TAL TECH

DOCTORAL THESIS

Applying Privacy-Preserving Computations in Digital Identity

Jelizaveta Vakarjuk

TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY
TALLINN 2026

TALLINN UNIVERSITY OF TECHNOLOGY
DOCTORAL THESIS
70/2026

Applying Privacy-Preserving Computations in Digital Identity

JELIZAVETA VAKARJUK

TAL TECH PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Software Science
This dissertation was accepted for the defence of the degree 08/09/2026

Supervisor: Prof. Dr. Ahto Buldas
Department of Software Science, School of Information Technologies
Tallinn University of Technology
Tallinn, Estonia

Co-supervisor: Dr. Alisa Pankova
Cybernetica AS
Tartu, Estonia

Opponents: Prof Dr. Olivier Blazy
École Polytechnique
Palaiseau, France

Prof Dr. Berry Schoenmakers
TU Eindhoven
Eindhoven, Netherlands

Defence of the thesis: 26/10/2026, Tallinn

Declaration:
Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology has not been submitted for doctoral or equivalent academic degree.

Jelizaveta Vakarjuk

---
signature



Copyright: Jelizaveta Vakarjuk, 2026
ISSN 2585-6898 (paperback)
ISBN 978-9916-80-569-5 (paperback)
ISSN 2585-6901 (PDF)
ISBN 978-9916-80-568-8 (PDF)
https://doi.org/10.23658/taltech.70/2026
Printed by Koopia Niini & Rauam

Vakarjuk, J. (2026). *Applying Privacy-Preserving Computations in Digital Identity* [TalTech Press]. https://doi.org/10.23658/taltech.70/2026

TALLINNA TEHNIKAÜLIKOOL
DOKTORITÖÖ
70/2026

Privaatsust säilitavate tehnoloogiate kasutamine digitaalidentiteedis

JELIZAVETA VAKARJUK

TAL TECH
KIRJASTUS

Contents

List of Publications 7

Author's Contributions to the Publications 8

Abbreviations 9

1 Introduction 10
1.1 Background and motivation 10
1.2 Approach and summary of results 11

2 Related work 15
2.1 EU Digital Identity Wallet 15
2.1.1 Credential formats 16
2.2 Attribute management systems 16
2.3 Threshold/blind decryption 17
2.4 Comparison of certified systems 18

3 Preliminaries 19
3.1 Notation 19
3.2 Game based definitions 19
3.2.1 Algebraic Group Model 20
3.2.2 Security assumptions 21
3.2.3 Random Oracle Model 22
3.3 Digital signatures 22
3.4 Public key encryption 23
3.5 Commitment schemes 25
3.6 Zero-knowledge proofs 25
3.7 Searchable Symmetric Encryption 27
3.8 Universal Composability 29
3.9 Types of corruption 30
3.10 BPMN 30

4 A Cloud Privacy-Preserving Attribute Management System 32
4.1 Plain model 32
4.2 System with searchable symmetric encryption 34
4.3 System with threshold decryption 35
4.4 Comparison and discussion 36
4.5 Chapter summary 37

5 Privacy-preserving server-supported decryption 38
5.1 Supporting algorithms 38
5.2 Ideal functionality 41
5.3 Description of the scheme 44
5.4 Security 47
5.5 Performance 50
5.6 Discussion 50
5.7 Chapter summary 50

6 Privacy-preserving attribute management system 51

6.1 Design principles and threat model 51
6.2 Ideal functionality 52
6.2.1 Properties of ideal functionality 54
6.3 System definition 59
6.4 Security 65
6.5 Performance 67
6.6 Chapter summary 68

7 Comparison-Based Methodology for the Security Assurance of Novel Systems ... 69
7.1 Methodology 69
7.2 Comparison of authentication methods 71
7.3 Comparison of private key protection 76
7.3.1 Description of compared systems 76
7.3.2 Analysis 78
7.4 Other applications 79
7.5 Chapter summary 80

8 Conclusions 81

List of Figures 83

List of Tables 84

References 85

Acknowledgements 93

Abstract 94

Kokkuvõte 96

Appendix 1 99

Appendix 2 111

Appendix 3 129

Appendix 3 151

Appendix 4 221

Appendix 6 243

9 Formal definition of Prettiness system 262

Curriculum Vitae 270

Elulookirjeldus 273

List of Publications

The present Ph.D. thesis is based on the following publications that are referred to in the text by Roman numbers.

I A. Kalu, B. C. Kus, P. Laud, K. L. Leung, N. Snetkov, and J. Vakarjuk. CACS: A cloud privacy-preserving attribute management system. In *Proceedings of the 18th International Conference on Availability, Reliability and Security, ARES 2023, Benevento, Italy, 29 August 2023- 1 September 2023*, pages 122:1–122:9. ACM, 2023

II P. Laud, A. Pankova, and J. Vakarjuk. Privacy-preserving server-supported decryption . In *2025 IEEE 38th Computer Security Foundations Symposium (CSF)*, pages 48–63, Los Alamitos, CA, USA, June 2025. IEEE Computer Society

III J. Vakarjuk and A. Pankova. Prettiness – privacy preserving attribute management system, 2026 (preprint)

IV P. Laud and J. Vakarjuk. A comparison-based methodology for the security assurance of novel systems. In S. K. Katsikas, F. Cuppens, C. Kalloniatis, J. Mylopoulos, F. Pallas, J. Pohle, M. A. Sasse, H. Abie, S. Ranise, L. Verderame, E. Cambiaso, J. M. Vidal, M. A. S. Monge, M. Albanese, B. Katt, S. Pirbhulal, and A. Shukla, editors, *Computer Security. ESORICS 2022 International Workshops - CyberICPS 2022, SECPRE 2022, SPOSE 2022, CPS4CIP 2022, CDT&SECOMANE 2022, EIS 2022, and SecAssure 2022, Copenhagen, Denmark, September 26-30, 2022, Revised Selected Papers*, volume 13785 of *Lecture Notes in Computer Science*, pages 625–644. Springer, 2022

V J. Vakarjuk, A. Pankova, and A. Kalu. Comparison of private key protection techniques for digital identity wallets. In *2026 IEEE European Symposium on Security and Privacy Workshops*, pages 396–412, 2026

Author's Contributions to the Publications

I In I, I was the main author, proposed attribute management systems, carried out the initial analysis and comparison, prepared the figures, and wrote the manuscript.

II In II, I contributed to the security definition and proofs of the proposed blind threshold decryption scheme, and contributed to the writing to the corresponding sections.

III In III, I was the main author, created the list of functional requirements and threat model, proposed attribute management system, defined security for the proposed system, wrote the security proof, prepared the figures, and wrote the manuscript.

IV In IV, I contributed to the creation of comparison methodology, and example comparison of smartcard-based and threshold cryptography based authentication.

V In V, I was the main author, performed comparison and analysis of private key protection mechanisms for identity management systems, prepared the figures, and wrote the manuscript.

Abbreviations

AGM	Algebraic Group Model
AMS	Attribute Management Service
ARF	Architecture Reference Document
BPMN	Business Process Model and Notation
CDH	Computational Diffie Hellman
DCR	Decisional Composite Residuosity
DEM	Data Encryption Mechanism
DLP	Discrete Logarithm Problem
DVNIZK	Designated Verifier Non-Interactive Zero-Knowledge
DVZKP	Designated Verifier Zero-knowledge proof
EUDIW	EU Digital Identity Wallet
GGM	Generic Group Model
HSM	Hardware Security Module
HVZK	Honest Verifier Zero Knowledge
IND-CCA	Indistinguishability under Chosen Ciphertext Attack
KEM	Key Encapsulation Mechanism
NIZK	Non-interactive zero-knowledge
PPT	Probabilistic Polynomial Time
RAMP	Rating Maintenance Phase
ROM	Random Oracle Model
RP	Relying Party
SD-JWT	Selective Disclosure for the JSON Web Token
SE	Secure Element
SSE	Searchable Symmetric Encryption
TCSEC	Trusted Computer System Evaluation Criteria
UC	Universal Composability
UF-CMA	Unforgeability under Chosen Message Attack
VC	Verifiable Credential
WSCD	Wallet Secure Cryptographic Device
ZKP	Zero-knowledge proof

1 Introduction

1.1 Background and motivation

Our interactions in the digital world heavily depend on the usage of digital identities. A digital identity can be viewed as a collection of claims about the person (attributes) that serves to represent a person in the digital space. These attributes may include various data including a unique identification code, a name, citizenship, biometric data, information about education and other data that authenticates and authorizes a user's online presence and activities. A collection of attributes describing a person is a credential. A person uses their digital identity on a daily basis to perform digital interactions with a variety of platforms, services and applications. Usually, there are the following parties involved in digital identity management systems:

- User – a party who is using the identity management system, has a set of attributes connected to them.
- Issuer – a party who is issuing a credential attestation (the cryptographic proof that a credential was issued by a trusted issuer) to the user.
- Relying Party (RP) – a party who requests and verifies the user's credentials.

The data flow between the main parties of the system is depicted in Figure 1. In this work,

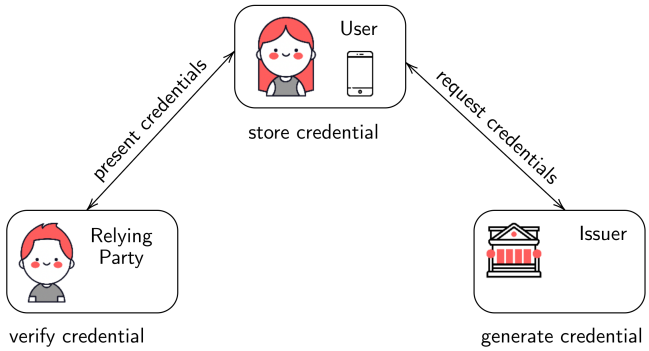


Figure 1: Main parties of the identity management system

we are using terminology consistent with the Verifiable Credentials Data Model [78] to describe the identity management system. Verifiable credentials (VC) are statements about the user's attributes that are digitally signed by the issuer. A verifiable presentation is a presentation of credential information to the RP such that the authorship of the credential data can be trusted after cryptographic verification. A verifiable presentation might contain the verifiable credential itself. Users may hold a large amount of credentials from different issuers. On the request from the relying parties, users are expected to create verifiable presentations of those credentials. The massive amount of interactions using digital identities raises concerns about the privacy of this data. Privacy concerns include, for example, whether the issuer of the credential learns to which RP the credential gets presented, whether the RP should learn all the data included in the credential or just some parts of it, etc. To address these issues, various cryptographic primitives can be used to minimise the amount of privacy leakage. Examples of cryptographic primitives that allow to achieve privacy properties include blind signatures [21], zero-knowledge proofs [14],

searchable symmetric encryption [70], blind decryption [33], etc. Depending on the exact privacy requirements of the system and the system architecture, different primitives may be chosen. Many of those primitives require the user to securely store cryptographic material. Therefore, another important concern to address is the storage of the cryptographic private keys on the user's devices. This can be done by either storing keys inside a secure element or using threshold cryptography[62]. In threshold cryptography, a private key is split between several parties and a certain subset of parties is required to cooperate to perform operations with the secret key (decryption, signing). An example of using threshold cryptography is the Smart-ID[1] signing and authentication service widely used in the Baltic states. The system relies on a server-supported RSA signature scheme [15]. Another example of using threshold cryptography for storing secret keys on a mobile device is the Yivi application, which is an identity-wallet solution[2].

The need for privacy-preserving digital identity management solutions is also dictated by the recent updates in the regulatory framework. The eIDAS2 regulation defines a concept called the European Digital Identity Wallet (EUDIW) that aims to provide users with a way to manage and present their credentials to relying parties, while trying to ensure interoperability of solutions between EU member states. According to the regulation, all EU member states should provide their residents with a digital identity wallet solution.

If we want to instantiate the identity management system with a threshold signature instead of a regular one to improve security guarantees, we would need to revise the certification of the system. A common way to ensure that the information system satisfies the defined security requirements is to go through the certification process. The certification process itself is very resource-consuming and requires a detailed analysis of the system components, functionality, security requirements. However, if the system under consideration relies on a novel technology or cryptographic primitive, such as threshold cryptography, the process of certification becomes even more complicated. Therefore, it would be helpful to have a simpler way of comparing a novel technology with the old standard technology in order to ease reasoning about the security of the novel systems.

1.2 Approach and summary of results

Identity management systems offer a solution that allows users to prove ownership of their attributes to relying parties. From the requirements posed on the system by the eIDAS2 regulation, one of the most important ones is selective disclosure – allowing the user to selectively disclose specific attributes for sharing with the RP. While many systems have been proposed in the literature, and some systems have practical implementations, these solutions often store attributes locally, on user devices. This thesis focuses on developing a centralised digital identity management system that allows users to present their credentials to relying parties in a privacy-preserving manner. The system introduces a centralised Attribute Management Service (AMS) server that helps users with interactions involving their credentials. The parties involved in the identity management system are depicted in Figure 2. Additionally, we propose cryptographic components that can be used in such a centralised identity management system. This work aims to answer the following question:

How to design privacy-preserving identity management systems?

RQ1 Which information should be disclosed to the issuer and relying party in the identity management systems?

[1] `https://www.smart-id.com`

[2] `https://docs.yivi.app/keyshare-protocol`

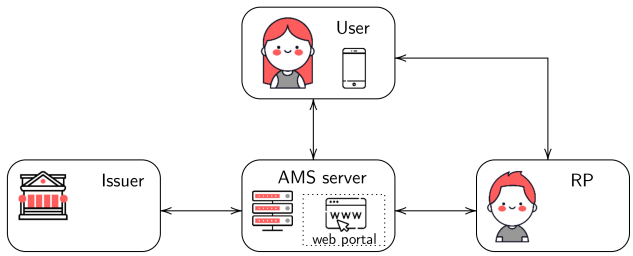


Figure 2: Main parties of the identity management system developed in this work

RQ2 What are advantages and disadvantages of storing the data on the user's device and the server?

RQ3 How to protect storage, retrieval and revocation of attributes?

RQ4 How to analyse security of a proposed identity management system?

We split the requirements for the identity management system, developed in this work, into essential and additional. The essential requirements are the basic requirements that the system needs to satisfy from the functional and security perspective. The additional requirements are harder to achieve, but help to improve the system's security and privacy guarantees. Essential requirements, motivated by the survey of Maximilian Richter et al. [59] include:

- **Completeness** – a valid credential should be accepted.
- **Soundness** – an invalid credential should be rejected. The RP should be able to verify that the presented credentials are valid and belong to the same person and that the issuers can be trusted.
- **Unforgeability of credentials** – a valid credential can only be created by the corresponding issuer.
- **Unforgeability of presentation** – a valid credential presentation can only be created by the corresponding user. Meaning that the user's credential should not be disclosed to the RP without the user's consent.
- **Freshness of the presentation** – every presentation has to be created fresh for every verification.
- **Impossibility to duplicate credentials** – no one (except the issuer) may create a valid copy of a credential. This includes the ability of one user to present the same credential from multiple devices. However, the problem of **non-transferability** of the credentials is deeper, as it would require introducing a mechanism that prevents a user from sharing their device with a friend.
- **Revocation** – a user should be able to revoke their credentials. Additionally, the issuer should be able to revoke the user's credentials without the user actively participating in the revocation protocol. An alternative approach to a separate revocation functionality is to set **short validity** time for the credentials, requiring credentials to be frequently regenerated. However, we believe that introducing separate revocation functionality is essential for the usability of the attribute management system in practice.

- **Efficiency** – the cryptographic components of the system should be efficient enough to use the system in the real-life applications. The focus of this thesis is on validating the novel cryptographic privacy-preserving mechanisms of the system. While efficiency is considered, the primary objective is to establish a secure and functional proof of concept, with further performance optimisation as a direction for subsequent work.

Additional requirements include:

- **Selective disclosure** – the user should be able to present only certain attributes of a credential.
- **Issuer participation** – the issuer of the credential should not be involved in the process of presenting credential to the RP.
- **Standard compliance** – the system should be compliant with standard credential presentation formats like mobile Driver's License [37], or Selective Disclosure for the JSON Web Token (SD-JWT) [25]. In this work, we do not create an identity management system fully compatible with the standards. Rather, in our design we stay as close as possible to the message flow and formats defined by those standards.

We do not set the goal to achieve:

- **Unlinkability of presentations** – the verifier should not be able to correlate several presentations of the same credential. In this work, we are considering the setting where the relying party *should* learn the user's identity, so we are not aiming for this property. This is in accordance with (Article 5a, §16), as in this case the user *explicitly* authorises linkability. We see our system in the context of legal transactions by officially recognised issuers, like presenting driving/fishing licence, or getting access to a building. We do not aim to use it in anonymous transactions (like shop purchases).
- **Repudiation** – RP should not be able to prove the authenticity of verified credential presentation to a third party. In this work, we are not aiming for this property. In contrary, we believe that we may need non-repudiation for later accountability of transactions (although we leave it out of scope of this paper).

An additional requirement introduced to the system is the central Attribute Management Service, due to the following reasons:

- This facilitates introducing the functionality of backing up a list of the user's attributes, attestations, and configurations. This requirement is also part of ARF (v. 2.8.0), defining the requirements and guidelines for implementing EU Digital Identity Wallet.
- The server can assist in managing revocations. Since revocation functionality is a crucial part of the identity management system, having a central server helps to facilitate the construction of such a system.
- A centralised server is needed for the server-supported decryption and signing functionalities.

The thesis is organized as follows. Chapter 2 introduces related work. Chapter 3 presents relevant notation, definitions and an explanation of the concepts used in this thesis. Chapter 4 proposes CACS – an initial design of the attribute management system for storing

credentials in the cloud, while satisfying privacy properties for users, and additional assurances for relying parties. Chapter 4 proposes three ways to construct such a system using different cryptographic primitives to protect users' identities and credentials presented to relying parties. The protocol for presentation of credentials to relying parties in the proposed systems follows the standard protocols of Mobile Driving Licence (mDL, ISO 18013-5). This chapter addresses research questions RQ1 and RQ3.

Chapter 5 introduces a threshold decryption protocol for two parties: the client and the server, with privacy against the server. The protocol is created to support the usage of decryption from mobile devices. Threshold cryptography serves as a measure of protection for the storage of private key material on the user's mobile device. Privacy against the server ensures that the supporting server does not learn which ciphertext is being decrypted by the user. This chapter addresses research question RQ3.

Chapter 6 defines an improved attribute management system – Prettiness, which improves the work done in CACS. The definition of the new system is more formal and detailed. It starts with defining the threat model and the system design principles. The new system discloses less information to the attribute management service while still helping users to manage their credentials. Additionally, it uses a revocation protocol that allows revocation by the user and the issuer. Further, the chapter introduces the security definition for the attribute management system in the form of an ideal functionality. This chapter addresses research questions RQ2, RQ3, RQ4.

Chapter 7 focuses on the problem of certification of the systems relying on novel cryptographic primitives. It proposes a methodology that allows to compare the already certified systems to the novel/improved ones that achieve the same functionality and share the same security target. This methodology allows to argue that the systems relying on non-standard cryptographic primitives allow to achieve at least the same security level as the systems relying on standard primitives. Additionally, it presents an example application of the proposed methodology to perform a comparison of private key protection mechanisms in identity management systems. This chapter addresses the RQ4 research question.

2 Related work

2.1 EU Digital Identity Wallet

The concept of the EU Digital Identity Wallet (EUDIW) is introduced by the eIDAS2 regulation [1]. The regulation defines a

> *fully mobile, secure and user-friendly*

wallet, that enables users to identify themselves to online services. The regulation requires that all member states should offer such an EUDI wallet to all their citizens and residents by the end of 2026. Further, the regulation lists security and privacy requirements for digital identity wallets, including

- *selective disclosure of data*, *generating pseudonyms and storing them encrypted*, *accessing a log of all transactions* (Article 5a, §4)
- *not allowing any party, after issuance of attestations of attributes, to obtain data that allows transactions or user behaviour to be tracked, linked or correlated unless explicitly authorised by the user* and *enabling privacy-preserving techniques which ensure unlinkability* (Article 5a, §16).

While developing the architecture reference framework [3] (ARF), which consists of common guidelines and best practices for implementing the European Digital Identity Framework, the EUDI wallet team of the European Commission invited subject-matter experts to participate in a meeting, where the current design of the EUDI wallet (ARF version 1.4.0[3]) was presented. Subject-matter experts were asked to provide feedback on the attestations and zero-knowledge proofs in the EUDI wallet. As a result, the feedback was published in [9], raising concerns about the proposed solution not being compliant with the properties stated above. In ARF version 2.7.3, the concerns defined by the experts have been acknowledged and some solutions aiming to mitigate linkability issues have been proposed. However, the design still relies on simpler salted-hash attestations, avoiding the introduction of zero-knowledge proofs.

Moreover, there is an important constraint on the cryptographic primitives that can be used to instantiate the EUDI wallet. The EU Member States rely on the set of algorithms provided by the Agreed Cryptographic Mechanisms for cross-border assurance [27]. Currently, this list includes only RSA, ECDSA, Schnorr, ML-DSA, SLH-DSA, XMSS, LMS digital signatures and does not include any advanced cryptographic algorithms such as those built on pairings. However, many existing anonymous credential systems [18, 60, 7] rely on pairing-based cryptography or non-standard signature schemes to guarantee privacy properties.

A further requirement is that the credential issued by the wallet must be bound to a hardware-protected device key and requires the Level of Assurance High Secure Elements [1] (§29). The ARF [3] lists the following options for the implementation of the Wallet Secure Cryptographic Device (WSCD):

1. a remote device, such as a Hardware Security Module (HSM), accessed over a network,
2. an external device, such as a smart card issued to the User specifically for this purpose,

[3] `https://eu-digital-identity-wallet.github.io/eudi-doc-architecture-and-reference-framework/1.4.0/arf/`

3. a component within the User device, such as a SIM, e-SIM, or embedded Secure Element,

4. a component embedded in the User device and accessed via an API provided by the operating system.

The first approach has been explored in the Germany Wallet example [4]. Another approach that is not listed in ARF [3], but explored in the literature is the usage of threshold cryptography, which allows the splitting of a user's private key between the mobile device and the supporting server [5].

2.1.1 Credential formats

There are different formats for attestations of attributes available. In this work, we follow closely the formats that are referenced within the EUDI Wallet ecosystem:

- mobile Driver's Licence (mDL) [37];
- Selective Disclosure JSON Web Tokens (SD-JWTs) [25];
- W3C Verifiable Credentials Data Model v2.0 [78].

The mDL format was originally developed as a standard for mobile driving licences (mDL) and mDL readers (relying parties). Its primary use case is proximity attestations presentation, where a wallet and a RP set up a communication channel using a QR code or NFC, to exchange and verify attestations. In this context, *mdoc* represents a verifiable presentation encoded in the mDL document format. By comparison, another presentation format is *pkpass*, which is used for passes in Apple Wallet (not specifically for credential presentations).

Selective Disclosure JSON Web Tokens (SD-JWTs) is a token format designed for privacy-preserving credentials. An SD-JWT contains a standard signed JWT payload where each claim (representing a user attribute) can be hashed with a salt and paired with separate *disclosures* that contain the plaintext values. This mechanism enables the user to reveal only selected attributes to each RP. The primary goal of this format is to enable remote credential presentations, where minimizing the amount of shared personal data is important. It enables issuers to generate credentials as JWTs but allows users to present only the necessary attributes to RPs, preventing oversharing of personal information. Both SD-JWT and mDL provide mechanisms for selective disclosure of user attributes.

The W3C Verifiable Credentials framework defines a general data model for digital credentials, providing a high-level structure of identity management systems. However, the model itself is format-agnostic and does not prescribe specific cryptographic mechanisms that should be used to instantiate the system. To implement a complete system, additional specifications are required to define security mechanisms, proof formats, and signature schemes.

2.2 Attribute management systems

Several previous studies have focused on credential systems that maintain the privacy of users. Among these systems is Idemix, an anonymous credential system which allows users to prove possession of credentials without disclosing any additional information

[4] `https://bmi.usercontent.opencode.de/eudi-wallet/wallet-development-documentation-public/v0.5.1/`

[5] `https://github.com/cleverbase/scal3`

(just the fact they possess a credential) [17]. There are several real-world implementations of Idemix, such as PrimeLife[6], and the Yivi app with the IRMA backend system[7]. An additional example is the U-Prove system that enables minimal disclosure of private information and unlinkability, which means that an issuer cannot link different transactions to a particular individual [57].

Further, several more recent works, such as [49, 51], focus on privacy-preserving identity systems. Lee et al. [49] proposed a privacy-preserving identity model based on zero-knowledge proofs that achieves data minimisation. To eliminate a single point of failure and to increase system security, Moreno et al. [51] used threshold cryptography to distribute tasks among several identity providers.

In most of these above-mentioned systems, one of the main cryptographic building blocks is zero-knowledge proofs (ZKP) and digital signatures (that support combination with ZKPs). However, it is also possible to introduce more privacy-preserving properties to the system by adding other cryptographic techniques such as searchable encryption, threshold decryption, and proxy re-encryption.

Recently, there have been various cloud-based attribute-management systems proposed, all trying to provide additional privacy guarantees to the system. The work by Krenn et al. [44] proposes a cloud attribute-based credential system that uses proxy re-encryption [12] and structure-preserving signatures [4]. Later, an improved version of this system [44] was introduced by Haböck et al. [35], reducing the ambiguity of the original work but introducing additional cryptographic complexity to the solution with blind signatures and additional zero-knowledge proofs. A similar approach to build an attribute-based credential system was used in the CREDENTIAL project [8] [40, 39, 43, 77].

There exist a number of proposed systems for decentralized management of credentials [5, 50, 36] that rely on blockchain technologies while still delivering privacy to the users. The goals of those systems are different from the ones this work tries to achieve, aiming to centralize the collection of attributes, management of cryptographic keys, and maintenance of trust relationships. In addition, these systems lack standard compliance, which is one of the important properties that this thesis tries to maintain.

2.3 Threshold/blind decryption

There are several use-cases when having a regular public key encryption scheme such as RSA [61], ElGamal [31] or Paillier [54] is insufficient, since the ability to decrypt ciphertexts is given to a single party. In order to distribute the trust between several system components or entities, threshold cryptography can be used to split the decryption key between several parties. There are various encryption schemes with threshold decryption proposed in the literature; some of those provide security only against chosen-plaintext attacks [62], some provide stronger security against chosen-ciphertext attacks [63, 28].

Another primitive that is built on top of public key encryption is an encryption scheme with blind decryption functionality. In these schemes, there is a decrypting party holding the private key, and the client issues blind decryption queries that do not leak the actual ciphertext nor the plaintext to the decrypting party. There are two relevant previous works that focused on blinding ciphertexts before sending them for decryption to the holder of the private key (share). Green [33] proposed a scheme where the client sends decryption requests to a server that holds the private key. Blazy et al. [13] proposed an encryption

[6] `http://primelife.ercim.eu/about/factsheet`

[7] `https://irma.app/docs/what-is-irma/`

[8] `https://credential.eu/`

system where the decryption capability and authorization are shared among three parties with unequal roles, offering privacy against the server.

2.4 Comparison of certified systems

The certification of systems that are being updated/improved over time and the reuse of previous security evaluation results have been the topic of a statement from the Common Criteria (CC) Recognition Arrangement Management Committee [58]. There have been several methods proposed for using CC together with agile product development [64, 65], and also for the continued certification of various cybersecurity products [24]. However, there have been only a few attempts to show the security of one system by comparing it to another one that has already been evaluated to be secure [16]. An additional example is the Trusted Computer System Evaluation Criteria (TCSEC), where the system evaluation process included an additional Rating Maintenance Phase (RAMP). The goal of this phase is to provide an instrument that extends the security evaluation of a certified system to an updated version of that system by analysing changes that were introduced [52].

3 Preliminaries

3.1 Notation

We use $\{0,1\}^n$ to denote a binary string of length n. The notation $x \leftarrow\$ S$ means that x is obtained by sampling it uniformly at random from the set S and $y \leftarrow A(x)$ denotes that y is obtained as the output of an algorithm A run on an input x. We use $\mathbb{Z}_q$ to denote the integers modulo q. The $\perp$ symbol denotes rejection or an empty output in the protocol. The $\langle \mathsf{P}_1 | \mathsf{P}_2 \rangle$ denotes composition of a protocol P_1 with P_2. The symbol $\oplus$ denotes the exclusive-or of two values. We use the Iverson bracket notation $[\![P]\!]$ to encode a boolean condition, where the expression evaluates to 1 if the statement P is true and to 0 otherwise. We use the notation $(\cdot)$ to denote a placeholder argument of a function or an algorithm.

Throughout this work, we use the notion of negligible value, which is formally defined as follows.

Definition 3.1. A function $\mu(n) : \mathbb{N} \to \mathbb{R}$ so that for every $c \geq 0$, there exists $N \in \mathbb{N}$ so that $|\mu(n)| < n^{-c}$ for every $n > N$.

Intuitively it means that a negligible quantity is so small that it can be safely ignored in practice. As the security parameter of the scheme grows, it vanishes faster than any inverse polynomial, making successful attacks very unlikely.

In this work, we define security of the schemes with respect to a probabilistic polynomial time adversary (PPT) that is defined as follows.

Definition 3.2. A probabilistic polynomial-time (PPT) adversary is a randomized algorithm $\mathcal{A}(\lambda, \cdot)$ parametrised by a security parameter λ, such that there is a polynomial p, so that for every input x, the running time of $\mathcal{A}(\lambda, \cdot)$ is upper bounded by $p(\lambda)$.

Such adversary models any efficient real-world attacker, where *probabilistic* means the adversary can use randomness in its computation, allowing it to make random choices or guesses. Polynomial-time means its running time grows at a reasonable rate (i.e., not exponentially) with respect to the security parameter λ, capturing the idea that the adversary runs in computationally feasible time.

3.2 Game based definitions

The game-based approach is one of the most widely adopted ways of performing security analysis of cryptographic schemes [11]. In this framework, security of the scheme is defined through a game between a challenger and an adversary. An intuitive illustration of the security game is presented in Figure 3. The game consists of a set of well-defined procedures that specify how the cryptographic scheme is used and how the adversary can interact with it. The adversary is modelled as a program, consisting of a single procedure. The main procedures in the game are oracle interfaces, functions that the adversary can query. The goal of these oracles is to simulate access to various parts of the cryptographic system, such as encryption, decryption, or signing operations. The adversary interacts with the system by making queries to these oracles, providing inputs and receiving outputs. The goal of the adversary is to exploit the scheme in some way (e.g., to distinguish between two scenarios or to forge a signature), while the game defines the rules and constraints for this interaction.

Further in this work, we write $\mathcal{A}^{\mathsf{O}(\cdot)}(\mathsf{in})$ to denote an adversary accessing oracle $\mathsf{O}(\cdot)$ and running on input in. In the security definitions described in this work, we use the term *experiment* to define a security game between a challenger and an adversary.

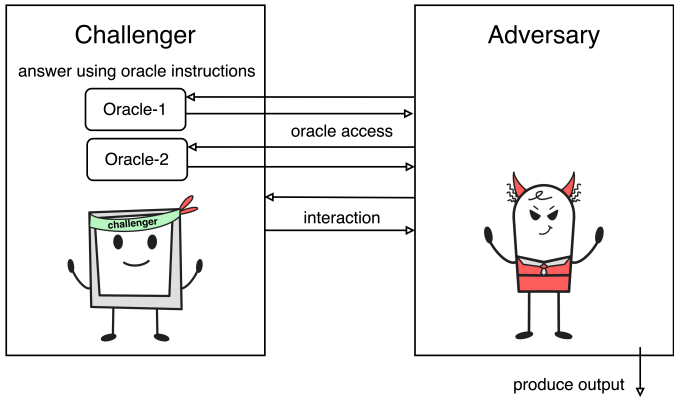


Figure 3: Security game overview

The first papers following the game-based methodology for the security proofs can be traced back to works by Goldwasser and Micali [32] and Yao [79]. Later, Kilian and Rogaway [41] formalized the methodology by defining security games through code-like specifications. This approach can be summarized in the following steps:

- Assume that we want to find an upper bound on the advantage of the adversary $\mathcal{A}$ in attacking the scheme that we are analysing. Let us define a special flag bad that gets set once something unusual or undesirable happens during the simulation (e.g. a collision, guess, or inconsistency).
- The advantage of the adversary is defined as the difference between the probability that $\mathcal{A}$ outputs 1 in each of two distinct scenarios or *worlds*.
- Define games though the code-like notation, this includes describing how to initialise the scheme's variables and how to interact with the adversary. Define two games, Game 0 and Game 1, each modelling the interaction between the adversary and the system in its respective world. These games are syntactically identical except for the statements that determine whether a flag bad is set to true. The adversary's advantage is then upper-bounded by the probability that bad is set during the execution of either game.
- The proof proceeds by transforming one game into another *intermediate game* through a sequence of small modifications. Each intermediate game is designed so that the probability of setting bad either remains unchanged or changes by a negligible amount. The goal of introducing those intermediate games is to gradually transform the initial game into a final game whose behaviour is easily analysable.
- By introducing those intermediate games, we obtain a sequence of games from Game 0 to the final game. The advantage of the adversary is bounded by the probability that bad is set in the final game.

3.2.1 Algebraic Group Model

The Algebraic Group Model (AGM) [29] is a theoretical framework used to analyse the security of cryptographic schemes that rely on algebraic group structures, such as those used in discrete logarithm–based cryptography and elliptic curve cryptography.

Usually, security proofs are conducted in the standard model, which makes no idealized assumptions. AGM is considered to have stronger assumptions than the standard model, but weaker than the Generic Group Model (GGM). One of the main differences between GGM and AGM is that AGM allows the analysis of the security of algorithms that exploit the structure of the group. In the AGM, adversaries are required to behave algebraically: whenever an adversary outputs a group element, it must also provide the representation. The representation is defined as a linear combination of previously seen group elements. Intuitively, the adversary is required to explain how each element it outputs depends algebraically on the earlier ones. This additional requirement from the adversary allows security proofs to track dependencies between the group elements and extract values that would be impossible to get otherwise.

3.2.2 Security assumptions

Unlike information-theoretic cryptography, which provides unconditional security, most modern cryptographic constructions are only computationally secure. This means that their security relies on the assumption that certain underlying mathematical problems are computationally hard to solve. As a result, the security of the cryptographic scheme is guaranteed only against adversaries with bounded computational resources.

Definition 3.3 (Discrete Logarithm Problem (DLP))**.** Let $\mathbb{G}$ be a cyclic group of prime order p with generator g. The discrete logarithm problem is defined as follows. Given (g, h) for $h = g^x$ where $x \leftarrow\$ \mathbb{Z}_p$, compute x.

Definition 3.4 (Computational Diffie-Hellman (CDH) Problem)**.** Let $\mathbb{G}$ be a cyclic group of prime order p with generator g. The Computational Diffie-Hellman problem is defined as follows. Given (g, g^x, g^y), where $x, y \leftarrow\$ \mathbb{Z}_p$, compute g^{xy}.

The hardness of the discrete logarithm problem implies the hardness of the Computational Diffie-Hellman problem. If one can solve DLP, then given (g, g^x, g^y), one can recover x (or y) and compute

$$g^{xy} = (g^y)^x.$$

Definition 3.5 (One-more computational Diffie-Hellman Problem)**.** Let $\mathbb{G}$ be a cyclic group of prime order p with generator g. In the *one-more (static) CDH problem with tests*, the experiment is defined as follows. Given

1. (g, g^x) with $x \leftarrow\$ \mathbb{Z}_p$
2. independent random elements $h_0, \ldots, h_n \in \mathbb{G}$
3. the exponentiation oracle $(\cdot)^x$
4. the oracle $\mathsf{Check}(\cdot)$, where $\mathsf{Check}(h)$ returns i such that $h = h_i^x$, and $\perp$ if there is no such i

The goal of a solver is to invoke $\mathsf{Check}(h_i^x)$ for all $i \in \{0, \ldots, n\}$, while making at most n calls to the oracle $(\cdot)^x$. The calls to the Check-oracle are not restricted in number.

It can be noticed that the one-more CDH problem generalizes the standard CDH problem. In particular, when $n = 0$, the adversary is given (g, g^x) and a single challenge element $h_0 \in \mathbb{G}$, with no access to the exponentiation oracle $(\cdot)^x$. The goal of the solver is then to compute h_0^x, which is exactly an instance of the Computational Diffie-Hellman problem. Hence, CDH can be viewed as a special case of the one-more CDH problem.

While the Check-oracle may give the solver some extra power in the Standard Model, Bauer et al. [8, Thm. 5] show that nothing additional is gained in the Algebraic Group Model (AGM) [30].

3.2.3 Random Oracle Model

The Random Oracle Model (ROM) is a heuristic that is used to model the use of hash functions [10]. The idea is to model a hash function H as if it were a truly random function $\mathsf{O}(\cdot) : \mathcal{X} \to \mathcal{Y}$, where for each $x \in \mathcal{X}$, the output $\mathsf{O}(x)$ is sampled uniformly at random from $\mathcal{Y}$. Then the challenger uses $\mathsf{O}(\cdot)$ in place of H to answer the adversary's queries and perform computations. The adversary is allowed to obtain the output of $\mathsf{O}(\cdot)$ for an arbitrary input of the adversary's choosing. The function $\mathsf{O}(\cdot)$ is called a random oracle and the security of the scheme analysed in this setting is said to hold in the random oracle model.

While security proofs in the Random Oracle Model are mathematically rigorous, they are heuristic and do not imply security for concrete instantiations with specific hash functions. Instead, they serve to exclude generic attacks that would be possible if the hash function were modelled as a random oracle.

To analyse the security of system S in the random oracle model, the attack game defining security is modified so that the hash function is replaced by a random oracle $\mathsf{O}(\cdot)$. Both the adversary and the challenger have oracle access to this function. The ROM supports security proofs for many modern protocols by allowing reductions to use properties that are only partially achievable in the standard model. One such useful property is *programmability* that allows one to program the random oracle – adaptively choose its outputs in response to adversarial queries.

3.3 Digital signatures

Definition 3.6 (Digital signature scheme)**.** A digital signature scheme consists of the following sets, algorithms, and protocols. Here and in the definitions below, all algorithms and sets are parametrised by λ , but we omit it from the notation for readability.

- Sets of signatures S, public keys PK, and the private keys SK.
- Key generation algorithm KeyGen, returns $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{SK} \times \mathsf{PK}$.
- Signing algorithm Sign takes as input $m \in \mathsf{M}$, $\mathsf{sk} \in \mathsf{SK}$ and $\mathsf{pk} \in \mathsf{PK}$ and returns $\sigma \in \mathsf{S}$.
- Verification algorithm Ver takes as input a public key $\mathsf{pk} \in \mathsf{PK}$, a message $m \in \mathsf{M}$ and signature $\sigma \in \mathsf{S}$ and returns a verification outcome $b \in \{0, 1\}$.

Here and in the definitions below, all algorithms are parametrised by λ, but we omit it from the notation for readability.

A *blind signature* is a cryptographic primitive that is widely used to enhance privacy of the systems. The signing protocol is defined between the two parties – Sender and Signer. Sender wants to sign a message and the Signer holds the signing private key. In a blind signature scheme, Sender blinds the input message and sends it to the Signer. The Signer signs the blinded message using their private key and sends the result to the Sender. The Sender unblinds the received signature and outputs it. In this protocol, the Signer does not learn the content of the message being signed.

Definition 3.7 (Server-supported digital signature scheme)**.** A digital signature scheme with the client-server signing consists of the following sets, algorithms, and protocols, parametrised with the security parameter λ.

- Sets of signatures S, public keys PK, the client's private keys SK_C, and the server's private keys SK_S.

- Key generation protocol $\langle \mathsf{KeyGen}_C | \mathsf{KeyGen}_S \rangle$, run by both parties. It returns a key pair $(\mathsf{sk}_C, \mathsf{pk}) \in \mathsf{SK}_\mathrm{C} \times \mathsf{PK}$ to the client, and a key pair $(\mathsf{sk}_S, \mathsf{pk}) \in \mathsf{SK}_\mathrm{S} \times \mathsf{PK}$ to the server.

- Signing protocol $\langle \mathsf{Sign}_C | \mathsf{Sign}_S \rangle$ run by the client and the server. Client's inputs to Sign_C are $m \in \mathsf{M}$, $\mathsf{sk}_C \in \mathsf{SK}_\mathrm{C}$, and $\mathsf{pk} \in \mathsf{PK}$. Server's inputs to Sign_S are $m \in \mathsf{M}$, $\mathsf{sk}_S \in \mathsf{SK}_\mathrm{S}$ and $\mathsf{pk} \in \mathsf{PK}$. The protocol returns either $\sigma \in \mathsf{S}$ or the failure notice $\perp$ to the client and the server.

- Verification algorithm Ver takes as input a public key $\mathsf{pk} \in \mathsf{PK}$, a message $m \in \mathsf{M}$ and signature $\sigma \in \mathsf{S}$ and returns a verification outcome $b \in \{0, 1\}$.

It is possible to construct a variant of the server-supported digital signature scheme, where the message being signed is hidden from the supporting server [69]. The scheme with such a property is useful in the context of identity management systems, where on one hand, we want to ensure the security of the user's private key and on the other hand, we want to hide the content of the message from the supporting server.

The intuition behind the security requirement for the digital signatures is that only the signer should be able to generate valid signatures. The formal definition is presented below.

Definition 3.8 (UF-CMA)**.** The digital signature scheme SIGN provides unforgeability under chosen message attacks, if the experiment $\mathsf{UF\text{-}CMA}^{\mathcal{A},\mathsf{SIGN}}$ is successful with at most negligible probability for all PPT adversaries $\mathcal{A}$. In the experiments introduced below, we assume $\mathcal{A}$ remembers its internal state.

Experiment $\mathsf{UF\text{-}CMA}^{\mathcal{A},\mathsf{SIGN}}$	$\mathsf{Sign}(m, \mathsf{sk})$
$\mathsf{M} \leftarrow \emptyset$	$\mathsf{M} \leftarrow \mathsf{M} \cup \{m\}$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{SIGN.KeyGen}()$	**return** $\sigma \leftarrow \mathsf{SIGN.Sign}(m, \mathsf{sk})$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,\mathsf{sk})}()$	
return $[\![m^* \notin \mathsf{M}]\!] \wedge \mathsf{SIGN.Ver}(\mathsf{pk}, m^*, \sigma^*)$	

An experiment $\mathsf{UF\text{-}CMA}^{\mathcal{A},\mathsf{SIGN}}$ can intuitively be described as follows. The challenger generates a keypair for the signature scheme consisting of the public key pk and private key sk. The adversary receives a public key pk from the challenger and is allowed to make signing requests. The goal of the adversary is to produce a signature message pair (m^*, σ^*) that passes the verification algorithm and has not been queried through the signing queries before.

3.4 Public key encryption

Definition 3.9 (Public key encryption)**.** A public key encryption scheme consists of the following sets, algorithms, and protocols, parametrised with the security parameter λ.

- Sets of ciphertexts C, public keys PK, and private keys SK.

- Key generation algorithm KeyGen. It returns $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{SK} \times \mathsf{PK}$.

- Encryption algorithm Enc takes as input a public key $\mathsf{pk} \in \mathsf{PK}$, a plaintext $m \in \mathsf{M}$ and returns a ciphertext $c \in \mathsf{C}$.

- Decryption algorithm Dec takes a ciphertext $c \in \mathsf{C}$, private key $\mathsf{sk}_C \in \mathsf{SK}_\mathrm{C}$, and public key $\mathsf{pk} \in \mathsf{PK}$. It returns either $m \in \mathsf{M}$ or the failure notice $\perp$.

Some encryption schemes have *homomorphic properties*, which allows performing operations like addition or multiplication on ciphertexts such that, after decryption, the outcome is equivalent to applying those (or corresponding) operations to the plaintexts. The Paillier encryption scheme [54] is a well-known example of an additively homomorphic encryption scheme, allowing efficient computation on the encrypted data. The security of the Paillier cryptosystem builds on the Decisional Composite Residuosity (DCR) assumption.

A primitive that is connected to the public key encryption and allows the parties to establish the shared secret is a Key Encapsulation Mechanism (KEM).

Definition 3.10 (Key encapsulation mechanism)**.** A key encapsulation mechanism consists of the following sets, algorithms, and protocols, parametrised with the security parameter λ.

- Sets of encapsulations C, public keys PK, private keys SK, keys K
- Key generation algorithm KeyGen. It returns $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{SK} \times \mathsf{PK}$.
- Encapsulation algorithm Encaps takes as input a public key $\mathsf{pk} \in \mathsf{PK}$, and returns a key $K \in \mathsf{K}$ and encapsulation $c \in \mathsf{C}$ of K.
- Decapsulation algorithm Decaps takes an encapsulation $c \in \mathsf{C}$, private key $\mathsf{sk}_C \in \mathsf{SK}_\mathrm{C}$, and public key $\mathsf{pk} \in \mathsf{PK}$. It returns either $K \in \mathsf{K}$ or the failure notice $\perp$.

Definition 3.11 (Encryption scheme with the server supported decryption)**.** An encryption scheme with the client-server decryption consists of the following sets, algorithms, and protocols, parametrised with the security parameter λ.

- Sets of ciphertexts C, public keys PK, the client's private keys SK_C, and the server's private keys SK_S.
- Key generation protocol $\langle \mathsf{KeyGen}_C | \mathsf{KeyGen}_S \rangle$, run by both parties. It returns a key pair $(\mathsf{sk}_C, \mathsf{pk}) \in \mathsf{SK}_\mathrm{C} \times \mathsf{PK}$ to the client, and a key pair $(\mathsf{sk}_S, \mathsf{pk}) \in \mathsf{SK}_\mathrm{S} \times \mathsf{PK}$ to the server.
- Encryption algorithm Enc takes as input a public key $\mathsf{pk} \in \mathsf{PK}$, a plaintext $m \in \mathsf{M}$ and returns a ciphertext $c \in \mathsf{C}$.
- Decryption protocol $\langle \mathsf{Dec}_C | \mathsf{Dec}_S \rangle$, run by the client and the server. Client's inputs to Dec_C are $c \in \mathsf{C}$, $\mathsf{sk}_C \in \mathsf{SK}_\mathrm{C}$, and $\mathsf{pk} \in \mathsf{PK}$. Server's inputs to Dec_S are $\mathsf{sk}_S \in \mathsf{SK}_\mathrm{S}$ and $\mathsf{pk} \in \mathsf{PK}$. The protocol returns either $m \in \mathsf{M}$ or the failure notice $\perp$ to the client. It returns the success notice $\top$ or the failure notice $\perp$ to the server.

Definition 3.12 (IND-CCA)**.** The public key encryption scheme PKE provides indistinguishability against chosen ciphertext attacks, if the experiment $\mathsf{IND\text{-}CCA}^{\mathcal{A},\mathsf{PKE}}$ is successful with probability at most negligibly larger than $1/2$ for all PPT adversaries $\mathcal{A}$.

Experiment IND-CCA$^{\mathcal{A},\mathsf{PKE}}$	$\mathsf{Dec}(c,\mathsf{sk})$
$\mathsf{C} \leftarrow \emptyset$	**if** $c \in \mathsf{C}$, **return** $\perp$
$(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{PKE.KeyGen}()$	**return** $\mathsf{PKE.Dec}(c,\mathsf{sk})$
$(m_0,m_1) \leftarrow \mathcal{A}(\mathsf{pk})$	
$b \leftarrow\$ \{0,1\}$	
$c \leftarrow \mathsf{PKE.Enc}(\mathsf{pk},m_b), \mathsf{C} \leftarrow \mathsf{C} \cup \{c\}$	
$b^* \leftarrow \mathcal{A}^{\mathsf{Dec}(\cdot,\mathsf{sk})}(c)$	
return $b = b^*$	

An experiment IND-CCA$^{\mathcal{A},\mathsf{PKE}}$ can be intuitively described as follows. The challenger generates a key pair consisting of a public pk and private key sk and sends the public key to the adversary. The adversary selects two messages m_0, m_1 and sends them through an encryption query to the challenger. The challenger randomly chooses which message to encrypt by selecting bit b. The challenger sends a ciphertext c that corresponds to the encryption of message m_b to the adversary. The Adversary is allowed to make decryption queries to the challenger (but not allowed to submit c) and is asked to guess the bit b.

3.5 Commitment schemes

A commitment is a cryptographic scheme that enables a party, called the committer, to commit to a chosen value while keeping it hidden from others, with the ability to reveal it later. A commitment scheme typically consists of two algorithms:

1. a commitment algorithm, which on input a message m and randomness r produces a commitment $c \leftarrow \mathsf{Commit}(m,r)$;
2. a verification (or opening) algorithm, which checks that a given opening (m,r) corresponds to a commitment c.

The security of commitment schemes is captured by two main properties:

- **hiding** ensures that the commitment c reveals no information about the committed value m prior to the opening phase.
- **binding** ensures that, once a commitment c is generated, the committer cannot open it to two different values.

The simplest example of a commitment scheme is a hash-based commitment. To commit to a value m, the committer samples randomness r and computes the commitment as $c = H(m \,\|\, r)$, where H is a RO. To open the commitment, the committer reveals (m,r), and the verifier checks that $c = H(m \,\|\, r)$.

3.6 Zero-knowledge proofs

A broader class of protocols that allow a prover to convince a verifier of the validity of a statement without revealing any additional information beyond its correctness is called Zero-knowledge Proofs (ZKP). For a given statement x, the goal of the prover is to convince the verifier that $x \in \mathcal{L}$ and w is a witness proving that $x \in \mathcal{L}$. The security requirements for ZKPs are:

- **soundness** – it should be infeasible for any prover to make the verifier accept a statement that is not true.

- **zero-knowledge** (ZK) – ensures that the verifier learns nothing more than the truth of the statement from the proof. This requirement is defined through the existence of a special party called Simulator, that, given some extra information but not the secret input of the prover, can generate a proof that is indistinguishable from the proof that the honest prover generates.

A special type of ZKPs are Designated Verifier Zero-knowledge proofs (DVZKP). In this case, the proof can only be verified by a single party, who must be identified before the proof is produced. This designated verifier generates and holds the secret state. In the DVZKP setting, the soundness property only holds for a designated verifier that has a secret state.

A **Σ-protocol** is a protocol that allows a prover (P) to convince a verifier (V) that it knows a secret that satisfies some relation, without revealing any useful information to V about the secret. More formally, a $\Sigma-$protocol is a 3-move interactive proof for a relation $R \subseteq \mathcal{X} \times \mathcal{W}$. The statement $x \in \mathcal{X}$ (public input) and a witness $w \in \mathcal{W}$ (secret known only to the prover) satisfy $(x, w) \in \mathcal{R}$. The protocol between P and V (illustrated in Figure 4) proceeds as follows:

- To initialise the protocol, P computes a message α (commitment), and sends it to V;
- Upon receiving commitment α, V chooses a challenge β at random from a finite challenge space $\mathcal{CH}$, and sends β to P;
- Upon receiving challenge β, P computes a response γ, and sends γ to V;
- Upon receiving response γ, V outputs either accept or reject, which is computed as a function of the statement y and the conversation transcript (α, β, γ).

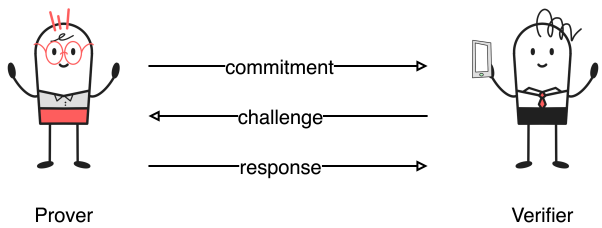


Figure 4: $\Sigma-$ protocol message flow

One of the most commonly used examples of $\Sigma-$protocols is Schnorr's $\Sigma-$protocol. Let $\mathbb{G}$ be a group where the discrete logarithm problem is hard. A statement for Schnorr's $\Sigma-$protocol is a group element $u \in \mathbb{G}$, and a witness for u is $x \in \mathbb{Z}_q$ such that $g^x = u$. One of the security requirements for a $\Sigma-$protocol is **special soundness**. A $\Sigma-$protocol provides special soundness if there is an efficient deterministic algorithm Ext (witness extractor) – whenever Ext is given as input a statement $x \in \mathcal{X}$, and two accepting conversation transcripts (α, β, γ) and $(\alpha, \beta', \gamma')$, with $\beta \neq \beta'$, algorithm Ext always outputs $w \in \mathcal{W}$ such that $(x, w) \in \mathcal{R}$ (meaning that w is a witness for x). This requirement ensures that no cheating prover can succeed without *knowing* a valid witness. This is a requirement for the Proof-of-Knowledge (PoK) systems.

The other security requirement is called **special honest verifier zero knowledge** (HVZK). Intuitively, the requirement says that for $(x, w) \in \mathcal{R}$, a conversation between P and V

should not reveal anything about the witness w. More formally, we say that a $\Sigma-$protocol is special honest verifier zero knowledge (special HVZK), if there exists an efficient probabilistic algorithm Sim(simulator) that takes as input $(x, \beta) \in \mathcal{X} \times \mathcal{CH}$, and satisfies the following properties:

- for all inputs $(x, \beta) \in \mathcal{X} \times \mathcal{CH}$, Sim always outputs a pair (α, γ) such that (α, β, γ) is an accepting conversation transcript for x;
- for all $(x, w) \in \mathcal{R}$, if we compute

$$\beta \leftarrow\$ \mathcal{CH}, (\alpha, \gamma) \leftarrow \mathsf{Sim}(x, \beta),$$

  then (α, β, γ) has the same distribution as that of a transcript of a conversation between P and V.

This definition assumes that the challenge β is sampled honestly uniformly at random from the challenge set. Since in $\Sigma-$protocols β is picked by the verifier, the protocol guarantees ZK as long as the verifier picks it honestly.

It is possible to transform an interactive $\Sigma-$protocol into a non-interactive using the Fiat-Shamir transform [26]. The idea is to generate a challenge using the hash function as $\beta \leftarrow H(x, \alpha)$, where x is a statement and α is a commitment. The verifier when needs to verify that (α, β, γ) is an accepting conversation transcript for x, where $\beta = H(x, \alpha)$.

3.7 Searchable Symmetric Encryption

The goal of the Searchable Symmetric Encryption (SSE) scheme is to allow the client making keyword queries on the encrypted data, stored on the server, ensuring a minimal amount of information leakage to the server in the process. SSE schemes can be divided into two categories – *static* that only support keyword searches for a fixed collection of data, and *dynamic* that also support updating of the encrypted data collection.

Definition 3.13 (Static Searchable Symmetric Encryption [34])**.** A static searchable symmetric encryption consists of the following sets, algorithms, and protocols, parametrised with the security parameter λ.

- Sets of secret keys SK, queries Q
- Setup protocol $\langle \mathsf{Setup}_C | \mathsf{Setup}_S \rangle$, run by client and server. Client's input to Setup_C is database DB and the output is $(\mathsf{EDB}, \mathtt{state}_C, \mathsf{sk})$ with EDB being encrypted database, $\mathtt{state}_C$ being client's internal state and sk being secret key. Server's input to Setup_S is encrypted database EDB and output is $\top$ on successful setup and $\perp$ otherwise.
- Keyword query protocol $\langle \mathsf{KWQuery}_C | \mathsf{KWQuery}_S \rangle$, run by the client and the server to support a single-keyword query. Client's inputs are secret key sk, query q and state $\mathtt{state}_C$. Server's input is encrypted database EDB. The client's output is $(r, \mathtt{state}_C)$, where $r \in \mathsf{DB}$ being the result of the query and $\mathtt{state}_C$ being the new state of the client. The server's state EDB may also be modified following the execution.

Definition 3.14 (Dynamic Searchable Symmetric Encryption [34])**.** A dynamic searchable symmetric encryption consists of the following sets, algorithms, and protocols that is parametrised with the security parameter λ.

- Sets of secret keys SK, queries Q

- Setup protocol $\langle \text{Setup}_C | \text{Setup}_S \rangle$, run by client and server. Client's input to Setup_C is database DB and the output is $(\text{EDB}, \text{state}_C, \text{sk})$ with EDB being encrypted database, state_C being client's internal state and sk being secret key. Server's input to Setup_S is encrypted database EDB and output is $\top$ on successful setup and $\bot$ otherwise.

- Keyword query protocol $\langle \text{KWQuery}_C | \text{KWQuery}_S \rangle$, run by the client and the server to support a single-keyword query. Client's inputs are secret key sk, query q and state state_C. Server's input is encrypted database EDB. The client's output is (r, state_C), where $r \in$ DB being the result of the query and state_C being the new state of the client. The server's state EDB may also be modified following the execution.

- Insert protocol $\langle \text{Insert}_C | \text{Insert}_S \rangle$, run by the client and the server to update EDB. Client's inputs are secret key sk, an update query q, and internal state state_C. The server's input is encrypted database EDB. The client's output gets (r, state_C), where r being the response from the server and state_C being the new state of the client. The server's output is EDB′ where EDB′ is the encrypted database after the insertion operation.

- Delete protocol $\langle \text{Delete}_C | \text{Delete}_S \rangle$, run by the client and the server to update the encrypted database. The client's inputs are secret key sk, update query q, and internal state state_C. The server's input is encrypted database EDB. The client's output is (r, state_C) with r being the response from the server and state_C being the new client's state. The server's output is EDB′ where EDB′ is the encrypted database after the deletion operation.

The security definitions for SSE are formulated in terms of a leakage function that captures the information an adversary can unavoidably learn through the user's queries. The leakage for the SSE schemes includes:

- access pattern leakage – reveals which elements from the encrypted database match a query

- search pattern Leakage – reveals whether two search queries were for the same keyword

- volume leakage – reveals the number of encrypted database elements returned by the search query for a certain keyword

- co-occurrence leakage – reveals the co-occurrence information of the keywords when queries for different keywords return the documents common between these queries

Additionally, for the dynamic SSE schemes there are two main notions of privacy – forward privacy and backward privacy. *Forward privacy* poses the requirement that updating (insert or delete operation) an element in the database should not reveal whether the updated element contains keywords that have been previously searched for. *Backward privacy* poses the requirement that searching for a keyword should reveal no information about elements of the database previously containing this keyword that have subsequently been deleted from the database. In [34], however, there is an additional security notion for the dynamic SSE – *obliviousness of operations*. This is a stronger notion compared to the previously defined ones. It implies forward privacy, but additionally requires update operations to hide the number of keywords of the database elements being updated. Additionally, it implies backward privacy, requiring to completely hide both the result pattern and the update history associated with the underlying keyword.

3.8 Universal Composability

In the universal composability model [19], security of a protocol is defined in the real-ideal world paradigm. It means that the protocol π is considered to be securely implementing some task if the execution of the protocol with a real adversary is indistinguishable from simulating an ideal protocol with the appropriate trusted party. The instructions that are run by this trusted party are defined in the form of an ideal functionality $\mathcal{F}_{\text{ideal}}$. The additional component of the UC model is *environment* $\mathcal{Z}$ that provides inputs to the participants of the protocol and receives corresponding outputs. The environment $\mathcal{Z}$ models everything external to the protocol execution, including inputs provided to the parties, auxiliary information, and the external context in which the protocol is used. Intuitively, $\mathcal{Z}$ acts as an observer and orchestrator. It can interact with the parties, providing inputs and receiving outputs, and communicate with the adversary. The role of $\mathcal{Z}$ is to capture any possible real-world usage of the protocol.

In the real protocol execution, honest parties run the protocol π over a network controlled by an adversary $\mathcal{A}$, who can also corrupt some parties. In the ideal world, honest parties are viewed as *dummy parties* who forward their inputs directly to the ideal functionality $\mathcal{F}_{\text{ideal}}$ that internally executes the main task of the protocol. The protocol π securely realises functionality $\mathcal{F}$ if for every real adversary $\mathcal{A}$ attacking the protocol there exists an ideal adversary (simulator) $\mathcal{S}$, such that no environment $\mathcal{Z}$ can distinguish whether it is interacting with the protocol and the real adversary or with the ideal adversary and simulator. The *universality* in the UC framework comes from the fact that security is defined with respect to all possible environments $\mathcal{Z}$. This guarantees that a protocol remains secure when composed with any other protocols or used in any larger system. In particular, if a protocol securely realizes an ideal functionality in the UC framework, then it can be substituted for the functionality in any context without compromising overall security. Intuitively, security in the UC framework means that for every real-world attack on the protocol, there exists a corresponding attack in the ideal world that produces an indistinguishable effect. The ideal functionality $\mathcal{F}_{\text{ideal}}$ is designed to capture all sensible behaviours of an adversary, ensuring that any attack possible in the real execution can already be represented within the ideal model.

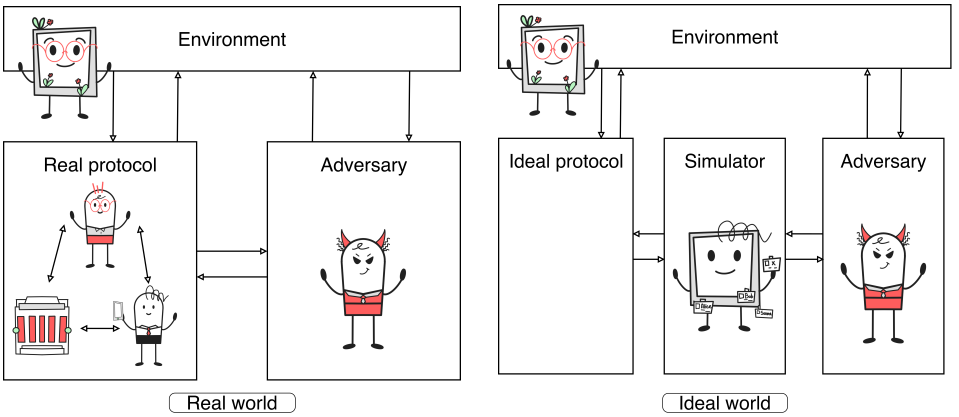


Figure 5: Universal Composability framework idea

3.9 Types of corruption

In cryptographic security models, corruption describes how the adversary controls the parties. There are different types of corruption that capture how powerful the adversary is. From the point of view of when the corruption occurs, we differentiate between static and adaptive corruption:

- **Static corruption** – the adversary chooses which parties to corrupt before the protocol execution begins, and this set remains fixed throughout the entire execution.
- **Adaptive corruption** – the adversary may choose which parties to corrupt throughout the execution of the protocol, possibly based on information learned so far.

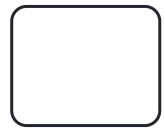
From the adversary's interaction and control of the parties during the protocol, we differentiate between passive, active and covert corruption:

- **Passive (semi-honest) corruption** – corrupted party follows the protocol honestly, but the adversary learns its internal state, including inputs, randomness used for computation, results of intermediate computations, and outputs.
- **Active (malicious) corruption** – corrupted party is fully controlled by the adversary and may arbitrarily deviate from the protocol.
- **Covert corruption** – corrupted party may deviate from the protocol, but will be following the protocol if the probability of being detected is non-negligible.

3.10 BPMN

BPMN (Business Process Model and Notation) is a widely-adopted notation, used to model and visualize business processes [53]. It provides a set of symbols (such as events, activities, gateways, flows, and data objects) that help to clearly document, analyse, and improve processes. Since BPMN is designed to be easily understood, it is a useful notation for process analysis. In the context of analysis of cryptographic systems, BPMN diagrams can be used to describe protocols, allowing for detailed analysis of data storage, movement, or processing steps. Table 1 defines the set of BPMN symbols that will be further used in this work.

Table 1: BPMN notation used in this work

Element	Description	Notation
Event	Events occur during a process and influence flow via triggers or results	
Activity	Activities represent action steps that are part of processes	
Gateway	Gateways are used to determine branching, forking, merging, and joining of paths in the process	
Data object	Data Objects define information required by or produced by activities	
Pool	Pool represents a participant of a process	

4 A Cloud Privacy-Preserving Attribute Management System

This section introduces CACS – A Cloud Privacy-Preserving Attribute Management System, where all the credential requests are handled by the central AMS server. The interaction between the parties (issuer, holder, verifier) follows the way defined by the mDL specification, where the holder component consists of two sub-components – user, who interacts with system through wallet application, and the central AMS server. Users interact with the system through wallet application on their mobile device. The wallet application informs the user which attributes are requested by the RP during the presentation phase and allows user to select which attributes they wish to disclose. Additionally, wallet application enables user to access to the log of transactions associated with the user's attributes.

In this system, the central AMS server is assumed to be a trusted party as it is allowed to learn the user-RP interactions, as well as the attributes that users present to RPs. Internally, the AMS maintains two databases, one for storing user's credentials and one for storing the logs of accessing the credentials. These databases are assumed to be untrusted (passively corrupted) cloud components of the CACS systems. The design goal for the system is to keep mDL data flow of the protocol unmodified while ensuring privacy against the untrusted database component that can be viewed, for example as a cloud service.

The system operates with individual attributes separately, rather than receiving credential with multiple attributes from the issuer. This offers modularity of the system, allowing user to selectively disclose their attributes and combine those into credential only at the moment of presentation. The problem of the attribute revocation is solved by allowing the user to submit revocation requests to the AMS server and allowing AMS server to revoke attributes on the request from authorities. Since AMS server is responsible for storing revocation information, it should notify RP if the revoked credential is being presented.

We start with defining the message flow between the system components in the plain model and then outline the way it changes once we add SSE or threshold decryption for protecting storage of attributes. The system consists of two main protocols – issuing and presentation (which is followed by the verification).

Notation This section presents notation that is used for defining attribute management system. Notation is defined in Table 2. We note that the same notation is applicable to the system defined in Section 6.

4.1 Plain model

Before running issuing protocol, we assume that users have downloaded the wallet applications on their mobile devices. The message flow between the parties during the issuing protocol is presented in Figure 6.

Issuing:

1. The user authenticates to the AMS web portal and selects the attributes $\overrightarrow{\mathsf{aID}}$ that they wish to request from the issuer IID.
2. The AMS server sends the consent request to the user, which can be done in the form of an authentication or signing request.
3. Upon receiving user's consent, AMS server verifies it and saves the consent to the Log database.

Table 2: Notation for attribute management system

Notation	Explanation
aID	attribute identifier
$\overrightarrow{\text{aID}} = (\text{aID}_1, \ldots, \text{aID}_n)$	list of attribute identifiers
atr	attribute value
c_{atr}	encrypted attribute value
$\overrightarrow{\text{atr}} = (\text{atr}_1, \ldots, \text{atr}_n)$	list of attribute values
cID	credential identifier
$\overrightarrow{\text{cID}} = (\text{cID}_1, \ldots, \text{cID}_n)$	list of credential identifiers
$\text{cred} = \langle(\text{aID}_1, c_{\text{atr}_1}), \ldots (\text{aID}_n, c_{\text{atr}_n})\rangle$	credential
present_cred	credential presentation
uID	user identifier
IID	issuer identifier

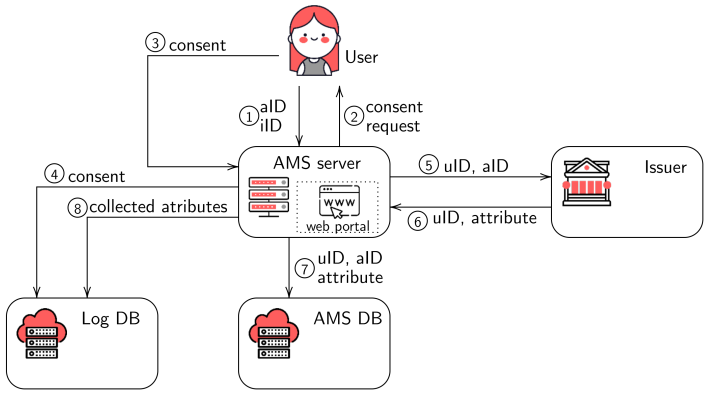


Figure 6: Issuing of attributes in plain model

4. The AMS server creates an attribute collection request $\langle \text{uID}, \overrightarrow{\text{aID}} \rangle$ and sends it to the issuer.

5. The issuer provides signed attributes $\overrightarrow{\text{atr}}$.

6. The AMS server verifies issuer signature on attributes and saves the received attributes to the AMS database.

7. The AMS server logs attribute collection by saving a new record in the Log database.

8. The AMS server informs the user of attribute collection through the AMS web portal.

Presentation: The message flow between the participants of presentation protocol is depicted in Figure 7.

1. The user with identifier uID authenticates to the AMS web portal and selects which attributes $\overrightarrow{\text{aID}}_{\text{dscl}}$ they want to disclose to the RP.

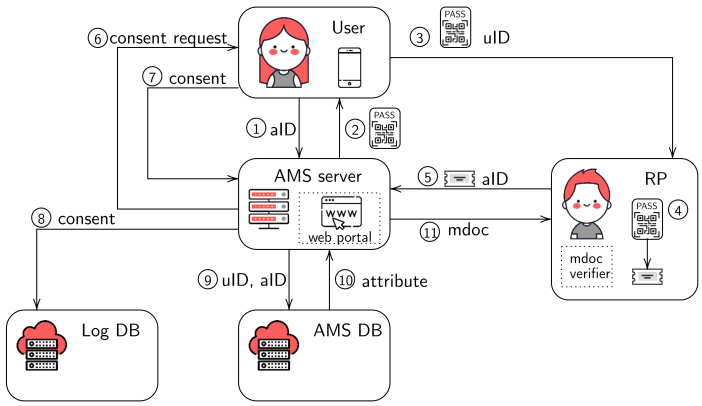


Figure 7: Presentation in plain model

2. The AMS server generates a QR code which includes the server retrieval token. Then the AMS generates an Apple Wallet pass in pkpass format including previously generated QR code.

3. The user downloads and saves the pkpass file, opens it and presents the QR code to the RP.

4. The RP uses the mdoc verifier to retrieve the server retrieval token by scanning the QR code.

5. The RP requests the attributes $\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}$ from the AMS server by presenting the server retrieval token and the user identifier uID.

6. The AMS server verifies the mdoc request and sends a consent request to the user.

7. Upon receiving consent, AMS server verifies it and saves it into the Log database.

8. AMS server gets requested attributes $\overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}$ from the AMS database.

9. The AMS forms and signs an mdoc document that includes the user's chosen attributes $\overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}$ and sends the mdoc response to the RP.

10. The RP verifies the AMS server signature on the server retrieval mdoc response. The RP reads the mdoc data and performs attribute verification.

4.2 System with searchable symmetric encryption

To protect storage of attributes in the AMS server's database, we propose to use searchable symmetric encryption. We propose to instantiate the system with SWiSSSE [34] scheme, which is a dynamic SSE scheme. The scheme satisfies obliviousness of operations, meaning that the search and update queries are computationally indistinguishable to the server. When the SWiSSSE database is instantiated, together with the actual encrypted data there are dummy entries created. These entries are not containing real data, they are intentionally inserted to implement frequency bucketisation, which mitigates volume leakage. By padding the encrypted database with fake occurrences of keywords and fake documents, the system ensures that every keyword in a specific bucket appears to have

the same frequency (the "bucket frequency") to an observer. This hides the true frequency of keywords. Additionally, dummy entries are used in the dynamic version of the system to mask when a new document is inserted, it prevents an adversary from noticing a change in the total number of document entries. To hide an insertion, the system avoids creating a fresh entry in the encrypted document array and instead converts an existing dummy entry into a real one. Deletions work in a complementary way by converting a real entry back into a dummy entry containing garbage ciphertext.

Issuing: Once the AMS server receives issued attributes $\overrightarrow{\mathsf{atr}}$ from the AMS server, it verifies issuer signature and saves attributes to its database using an SWiSSSE insert query.

Presentation: Once AMS server receives and verifies consent to present attributes from the user, AMS server performs an SWiSSSE search query with the AMS database. The AMS server uses $\mathsf{uID}||\mathsf{aID}_i$ as keywords to search for, where $\mathsf{aID}_i \in \overrightarrow{\mathsf{aID}}$. The AMS server performs $n = |\overrightarrow{\mathsf{aID}}|$ search queries. Each search query consists of two rounds:

1. With the first round, the AMS server receives identifiers of the entries that contain the searched keyword.

2. With the second round, the AMS server receives the entries themselves using the identifiers received in the first round.

AMS server decrypts received attributes $\overrightarrow{\mathsf{atr}}$ and proceeds to forming an mDoc document.

Revocation: When the AMS server wants to revoke specific attributes (as per the request from authority), the corresponding attribute is searched for using an SWiSSSE search query to the AMS database. The entry with attribute is updated to contain the status *revoked* and is uploaded back to the AMS database as a dummy entry. We propose not to delete revoked attributes from the database entirely, but to convert them into dummy entries. This way, we ensure that there will be enough space for the new attributes in the encrypted database. This works because the SWiSSSE insertion query rewrites the dummy entries of the database to contain actual encrypted data.

Updating attributes: When the AMS receives a new attribute for the user, it performs an SWiSSSE insertion query with the AMS database. A new encrypted attribute replaces one of the dummy entries in the AMS database. Similarly, when the attribute value is updated, the old version of the attribute is updated to contain the status *revoked* (turning it into a dummy entry). The new version of the attribute is added to the AMS database using an SWiSSSE insertion query.

This approach ensures that AMS database that can be hosted by a passively corrupted cloud provider, does not learn plaintext values of the user's attributes as well as access patterns to its entries.

4.3 System with threshold decryption

Alternative approach to protecting storage of the user's attributes is usage of threshold decryption schemes. The decryption key will be distributed between the user (mobile device) and AMS. With this approach, we can ensure that the user's attributes cannot be decrypted without user participating in the decryption process. One possible way to instantiate the system is using scheme by Kirss et al. [42].

User's device setup: The user's device (wallet app) runs the threshold key generation protocol with the AMS server. As a result, the user device stores one share sk_1 of the private key and the AMS server stores the second share sk_2. The user's share of the private key is stored protected by the user's PIN code. The public encryption key pk is stored by the AMS server.

Issuing: Instead of sending plaintext attributes as in the plain model, the issuer encrypts requested attributes under the user's public key pk as $c_{\mathsf{atr}_i} = \mathsf{Enc}_{\mathsf{pk}}(\mathsf{atr}_i)$. Signature is issued on the attribute plaintext value.

Presentation: Upon receiving user's consent to present certain attributes $\overrightarrow{\mathsf{aID}}$, the AMS server requests encrypted attributes from the AMS database by uID and $\overrightarrow{\mathsf{aID}}$. Next, the AMS initiates a threshold decryption protocol with the user device. The user device applies its share sk_1 of the private key and the AMS server finishes the decryption process by applying its share sk_2 of the private key and obtains a plaintext version of the requested attributes $\overrightarrow{\mathsf{atr}}$.

This approach ensures that the AMS server is not able to decrypt attributes without the user participation in the decryption protocol.

4.4 Comparison and discussion

When examining both systems, distinct advantages and disadvantages can be identified. Summary of the values disclosed to different parties during the protocol run is presented in Table 3. "In storage" column highlights the values that corrupted party can learn through access to storage.

Table 3: User data disclosed to different parties. Highlighted in grey disclosures (yes) are intentional and needed for functionality. Since AMS is sending attributes to RP, we assume it knows plaintext values of attributes during presentation. Additionally, since all the interactions with attributes in the system happen through aID*, we intentionally disclose it. Highlighted in red disclosures (yes) are the ones whose leakage we would like to avoid.*

	Issue		**Present**		**In storage**	
corrupted parties	$\overrightarrow{\mathsf{aID}}$	$\overrightarrow{\mathsf{atr}}$	$\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}$	$\overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}$	$\overrightarrow{\mathsf{aID}}$	$\overrightarrow{\mathsf{atr}}$
Plain model						
issuer	yes	yes	no	no	no	no
AMS	yes	yes	yes	yes	yes	yes
AMS DB	yes	yes	yes	yes	yes	yes
RP	no	no	yes	yes	no	no
Threshold decryption-based						
issuer	yes	yes	no	no	no	no
AMS	yes	no	yes	yes	yes	no
AMS DB	yes	no	yes	no	yes	no
RP	no	no	yes	yes	no	no
Searchable symmetric encryption based						
issuer	yes	yes	no	no	no	no
AMS	yes	yes	yes	yes	yes	yes
AMS DB	no	no	no	no	no	no
RP	no	no	yes	yes	no	no

The main advantage of SSE-based system is in hiding search patterns from the external database and offloading cryptographic computations from the user's device. However, it allows AMS server to have full control over the user's data and access it any time. The threshold decryption-based system does not allow AMS server to decrypt user's attributes on their own and requires communication with the user's device. However, it does not stop AMS server from saving decrypted attributes for later use. This raises the further research question, whether it is possible to modify the message flow such that the user

is the recipient of the plaintext attribute, not the AMS server. Although both presented CACS systems improve some security aspects of the plain model, they still do not provide reasonable security to the user nor the privacy against AMS server. Both CACS system presented above assume that the AMS server is trusted and leak information about the user's interactions with attributes, revocation status as well as the attribute plaintext values. As the communication model of the CACS system assumes that all the interactions with the user's attributes happen through the AMS server, it is hard to hide the user's private information without introducing heavy-weight cryptographic primitives. To improve security and privacy guarantees, the subsequent work changes the communication system making the user themselves responsible for sharing attributes with the RP.

4.5 Chapter summary

This chapter introduces CACS, a centralized identity management system designed to store user credentials in the cloud while preserving privacy against untrusted storage providers. It evaluates three cryptographic approaches: plain model, searchable symmetric encryption using SWiSSSE, and threshold decryption, alongside a combined approach. The analysis reveals that while SSE hides search patterns from the untrusted database, it inherently trusts the AMS with the attribute plaintext data. Conversely, usage of threshold decryption restricts the AMS from decrypting attributes without user consent but leaks to the untrusted database which ciphertext is accessed. The combined approach successfully mitigates both weaknesses, ensuring that database does not learn access patterns and AMS cannot initiate decryption of user's attributes without user's consent. However, due to the limitations of communication model all the interactions with the user's attributes happen through the AMS. Thus, it is hard to hide the plaintext values of user's attributes without introducing heavy-weight cryptographic primitives. Therefore, the subsequent work focuses on changing the communication model within the system making the user themselves responsible for sharing attributes with the RP.

5 Privacy-preserving server-supported decryption

When using public-key encryption scheme that supports threshold decryption for storing credentials, one more security concern appears. Even if only the user learns the plaintext of their credential, the ciphertext still needs to be sent to the supporting server for decryption. Having this information, supporting server can still learn which credential is being presented to the RP, which violates user privacy. In this section, we present construction of an encryption scheme with server-supported decryption. This scheme helps to protect storage of the private key by generating it distributively between the client and server. Additionally, it provides privacy against the server ensuring that the server does not learn which ciphertext is being decrypted by the client. The security of the scheme is analysed in the UC framework. As a separate contribution, we propose an ideal functionality for the encryption with server-supported blind threshold decryption.

5.1 Supporting algorithms

Notation For the reference on notation used to denote supporting algorithms, we introduce Table 4. In the context ctx means that this additional input to the algorithm binds the proof to the protocol.

Table 4: Notation and sub-routines used in blind server supported decryption construction

Notation	Explanation
H and $\tilde{H}$	hash functions
π	proof
$\mathsf{SE}(k,m)$ and $\mathsf{SD}(k,c)$	encryption and decryption procedures of Data Encryption Mechanism (DEM) for a key k, a plaintext m and a ciphertext c
$\mathsf{DHP}^{H}[r\mid^{u,v}_{g,h}\|ctx]$ and $\mathsf{ChP}^{H}[\pi\mid^{u,v}_{g,h}\|ctx]$	proof generation and verification for the proof that $\log_g u = \log_h v$, given in *context* ctx, where $r \in \mathbb{Z}_p$ is that discrete logarithm
$\mathsf{KnE}^{H,\tilde{H}}_{\mathsf{d}}[r\mid^{u}_{g}\|ctx]$ and $\mathsf{ChE}^{H,\tilde{H}}_{\mathsf{d}}[\pi\mid^{u}_{g}\|ctx]$	proof generation and verification for the proof of knowledge of (extractable) exponent $r = \log_g(u)$ in context ctx
$\mathsf{DVP}(r,r',\mathbf{r}\mid^{\alpha,\Gamma}_{g,u}\|B)$ and $\mathsf{DVC}(\pi\mid^{\alpha,\Gamma}_{g,u}\|B)$	proof generation and verification for the proofs of validity of DVNIZK proof (α,Γ) that uses randomness $r = \log_g(u)$, $r' = \log_g(\alpha)$ and $\mathbf{r}$ in the context of encrypted challenge B, where B is a DVNIZK challenge used in generation of Γ
$(\mathcal{K},\mathcal{E}_{\mathsf{ek}},\mathcal{D}_{\mathsf{vk}})$	Paillier key generation, encryption and decryption
$\mathbf{I}(n)$	set $\{0,1,\ldots,2^{n-1}\}$
$\mathbf{Cn}$	set of possible random coins used during encryption
assert condition	indicates verification steps (reject if false)
$\longrightarrow \mathrm{P} : m$	Party receives message m from party P
$\mathrm{P} \longrightarrow : m$	Party P sends message m

The proposed scheme is constructed from the ElGamal key encapsulation mechanism (KEM) presented in Figure 8. The security of the scheme is based on hardness of computational Diffie-Hellman problem. This scheme was selected as it allows to re-randomise ciphertexts, which helps to achieve blinding property as well as allows creating threshold

schemes. For the scheme presented in Figure 8, c^z for a random z results in valid encapsulation under the same pk with changed randomness ($r \cdot z$ instead of r).

The protocol that we base proposed scheme on consists of two steps – key encapsulation that is followed by data encapsulation as $\mathsf{SE}(k,m) := (H'(k) \oplus m, H''(k,m))$ for some hash functions (modelled as random oracles) H', H'' denoting symmetric encryption of a message m with a key k derived through KEM.

Key Generation:	Encapsulation:	Decapsulation:
1 : $\mathsf{sk} \leftarrow\!\!\$\, \mathbb{Z}_p$	1 : $r \leftarrow\!\!\$\, \mathbb{Z}_p$	1 : $ss = c^{\mathsf{sk}}$
2 : $\mathsf{pk} = g^{\mathsf{sk}}$	2 : $ss = \mathsf{pk}^r$	2 : Return ss
3 : Return $(\mathsf{sk}, \mathsf{pk})$	3 : $c = g^r$	
	4 : Return (ss, c)	

Figure 8: ElGamal KEM

In order to obtain IND-CCA security for ElGamal, we use proofs of knowledge of encryption randomness. Additionally, we need to introduce other zero-knowledge proofs to support blinding ciphertext from the server.

For the publicly verifiable proofs of knowledge of the exponent r in an ElGamal ciphertext, we use Σ-protocols that are made non-interactive using the Fiat-Shamir transform.

We start a NIZK proof that $\log_g u = \log_h v$ given in *context* of certain ctx presented in Figure 9.

$\mathsf{DHP}^H[r \mid^{u,v}_{g,h} \mid ctx]$:	$\mathsf{ChP}^H[\pi \mid^{u,v}_{g,h} \mid ctx]$:
1 : $s \leftarrow\!\!\$\, \mathbb{Z}_p$	1 : $\alpha \leftarrow g^\gamma / u^\beta$
2 : $\alpha \leftarrow g^s, \quad \alpha' \leftarrow h^s$	2 : $\alpha' \leftarrow h^\gamma / v^\beta$
3 : $\beta \leftarrow H(g,h,u,v,\alpha,\alpha',ctx) \in \mathbb{Z}_p$	3 : **assert** $\beta = H(g,h,u,v,\alpha,\alpha',ctx) \in \mathbb{Z}_p$
4 : $\gamma \leftarrow s + r \cdot \beta$	
5 : **return** $\pi \leftarrow (\beta, \gamma)$	

Figure 9: NIZK proof that $\log_g u = \log_h v$

Next, we define simulatable proofs of knowledge of exponent on the Figure 10. These are used to prove that one knows the value $r = \log_g u$ and in the security proof extract the ability to raise to the power of r (or $1/r$) from the proof itself. The intuition is that a prover would not be able to generate the proof without being able to raise to the power of r (with $\mathrm{d} = 1$) or $1/r$ (with $\mathrm{d} = -1$).

$\mathsf{KnE}^{H,\tilde{H}}_{\mathrm{d}}[r \mid^{u}_{g} \mid ctx]$:	$\mathsf{ChE}^{H,\tilde{H}}_{\mathrm{d}}[\pi' \mid^{u}_{g} \mid ctx]$:
1 : $h \leftarrow \tilde{H}(g,u,ctx) \in \mathbb{G}$	1 : $h \leftarrow \tilde{H}(g,u,ctx) \in \mathbb{G}$
2 : $v \leftarrow h^{r^{\mathrm{d}}}$	2 : $(\pi, v) \leftarrow \pi'$
3 : **if** $\mathrm{d} = 1$ **then** $\pi \leftarrow \mathsf{DHP}^H[r \mid^{u,v}_{g,h} \mid ctx]$	3 : **if** $\mathrm{d} = 1$ **then assert** $\mathsf{ChP}^H[\pi \mid^{u,v}_{g,h} \mid ctx]$
4 : **if** $\mathrm{d} = -1$ **then** $\pi \leftarrow \mathsf{DHP}^H[r \mid^{u,h}_{g,v} \mid ctx]$	4 : **if** $\mathrm{d} = -1$ **then assert** $\mathsf{ChP}^H[\pi \mid^{u,h}_{g,v} \mid ctx]$
5 : **return** $\pi' \leftarrow (\pi, v)$	

Figure 10: Proof of knowledge of exponent in a group $\mathbb{G}$

Since in the decryption protocol, the ciphertexts and the corresponding proofs get blinded (and later unblinded), when the client asks the server to assist in decryption, we need a NIZK proof that supports malleability. Meaning that the proof should remain valid even after the ciphertext and proof are blinded by the client. For this purpose, we rely on a construction from Damgård–Fazio–Nicolosi (DFN) [23], which yields DVNIZK proofs fitting well in to our construction. This proof aims to convince the server that the encrypting party knows the randomness used to generate ElGamal ciphertext. The initial DVNIZK proofs (generated by encrypting party) get blinded by the client and the server gets to verify blinded version of DVNIZK proof.

To support designated verifier proofs, we need to use additively homomorphic public-key encryption scheme that consists of key generation algorithm $\mathcal{K}()$, outputting key pair of encryption key ek and verification key vk, encryption algorithm $\mathcal{E}_{\mathsf{ek}}(\cdot)$, producing ciphertext and decryption algorithm $\mathcal{D}_{\mathsf{vk}}$, outputting plaintext. We propose to instantiate it with the Paillier encryption scheme as it has necessary properties.

Finally, we need a proof that a DVNIZK proof is valid which is presented in Figure 11. The aim of this proof is to convince client that the DVNIZK proof will be successfully verified by the server holding corresponding private verification key. Prover (encrypting party) wants to convince the verifier (client) that values (u, α_1, Γ) have been constructed as in DFN construction and the prover knows the used randomness $r = \log_g u \in \mathbf{I}(d)$, $r_1 = \log_g \alpha_1 \in \mathbf{I}(\rho + d + \kappa)$, and the random coins $\mathbf{r}$ for the encryption.

The Σ-protocol for this is similar to proving the knowledge of exponent, where the prover constructs another similar triple (α_2, α_3, A) using fresh randomness $(r_2, r_3, \mathbf{r}')$ and sends to the verifier, also sends a linear combination (with coefficients chosen by the verifier) of the original and fresh randomness to the verifier, who can then use the homomorphic properties of the operations to match the original triple against the fresh one. We present the protocol in Fig. 11 as a NIZK proof, replacing verifier's choice of the coefficients of the linear combination with the invocation of a random oracle.

$\mathsf{DVP}(r, r_1, \mathbf{r} \mid^{\alpha_1,\Gamma}_{g,u} \mid B)$

$r_2 \leftarrow\$ \mathbf{I}(\rho + d + \kappa)$
$r_3 \leftarrow\$ \mathbf{I}(2\rho + d + 2\kappa)$
$\mathbf{r}' \leftarrow\$ \mathbf{Cn}$
$\alpha_2 \leftarrow g^{r_2 \bmod p}$
$\alpha_3 \leftarrow g^{r_3 \bmod p}$
$A \leftarrow \mathcal{E}_{\mathsf{ek}}(r_3; \mathbf{r}') \cdot B^{r_2}$
$\beta' \leftarrow H(g, u, \alpha_1, \Gamma, \alpha_2, \alpha_3, A, B)$
$\gamma_2 \leftarrow r_2 + \beta' \cdot r$
$\gamma_3 \leftarrow r_3 + \beta' \cdot r_1$
$\gamma_{\mathbf{c}} \leftarrow \mathbf{r}' \boxplus (\beta' \boxdot \mathbf{r})$
$\pi \leftarrow (\beta', \gamma_2, \gamma_3, \gamma_{\mathbf{c}})$
return π

$\mathsf{DVC}(\pi \mid^{\alpha_1,\Gamma}_{g,u} \mid B)$

$(\beta', \gamma_2, \gamma_3, \gamma_{\mathbf{c}}) \leftarrow \pi$
$\alpha_2 \leftarrow g^{\gamma_2} / u^{\beta'}$
$\alpha_3 \leftarrow g^{\gamma_3} / \alpha_1^{\beta'}$
$A \leftarrow \mathcal{E}_{\mathsf{ek}}(\gamma_3; \gamma_{\mathbf{c}}) \cdot B^{\gamma_2} / \Gamma^{\beta'}$
assert $\beta' = H(g, u, \alpha_1, \Gamma, \alpha_2, \alpha_3, A, B)$

Figure 11: Proving the validity of a DVNIZK proof. Notation $\boxdot$ and $\boxplus$ means scalar multiplication and addition of coins.

5.2 Ideal functionality

The security for the scheme has been defined through the ideal functionality in the universal composability framework. The ideal functionality $\mathcal{F}_{\text{tdecrypt}}$ is presented in Figure 12, Figure 13, and Figure 14. The ideal functionality has the following interfaces – key generation, encryption, and decryption. There are two main parties – P_1 (client) and P_2 (server) that initiate key generation and decryption protocols. There are many parties P_i that can initiate encryption process.

Key generation
On message (KeyGen, sid) from both P_1 and P_2:

1. Send (KeyGen, sid) to $\mathcal{S}$ and wait for $(\text{Key}, sid, \text{pk})$ from $\mathcal{S}$.
2. If $\text{pk} \neq \perp$, create an empty table T for storing encrypted messages, and set decryption counter $ctr_{\text{dec}} \leftarrow 0$.
3. For $i \in \{1,2\}$, if the party P_i is corrupted, wait for (Output, i, y_i) from $\mathcal{A}$. Otherwise, take $y_i = \text{pk}$.
4. Output (Key, sid, y_1) to P_1 and (Key, sid, y_2) to P_2.

Encryption
On message $(\text{Encrypt}, \text{sid}, m)$ from some party P_i:

1. If P_i is corrupted, send $(\text{Encrypt}, \text{sid}, i, m)$ to $\mathcal{S}$. Otherwise, send $(\text{Encrypt}, sid)$ to $\mathcal{S}$.
2. Wait for $(\text{Encrypt}, sid, c)$ from $\mathcal{S}$ (the adversary chooses the ciphertext for the given plaintext).
3. Store the pair (m, c) in table T.
4. If the party P_i is corrupted, wait for (Output, i, y) from $\mathcal{A}$. Otherwise, take $y = c$.
5. Send $(\text{Encrypted}, sid, y)$ to P_i.

Figure 12: Ideal encryption functionality $\mathcal{F}_{tdecrypt}$.

An important functional requirement for such system is in the decryption process. It should start with client blinding the ciphertext and sending it to the server for partial decryption. This, in turn, requires client to validate the ciphertext before sending it to the server.

The key generation and encryption interfaces are similar to the existing functionalities for public key encryption [20]. Key generation is initialised with inputs from both parties P_1 (client) and P_2 (server). $\mathcal{F}_{\text{tdecrypt}}$ allows adversary $\mathcal{S}$ to choose pk. In the definition of ideal functionality, public key pk serves as an identifier for the public key related to concrete instance of the functionality. Thus, pk value can be chosen arbitrarily by the attacker. Additionally, when designing ideal functionality, we make corresponding security proofs easier if we give more power (more values) to the simulator, but we need to be careful that it does not make or ideal world weaker than we would like to.

Decryption procedure $Decrypt(\mathsf{Dec}, c)$ for a function $\mathsf{Dec} : \mathsf{C} \to \mathsf{M}$ and a ciphertext $c \in \mathsf{C}$:

- If there exists a pair (m, c) in T: (for correctness we need that c would decrypt to m)
  - If m is not unique (there are two different m for the same c), take $m' = \perp$.
  - If m is unique, take $m' = m$.
- Otherwise, compute $m' := \mathsf{Dec}(c)$, running it only a polynomial number of steps (the polynomial is a parameter of $\mathcal{F}$). Add (m', c) to T.
- Return m'.

Decryption (honest client)
On message $(\text{Decrypt}, sid, c)$ from party P_1 and $(\text{Decrypt}, sid)$ from party P_2:

1. Send $(\text{Decrypt-init}, sid)$ to $\mathcal{S}$.
2. Upon receiving $(\text{Decrypt-init}, sid, \mathsf{Ver})$ from $\mathcal{S}$, where $\mathsf{Ver} : \mathsf{C} \to \{\text{true}, \text{false}\}$, compute $b \leftarrow \mathsf{Ver}(c)$.
   - If $b = \text{true}$, send $(\text{Decrypt-good-c}, sid)$ to $\mathcal{S}$.
   - If $b = \text{false}$ or $b = \perp$, send $(\text{Decrypt-bad-c}, sid)$ to $\mathcal{S}$. Proceed to the point 4) with $y_1 = y_2 = \perp$.
3. The final output y_1 for P_1 depends on whether the server is corrupt.

   If the server is *honest*, the decryption can only be delayed:
   - Upon receiving $(\text{Decrypt-complete}, sid, \mathsf{Dec})$ from $\mathcal{S}$, take $y_1 \leftarrow Decrypt(\mathsf{Dec}, c)$, and $y_2 \leftarrow \top$.

   If the server is *corrupted*, $\mathcal{S}$ tells whether decryption succeeded.
   - Upon receiving $(\text{Decrypt-complete}, sid, \mathsf{Dec})$ from $\mathcal{S}$, take $y_1 \leftarrow Decrypt(\mathsf{Dec}, c)$.
   - Upon receiving $(\text{Decrypt-fail}, sid)$ from $\mathcal{S}$, take $y_1 \leftarrow \perp$.
4. If the server is corrupted, wait for $(\mathtt{Output}, 2, y_2')$ from $\mathcal{A}$ and take $y_2 \leftarrow y_2'$.
5. Send $(\text{Decrypted}, sid, y_1)$ to P_1 and $(\text{Decrypted}, sid, y_2)$ to P_2.

Figure 13: Ideal encryption functionality $\mathcal{F}_{\text{tdecrypt}}$ (continued).

Encryption can be invoked by any party P_i, knowing the correct public key pk. $\mathcal{S}$ is allowed to generate ciphertext c without seeing input m. $\mathcal{F}_{\text{tdecrypt}}$ sends m to $\mathcal{S}$ only when the encrypting party P is corrupted. This allows us to capture the fact that the ciphertext c does not leak anything about m and blinding, since corrupted server does not learn m

Decryption (corrupted client)
On message $(\text{Decrypt}, sid, c)$ from party P_1 and $(\text{Decrypt}, sid)$ from party P_2:

1. Set $ctr_{\text{dec}} \leftarrow ctr_{\text{dec}} + 1$ and send $(\text{Decrypt-init}, sid, c)$ to $\mathcal{S}$.
2. $\mathcal{S}$ tells whether decryption succeeded:
   - Upon receiving $(\text{Decrypt-complete}, sid)$ from $\mathcal{S}$, take $y_2 \leftarrow \top$.
   - Upon receiving $(\text{Decrypt-fail}, sid)$ from $\mathcal{S}$, take $y_2 \leftarrow \bot$.
3. Wait for $(\text{Output}, 1, y_1)$ from $\mathcal{A}$.
4. Send $(\text{Decrypted}, sid, y_1)$ to P_1 and $(\text{Decrypted}, sid, y_2)$ to P_2.

At any point of time (corrupted client). On message $(\text{Decrypt-msg}, sid, \text{Dec}, c')$ from $\mathcal{S}$:

1. Compute $m = Decrypt(\text{Dec}, c')$.
2. Set $ctr_{\text{dec}} \leftarrow ctr_{\text{dec}} - 1$. If $ctr_{\text{dec}} < 0$, then stop.
3. Send $(\text{Decrypted}, sid, m)$ to $\mathcal{S}$.

Figure 14: Ideal encryption functionality $\mathcal{F}_{tdecrypt}$ (continued).

either. The message-ciphertext pairs (m, c) generated through interaction with $\mathcal{F}_{\text{tdecrypt}}$ are stored in a table T, this allows to ensure the correctness of decryption queries. For simplicity of the ideal functionality, we assume that party P makes encryption queries once it has learned the correct pk. Thus, pk is not submitted as input for the encryption query.

Decryption interface is more nuanced compared to the previous works and its functionality depends on the corrupted parties. Intuitively, decryption through $\mathcal{F}_{\text{tdecrypt}}$ corresponds to finding the plaintext m from a ciphertext c that was created using the key pk. Decryption is initiated by the parties having access to the private key shares – P_1 and P_2, with P_1 providing the ciphertext.

There are two main approaches of modelling ideal functionality for public key encryption. In one approach, $\mathcal{S}$ provides encryption and decryption algorithms during the key generation and then the ideal functionality uses them to answer queries in the encryption and decryption interfaces. In the second approach, $\mathcal{S}$ performs encryption and decryption itself and provides values to the $\mathcal{F}$. None of those existing approaches works straightforwardly for modelling security of our scheme. If the client is honest and the $\mathcal{S}$ is expected to provide the descriptions of the encryption and decryption algorithms to $\mathcal{F}$, then it is difficult to build simulators in the Random Oracle Model (ROM). The ciphertext outputted by the scheme, is going to contain zero-knowledge proofs, which are in turn based on hash functions, modelled as random oracles. To be able to correctly decrypt ciphertext, the decryption function needs to know how these hashes have been evaluated.

Another approach is to ask $\mathcal{S}$ to provide the ciphertext values during encryption, and to decrypt unknown ciphertexts during decryption. This, however, contradicts our intuition of "blinded decryption", where $\mathcal{S}$ should not learn the ciphertexts that an honest client wants to decrypt. The solution that we propose is requesting $\mathcal{S}$ to perform the encryp-

tion process (internally programming the random oracles as necessary), and providing the $\mathcal{F}_{\text{tdecrypt}}$ with an *updated* description of Dec algorithm for each decryption request. This approach helps to ensure that $\mathcal{S}$ does not learn the ciphertext that is being decrypted, but the updated state of the random oracles is included in the description of Dec and affects the result of decryption.

In order to ensure correctness of all decryption queries, $\mathcal{F}_{\text{tdecrypt}}$ adds to T all (m, c) pairs obtained during the decryption as well. Since $\mathcal{S}$ cannot verify the correctness of the input ciphertext, $\mathcal{S}$ additionally sends to $\mathcal{F}_{\text{tdecrypt}}$ a description of a function Ver. This is a helper function that verifies the ciphertext, returning either true or false. This function is required for $\mathcal{F}_{\text{tdecrypt}}$ since the functional requirement for the protocol is that the decryption should not start if the input ciphertext is incorrect.

If the client is corrupted, then privacy of the ciphertext does not have to be protected and the ciphertext can be given to $\mathcal{S}$. But the difference from the existing literature is that in this case, a corrupted party is expected to learn the plaintext. Hence the simulator is also expected to learn m to be able to correctly simulate honest party's view. However, if the ciphertext c was created using the encryption interface of $\mathcal{F}$, then c is actually independent of m, and the only location containing m is the table T of $\mathcal{F}$. It means that $\mathcal{S}$ should have an ability, to read that table.

If we would need to model threshold decryption, it would be sufficient to allow $\mathcal{S}$ query for the row (m, c), where c is the ciphertext that is being decrypted. However, since we are modelling *blinded* decryption, the adversary corrupting the client can start the decryption protocol for an arbitrary ciphertext, hence the row(s) queried by $\mathcal{S}$ can no longer be restricted like that. $\mathcal{S}$ should be allowed to query the row (m, c) for a ciphertext c of its own choice. Moreover, we allow to make this query *after* the decryption session has ended, since the corrupted client may potentially be able to undo his own blinding in multiple ways after getting response from the server, thus being able to choose between decrypting several different ciphertexts later. Nevertheless, the adversary is only allowed to decrypt *at most one message per decryption query* coming from the environment.

5.3 Description of the scheme

Key generation Client and server initiate key generation process by sampling their shares of the private key sk_i and constructing π_i – proof of knowledge of sk_i corresponding to the public key share pk_i. Next, they exchange hash commitments on (pk_i, π_i) to avoid another party adaptively choosing their values. Once commitments are received, parties exchange opening of commitments (pk_i, π_i) and verify values (commitment openings and proofs) received from each other. Parties define combined public key as $\mathsf{pk} \leftarrow \mathsf{pk}_{3-i}^{sk_i}$.

Next, server generates two encryption key pairs $(\mathsf{ek}_1, \mathsf{vk}_1)$ and $(\mathsf{ek}_2, \mathsf{vk}_2)$ that are used for the designated verifier zero-knowledge proofs. Server proceed by encrypting random value β under both key pairs to get B_1, B_2 and constructs the following proofs. With the first one, server proves that the value under the encryption indeed belongs to the set $\mathbf{I}(\rho)$. With the second one, server proves that both ciphertexts B_1 and B_2 encrypt the same value. The reason for generating two ciphertexts is that during the encryption process, we need *two* DVNIZK proofs for special soundness property. Key generation protocol is defined in Figure 15.

Encryption Encryption can be initiated by any party, it starts by following the standard ElGamal encryption procedure by computing ciphertext (u, c_2). Next, it proceeds with computing:

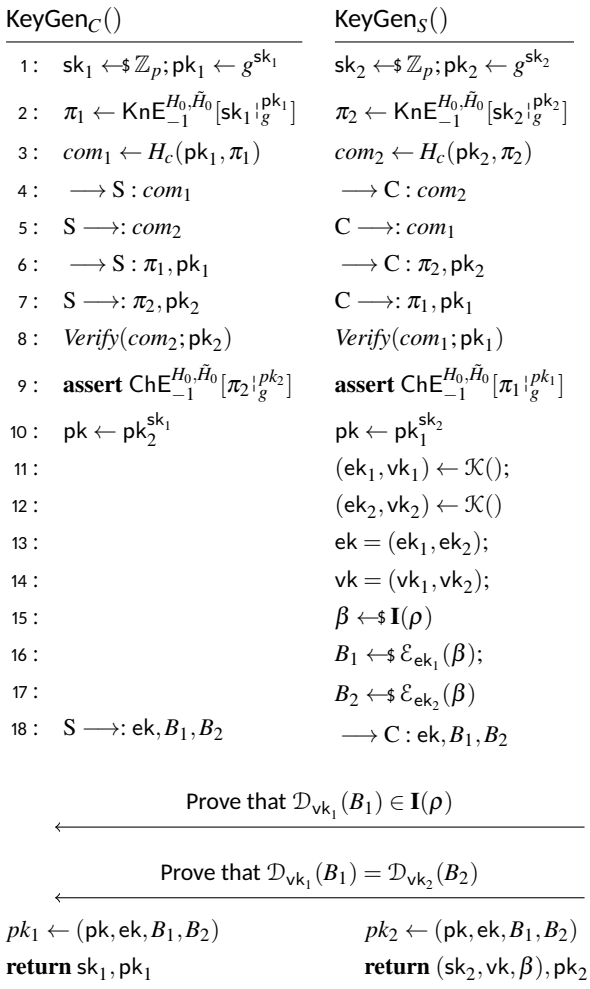


Figure 15: Key generation for client and server privacy-preserving threshold decryption

- π – the NIZK proof of knowledge of the exponent r, will be verified by client;
- $(\alpha_1, \Gamma_1, \Gamma_2)$ – the DVNIZK proof of knowing r, will be verified by supporting server;
- π_t – the NIZK proofs that (u, α_1, Γ_t) will be accepted by the verifier, will be verified by the client.

The final ciphertext is comprised of c_1 containing all the auxiliary values and c_2 being an actual ciphertext. Encryption algorithm is defined in Figure 16.

Decryption Decryption protocol is run between the client and server. Client's input is the ciphertext that should be decrypted, while the server does not get ciphertext as input. Client verifies proofs π and π_1, π_2, generates masking values z, z' and constructs the blinded ciphertext u', and the blinded DVNIZK proof $(\alpha_1', \Gamma_1', \Gamma_2')$.

Server gets blinded ciphertext from the client together with the blinded proofs. While verifying the DVNIZK proof, the server checks that γ is found from both Γ_1' and Γ_2'. Next,

$\mathsf{Enc}(m, \mathsf{pk}, \mathsf{ek}_1, \mathsf{ek}_2, B_1, B_2)$

$r \leftarrow\!\$\ \mathbb{Z}_p$
$u \leftarrow g^r$
$c_2 \leftarrow \mathsf{SE}(\mathsf{pk}^r, m)$
$r_1 \leftarrow\!\$\ \mathbf{I}(\rho + d + \kappa)$
$\mathbf{r}_\iota \leftarrow\!\$\ \mathbf{Cn}$
$\alpha_1 \leftarrow g^{r_1 \bmod p}$
$\Gamma_\iota \leftarrow \mathcal{E}_{\mathsf{ek}_\iota}(r_1; \mathbf{r}_\iota) \cdot B_\iota^r$
$\pi \leftarrow \mathsf{KnE}_1^{H_1, \tilde{H}_1}[r \,|^{u}_{g}\, | \alpha_1, \Gamma_1, \Gamma_2]$
$\pi_\iota \leftarrow \mathsf{DVP}(r, r_1, \mathbf{r}_\iota \,|^{\alpha_1, \Gamma_\iota}_{g, u}\, | B_\iota)$
$c_1 \leftarrow (u, \alpha_1, \Gamma_1, \Gamma_2, \pi, \pi_1, \pi_2)$
return c_1, c_2

$\iota \in \{1, 2\}$

Figure 16: Encryption in our construction

server verifies the proof of knowing r, and then applies its share of the private key on the blinded ciphertext $w \leftarrow u^{\mathsf{sk}_2}$. The value w is returned together with a proof that it has been correctly computed – that $\log_u w = \log_g \mathsf{pk}_2$. The client verifies this proof, applies its private key share sk_1 to u, and combines the result with the plaintext share w obtained from the server. Finally, client unblinds the value and derives the plaintext message m. Decryption protocol is defined in Figure 17.

$\mathsf{Dec}_C(c_1, c_2, \mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}, \mathsf{ek}_1, \mathsf{ek}_2, B_1, B_2)$

$(u, \alpha_1, \Gamma_1, \Gamma_2, \pi, \pi_1, \pi_2) \leftarrow c_1$
assert $\mathsf{ChE}_1^{H_1, \tilde{H}_1}[\pi \,|^{u}_{g}\, | \alpha_1, \Gamma_1, \Gamma_2]$
assert $\mathsf{DVC}(\pi_\iota \,|^{\alpha_1, \Gamma_\iota}_{g, u}\, | B_\iota)$
$z \leftarrow\!\$\ \mathbb{Z}_p,\ z' \leftarrow\!\$\ \mathbf{I}(\rho + 2d + 2\kappa)$
$\mathbf{r}'_\iota \leftarrow\!\$\ \mathbf{Cn}$
$u' \leftarrow u^z$
$\alpha'_1 \leftarrow \alpha_1^z \cdot g^{z' \bmod p}$
$\Gamma'_\iota \leftarrow \Gamma_\iota^z \cdot \mathcal{E}_{\mathsf{ek}_\iota}(z'; \mathbf{r}'_\iota)$
$\pi' \leftarrow\!\$\ \mathsf{KnE}_{-1}^{H_2, \tilde{H}_2}[\mathsf{sk}_1 \,|^{\mathsf{pk}_1}_{g}\, | u', \alpha'_1, \Gamma'_1, \Gamma'_2]$
$\longrightarrow \mathrm{S} : u', \alpha'_1, \Gamma'_1, \Gamma'_2, \pi'$
$\mathrm{S} \longrightarrow : w, \pi''$
assert $\mathsf{ChP}^{H_3}[\pi'' \,|^{\mathsf{pk}, w}_{\mathsf{pk}_1, u'}]$
return $\mathsf{SD}(w^{\mathsf{sk}_1 / z}, c_2)$

$\mathsf{Dec}_S(\mathsf{sk}_2, \mathsf{pk}_1, \mathsf{pk}, \mathsf{vk}_1, \mathsf{vk}_2, \beta)$

$\mathrm{C} \longrightarrow : u', \alpha'_1, \Gamma'_1, \Gamma'_2, \pi'$
$\gamma' \leftarrow \mathcal{D}_{\mathsf{vk}_1}(\Gamma'_1)$
assert $\gamma' = \mathcal{D}_{\mathsf{vk}_2}(\Gamma'_2)$
assert $g^{\gamma'} = \alpha'_1 \cdot (u')^\beta$
assert $\mathsf{ChE}_{-1}^{H_2, \tilde{H}_2}[\pi' \,|^{\mathsf{pk}_1}_{g}\, | u', \alpha'_1, \Gamma'_1, \Gamma'_2]$
$w \leftarrow (u')^{\mathsf{sk}_2}$
$\pi'' \leftarrow \mathsf{DHP}^{H_3}[\mathsf{sk}_2 \,|^{\mathsf{pk}, w}_{\mathsf{pk}_1, u'}]$
$\longrightarrow \mathrm{C} : w, \pi''$
return $\top$

Figure 17: Decryption in our construction

5.4 Security

The initial security proof for the scheme was game-based and written in the Algebraic Group Model (AGM). However, corresponding security definition for blindness required introducing AGM in the definition itself, making it less suitable for proving security of other protocols, not relying on AGM.

Therefore, to be able to capture all the required properties of the system as well as removing the need of AGM, it was decided to write security proof in the UC model and rely on one-more CDH assumption.

Theorem 1. The protocol set π_{btdec} in Figure 15, 16 and 17 securely implements functionality $\mathcal{F}_{\mathsf{tdecrypt}}$ in presence of malicious static adversary under one-more CDH assumption with tests.

The proof sketch of Theorem 1 is presented below. The full proof is presented in the extended version of paper [46].

Proof sketch In order to show that our threshold decryption protocol π_{btdec} securely implements defined functionality $\mathcal{F}_{\mathsf{tdecrypt}}$, we construct a simulator $\mathcal{S}$ such that for each adversary $\mathcal{A}$ attacking the protocol, the environment $\mathcal{Z}$ cannot distinguish whether it is interacting with the real protocol π_{btdec} and the adversary $\mathcal{A}$, or the ideal functionality $\mathcal{F}_{\mathsf{tdecrypt}}$ and the ideal adversary. Where the ideal adversary consists of $\mathcal{A}$ and $\mathcal{S}$. Below, we sketch the idea of how we define simulator $\mathcal{S}$ and how it interacts with the functionality $\mathcal{F}_{\mathsf{tdecrypt}}$ and the adversary in case of different corruptions. Once the simulator is constructed, we discuss why the encryption and decryption performed by the ideal functionality interacting with the $\mathcal{S}$ is indistinguishable from the real protocol. The main intuition being that if the environment is able to distinguish between the real protocol and the ideal protocol, then the simulator has enough information to solve one-more CDH problem that it received as an input from the external challenger.

Initialization: We assume that the corruptions of the parties are known already in the initialization phase. Let n be a parameter denoting the number of encryptions considered in session sid. $\mathcal{S}$ receives the following inputs from the external challenger – $\mathsf{pk} = g^{\mathsf{sk}}$ (where the corresponding sk is only known to the challenger), access to the oracle Check$(\cdot)$ and also depending on corruption:

- If the client is corrupted, $\mathcal{S}$ receives $(u_0, \ldots, u_n)$ for some parameter n, such that $(g, \mathsf{pk}, u_0, \ldots, u_n)$ is an instance of a one-more CDH problem. Further, $\mathcal{S}$ gets access to a decryption oracle $(\cdot)^{\mathsf{sk}}$.

  // If the number of decrypted ciphertexts exceeds the number of backdoor decryption queries through oracle $(\cdot)^{\mathsf{sk}}$ allowed by the $\mathcal{F}_{\mathsf{tdecrypt}}$, then the $\mathcal{S}$ finds solution to the one-more CDH problem.

- If the client is honest, $\mathcal{S}$ receives pk and u_0, such that (g, pk, u_0) is an instance of one-more CDH problem with $n = 0$. $\mathcal{S}$ sets $u_1 = \cdots = u_n \leftarrow u_0$.

  // If the adversary manages to decrypt ciphertext without interaction with user, it means breaking the CDH problem.

Additionally, in the initialisation phase, $\mathcal{S}$ prepares tables for storing the random oracle data, and in the background it runs processes that are constantly observing accesses of the adversary $\mathcal{A}$ and the environment $\mathcal{Z}$ to these oracles, programming them on demand to extract certain values as described further. Table 5 defines oracles used in the protocol. Table 6 defines which values the simulator can extract from which proof type.

Table 5: Usage of random oracles in protocol

Hash function	Usage in system
H_c	Hash commitment
$H_0, \tilde{H}_0$	the proof of knowledge of private key (key generation)
$H_1, \tilde{H}_1$	proof of knowledge of encryption exponent (encryption)
$H_2, \tilde{H}_2$	the proof of knowledge of private key (decryption)
H_3	proof of discrete logarithm equality (decryption)

Table 6: Values the simulator can extract from different proof types

Proof type	Values that can be extracted
KnE$_{-1}$	computing $\mathsf{pk}^{\mathsf{sk}_{3-i}^{-1}}$ (key generation)
KnE$_{-1}$	computing $u'^{\mathsf{sk}_1^{-1}}$ (decryption)
$(\alpha_1, \Gamma_1, \Gamma_2)$	$r' = \log_g(u')$ (decryption)

Key generation is simulated as follows:

- $\mathcal{S}$ uses random oracle H_c to extract the corrupted party's share of pk and the proof π from the commitment.
- $\mathcal{S}$ uses knowledge extraction w.r.t. π to construct the honest party's share of the public key, ensuring that it is consistent with input pk.
- If the client is corrupted, $\mathcal{S}$ needs to additionally simulate the proofs related to β:
  - $\mathcal{S}$ sets β_1 and β_2 such that $\beta_1 \neq \beta_2$ (to be able to apply special soundness later). The simulator creates two views of the same proof with different challenges, so it can later extract secret value — and this only works if $\beta_1 \neq \beta_2$. From the adversary's view it is indistinguishable as both values are encrypted and corresponding proofs are simulated.

    // As $\mathcal{S}$ has access to trapdoors, the proofs of equality can be simulated even if $\beta_1 \neq \beta_2$.

When $\mathcal{F}_{\text{tdecrypt}}$ receives an encryption query, it asks for a ciphertext from $\mathcal{S}$, without providing message as input. **Encryption** is simulated as follows:

- $\mathcal{S}$ constructs the ciphertext from the next challenge for one-more CDH problem:
  - $\mathcal{S}$ computes $u = (u_j)^r$ for a challenge u_j // instead of taking $u = g^r$ for a known r
  - $\mathcal{S}$ outputs a random c_2 // instead of encrypting the true message as it is not given to $\mathcal{S}$ as input

  The random oracles of H' and H'' can be programmed later on corresponding inputs. All related proofs can be simulated.

If adversary tries to encrypt some ciphertexts without accessing $\mathcal{F}_{\text{tdecrypt}}$, it will also need to construct corresponding proofs to ensure successful decryption. It implies accessing oracles H_1 and $\tilde{H}_1$, which allows $\mathcal{S}$ to prepare random oracles H_1 and $\tilde{H}_1$ to compute pk^r.

For the simulation of decryption, we differentiate corrupted client and corrupted server cases.

Decryption (honest client):

- $\mathcal{F}_{\text{tdecrypt}}$ waits for $\mathcal{S}$ to provide a verification function Ver, which should output to the $\mathcal{F}_{\text{tdecrypt}}$ whether ciphertext submitted for decryption is correct. This procedure includes the verification of DVNIZK proofs on behalf of the client. Since this functionality requires to evaluate H_1 and $\tilde{H}_1$, the contents of these hash tables should be embedded as a part of Ver. The values of B_1 and B_2 are also included in Ver.

- Since $\mathcal{S}$ does not receive the true ciphertext as input, interaction with the server is simulated for a ciphertext generated from a *random plaintext*.

  // Due to the statistical blinding of the ciphertext, the environment detects the difference only with the negligible probability.

- $\mathcal{F}_{\text{tdecrypt}}$ waits for a decryption function Dec from $\mathcal{S}$. The simulator provides to $\mathcal{F}_{\text{tdecrypt}}$ auxiliary table that matches π to an appropriate pk^r pre-extracted by the simulator.

  // For any ciphertext generated *inside* $\mathcal{F}_{\text{tdecrypt}}$, the adversary could notice difference in decryption process only if it manages to construct a ciphertext $((u^*,\ldots),c_2^*)$ which would decrypt to something that is related to a message that is encrypted with $\mathsf{pk}^r = u^{\mathsf{sk}}$ where u is a CDH challenge, but $u^* \neq u$ or $c_2^* \neq c_2$. To construct such a ciphertext the attacker would need to solve CDH.

Decryption (honest server):

- $\mathcal{S}$ gets from $\mathcal{A}$ a blinded ciphertext and needs to come up with a response consisting from w and a proof π'. The $\mathcal{S}$ should ensure that the response should be valid for a valid ciphertext. The simulation approach is different depending on the way ciphertext has been generated:

  - For messages generated inside $\mathcal{F}_{\text{tdecrypt}}$, using special soundness, $\mathcal{S}$ can compute $r' = \log_g(u')$ from a valid sigma-proof, which allows to compute response $w = \mathsf{pk}_2^{r'}$.
  - For messages generated outside of $\mathcal{F}_{\text{tdecrypt}}$, it is impossible to extract the exponent, so $\mathcal{S}$ needs to compute $w = (u')^{\mathsf{sk}_2} = (u'^{1/\mathsf{sk}_1})^{\mathsf{sk}}$. For this, it needs to access the CDH oracle $(\cdot)^{\mathsf{sk}}$. Next, $\mathcal{S}$ needs obtain u'^{1/sk_1} from u'. For that, it uses special procedure for raising $z \in \mathbb{G}$ to the power of r^d for $d \in \{1,-1\}$ using properties of KnE proof.
  - In general, homomorphic encryption allows to obtain a valid ciphertext as a linear combination of ideal ciphertexts. However, since $\mathcal{S}$ has used different γ_1 and γ_2 in Γ_1 and Γ_2 of the challenge ciphertexts, special soundness allows to extract the term r', splitting $u' = u'_1 \cdot u'_2$ where $u'_1 = g^{r'}$ and $u'_2 = \prod_j u_j^{z_j}$ is a linear combination of challenges. $\mathcal{S}$ can further compute $w = \mathsf{pk}_2^{r'} \cdot (u'_2)^{\mathsf{sk}}$, thus ensuring that $(\cdot)^{\mathsf{sk}}$ is only applied to the linear combinations of challenges and is not misused for breaking some proof.

- There will be no more calls to oracle $(\cdot)^{\mathsf{sk}}$ than there are decryptions approved by the server. When the adversary attempts to actually decrypt a ciphertext, $\mathcal{S}$ has control over it as it controls oracles H' and H'' that are evaluated for a valid decryption. $\mathcal{S}$ programs H' and H'' so that it would decrypt to the true message. In case ciphertext is one of the previous ciphertexts by $\mathcal{F}$ (this can be verified using $\mathsf{Check}(\cdot)$ oracle), the simulator can use the "backdoor" decryption of $\mathcal{F}$ to get the plaintext.

// If the adversary manages to obtain more plaintexts than there have been decryption sessions, $\mathcal{S}$ has solved the one-more-CDH problem.

5.5 Performance

For the performance evaluation, we use the elliptic curve group P-256 as $\mathbb{G}$, and 3072-bit RSA modulus for Paillier encryption scheme. Protocols presented in the paper have been implemented in Go, which gives better benchmarks compared to the Python implementation of [47]. The protocol was executed locally on Apple M1 with 8GB RAM. The times do not include communication and network latency. The ciphertext includes 2586-byte encapsulation and 64-byte ciphertext, encrypting 32-byte message.

*Table 7: Performance of blind threshold decryption for 32-byte message. Notice: * without network delay*

Ciphertext Size	**Client decryption message**	**Server decryption message**	**Encryption time**	**Decryption time (Client)**	**Decryption time (Server)**
2650 bytes	1258 bytes	106 bytes	175 ms	189 ms*	30 ms*

5.6 Discussion

This scheme was created primarily for the use-cases, where the client is the user's mobile device. Generating keys in a threshold way aims to protect the user's private key from adversary. However, generating keys in threshold way is not enough to ensure full protection. Therefore, the approach similar to [15] can be used to add protection against other attacks, such as memory cloning.

Additional problem raises if the user loses their mobile device and hence the access to the private key share. This issue is manageable with digital signature schemes, since all the previously created signatures still remain verifiable. But if the decryption key is lost, user loses ability to decrypt their ciphertexts. Possible solution would be to further split client's key share and store one of the shares as a key recovery option on the other device. For each decryption session, only one of the client's key shares will be required.

However, those optimisations are out of scope of this work and should be separately studied to ensure security of the protocol. Additionally, using encryption with external storage of the recovery share would prevent the system from remaining UC-secure. This is because, in realistic settings, encryption is typically reused across multiple contexts, and cannot be assumed to be restricted to this single purpose.

5.7 Chapter summary

To resolve the ciphertext leakage to the supporting server in standard threshold decryption, this chapter develops a blind server-supported decryption protocol. The scheme allows a client to request decryption assistance from a server without the server learning which specific ciphertext is being decrypted. Security of the proposed scheme is formally defined via an ideal functionality in the universal composability framework, with a proof that proposed protocol realises defined ideal functionality under the one-more computational Diffie-Hellman assumption. The scheme is designed by integrating ElGamal key encapsulation with non-interactive and designated-verifier zero-knowledge proofs, resulting in strong privacy guarantees against malicious server.

6 Privacy-preserving attribute management system

PRETTINESS (Privacy pResErving aTTrIbute maNagEment SyStem) is a privacy-preserving attribute management system that improves concerns raised in the design of CACS system. It is a centralised system that is built around Attribute Management Server (AMS) that helps the user with the storage of their credentials as well as plays essential role in the revocation phase. The system makes use of the following sub-protocols:

- Blind server-supported signature scheme [69] for user authentication and creating verifiable credential presentation;
- Server-supported blind decryption (Section 5) for storing credentials on the AMS side, such that the credentials cannot be decrypted without the user participation;
- Revocation protocol [55] that allows both user and issuer revocation, while hiding identifiers of revoked credentials from the AMS.

Notation The notation used in this section is presented in Table 2 and additional notation is presented in Table 8. The real protocol routines will need to internally store some additional values, which are not a part of input/output interface. For clarity, we highlight such values.

Table 8: Additional notation for attribute management system

Notation	Explanation
$\mathcal{F}(cmd, \mathsf{sid}, x)$	run the subroutine cmd of functionality $\mathcal{F}$ with sid and input x
$\mathsf{aID}_{\mathsf{dscl}} = (\mathsf{aID}_1, \ldots, \mathsf{aID}_n)$	list of disclosed attribute identifiers
$\overrightarrow{\mathsf{atr}}_{\mathsf{dscl}} = (\mathsf{atr}_1, \ldots, \mathsf{atr}_n)$	list of disclosed attribute values
rev	revocation token
$\mathcal{CH}$	set of challenges
$(\_, \ldots)$	pattern matching for any value in the tuple of values
$\mathcal{X}$	set of revocation tokens
X^-	set of revocation tokens that have been revoked

6.1 Design principles and threat model

Recall that we have the following parties: user, AMS, Issuer, Relying Party. In this section, we describe threat model for the parties in more details.

User is interacting with system using their mobile device. Therefore, there might be different levels of device corruption, including partial access to the device memory or full access to the device memory. In the threat model, we assume that user can be corrupted throughout the protocol, achieving the following corruption levels:

- $c_{\mathsf{U}} = 0$ corresponds to no corruption;
- $c_{\mathsf{U}} = 1$ corresponds to partial corruption, where adversary copies device memory after the keys have been generated;
- $c_{\mathsf{U}} = 2$ corresponds to partial corruption, where adversary copies device memory during the process of creating authentication. In this case, the adversary gets as

much power as in the case of full corruption, but only for a limited number of authentications;

- $c_{\mathsf{U}} = 3$ corresponds to full corruption of device (having dynamic and unlimited access to the device and its memory).

Such corruption model allows us to rely on additional security properties introduced by the server-supported signing protocol [69]. To make use of the user's key protection mechanisms, we need to assume that the user has not been corrupted during the key generation and adversary can copy device's memory.

In this case, we consider user to be statically corrupted with levels $c_{\mathsf{U}} = 0$ and $c_{\mathsf{U}} = 3$ from the beginning of protocol. However, the user's device memory is prone to certain attacks throughout the protocol execution, which corresponds to restricted variant of adaptive corruption. It means that the corruption level can only increase (to $c_{\mathsf{U}} = 1$ and $c_{\mathsf{U}} = 2$), which corresponds to adversary learning more information about authentication secret key.

AMS can be statically corrupted throughout the protocol. In particular, we assume a covertly [6] corrupted server. Meaning that the AMS can choose not to follow the protocol, but will not do it if the probability of being caught is sufficiently large. AMS is acting as supporting server in both blind server-supported signing and decryption protocols. Due to the blinding property of both protocols, supporting server does not learn which message is being signed or which ciphertext is being decrypted.

Issuer can be statically corrupted throughout the protocol. We set the requirement that the issuer should not be able to track credential presentations (unless colluded with RP) and revocations by the user (unless colluded with AMS) of its credentials.

Relying party is a party that requests to verify certain user's attributes to authenticate user or to give access to certain services. We assume that RP can be statically corrupted. We require that RP should not be able to learn anything except the attributes than the user has explicitly shown to it from the credential presentation. However, in this system, we do not aim for unlinkability of presentations, as this can be tracked through the issuer's signature. Additionally, in the system we aim for compatibility with mDL and ISO standards which do not hide the signature. Changing that will make our construction considerably deviate from those standards.

Functional requirements:

- User should be able to make queries connected to their credentials by credential identifier cID. To make querying by the credential identifier usable in practice, we need to define it to be self-explanatory.
- AMS should be logging information about transactions with the user credentials (identified by cID). The user should be able to request information about transactions with their credentials.
- User allows AMS to communicate with issuers per user's requests, accompanied with the user's consent.

6.2 Ideal functionality

Ideal functionality for the centralised identity management system is defined in Figure 18, Figure 19, and Figure 20.

Initialisation is used to initialise $\mathcal{F}_{\mathsf{ideal}}$ internal tables:

- T_a for storing attributes,

- T_r for storing revocation information,
- T_p for storing credential presentations,
- T_v for storing verification information.

Next, $\mathcal{F}_{\mathsf{ideal}}$ initialises the flag $\mathrm{c_U} = 0$ that is used to store information about the level of user corruption. Finally, $\mathcal{F}_{\mathsf{ideal}}$ lets adversary to define Present and Ver functions. Present function is used to generate credential presentation for the presentation phase, and Ver is the corresponding verification function. $\mathcal{F}_{\mathsf{ideal}}$ will later use these functions to be able to output a particular presentation to $\mathcal{Z}$, which can later be forwarded to a verifier, who can verify it using verification interface of $\mathcal{F}_{\mathsf{ideal}}$.

Silent request is a special interface for $\mathcal{A}$ to query Issue, Revoke or Present interfaces on behalf of corrupted initiator (user or issuer). If it is queried on behalf of a user, $\mathcal{F}_{\mathsf{ideal}}$ firstly calls the authentication sub-routine which verifies user corruption level:

- if *user is not fully corrupted* ($\mathrm{c_U} < 2$), *adversary is not allowed to make those requests* to the $\mathcal{F}_{\mathsf{ideal}}$;
- if user is fully corrupted ($\mathrm{c_U} \geq 2$), $\mathcal{F}_{\mathsf{ideal}}$ proceeds with the requested interface using input provided by the adversary.

$\mathcal{F}_{\mathsf{ideal}}$ proceeds with forwarding the user's output to the adversary. By defining functionality in this way, we can ensure that even though the adversary will see all the inputs and outputs of partially corrupted users, it cannot initiate any routines on their behalf (unless the user is fully corrupted).

Issue is interface for user to request attributes from the issuer. It is initiated by the user's input providing information about the requested attributes. If no-one is corrupted, $\mathcal{A}$ just gets notified that the issuing process has been initialised and is able to interrupt the communication. If storage (ST) is corrupted, we are not hiding issuer's identity and the issued $\overrightarrow{\mathsf{aID}}$ from it. The issuer identity is not hidden from the AMS due to the systems functional requirements to enable AMS communicate with issuers per user's requests. Since, ST knows the issuer identifier, it can likely guess which kinds of attributes have been requested by the user. Therefore, we are not hiding $\overrightarrow{\mathsf{aID}}$ from corrupted ST.

Next, $\mathcal{F}_{\mathsf{ideal}}$ sends attribute request to the issuer and, if not interrupted by the adversary, the issued credential consisting of attributes $\overrightarrow{\mathsf{atr}}$ is recorded to the table T_a. We let $\mathcal{A}$ choose credential identifier cID and $\mathcal{F}_{\mathsf{ideal}}$ checks that it is unique for an honest user and an honest AMS. Corrupted AMS learns only cID, we do not leak values of attributes to it.

Revoke interface can be invoked by either the user or the issuer on input message containing cID – identifier of credential that should be revoked. Firstly, $\mathcal{F}_{\mathsf{ideal}}$ verifies that such credential exists. The leakage to adversary in the revocation process depends on several properties:

- if both AMS and RP are corrupted and the credential has been presented to the corrupted RP before – $\mathcal{A}$ learns cID. In this case, $\mathcal{A}$ has already seen credential presentation, including the corresponding revocation token and knows that specific token corresponds to this cID;
- if user is corrupted – $\mathcal{A}$ learns cID;
- otherwise, $\mathcal{A}$ is just notified that the protocol has been initialised.

$\mathcal{F}_{\text{ideal}}$ also leaks to corrupted AMS the identity of the user, whose credential is being revoked. This leakage comes form the system functional requirement of logging information about the transactions with user credentials. To be able to save this information to the logs, AMS server should know who has been the initiator of revocation process and in case of issuer revocation, whose credential is being revoked. In this way AMS can show to the user only those revocation logs which concern that user.

If the protocol has not been interrupted by $\mathcal{A}$, $\mathcal{F}_{\text{ideal}}$ records that cID has been revoked by user or issuer in table T_r and confirms successful revocation to the revoker.

Present interface can be initialised by the user on message containing credential identifier cID and identifiers of attributes that will be disclosed $\overrightarrow{\text{aID}}_{\text{dscl}}$. Firstly, $\mathcal{F}_{\text{ideal}}$ verifies that such credential exists. Next, $\mathcal{F}_{\text{ideal}}$ notifies $\mathcal{A}$ that presentation process has been initialised and leaks only the number of disclosed attributes even if AMS and the issuer are corrupted.

If presentation is approved by $\mathcal{A}$, $\mathcal{F}_{\text{ideal}}$ verifies that credential has not been revoked, by consulting table T_r. $\mathcal{F}_{\text{ideal}}$ proceeds by generating presentation using Present function provided by adversary. The obtained credential presentation is recorded in internal table T_v and is outputted to the user.

Verify is initiated by the user on message containing presentation present and identifier of RP. For honest user, $\mathcal{F}_{\text{ideal}}$ verifies that the presentation has been produced by the $\mathcal{F}_{\text{ideal}}$ by consulting table T_v. For corrupted user, $\mathcal{F}_{\text{ideal}}$ verifies that credential being presented has been recorded by the issuer, that presentation verifies and that credential has not been revoked. As far as no revocations took place between a presentation and a verification, both checks always verify for the presentations generated through the $\mathcal{F}_{\text{ideal}}$. Next, $\mathcal{F}_{\text{ideal}}$ produces leakage to $\mathcal{A}$:

- if RP is corrupted (with or without AMS), $\mathcal{A}$ learns presentation and identifiers of disclosed attributes;
- if AMS is corrupted (with or without the issuer of presented credential), $\mathcal{A}$ learns neither RP identifier nor credential identifier.

$\mathcal{F}_{\text{ideal}}$ waits for confirmation from $\mathcal{A}$, containing presentation challenge ch and verifies it in case of honest RP. The value ch is introduced to the functionality to ensure freshness of credential presentation, which is checked for an honest RP. Finally, $\mathcal{F}_{\text{ideal}}$ outputs presentation along with outcome of verification v to user and RP.

Corrupt allows adversary to make corruption requests identifying which party is being corrupted. User corruption is handled as described in Sec. 6.1.

6.2.1 Properties of ideal functionality

Table 9 presents which values are leaked to the adversary in the protocols given corruption of certain parties.

Additionally, we claim that functionality $\mathcal{F}_{\text{ideal}}$ satisfies the following properties, formally stated and proved in below:

- The user and the AMS server cannot convince RP of possession of attributes that have not been issued to the user using issuing interface of $\mathcal{F}_{\text{ideal}}$.
- The user cannot convince RP of possession of the credential that has been revoked by the user themselves or by the issuer of that credential.
- An adversary who has only partially corrupted the user cannot make issuing, presentation, or revocation requests.

Initialisation. On input $(\mathsf{setup}, \mathsf{sid})$ from $\mathcal{A}$:

- Create empty tables $\mathsf{T}_a, \mathsf{T}_r, \mathsf{T}_p, \mathsf{T}_v$, set flag $\mathsf{c}_\mathsf{U} = 0$.
- Send $(\mathsf{setup\text{-}ok}, \mathsf{sid})$ to $\mathcal{A}$ and wait for $(\mathsf{setup\text{-}ok}, \mathsf{sid}, \mathsf{Ver}, \mathsf{Present})$ from $\mathcal{A}$.

Corrupted inputs. At any time when $(\mathsf{routine}, \mathsf{sid}, x)$ is called from $\mathcal{Z}$ by a party $P \in \mathcal{C}$, send $(\mathsf{input}, P, \mathsf{routine}, \mathsf{sid}, x)$ to $\mathcal{A}$. Wait for $(\mathsf{input}, P, \mathsf{routine}, \mathsf{sid}, x')$ from $\mathcal{A}$ and run routine with input x' instead of x.
Silent request. At any time, on input $(\mathsf{invoke}, \mathsf{routine}, \mathsf{sid}, P, \mathsf{inputs})$ from $\mathcal{A}$:

- If $\mathsf{ST} \in \mathcal{C}$, or $P = \mathsf{I}$, authentication is not required.
- Otherwise authenticate$(\mathsf{sid}, \mathsf{U})$, proceed if successful.

Wait for $(\mathsf{input}, P, \mathsf{routine}, \mathsf{sid}, x)$ from $\mathcal{A}$ and follow routine interface x as input of party P. Return output of routine to $\mathcal{A}$.
Issue. On input $(\mathsf{issue\text{-}req}, \mathsf{sid}, \overrightarrow{\mathsf{aID}}, \mathsf{I})$ from U:

- If $\mathsf{I} \in \mathcal{C}$ or $\mathsf{ST} \in \mathcal{C}$ send $(\mathsf{issue\text{-}req}, \mathsf{sid}, \mathsf{U}, \mathsf{I}, \overrightarrow{\mathsf{aID}})$ to $\mathcal{A}$. // I and ST learn $\overrightarrow{\mathsf{aID}}$
  Otherwise, send $(\mathsf{issue\text{-}req}, \mathsf{sid}, \mathsf{U}, \mathsf{I}, |\overrightarrow{\mathsf{aID}}|)$ to $\mathcal{A}$.
- Wait for $(\mathsf{issue\text{-}ok}, \mathsf{sid}, \mathsf{cID})$ from $\mathcal{A}$. Check that cID is unique.
  Send $(\mathsf{issue\text{-}req}, \mathsf{sid}, \mathsf{U}, \mathsf{cID}, \overrightarrow{\mathsf{aID}})$ to I.
- Upon receiving $(\mathsf{issue\text{-}atr}, \mathsf{sid}, \overrightarrow{\mathsf{atr}})$ from I:
  - If $\mathsf{U} \in \mathcal{C}$ send message $(\mathsf{issue\text{-}atr}, \mathsf{sid}, \overrightarrow{\mathsf{atr}})$ to $\mathcal{A}$. // U learns $\overrightarrow{\mathsf{atr}}$
  - Otherwise, send message $(\mathsf{issue\text{-}atr}, \mathsf{sid})$ to $\mathcal{A}$.
- Wait for $(\mathsf{atr\text{-}ok}, \mathsf{sid})$ response from $\mathcal{A}$.
- Record $(\mathsf{U}, \mathsf{I}, \mathsf{cID}, \overrightarrow{\mathsf{aID}}, \overrightarrow{\mathsf{atr}})$ to table T_a. Send $(\mathsf{issue\text{-}resp}, \mathsf{sid}, \mathsf{cID})$ to U and I.

Get all credential identifiers. On input $(\mathsf{getcids\text{-}req}, \mathsf{sid})$ from U:

- Send $(\mathsf{getcids\text{-}req}, \mathsf{sid}, \mathsf{U})$ to $\mathcal{A}$. Wait for $(\mathsf{getcids\text{-}ok}, \mathsf{sid})$ from $\mathcal{A}$
- Let $\overrightarrow{\mathsf{cID}}$ be all identifiers stored in T_a. Output $(\mathsf{getcids\text{-}resp}, \mathsf{sid}, \overrightarrow{\mathsf{cID}})$ to U.

Figure 18: Ideal functionality $\mathcal{F}_{ideal}$ (Part 1)

Revoke. On input $(\mathsf{revoke\text{-}req}, \mathsf{sid}, \mathsf{cID})$ from P:

- Find an entry $(\mathsf{U}, \mathsf{I}, \mathsf{cID}, \_, \_) \in \mathsf{T}_a$, s.t. $P \in \{\mathsf{I}, \mathsf{U}\}$.
  If there is no such entry, send $(\mathsf{revoke\text{-}resp}, \mathsf{sid}, \bot)$ to the party P.
- Produce leakage to the adversary:
  - If $\mathsf{ST} \in \mathcal{C}$ and $(\mathsf{RP}, \mathsf{cID}) \in \mathsf{T}_p$ for $\mathsf{RP} \in \mathcal{C}$, send $(\mathsf{revoke\text{-}req}, \mathsf{sid}, P, \mathsf{cID})$ to $\mathcal{A}$.
    // credential has been presented to the corrupted RP before
  - If $\mathsf{U} \in \mathcal{C}$ send $(\mathsf{revoke\text{-}req}, \mathsf{sid}, P, \mathsf{cID})$ to $\mathcal{A}$.
    // corrupted user learns if their credential is revoked
  - Otherwise, send $(\mathsf{revoke\text{-}req}, \mathsf{sid}, P, \mathsf{U})$ to $\mathcal{A}$.
- Wait for $(\mathsf{revoke\text{-}ok}, \mathsf{sid})$ response from $\mathcal{A}$.
- If there is no entry $(\mathsf{U}, \mathsf{cID}, P)$ in table T_r, add $(\mathsf{U}, \mathsf{cID}, P)$ to the table T_r.
- Send $(\mathsf{revoke\text{-}resp}, \mathsf{sid}, \mathsf{ok})$ to the party P.

Present. On input $(\mathsf{present\text{-}req}, \mathsf{sid}, \mathsf{cID}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}})$ from U:
// values in $\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}$ are identifying attributes that will be disclosed to the RP

- Find an entry $\mathsf{cred} = (\mathsf{U}, \mathsf{I}, \mathsf{cID}, \overrightarrow{\mathsf{aID}}, \overrightarrow{\mathsf{atr}}) \in \mathsf{T}_a$ with $\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}} \subseteq \overrightarrow{\mathsf{aID}}$.
  If there is no such entry, send $(\mathsf{present\text{-}resp}, \mathsf{sid}, \bot)$ to U.
- Send $(\mathsf{present\text{-}req}, \mathsf{sid}, \mathsf{U}, |\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}|)$ to $\mathcal{A}$. Wait for $(\mathsf{present\text{-}ok}, \mathsf{sid})$ from $\mathcal{A}$.
- If $(\mathsf{U}, \mathsf{cID}, \_) \in \mathsf{T}_r$, send $(\mathsf{present\text{-}resp}, \mathsf{sid}, \bot)$ to U.
- Generate $\mathsf{present} = \mathsf{Present}(\mathsf{cred}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}})$.
  If $\mathsf{Ver}(\mathsf{cred}, \mathsf{present}) = 1$, add $(\mathsf{cred}, \mathsf{present}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, 1)$ to T_v.
  Otherwise, send $(\mathsf{present\text{-}resp}, \mathsf{sid}, \bot)$ to U
- Send $(\mathsf{present\text{-}resp}, \mathsf{sid}, \mathsf{present})$ to U.

Figure 19: Ideal functionality $\mathcal{F}_{ideal}$ (Part 2)

Verify. On input $(\text{verify-req}, \text{sid}, \text{present}, \text{RP})$ from U:

- If $\text{U} \notin \mathcal{C}$, find $(\text{cred} = (\text{U}, \_, \text{cID}, \_, \_), \text{present}, \overrightarrow{\text{aID}}_{\text{dscl}}, 1) \in \text{T}_v$. Add (RP, cID) to T_p.

  // honest users only input presentations previously produced by $\mathcal{F}_{\text{ideal}}$

  If $\text{U} \in \mathcal{C}$, find $\text{cred} = (\_, \_, \text{cID}, \_, \_) \in \text{T}_a$ s.t. $\text{Ver}(\text{cred}, \text{present}) = 1$ and $(\_, \text{cID}, \_) \notin \text{T}_r$.

  // presentation of non-issued attributes cannot be forged

  // revoked credentials cannot be presented

  In both cases, if there is no such entry, send $(\text{verify-resp}, \text{sid}, \perp)$ to U and RP.

- Produce leakage to the adversary.

  - If $\text{RP} \in \mathcal{C}$ and $\text{ST} \in \mathcal{C}$, send $(\text{verify-req}, \text{sid}, \text{U}, \text{cID}, \text{present}, \overrightarrow{\text{aID}}_{\text{dscl}}, S_{ID})$ to $\mathcal{A}$,
    where $S_{ID} \leftarrow \{\text{sid} \mid (\text{revoke-req}, \text{sid}, \text{U}, \text{cID})\text{was input to } \mathcal{F}_{\text{ideal}}\}$.
  - Otherwise, if $\text{RP} \in \mathcal{C}$, send $(\text{verify-req}, \text{sid}, \text{U}, \text{cID}, \text{present}, \overrightarrow{\text{aID}}_{\text{dscl}})$ to $\mathcal{A}$.
  - Otherwise, send $(\text{verify-req}, \text{sid}, \text{U})$ to $\mathcal{A}$.

- Wait for $(\text{verify-ok}, \text{sid}, \text{ch})$ from $\mathcal{A}$. If $\text{RP} \notin \mathcal{C}$, check $\text{ch} \in \mathcal{CH}$.

- Set $\text{present}' \leftarrow (\text{present}, \text{ch})$. Set $\mathcal{CH} \leftarrow \mathcal{CH} \setminus \{\text{ch}\}$.

- Send $(\text{verify-resp}, \text{sid}, \text{present}', v)$ to RP and to U.

Corrupt.

- // corruption in the beginning of the protocol

  On input $(\text{corrupt-party}, P, \text{sid})$ from $\mathcal{A}$ where $P \in \{\text{I}, \text{RP}, \text{ST}, \text{U}\}$,

  set $P \in \mathscr{C}$ (if $P = \text{U}$, set $\text{c}_\text{U} = 3$) and send $(\text{corrupted}, P, \text{sid})$ to $\mathcal{A}$.

- // corruption after initialisation for user

  On input $(\text{corrupt-party}, \text{U}, \text{sid})$ from $\mathcal{A}$:

  - if $\text{c}_\text{U} = 0$, set $\text{c}_\text{U} = 1$ and $\text{U} \in \mathcal{C}$;
  - if $\text{c}_\text{U} = 1$, set $\text{c}_\text{U} = 2$ and send $(\text{full-leakage}, \text{sid}, \text{U}, (\text{T}_a^\text{U}, \text{T}_r^\text{U}))$ to $\mathcal{A}$, where T^U is projection of table T to the user U.

authenticate(sid, U):

- Send $(\text{auth}, \text{sid})$ to $\mathcal{A}$ and wait for $(\text{auth-ok}, \text{sid})$.
- If $\text{c}_\text{U} \in \{0, 1\}$, ignore this query. // with partial corruption, no queries allowed
- If $\text{c}_\text{U} \in \{2, 3\}$, proceed with the protocol.

Figure 20: Ideal functionality $\mathcal{F}_{ideal}$ (Part 3)

The first two items are covered by Theorem 2. The third item is covered by Theorem 3.

Theorem 2. An adversary $\mathcal{A}$ and an environment $\mathscr{Z}$ running in parallel with $\mathcal{F}_{\text{ideal}}$, are not able to make a relying party accept an output $(\text{verify-resp}, \text{sid}, (\text{present\_cred}, \text{ch}), 1)$ if there is no $\text{cred} = (\_, \_, \text{cID}, \_, \_) \in \mathsf{T}_a$ that would satisfy $\text{Ver}(\text{cred}, \text{present})$ and $(\_, \text{cID}, \_) \notin \mathsf{T}_r$.

Proof. Let us assume an adversary $\mathcal{A}$ produced present, such that a relying party received an output $(\text{verify-resp}, \text{sid}, (\text{present}, \text{ch}), \overrightarrow{\text{aID}}_{\text{dscl}}, 1)$, but there is no $\text{cred} = (\_, \_, \text{cID}, \_, \_) \in \mathsf{T}_a$ s.t. $\text{Ver}(\text{cred}, \text{present}) = 1$ and $(\_, \text{cID}, \_) \notin \mathsf{T}_r$.

If $U \in \mathcal{C}$, then the checks $\text{cred} = (\_, \_, \text{cID}, \_, \_) \in \mathsf{T}_a$, $\text{Ver}(\text{cred}, \text{present}) = 1$, and $(\_, \text{cID}, \_) \notin \mathsf{T}_r$ are performed explicitly.

If $U \notin \mathcal{C}$, then present was produced as a response to the presentation command by $\mathcal{F}_{\text{ideal}}$. This implies that functionality has verified that there exists $\text{cred} \in \mathsf{T}_a$ such that $(\_, \text{cID}, \_) \notin \mathsf{T}_r$ before creating present. Passing subsequent verification check implies that $\text{Ver}(\text{cred}, \text{present}) = 1$. Altogether, this contradicts the assumption. □

Theorem 3. An adversary $\mathcal{A}$ and an environment $\mathscr{Z}$ running in parallel with $\mathcal{F}_{\text{ideal}}$, corrupting user for at most $\mathsf{c}_{\mathsf{U}} = 1$, are not able to issue requests (issuing, user revocation, presentation) to the ideal functionality.

Proof. Assume that $\mathcal{A}$, who partially corrupted user ($\mathsf{c}_{\mathsf{U}} = 1$) makes request on behalf of the user to the issuing, user revocation or presentation interface. Each of the interfaces firstly calls authenticate sub-routine with the user identifier U as input. To proceed with the interface instructions, it should hold that either the user is fully corrupted by the adversary ($c_{\mathsf{U}} = 2$), or the interface has been called by the honest user, not the adversary, which contradicts the assumption. □

Table 9: User data disclosed to corrupted parties, where cID *and* $\overrightarrow{\text{aID}}$ *are credential and attribute identifiers,* $\overrightarrow{\text{atr}}$ *are attribute values,* $\overrightarrow{\text{aID}}_{\text{dscl}}$ *are the attributes used in a presentation, and* $\overrightarrow{\text{atr}}_{\text{dscl}}$ *is the disclosure of attribute values coming from that presentation. Disclosures marked by* * *are possible only for the given coalition of corrupted parties, and are* not *derived from smaller coalitions. Highlighted disclosures (yes) are intentional and needed for functionality.*

	Issue			**Revoke**	**Present**		
corrupted parties	cID	$\overrightarrow{\text{aID}}$	$\overrightarrow{\text{atr}}$	cID	$\overrightarrow{\text{aID}}$	$\overrightarrow{\text{atr}}$	cID
issuer	yes	yes	yes	no	no	no	no
AMS	yes	yes	no	no	only $\lvert\overrightarrow{\text{aID}}_{\text{dscl}}\rvert$	no	no
RP	no	no	no	no	only $\overrightarrow{\text{aID}}_{\text{dscl}}$	only $\overrightarrow{\text{atr}}_{\text{dscl}}$	yes
issuer + RP	yes	yes	yes	no	yes* (via cID)	yes* (via cID)	yes
AMS + issuer	yes	yes	yes	no	only $\lvert\overrightarrow{\text{aID}}_{\text{dscl}}\rvert$	no	no
AMS + RP	yes	yes	no	yes*	yes* (via cID)	only $\overrightarrow{\text{atr}}_{\text{dscl}}$	yes
AMS + issuer + RP	yes	yes	yes	yes	yes	yes	yes

6.3 System definition

We define the system through the sub-protocols used to instantiate the system – signing (described through $\mathcal{F}_{\text{sign}}$), server-supported blind signing (described through $\mathcal{F}_{\text{tsign}}$), server-supported blind threshold decryption (described through $\mathcal{F}_{\text{tdecrypt}}$), and revocation (described through $\mathcal{F}_{\text{rev}}$). In this section, we present less formal definition of system phases in this section. The formal definition of the system through the calls to corresponding sub-functionalities is presented in Appendix 9. The AMS server uses revocation protocol to manage revocations. It also serves as the supporting server for server-supported blind threshold decryption and server-supported blind signing. The issuer uses $\mathcal{F}_{\text{sign}}$ (representing non-threshold signature) to sign issued credentials.

For regular signatures, we adopt the ideal digital signature functionality $\mathcal{F}_{\text{sign}}$ proposed by Canetti [19]. A signer can ask it to sign a message, and anyone can ask it to verify a signature, and the functionality keeps the result consistent once it has answered for a given message/signature pair.

$\mathcal{F}_{\text{rev}}$ is the revocation functionality, with the following participants. The server S (the AMS server) who maintains the set X^- of revoked tokens, the revoker R (can either be the issuer or the user), who receives the issued token and may revoke it later, the user U who can query token revocation status, and the verifier V (the RP) who learns revocation token x and whether $x \in X^-$.

$\mathcal{F}_{\text{tsign}}$ represents blind server-supported signing functionality. To initialise signing, inputs from both client and supporting server are required. Since it models blind signing, the message is input from the client side, supporting server does not learn the message. Produced signature can be verified by any party, and the verification procedure is similar to the one from $\mathcal{F}_{\text{sign}}$.

We write the building blocks with superscripts to indicate per-party instances – each issuer IID gets its own $\mathcal{F}_{\text{sign}}^{\text{IID}}$, and each user uID gets its own $\mathcal{F}_{\text{tsign}}^{\text{uID}}$ and $\mathcal{F}_{\text{tdecrypt}}^{\text{uID}}$.

Initialisation The protocol is presented in Figure 21. User and AMS are initialising the system by generating signing key pair for the threshold signing with $\text{sk}_{\text{ss-sign}}^{\text{U}}$ being user's private key share and $\text{sk}_{\text{ss-sign}}^{\text{S}}$ being server's private key share; and decryption key pair for threshold decryption with $\text{sk}_{\text{ss-sign}}^{\text{U}}$ being user's private key share and $\text{sk}_{\text{ss-dec}}^{\text{S}}$ being server's private key share.

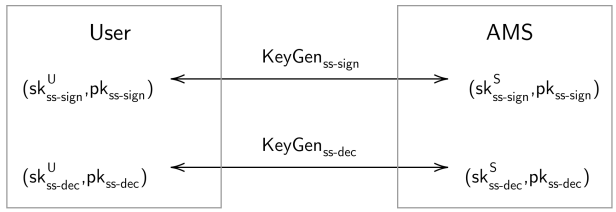


Figure 21: Prettiness initialisation phase

Authentication Each user query to the AMS server is initiated with authentication, which is performed in the form of generating and verifying digital signature. User creates a threshold signature σ_{req}, using PIN as input with AMS server participating in signature creation as supporting server. User adds tag_x input to the signing process that serves as

routine identifier. Threshold signature generation itself includes user authentication to the server. It is achieved through user proving ownership of the secret key share. Moreover, obtained signature serves as a consent from the user to request, share or revoke their credentials.

Issuing The protocol is presented in Figure 22. User starts with preparing issuing request req that consists of requested attributes $\overrightarrow{\mathsf{aID}}$, the issuer identifier IID and a fresh identifier id_{cID} for the credential. Upon successful user authentication, AMS server forwards attribute request to the issuer. To avoid repetition attacks by a corrupted AMS, the issuer keeps all received signatures and tracks their repetitions.

AMS provides to the user a link that they can use to directly communicate with the issuer IID (i.e. links IID to a communication channel of I), which is needed to generate revocation tokens rev. First, the user generates a token rev_U whose revoker will be the user. Then, the user and the issuer jointly generate a token rev_I, whose revoker will be the issuer.

For each requested attribute aID_i, the issuer comes up with a value atr_i. The issuer creates the credential by encrypting each atr_i under the user's public key $\mathsf{pk}_{\text{ss-dec}}$, including revocation tokens rev_I and rev_U. Next, the issuer signs the credential using $\mathsf{sk}^{\mathsf{IID}}_{\mathsf{sign}}$ and forwards it to the AMS server. AMS server verifies the issuer signature and forwards issued credential to the user for approval. User decrypts attributes from the credential using threshold decryption and verifies:

- that the attributes are not malformed (i.e. that the value atr_i belongs to the set of valid attribute values $\mathcal{V}_{\mathsf{aID}_i}$),
- that revocation token data is correct,
- the correctness of issuer signature.

If all the verification steps succeed, AMS server saves the issued credential with identifier $\mathsf{cID} = (id_{\mathsf{cID}}, \mathsf{summary}(\mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID}))$, where id_{cID} ensures uniqueness of cID, and function summary generates an intuitive short description of the credential. The user includes credential data into a small checksum digest that it maintains locally.

Get all cID**-s** Protocol is presented in Figure 23. It allows user to get all the previously issued credential identifiers cID and all the stored credentials $\overrightarrow{\mathsf{cred}}$. The user receives and verifies each individual cred in the issuing phase and updates local checksum accordingly. Therefore, user can verify whether the data received from the possibly corrupted AMS corresponds to the checksum. The issuer's signature on credentials protects each individual credential and the checksum prevents AMS from being silent about the existence of some credential. As an output of this protocol, user receives *encrypted* credentials, and will need further access to the decryption functionality to get their contents.

Revocation Revocation protocol can be initiated by either user (Figure 24) or issuer (Figure 25). Issuer revocation is initiated by the issuer's signed revocation request, which confirms that it is an authorised revocation and it is needed in the system to allow law enforcement to apply. Issuer revocation does not involve user participation. In both cases, revocation of cID happens through revoking the corresponding revocation token (rev_U or rev_I).

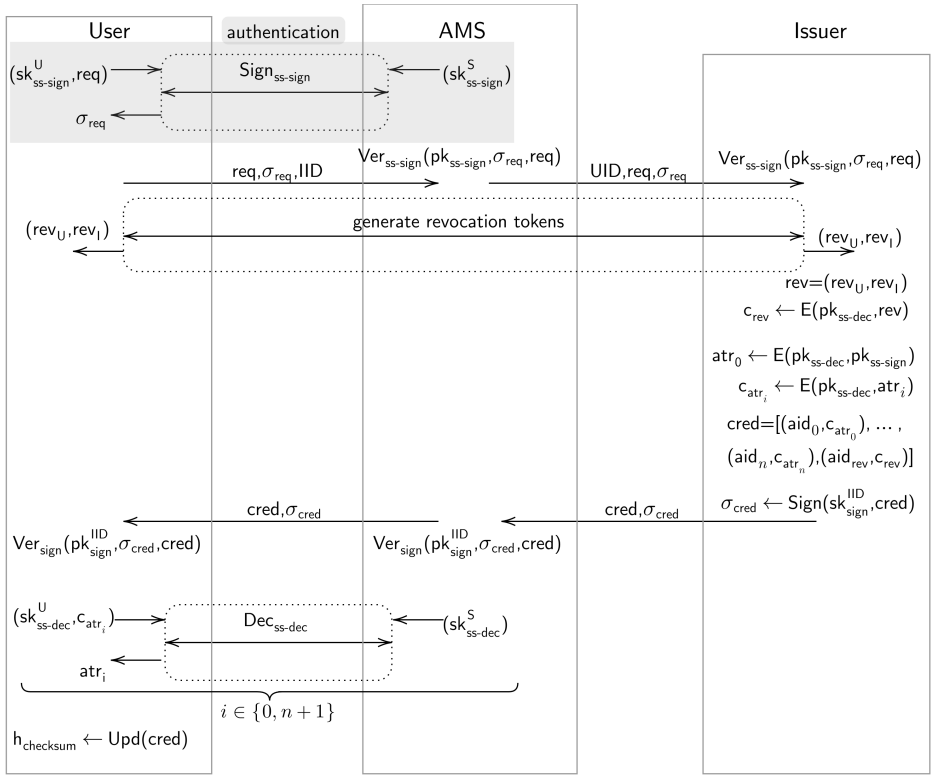


Figure 22: Prettiness issuing phase

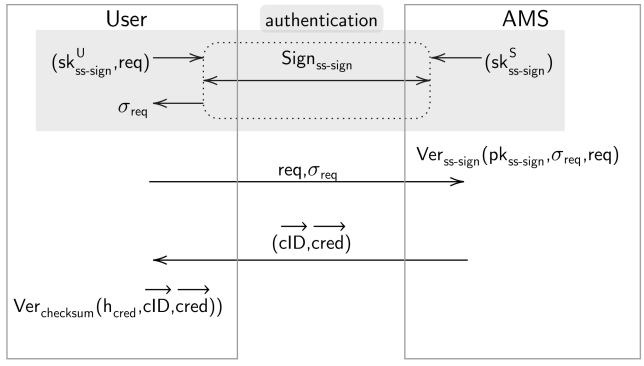


Figure 23: Prettiness get all credentials

It is allowed that credential gets revoked twice – by the user and by the issuer. The verification procedure by RP includes verifying that both rev_I and rev_U have not been revoked. As the user will check revocation status of credential immediately before presenting it, RP does not learn whether credential has been revoked by the user, by the issuer, or by both. This leakage is only possible if the AMS server collaborates with a corrupted RP.

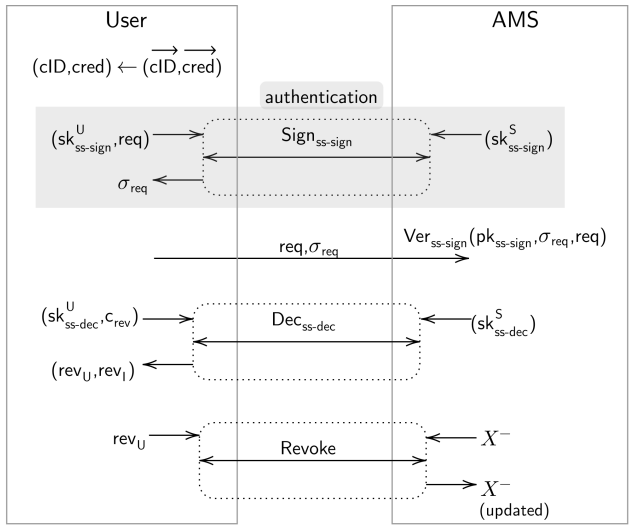


Figure 24: Prettiness revocation by the user

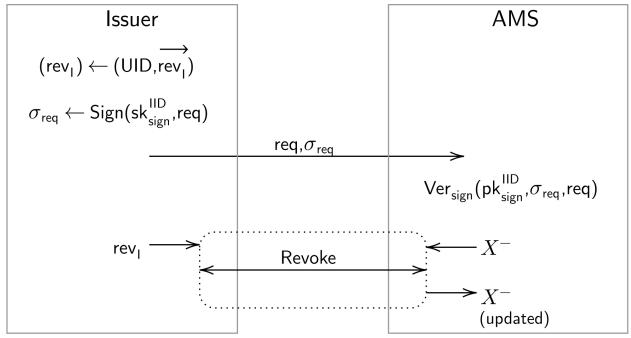


Figure 25: Prettiness revocation by the issuer

Presentation The protocol of Figure 26 allows the user to generate presentation for the credential with identifier cID and selectively disclose attributes $\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}$. The user retrieves credential cred identified by cID. Next, user proceeds with authentication which is followed by decryption of revocation token pair $\mathsf{rev} = (\mathsf{rev}_I, \mathsf{rev}_U)$ using threshold decryption protocol with key share $\mathsf{sk}^{\mathsf{U}}_{\mathsf{ss\text{-}dec}}$. User verifies that the revocation tokens $\mathsf{rev} = (\mathsf{rev}_I, \mathsf{rev}_U)$ have not been revoked. Finally, user decrypts selected attributes $\mathtt{atr}_i$ using threshold decryption protocol with key share $\mathsf{sk}^{\mathsf{U}}_{\mathsf{ss\text{-}dec}}$. User generates credential presentation present

which contains only the selected disclosed attributes. All auxiliary data that will be needed to verify presentation validity will be provided in the verification phase (including user's signature on presentation).

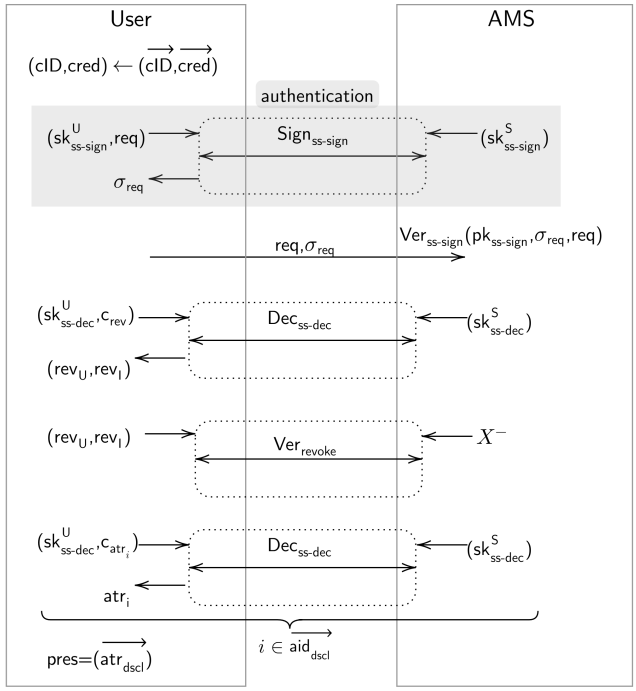


Figure 26: Prettiness presentation phase

Verification The protocol is presented in Figure 27. Before initiating the verification, RP receives updated state X^- from the revocation functionality using notification interface of revocation protocol. It is needed to get fresh information about the tokens that have been revoked. If RP requests updated revocation state immediately before the user's presentation, we cannot guarantee the unlinkability of that presentation to the RP by AMS server, so we require that it should be run independently. RP generates fresh presentation challenge and sends it to the user for signing. This step is required to guarantee freshness of the presentation.

User proceeds with generating signature on credential presentation with challenge received from RP using $\mathsf{sk}^{\mathsf{U}}_{\text{ss-sign}}$. The blindness property of the threshold signing protocol ensures that the AMS server does not learn anything about the credential presentation, except the fact that the user has presented some credential to some party. To make flow of the protocol similar to the SD-JWT, we include cred and σ_{cred} to the presentation message that gets signed.

Once RP receives credential presentation from the user, it verifies the user's signature on presentation and the issuer's signature on the credential. Next, RP verifies revocation tokens to ensure that the credential has not been revoked. Finally, it verifies whether the selectively disclosed attributes are indeed contained in the signed credential.

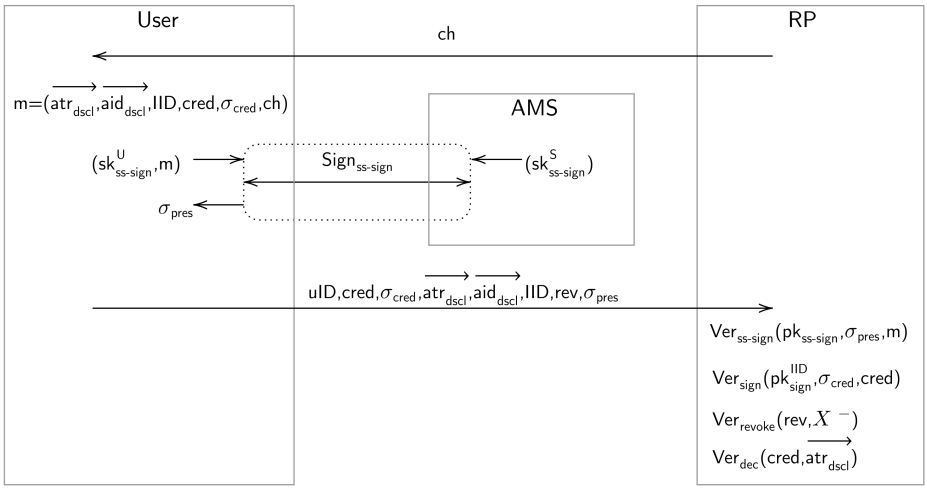


Figure 27: Prettiness presentation phase

Logging We are not defining the logging functionality of the system explicitly, but we assume that the AMS logs all request messages along with the time at which these requests have been made. This is needed to enable user to get an overview of transactions performed to their credentials. The list of logs and information that the user can extract from these is given in Table 10.

However, if the AMS is corrupted it can refuse providing log of transactions to the user. Therefore, logging functionality is useful under assumptions of a passively corrupted server. Additionally, if the issuer is corrupted, then there are no guarantees that valid revoked cID has been recorded. It is possible to introduce additional measures to improve threat model for the logging process, but these enhancements would require more details, and are out of scope of this work.

Table 10: Logged records and their interpretation.

Protocol	Logged record	What can be read by user
Issue	$(\text{tag}_{issue}, id_{\text{cID}}, \text{uID}, \overrightarrow{\text{aID}}, \text{IID})$	issued cID
Get cIDs	$(\text{tag}_{cids}, \text{uID})$	fact of credential backup request
Revoke(user)	$(\text{tag}_{rev}, \text{uID}, c_{\text{cID}})$	revoked cID
Revoke(issuer)	$(\text{tag}_{rev}, \text{IID}, c_{\text{cID}})$	revoker IID and cID
Present	$(\text{tag}_{pres}, \text{uID}, n, c_{\text{cID}})$	presented cID

Expiration All the credentials have certain expiration date and the expired records need to be periodically removed (or archived). Let Δ be an overestimated time after which any credential will "definitely expire". For both the credential and the revocation tables, the server removes an entry as soon as the time Δ passes after it has been added to the corresponding table.

6.4 Security

We start with an overview of corruption models of supporting server (AMS) in the used sub-protocols. The overview is presented in Table 11. Although the PRETTINESS system overall assumes a covertly corrupted AMS, for each individual building block the AMS operates according to the specific security level and corruption model defined for that respective functionality. The covert adversary model originates from the revocation functionality, where explicit cheating detection and accountability interfaces are integrated to ensure the AMS cannot silently alter or hide revocation status without risking exposure to third parties.

Table 11: Corruption model of sub-protocols

Protocol	Corruption of AMS
Blind server-supported signing	adaptive active corruption
Blind server-supported decryption	static active corruption
Revocation	static covert corruption

Theorem 4. Assuming a static covert corruption for AMS, static malicious corruption for user device, static malicious corruption for issuer, static malicious corruption for RP, the protocol set $\pi_{\text{PRETTINESS}}$ of Sec. 6.3 UC-realizes $\mathcal{F}_{\text{ideal}}$ in $\mathcal{F}_{\text{sign}}^{\text{IID}}, \mathcal{F}_{\text{tsign}}^{\text{uID}}, \mathcal{F}_{\text{tdecrypt}}^{\text{uID}}, \mathcal{F}_{\text{rev}}$- hybrid model.

Below we present sketch of the security proof, the full security proof is presented in [71]. In the proof, we construct a simulator $\mathcal{S}$ that is mediating communication between the real adversary $\mathcal{A}$ and the ideal functionality $\mathcal{F}_{\text{ideal}}$. The goal of constructing simulator is to generate messages for $\mathcal{A}$ and $\mathcal{F}_{\text{ideal}}$ in such a way that $\mathcal{Z}$ gets convinced that it is communicating with the real protocol $\pi_{\text{PRETTINESS}}$. The simulator acts in a role of honest parties in the protocol, generating messages that are sent to the corrupted parties, which are controlled by $\mathcal{A}$. Additionally, $\mathcal{S}$ interacts with $\mathcal{F}_{\text{ideal}}$ to synchronise with its inputs and outputs.

Initialisation. On input from $\mathcal{F}_{\text{ideal}}$, $\mathcal{S}$ defines functions Present and Ver. Present is used to generate credential presentation as follows – on input $\text{cID}, \overrightarrow{\text{aID}}$, it takes the attribute values $\overrightarrow{\text{atr}}$ that correspond to cID and $\overrightarrow{\text{aID}}$ from the table T_a of issued attributes (as is done in the real protocol). Ver is used to verify credential presentation and is defined through checking that $\overrightarrow{\text{atr}}$ have indeed been stored in T_a for given cID and $\overrightarrow{\text{aID}}$.

Additionally, $\mathcal{S}$ also initializes all sub-functionalities $\mathcal{F}_{\text{tsign}}^{\text{uID}}$, $\mathcal{F}_{\text{tdecrypt}}^{\text{uID}}$ (a separate instance for each user of the system), $\mathcal{F}_{\text{rev}}$ (one instance for the entire system) and $\mathcal{F}_{\text{sign}}^{\text{IID}}$ (a separate instance for each issuer in the system).

Issuing. The goal of $\mathcal{S}$ in the issuing phase is to ensure that if the entry $(\text{cID}, \text{uID}, \_)$ for an honest user uID is added to T_a, then cID should indeed have been requested by uID. While simulating honest issuer, $\mathcal{S}$ treats credential requests from a malicious user as successful only if it could come up with some valid attribute values $\overrightarrow{\text{atr}}$ for that user and if generation of revocation token for that user has succeeded. Meaning that a credential will only be sent to the user if the issuer has possibility to revoke it later.

While simulating honest user, before accepting credential from a malicious issuer, simulator needs to make sure that successful issuing will be followed by successful presentation and revocation. Therefore, simulator verifies that:

- credential contains attributes from the valid set of the attributes;

- issued credential contains revocation token rev generated by the $\mathcal{S}$;
- credential contains valid issuer's signature.

Finally, all credential data except the party identifiers and the attribute identifiers $\overrightarrow{\mathsf{aID}}$ (as those values are explicitly leaked to $\mathcal{A}$) is encrypted, and can be simulated to a corrupted AMS. Also $\mathcal{S}$ maintains local table $\mathsf{T}_{\mathsf{issue}}$, where it saves credential record with additional protocol-specific data, like signatures and revocation tokens.

Get all CID-s. While simulating honest AMS, $\mathcal{S}$ simulates to a corrupted user all the cID-s and credentials that have been issued to the corrupted user so far. All these credentials have been already simulated to $\mathcal{A}$ during the issuing phase and hence the records are stored in internal table $\mathsf{T}_{\mathsf{issue}}$.

While simulating honest user, $\mathcal{S}$ verifies the entries provided by corrupted AMS against the digest representing the data that the AMS should have shown, this ensures consistency with the output of $\mathcal{F}_{\mathsf{ideal}}$.

Revocation. While simulating honest AMS, $\mathcal{S}$ runs the protocol with corrupted revoker, letting $\mathcal{A}$ choose the token to revoke. $\mathcal{A}$ cannot illegitimately revoke a credential cID of an honest user if it does not have rights on its revocation (provided by $\mathcal{F}_{\mathsf{rev}}$). While simulating honest revoker, $\mathcal{S}$ takes a random revocation token $\mathsf{rev} \leftarrow \mathcal{X}$, and simulates calls of $\mathcal{F}_{\mathsf{rev}}$ with that token. Randomly selected token will not coincide with any other token with probability $1/|\mathcal{X}|$. If the user of the revoked credential or the RP is corrupted, $\mathcal{S}$ receives cID from $\mathcal{F}_{\mathsf{ideal}}$ and can use the same token rev that was issued for cID. $\mathcal{A}$ will not notice the difference, until it gets more details of revocation. This can happen either upon reaching higher user corruption level $c_{\mathsf{U}} = 2$, or during presentation to a corrupted RP. In both cases, $\mathcal{F}_{\mathsf{ideal}}$ will provide all relevant revocation data to $\mathcal{S}$, so that it can add all relevant tokens rev to X^- of $\mathcal{F}_{\mathsf{rev}}$.

$\mathcal{S}$ maintains local set X^-, where it saves revocation tokens rev that have been revoked. The only case $\mathcal{S}$ cannot update its local set is when cID has not been leaked to $\mathcal{S}$ yet. This corresponds to the case where the user of cID is an honest party, and it will anyway not call verification of $\mathcal{F}_{\mathsf{rev}}$ with a revoked token.

Presentation. While simulating honest AMS, $\mathcal{S}$ interacts with corrupted user and lets $\mathcal{A}$ choose the ciphertexts of their choice for decryption.

When simulating honest user, $\mathcal{S}$ simulates presentation for a randomly chosen credential cID' from internal table $\mathsf{T}_{\mathsf{issue}}$, which is consistent with T_a of $\mathcal{F}_{\mathsf{ideal}}$. If there is no such cID', presentation trivially fails, as there have been no credentials issued to that user. $\mathcal{A}$ only learns the number of threshold decryptions, which corresponds to $|\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}|$. $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$ does not leak to AMS which ciphertexts exactly have been decrypted. Even though $\mathcal{S}$ used cID' for its internal simulation, the messages that it generated for $\mathcal{A}$ do not actually depend on cID'. $\mathcal{S}$ does not learn whether presentation of an honest user has eventually succeeded, but this fact does not need to be simulated to $\mathcal{A}$.

Verification. For the honest user, corrupted RP case, $\mathcal{S}$ receives disclosed presentation data from $\mathcal{F}_{\mathsf{ideal}}$. This is enough for $\mathcal{S}$ to construct all messages for RP, using additional data from internal table $\mathsf{T}_{\mathsf{issue}}$. Received cID may be different from cID' previously chosen for the presentation phase, but the simulation of presentation has not depended on cID' anyway.

For the case, when the user is corrupted with level $c_{\mathsf{U}} \geq 2$ (with $c_{\mathsf{U}} = 1$, corruptions are passive), it may try to present a revoked or invalid credential to an honest RP. $\mathcal{S}$ can detect that behaviour using its internal tables for revocation, signing and decryption for that user. If $c_{\mathsf{U}} = 3$, all these tables are valid for that user from the beginning. When user corruption increases to $c_{\mathsf{U}} = 2$, $\mathcal{F}_{\mathsf{ideal}}$ leaks internal records of that user to $\mathcal{S}$, allowing it

to update its tables accordingly. Hence, these tables are filled with data consistent with the internal tables of $\mathcal{F}_{\text{ideal}}$.

For the case where both RP and ST are corrupted, $\mathcal{A}$ expects to learn in which session the presented credential cID has been revoked (if such a session exists). In this case, $\mathcal{F}_{\text{ideal}}$ sends a corresponding index of the revocation table to $\mathcal{S}$, from which it can refer to the sid of that revocation.

6.5 Performance

We have estimated PRETTINESS performance. The benchmarks for sub-components were taken from implementations of protocols UC-realising $\mathcal{F}_{\text{rev}}$, $\mathcal{F}_{\text{tdecrypt}}$, $\mathcal{F}_{\text{tsign}}$, and $\mathcal{F}_{\text{sign}}$. We run the benchmarks on Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz, 4 cores, 2 threads per core, and 15GB RAM.

Blind server-supported signing The benchmarks for a protocol UC-realizing $\mathcal{F}_{\text{tsign}}$ has been taken from [15], where it is based on RSA signature. The protocol presented in [69] just adds blinding to it, therefore the benchmarks will be similar. Signing operation by client and server takes tens of milliseconds, which we for safety round up to 0.1 seconds. According to [15], the most time overhead comes from the network delay to transmit the signature share from the client to the server, which we do not include into the timing benchmark. The bandwidth has been estimated instantiating [15] with 2048-bit RSA and 128-bit randomness. The communication consists of a client signing message $\langle y, m, r, r' \rangle$, and a server signing message $\langle s, m \rangle$. Here y and s are 2048-bit RSA signatures, and $r, r' \in \{0,1\}^{\eta}$. The estimates are presented in Table 12.

Table 12: Efficiency of $\mathcal{F}_{\text{tsign}}$ realised by [69]. Notice: * without network delay. $|m|_b$ is the message length in bits.

Signature size	Client signing message	Server signing message	Signing time*
2048 bits	$2304 + \|m\|_b$	$2048 + \|m\|_b$	0.1 s

Revocation Efficiency estimation of revocation protocol is taken from [55]. While the issuing and verification processes of this protocol do not depend on the total number of revoked tokens, the efficiency of notifications and revocations depends on the total number of revoked elements. We reproduce the relevant benchmarks in Table 13 and Table 14. In this work, we assume that there are $m = 1,000,000$ revoked tokens. However, we also present results for smaller values of m to demonstrate efficiency during the initial phases when the number of revoked credentials remains low.

Table 13: Benchmarks (runtime) for different routines of revocation protocol. Notation "–" represent numbers that are the same as for $m = 0$.

Local computation times (ms)

m	0	100	10,000	1,000,000
$\mathcal{F}_{\text{rev}}$(issue)	0.00447	–	–	–
$\mathcal{F}_{\text{rev}}$(revoke)	0.108	16.5	23.8	743
$\mathcal{F}_{\text{rev}}$(unotify)	0.081	0.137	5.57	673
$\mathcal{F}_{\text{rev}}$(vnotify)	0.075	0.137	6.52	716
$\mathcal{F}_{\text{rev}}$(verify)	$\leq$2300	–	–	–

Table 14: Benchmarks (bandwidth) for different routines of revocation protocol. Notation "–" represent numbers that are the same as for $m = 0$.

Communication (bytes)				
m	0	100	10,000	1,000,000
$\mathcal{F}_{\text{rev}}(\text{issue})$	48	–	–	–
$\mathcal{F}_{\text{rev}}(\text{revoke})$	160	–	–	–
$\mathcal{F}_{\text{rev}}(\text{unotify})$	114	3,314	320,114	32,000,114
$\mathcal{F}_{\text{rev}}(\text{vnotify})$	114	3,314	320,114	32,000,114
$\mathcal{F}_{\text{rev}}(\text{verify})$	162	–	–	–

The benchmarks for blind server-supported decryption are presented in Section 5.5. Since the efficiency of PRETTINESS system depends on the actual number of credentials and their sizes, for the system estimation we assume a particular use case with $N = 100$ credentials per user, $n = 10$ attributes per credential, and $m = 1,000,000$ total revocations. In practice, such an m can be an overestimate if there are not too many urgent revocations of active credentials (for which the expiration time has not yet passed). The summary of results is given in Table 15. As user and RP notifications require a large download of the revocation database, which can be performed in advance, we treat this as a separate routine, denoted as **DB update**, in our estimates. This routine has similar efficiency for both the user and the RP.

Table 15: Summarised efficiency of the full system for $N = 100$ credentials per user, $n = 10$ attributes per credential, $m = 10^6$ revoked tokens.

	Issue	**Get Creds**	**Rev(I)**	**Rev(U)**	**Present**	**DB update**	**Verify**
comm (B)	90,716	3,906,474	3,355	6,688	7,590	32,000,114	122,170
time (s)	8.114	0.127	1.027	1.487	5.71	0.716	4.702

6.6 Chapter summary

Building on CACS limitations, this chapter presents PRETTINESS, an improved centralized identity system that minimizes information leakage to the Attribute Management Service. The architecture relies on blind server-supported signing and decryption to enable selective attribute disclosure and relies on dedicated revocation protocol that supports dual revocation (by both users and issuers) without compromising user privacy. Security is modelled through a UC ideal functionality and the system is formally proven to securely realise the proposed functionality under a covertly corrupted AMS assumption. Performance benchmarks confirm that the cryptographic overhead remains within acceptable bounds for real-world deployment. The constructed system is effectively balancing between regulatory compliance, usability, and privacy guarantees.

7 Comparison-Based Methodology for the Security Assurance of Novel Systems

For some applications relying on usage of cryptography, adherence to standards is a crucial requirement. Examples discussed in this thesis are threshold signatures and EUDIW. However, as was demonstrated in the previous sections, to achieve enhanced security guarantees, it is needed to use non-standard cryptographic constructions as building blocks. Usage of those constructions might complicate the standardisation process of the system. A notable example of a certified solution is Smart-ID, an authentication and signing service widely adopted in the Baltic states. The system is built upon a server-supported threshold signing scheme [15]. It successfully passed certification to be recognised as a QSCD (Qualified Signature Creation Device) [9]. This means that all digital signatures created using Smart-ID are equivalent to handwritten signatures. However, it is significantly more challenging to pass certification for a threshold signature scheme compared to a regular digital signature system due to the complexity of the trust model and the physical environment of the key shares being distributed. This chapter describes a methodology that allows comparing a previously certified (or standardised) system with an updated system that achieves the same functional and security requirements and shows that the novel system is at least as secure as the certified (or standardised) system. Additionally, we present an example application of the defined methodology to compare two authentication systems. In both systems, the RP server, knowing the user's public key, generates a challenge, which has to be signed with the corresponding private key held by the user. The systems differ in how the private key is stored, and how the signature is generated. One system uses a smart card to store the private key and generate a signature. The second system uses SplitKey technology [22] that builds upon a server-supported signature scheme.

7.1 Methodology

Let us define a certified system as S and a novel/updated system as S'. After identifying the systems to be compared, the comparison steps can be summarised as follows:

1. construct models describing processes of the systems S and S';
2. from the models, identify the main data elements, processes and decision points that can be targeted by the adversary;
3. identify the system weaknesses by analysing the identified data elements, processes and decision points in the systems S and S';
4. identify the relations between the listed weaknesses;
5. compare the sets of weaknesses for both systems;
6. identify security requirements for the system S' to be comparable with system S.

For the first step (modelling systems), we propose to use the Business Process Modelling Notation (described in Section 3.10). This approach enables modelling the processes happening in the system in enough detail, depicting visually the sequence of actions performed by the protocol participants. Moreover, BPMN diagrams allow explicitly showing

[9] https://cyber.ee/resources/news/splitkey-powered-smart-id-recognised-as-highest-level-of-electronic-signature-by-eu/

which data elements are used for which processes. This allows visually identifying all the information components (data objects), processes (tasks), and decision points (gateways) that can be targeted by the adversary (for Step 2).

For the third step (identify the system weaknesses), we proceed by analysing each identified data element in the storage, transmission and processing steps. Those are the steps when an adversary can tamper with a data element or change the outcomes of computation. Additionally, we analyse during which other processes and computations the adversary may get access to or tamper with the input and/or output data elements. We also analyse all the decision points from the process model, and identify how the adversary can interfere with the decision making process. The main types of weakness based on the diagram components are presented in Table 16. This is not a complete list of weaknesses that can be identified within a system. Depending on the system being analysed, there might be additional weaknesses identified, for example including attacks against privacy of the elements. An adversary may exploit one weakness either fully, or not exploit it at all, but there are no partial exploitations. The identified weaknesses may not be feasible to exploit in the real systems. However, we need to introduce them for purposes of comparing the systems, to identify whether there are other weaknesses with worse consequences or easier exploitation. The individual weaknesses are further grouped into sets based on which party's process or data element is targeted by the adversary.

Table 16: Main weakness types

Component of diagram	**Weakness types**
Data object	**Learn** – get unauthorised access to data object while in store or transit **Modify** – tamper with data object while in store or transit
Activity (Task)	**Interfere with** – change the way the task should be executed (including input and output data objects)
Gateway	**Make decision go other way** – influence the outcome of decision point

For the fourth step, we proceed by identifying relations between the sets of weaknesses. For this, we consider two types of relations – seriousness and difficulty of exploiting a weakness. We denote *seriousness* of a successful exploitation of a weakness as $\prec_{\mathrm{S}}$. For the weakness sets $\mathbf{w}_1$ and $\mathbf{w}_2$, we write $\mathbf{w}_1 \prec_{\mathrm{S}} \mathbf{w}_2$ to denote that the effects of exploiting all the weaknesses in $\mathbf{w}_1$ are no worse than the effects of exploiting $\mathbf{w}_2$. We denote *difficulty* of exploiting a weakness as $\prec_{\mathrm{D}}$. For the weakness sets $\mathbf{w}_1$ and $\mathbf{w}_2$, we write $\mathbf{w}_1 \prec_{\mathrm{D}} \mathbf{w}_2$ to denote that exploiting all the weaknesses in $\mathbf{w}_1$ is no more difficult than exploiting all the weaknesses in $\mathbf{w}_2$.

To be able to perform the fifth step (comparison of the systems), we require that both systems S and S' should be similar enough. This requirement is needed to be able to identify the relations between the weaknesses. This means that weakness sets for both systems should allow us to identify relations between them. However, it might not be the case in practice when comparing systems. To overcome this, we propose to follow an approach similar to the game-based proofs from cryptography. We construct intermediate systems, such that the weaknesses of two neighbouring systems are close enough to be able to identify the relations. As is the case with intermediate games in cryptographic proofs, intermediate systems introduced using this methodology are not necessarily realistic or deployable in real life. The goal of introducing them is to arrive from one system to another through changes small enough to allow for immediate analysis.

Finally, from the identified relations between the systems we can identify security requirements that should be in place in system S' for the identified attacks to be equivalent to each other.

7.2 Comparison of authentication methods

As an example of the proposed methodology, we present a comparison of two authentication methods. The first system S (Figure 28) is based on smart cards, where a private (signing) key is stored on a smart card.

The second system S' (Figure 29) is based on threshold cryptography. In a SplitKey system, the private key is split between the user's mobile device and the supporting server. The keyshare on the phone is encrypted with a low-entropy key that is derived from a PIN that the user enters. The system has implemented protection against offline guessing attacks – if the adversary has copied just the encrypted keyshare stored on the device, it cannot recognize which PIN is the correct one. The server can verify whether it has received a signature share from the phone that has been created with a wrong keyshare. When the phone receives an authentication challenge that needs to be signed, the device asks the user to input the PIN. In order to help the user understand whether it is a valid authentication request, the phone shows the user a short *control code* (CC), derived from the challenge. The same control code is shown to the user by the RP's website. The user is supposed to enter the PIN only if the control codes match.

As was explained above, systems S and S' cannot be compared directly as they are too different. Therefore, we introduce a sequence of intermediate systems, where we make small modifications to the SplitKey system S' until we arrive at a system that is close enough to S. In this example, close enough means that the signing process happens fully on one party's side with a single private key.

The intermediate systems are constructed as follows: we change system S' into system S_1 (Figure 30), where instead of computing the first signature share on the phone, we move all the computations to the SplitKey server. We let the SplitKey server store the user's private key share encrypted with the user's PIN, and we send the PIN together with the challenge from the phone to the SplitKey server.

Next, we change system S_1 into system S_2 (Figure 31), where instead of computing a combined signature from two signature shares, the SplitKey server generates a single signature using a single private key. The private key is not encrypted with the user's PIN. Instead, the server verifies the correctness of the user's PIN received together with the challenge by comparing the PIN received from the phone with the one stored on the server side.

System S_2 is close enough to the smart card system S and can be compared directly.

The comparison of the systems explained above helped to identify the system requirements that should be in place in the SplitKey system for the identified attacks to be equivalent to each other. The identified requirements are:

- the server should protect the private key share at least as strongly as the smart card;
- it should be as hard to interfere with the combined signature verification process in the SplitKey server as it is to interfere with the PIN verification in the smart card;
- it should be as hard to interfere with the signature creation/combination process in the SplitKey server as it is in the smart card system;
- it should be as hard to change the server's private key share that is used to create the signature in the SplitKey server as it is in the smart card system.

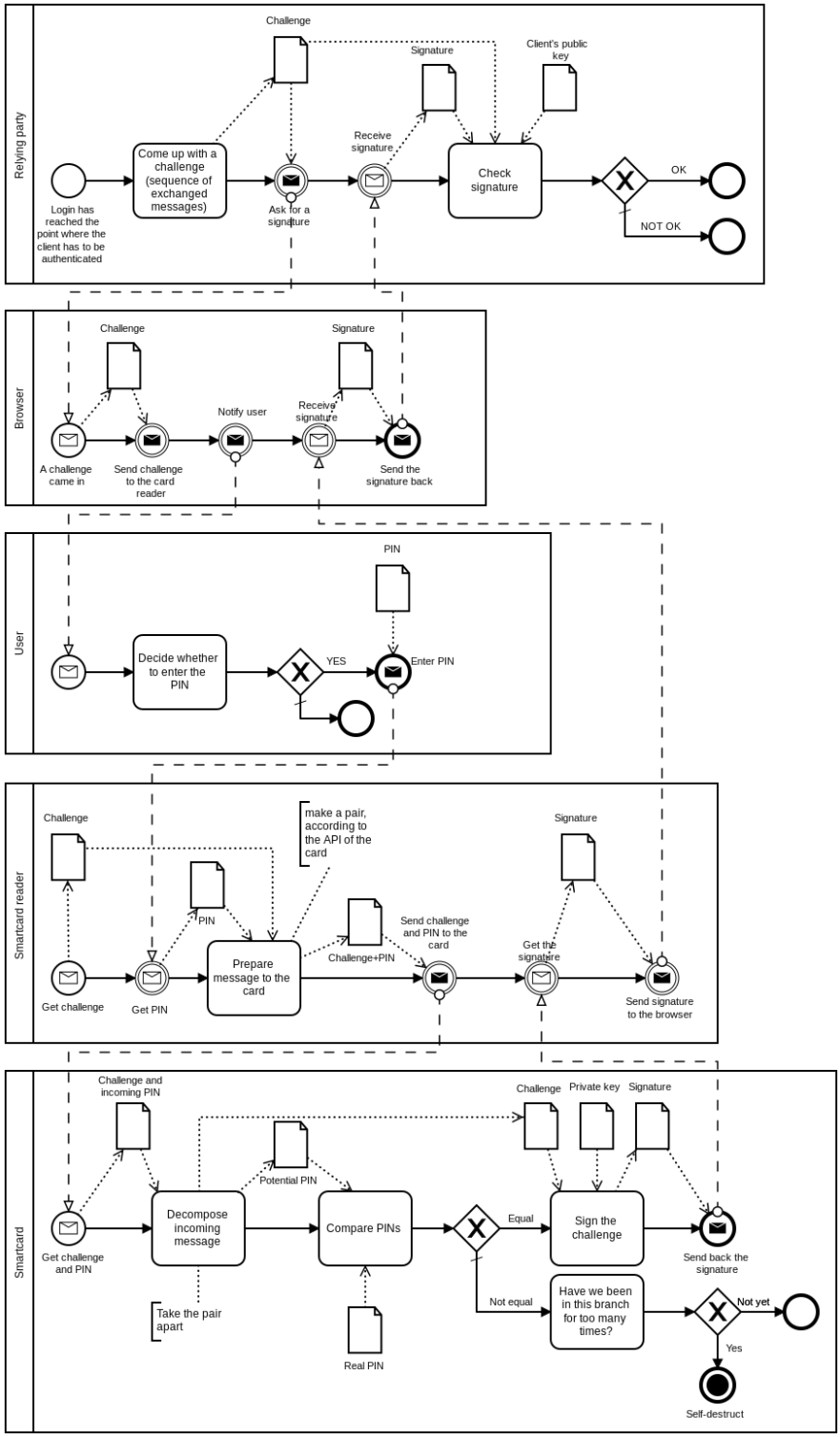


Figure 28: The smart card system

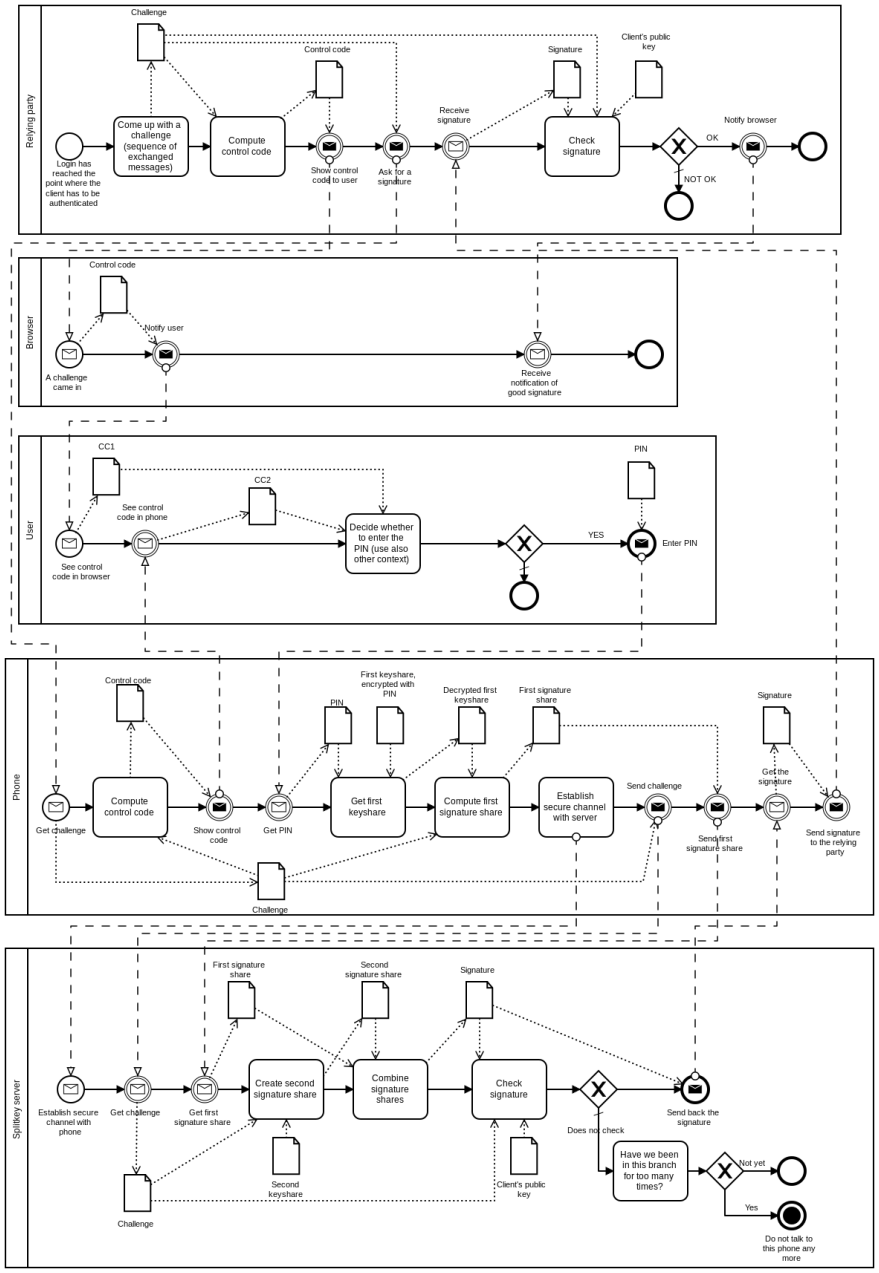


Figure 29: The SplitKey system

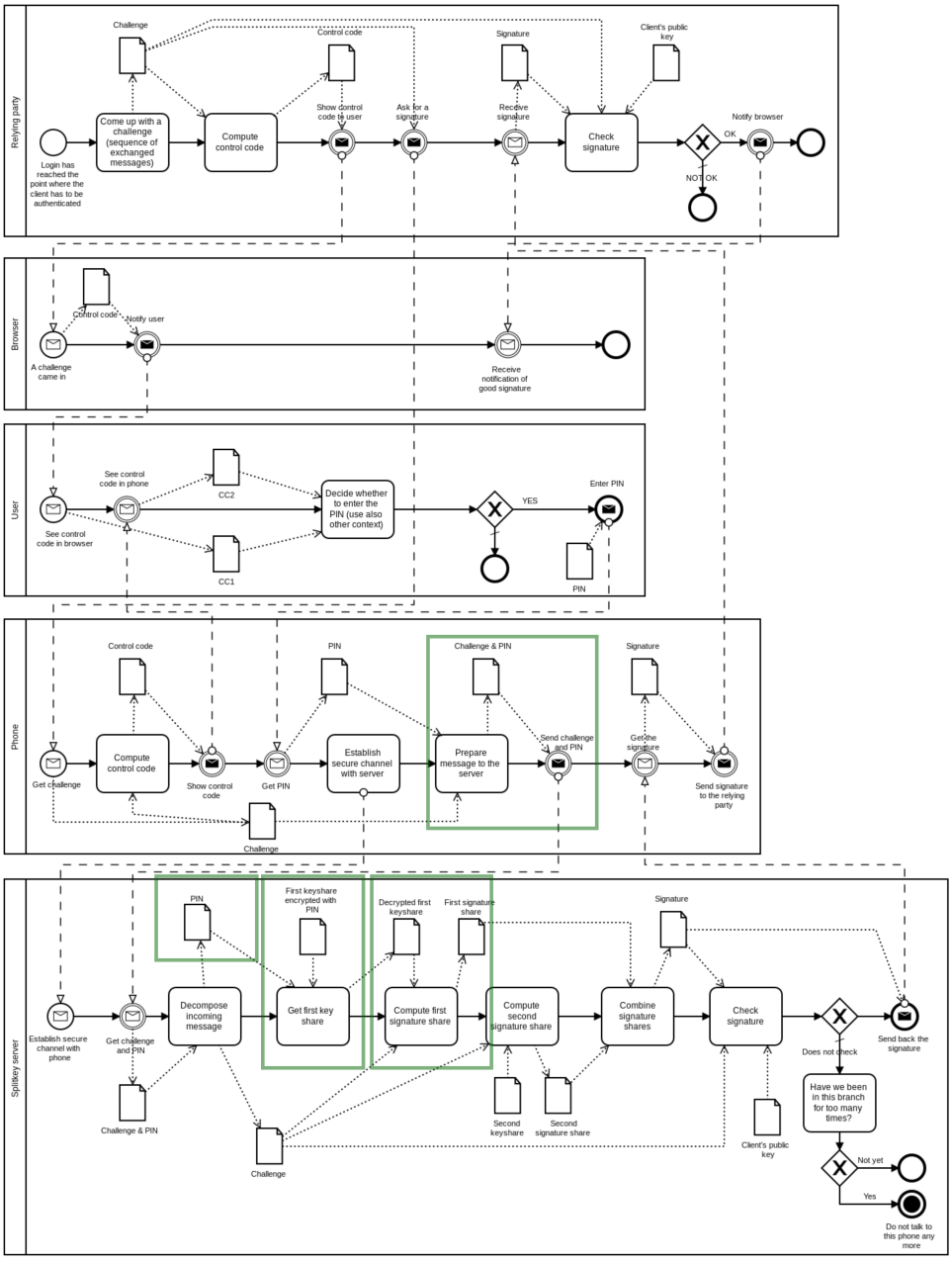


Figure 30: The first intermediate system. Highlighted in green boxes are main changes with the SplitKey system.

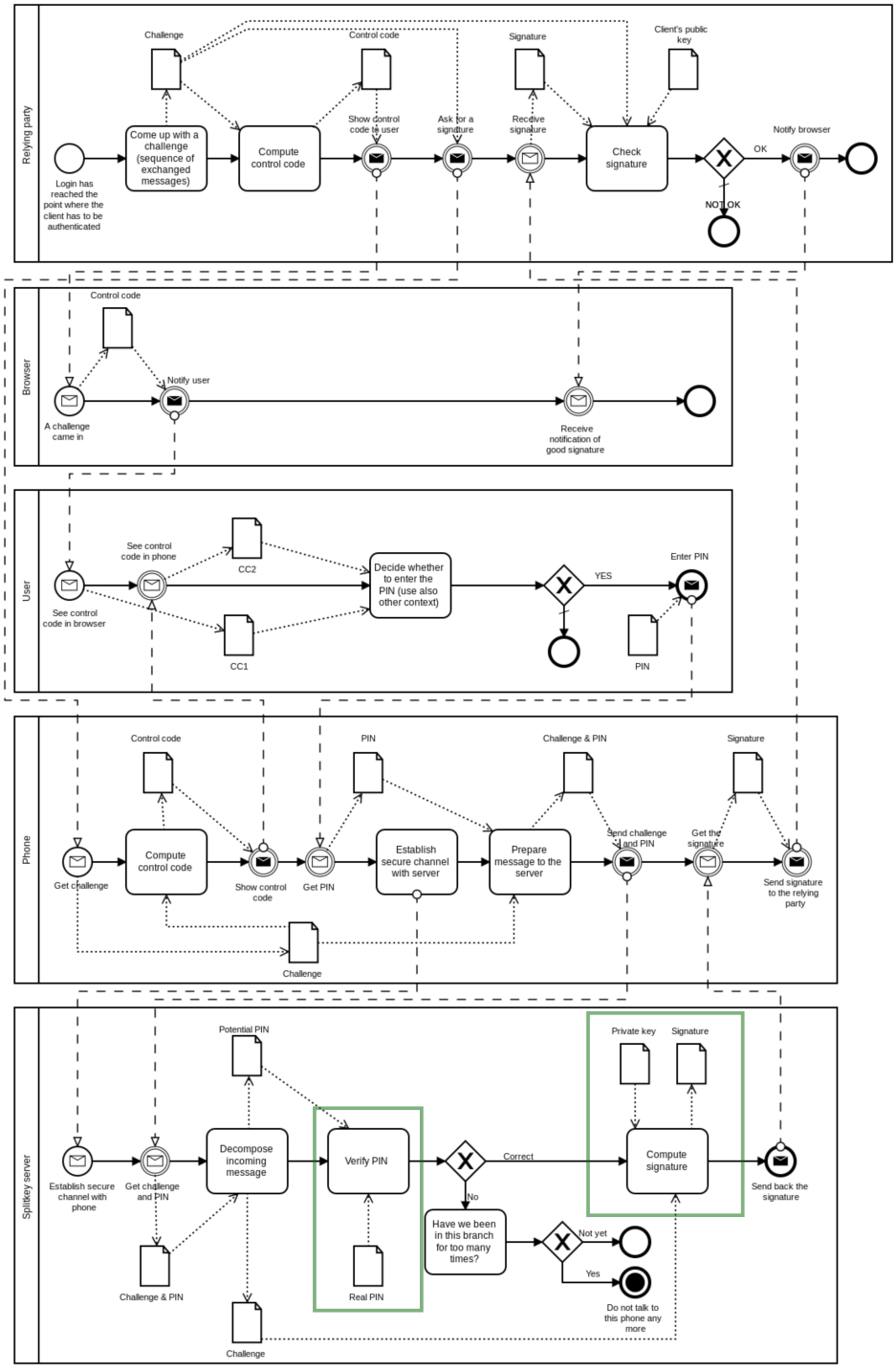


Figure 31: The second intermediate system. Highlighted in green boxes are main changes with the first intermediate system.

7.3 Comparison of private key protection

Another example application of the proposed comparison-based methodology is an analysis of private key protection mechanisms for digital identity wallets, specifically addressing compliance with the European Union's eIDAS2 regulation [72]. This analysis compares three approaches – remote Hardware Security Modules (HSM), threshold cryptography, and anonymous credentials with all-or-nothing non-transferability – against the established baseline of hardware-based protection using Secure Elements (SE) in mobile devices. This comparison highlights one of the main efforts of this thesis to demonstrate that novel systems relying on non-standard cryptographic components, such as threshold cryptography, can be argued to be at least as secure as certified standard systems. Two of the compared systems – remote HSM and hardware-based protection using Secure Elements (SE) are referenced in the EUDI Wallet ARF as possible options for implementation of the Wallet Secure Cryptographic Device (WSCD) [3].

7.3.1 Description of compared systems

We start by describing the main ideas behind the compared systems. In all four systems, the user generates a credential presentation using their mobile device.

Hardware-based credentials In this system, denoted as HW, the user's private key, which is used to create a verifiable credential presentation, is stored in an SE (Secure Element). For the credential presentation, the user needs to generate a signature. To access the signing functionality in the SE, the user needs to enter the correct knowledge factor (e.g., PIN/biometrics). The mobile device constructs the credential presentation from the verifiable credential and signature and sends it for verification to the Relying Party. The simplified presentation process for HW is presented in Figure 32.

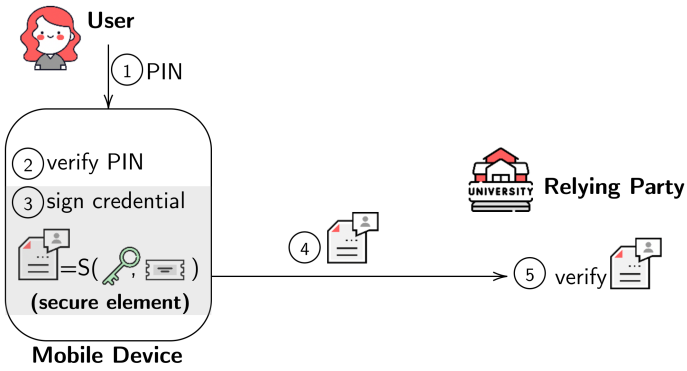


Figure 32: Hardware-based credential

Remote HSM The description of the remote HSM approach is based on the architecture document for the German National EUDI Wallet [2]. The main idea of this system, denoted as HSM, is that the user's signing private key is stored on a Remote HSM. The user needs to authenticate towards the Remote HSM using their knowledge factor. The knowledge factor is used to unlock the user's authentication key on the mobile device. Once the user is authenticated towards the Remote HSM, the user's mobile device submits a signing request to the Remote HSM server, which generates a signature for presentation. Finally,

the mobile device constructs the credential presentation and sends it for verification to the RP. The simplified presentation process for HSM is presented in Figure 33.

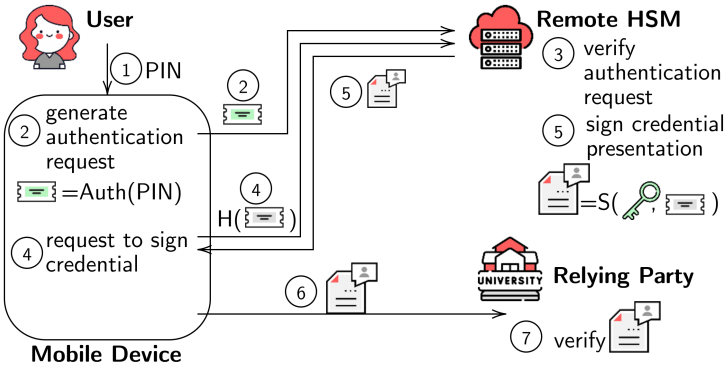


Figure 33: Remote HSM-based credential

Threshold cryptography In this system, denoted as SS, the user's private key is generated in a distributed way and the first share of the key is stored on the user's device, while the second share is stored on the supporting server. The presentation process with this approach is similar to HW, with the difference being in the way the user's signature is generated. Instead of generating a signature inside the SE, we assume the usage of a server-supported signature scheme (e.g., Buldas et al. [15]) to generate a signature. This signature can be verified as an ordinary non-threshold signature. In this system, the user's knowledge factor is used to authenticate the user towards the supporting server and unlock usage of the private key share stored on the mobile device. The simplified presentation process for SS is presented in Figure 34.

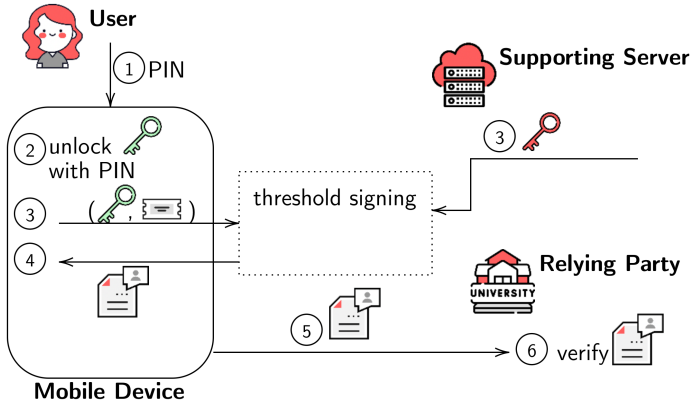


Figure 34: Threshold cryptography based credential

Anonymous credentials Anonymous credential systems enable the user to prove specific claims about the user's attributes to the RP while staying anonymous. The first prac-

tical and widely adopted construction of an anonymous credential system is the one proposed by Camenisch and Lysyanskaya [17]. Additionally to the main properties of anonymous systems, it discusses a mechanism of *all-or-nothing non-transferability*. The aim of this mechanism is to discourage the user from sharing their private key with other parties. This property can be achieved through connecting all the credentials with a single master secret key and publishing partial credentials. Later, if the user shares their master private key with a party (including an adversary), this party can look at published records of all the issuers and obtain information about all the user's credentials. For this analysis we use the system by Camenisch and Lysyanskaya [17] and assume that the master secret key is stored on the user's device, without additional protection mechanisms. The presentation process for this system differs from the previously described ones. In AN, the mobile device constructs zero-knowledge proofs, convincing the RP:

- of possessing a signature generated by an issuer on a credential,
- of knowing the master private key.

During the presentation, the user does *not* send the actual credential to the RP. The simplified presentation process for AN is presented in Figure 35.

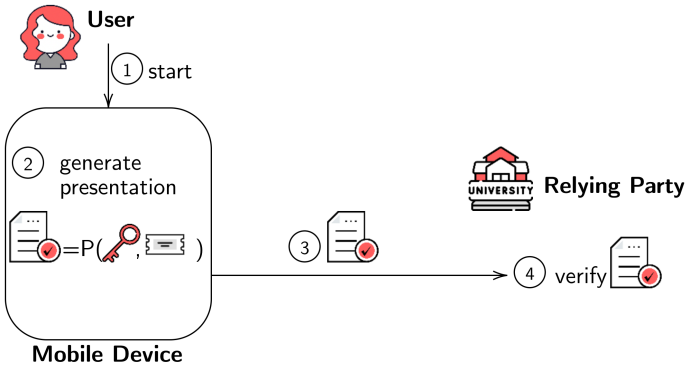


Figure 35: Anonymous credentials

7.3.2 Analysis

We have used the methodology described above to list weaknesses of the systems under comparison and identify relations between them. The analysis of the system weaknesses indicates that all four approaches can achieve comparable security when properly implemented. Hardware-based systems rely on SEs to store private keys securely, while remote HSM-based systems offload key storage and signing to a dedicated secure module. In threshold cryptography-based systems, the private key is split between the user's device and a supporting server, requiring the collaboration of both parties for the generation of a credential presentation. Anonymous credential systems enforce non-transferability by linking all the user credentials to a single master key, discouraging partial sharing.

The HSM offers stronger protection against guessing attacks on the user's PIN, as user authentication occurs on the Remote HSM. However, within the HSM architecture, gaining access to the Remote HSM is sufficient to perform signing operations, leaving the user without direct control. Additionally, a compromise of the Remote HSM component would

result in the exposure of signing keys for multiple system users. Consequently, safeguarding the Remote HSM against unauthorized access is critical for the security of HSM.

SS improves key protection over HW under the assumption that the second key share is stored in a server-side HSM. Moreover, it implements an additional property of clone detection, which helps to notice that the user's device memory has been cloned and to block this user. To gain access to the signing functionality in SS, the adversary is required to compromise both the user's mobile device and the supporting server. We note that the non-transferability property can be further added to the system via biometrics [56]. However, the underlying threshold signing protocol may result in compatibility issues with existing frameworks for verifiable credentials.

The solution in [15] offers only RSA signatures. Most of the attribute formats referenced in ARF, however, support ECDSA signatures [78, 37]. The work by Snetkov et al. [68] has demonstrated how to achieve an ECDSA-based protocol with similar properties. However, it may be harder to achieve compatibility with an HSM on the supporting server side with this scheme. Finally, blind signatures [69] can be employed to prevent the supporting server from inferring specific credential details.

For AN, the implementation of additional private key protection techniques against external adversaries is crucial. A compromise of the master secret key reveals all credentials and their content by the design of the scheme. However, this increases attack vectors. While key loss in AN is equivalent to losing the device and knowledge factor in HW, HSM, and SS, attacks are easier to execute in AN (without additional key protection mechanisms). Similarly to SS, the system relies on non-standard cryptographic protocols like circular encryption [17] to enable efficient zero-knowledge proofs. In this analysis, we do not focus on the cryptographic details of the protocols and additional nuanced properties they can provide. Table 17 presents the main weaknesses identified in the systems and proposes possible improvements to mitigate them.

Table 17: Main weaknesses of compared systems and proposed mitigation measures

System	The main weakness	Possible improvements
HW	All trust is placed on the user's device	Distribute storage of private key
HSM	All trust is placed on the Remote HSM	Distribute storage of private key between several remote servers
SS	Protection of supporting server's key share	Store server's share of the private key in HSM
	Leaking credential information to supporting server	Introduce blinding property towards supporting server
AN	Protection for the private key from external adversary	Distribute storage of private key
	User authentication	Introduce authentication using knowledge factor

7.4 Other applications

The introduced methodology has been applied to conduct a comparison of voting systems, in a study [76] that compares postal voting against internet voting using the Estonian voting system for analysis. The comparison and analysis revealed key differences between the two systems, concluding that the Estonian i-voting system offers stronger

security guarantees than the traditional paper-based postal voting system.

The analysis involved introducing three intermediate voting systems and comparing adjacent systems to identify, assess, and relate their respective weaknesses. This approach enabled a systematic evaluation of security properties, which showed that in the i-voting system, critical attacks targeting core security principles, such as vote secrecy, vote integrity, and ballot box integrity, are distributed across multiple system components. As a result, an adversary would need to compromise different parties and gather diverse information to succeed, making such attacks more difficult to execute. In contrast, the postal voting system involves fewer participants, and all of them are potential targets for critical attacks. This makes it significantly easier for an adversary to exploit weaknesses in the postal system. Consequently, the analysis demonstrated that the i-voting system provides a more resilient security model due to its distributed threat landscape, despite the complexity of its architecture.

7.5 Chapter summary

The final chapter introduces a structured methodology to improve the certification of novel cryptographic systems by comparing them against already certified systems. Using BPMN diagrams, the approach systematically identifies system weaknesses and maps them by comparing the difficulty and seriousness of potential exploits across certified and novel architectures. By employing intermediate system comparisons, the methodology demonstrates that advanced techniques like threshold cryptography can achieve security guarantees equivalent to or exceeding traditional hardware-based solutions. This framework facilitates regulatory compliance (e.g., eIDAS2, QSCD) by providing a formal basis to argue that novel systems maintain at least equivalent security levels without requiring full re-certification from scratch.

8 Conclusions

This work presents a centralised identity management system, which offers privacy against the central server. The work starts with defining security and functional requirements for such an identity management system as well as proposing the initial design of the system. The initial design relies on the usage of Searchable Symmetric Encryption (SSE) or threshold decryption to hide the values of the user's attributes from untrusted storage. Specifically, the Cloud Privacy-Preserving Attribute Management System (CACS) was proposed, which allows credentials to be stored in the cloud while maintaining privacy against the storage provider. However, the analysis of these proposed solutions revealed a critical limitation: in both initial systems, the centralised Attribute Management Service server inevitably learns the values of the user's attributes during the presentation phase to generate the mdoc document required for the relying party. This leakage contradicts the goal of preserving user privacy against the central server itself, raising the further research question whether it is possible to modify the message flow such that the user is the recipient of the plaintext attribute, not the AMS server.

Even in the case of the modified message flow, the work assumes that the AMS server should be assisting the user in the decryption process. This raises the concern of the AMS server learning which exactly ciphertext is being decrypted, violating the user's privacy. The work proceeds with defining a missing cryptographic component – blind server-supported decryption. The scheme is introduced along with the security definition in the form of an ideal functionality and the proof that the proposed scheme realises this functionality in the UC framework under the one-more CDH assumption.

Further, the work defines security for the centralised identity management system in the form of an ideal functionality and proves that the defined functionality captures the main security requirements defined for the identity management system. Next, the work proposes an improved centralised identity management system that relies on privacy-preserving cryptographic building blocks, such as server-supported blind decryption and server-supported blind signing. The work further proves that the proposed system securely implements the defined functionality.

Since reliance on cryptographic standards and the usage of agreed cryptographic mechanisms is an important requirement for such systems, this work proposes a methodology that allows comparing systems relying on non-standard cryptographic components to the already standardised/certified ones. Using this methodology, the thesis also presents a comparative analysis of private key protection mechanisms for digital identity wallets, evaluating hardware-based credentials with Secure Elements, remote HSMs, threshold cryptography, and anonymous credentials. The results indicate that while all four approaches can achieve comparable security when properly implemented, they each present distinct vulnerabilities requiring targeted mitigations. Hardware-based and remote HSM systems centralize trust on a single component, making distributed key storage a recommended upgrade. Threshold cryptography improves upon baseline hardware protection by splitting keys, though it necessitates securing the server-side share in an HSM. Anonymous credentials offer user privacy but suffer from master key compromise risks, requiring distributed storage and robust knowledge-factor authentication to mitigate external threats. The analysis demonstrates that novel systems relying on threshold cryptography can be formally argued to provide security guarantees at least equivalent to standard systems, particularly when implemented with secure server-side key management.

One of the main challenges encountered throughout the development of the thesis was the formal definition of security for both cryptographic components and the identity management system. Previous definitions for blind decryption based on game-based

models failed to fully capture the nuances of privacy and correctness in a distributed threshold setting. We found that the Universal Composability (UC) framework provided the necessary rigour to balance these properties, allowing us to define an ideal functionality that prevents the server from learning which ciphertext is being decrypted while ensuring a corrupted client cannot decrypt more messages than authorized. Another significant challenge was the integration of these novel cryptographic primitives into systems that must adhere to existing regulatory standards, such as eIDAS2 and mobile Driver's Licence (mDL) formats. The trade-off between enhanced security guarantees (e.g., threshold cryptography) and certification feasibility was addressed through the proposed comparison-based methodology, which allows novel systems to be argued as at least as secure as certified counterparts without requiring a full re-certification from scratch.

There are several directions for future work that stem from the work presented in this thesis. First, the issue of key recovery for the user share in threshold decryption needs further investigation to ensure usability without compromising security, potentially by encrypting the key share on a secondary device. Second, the efficiency of the encryption schemes can be improved by replacing the Paillier encryption component with more lightweight alternatives, which would reduce ciphertext sizes and computational overhead. Additionally, future work may include the development of post-quantum analogues of the proposed constructions. While the current constructions presented in this thesis rely on classical cryptographic primitives based on the hardness of the discrete logarithm problem and the factorisation problem, there is a need to switch to quantum-resistant alternatives. The work done in the area of quantum-safe protocols suggests that the foundational building blocks for PQC-based identity management systems are available. However, significant work remains to be done to adapt the specific privacy-preserving protocols proposed and used in this thesis to the post-quantum setting. Finally, the comparison-based methodology developed for security assurance has shown potential applications beyond identity management, such as comparing internet voting against postal voting, suggesting that this approach could help facilitate the certification of other novel cryptographic systems in critical infrastructure. By continuing to refine these cryptographic components and methodologies, we aim to contribute to the implementation of secure, privacy-preserving digital identity wallets that are compliant with evolving legislative frameworks like the eIDAS2 regulation.

List of Figures

1 Main parties of the identity management system 10
2 Main parties of the identity management system developed in this work ... 12
3 Security game overview 20
4 $\Sigma-$ protocol message flow 26
5 Universal Composability framework idea 29
6 Issuing of attributes in plain model 33
7 Presentation in plain model 34
8 ElGamal KEM 39
9 NIZK proof that $\log_g u = \log_h v$ 39
10 Proof of knowledge of exponent in a group $\mathbb{G}$ 39
11 Proving the validity of a DVNIZK proof. Notation $\boxdot$ and $\boxplus$ means scalar multiplication and addition of coins. 40
12 Ideal encryption functionality $\mathcal{F}_{\text{tdecrypt}}$ 41
13 Ideal encryption functionality $\mathcal{F}_{\text{tdecrypt}}$ (continued). 42
14 Ideal encryption functionality $\mathcal{F}_{\text{tdecrypt}}$ (continued). 43
15 Key generation for client and server privacy-preserving threshold decryption 45
16 Encryption in our construction 46
17 Decryption in our construction 46
18 Ideal functionality $\mathcal{F}_{\text{ideal}}$ (Part 1) 55
19 Ideal functionality $\mathcal{F}_{\text{ideal}}$ (Part 2) 56
20 Ideal functionality $\mathcal{F}_{\text{ideal}}$ (Part 3) 57
21 Prettiness initialisation phase 59
22 Prettiness issuing phase 61
23 Prettiness get all credentials 61
24 Prettiness revocation by the user 62
25 Prettiness revocation by the issuer 62
26 Prettiness presentation phase 63
27 Prettiness presentation phase 64
28 The smart card system 72
29 The SplitKey system 73
30 The first intermediate system. Highlighted in green boxes are main changes with the SplitKey system. 74
31 The second intermediate system. Highlighted in green boxes are main changes with the first intermediate system. 75
32 Hardware-based credential 76
33 Remote HSM-based credential 77
34 Threshold cryptography based credential 77
35 Anonymous credentials 78
36 Initialisation protocol (User, ST) 262
37 Issuing protocol (User, ST) 263
38 Issuing protocol (Issuer) 264
39 Protocol for downloading credential backup from the server 265
40 Revocation protocol initiated by user 265
41 Revocation protocol initiated by issuer 266
42 Presentation with selective disclosure 267
43 Verification of credential presentation – RP 268
44 Verification of credential presentation – User and AMS 269

List of Tables

1 BPMN notation used in this work 31
2 Notation for attribute management system 33
3 User data disclosed to different parties. Highlighted in grey disclosures (yes) are intentional and needed for functionality. Since AMS is sending attributes to RP, we assume it knows plaintext values of attributes during presentation. Additionally, since all the interactions with attributes in the system happen through aID, we intentionally disclose it. Highlighted in red disclosures (yes) are the ones whose leakage we would like to avoid.... 36
4 Notation and sub-routines used in blind server supported decryption construction 38
5 Usage of random oracles in protocol 48
6 Values the simulator can extract from different proof types 48
7 Performance of blind threshold decryption for 32-byte message. Notice: * without network delay 50
8 Additional notation for attribute management system 51
9 User data disclosed to corrupted parties, where cID and $\overrightarrow{\text{aID}}$ are credential and attribute identifiers, $\overrightarrow{\text{atr}}$ are attribute values, $\overrightarrow{\text{aID}}_{\text{dscl}}$ are the attributes used in a presentation, and $\overrightarrow{\text{atr}}_{\text{dscl}}$ is the disclosure of attribute values coming from that presentation. Disclosures marked by * are possible only for the given coalition of corrupted parties, and are *not* derived from smaller coalitions. Highlighted disclosures (yes) are intentional and needed for functionality. 58
10 Logged records and their interpretation. 64
11 Corruption model of sub-protocols 65
12 Efficiency of $\mathcal{F}_{\text{tsign}}$ realised by [69]. Notice: * without network delay. $|m|_b$ is the message length in bits. 67
13 Benchmarks (runtime) for different routines of revocation protocol. Notation "–" represent numbers that are the same as for $m = 0$. 67
14 Benchmarks (bandwidth) for different routines of revocation protocol. Notation "–" represent numbers that are the same as for $m = 0$. 68
15 Summarised efficiency of the full system for $N = 100$ credentials per user, $n = 10$ attributes per credential, $m = 10^6$ revoked tokens. 68
16 Main weakness types 70
17 Main weaknesses of compared systems and proposed mitigation measures 79

References

[1] Regulation (eu) 2024/1183 of the european parliament and of the council of 11 april 2024 amending regulation (eu) no 910/2014 as regards establishing the european digital identity framework, 2024.

[2] German National EUDI Wallet: Architecture Documentation v0.4.0, 2025. Accessed: 26.02.2026.

[3] The European Digital Identity Wallet Architecture and Reference Framework V2.4.0 , 2025.

[4] M. Abe, J. Groth, K. Haralambiev, and M. Ohkubo. Optimal structure-preserving signatures in asymmetric bilinear groups. In P. Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pages 649–666, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[5] A. Abid, S. Cheikhrouhou, S. Kallel, and M. Jmaiel. NovidChain: Blockchain-based privacy-preserving platform for COVID-19 test/vaccine certificates. *Softw Pract Exp.*, 52(4):841–867, 2022.

[6] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology*, 23(2):281–343, 2010.

[7] A. Barki, S. Brunet, N. Desmoulins, and J. Traoré. Improved algebraic macs and practical keyed-verification anonymous credentials. In R. Avanzi and H. M. Heys, editors, *Selected Areas in Cryptography - SAC 2016 - 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers*, volume 10532 of *Lecture Notes in Computer Science*, pages 360–380. Springer, 2016.

[8] B. Bauer, G. Fuchsbauer, and J. Loss. A classification of computational assumptions in the algebraic group model. In D. Micciancio and T. Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, pages 121–151, Cham, 2020. Springer International Publishing.

[9] C. Baum, O. Blazy, J. Camenisch, J.-H. Hoepman, E. Lee, A. Lehmann, A. Lysyanskaya, R. Mayrhofer, H. Montgomery, N. K. Nguyen, et al. Cryptographers' feedback on the eu digital identity's arf. *Tech. Rep.*, 2024.

[10] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, page 62–73, New York, NY, USA, 1993. Association for Computing Machinery.

[11] M. Bellare and P. Rogaway. Code-based game-playing proofs and the security of triple encryption. *IACR Cryptol. ePrint Arch.*, page 331, 2004.

[12] M. Blaze, G. Bleumer, and M. Strauss. Divertible protocols and atomic proxy cryptography. In K. Nyberg, editor, *Advances in Cryptology — EUROCRYPT'98*, pages 127–144, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[13] O. Blazy, L. Brouilhet, C. Chevalier, P. Towa, I. Tucker, and D. Vergnaud. Hardware security without secure hardware: How to decrypt with a password and a server. *Theor. Comput. Sci.*, 895:178–211, 2021.

[14] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications (extended abstract). In J. Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 103–112. ACM, 1988.

[15] A. Buldas, A. Kalu, P. Laud, and M. Oruaas. Server-Supported RSA Signatures for Mobile Devices. In S. N. Foley, D. Gollmann, and E. Snekkenes, editors, *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I*, volume 10492 of *Lecture Notes in Computer Science*, pages 315–333. Springer, 2017.

[16] A. Buldas and M. Saarepera. Electronic Signature System with Small Number of Private Keys. In C. M. Ellison, W. T. Polk, N. E. Hastings, and S. W. Smith, editors, *NISTIR 7085: 2nd Annual PKI Research Workshop Proceedings*, pages 110–122. National Institute of Standards and Technology (NIST), 2004.

[17] J. Camenisch and E. V. Herreweghen. Design and implementation of the *idemix* anonymous credential system. In V. Atluri, editor, *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 21–30. ACM, 2002.

[18] J. Camenisch and A. Lysyanskaya. Signature Schemes and Anonymous Credentials from Bilinear Maps. In M. K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer, 2004.

[19] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.

[20] R. Canetti, H. Krawczyk, and J. Nielsen. Relaxing chosen-ciphertext security. Cryptology ePrint Archive, Paper 2003/174, 2003. `https://eprint.iacr.org/2003/174`.

[21] D. Chaum. Blind signatures for untraceable payments. In D. Chaum, R. L. Rivest, and A. T. Sherman, editors, *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982*, pages 199–203. Plenum Press, New York, 1982.

[22] Cybernetica AS. Introduction to splitkey foundations. White Paper Version 1.0.1, Cybernetica AS, June 2020. Accessed 2026-03-10; technical overview of SplitKey cryptographic key splitting technology.

[23] I. Damgård, N. Fazio, and A. Nicolosi. Non-interactive zero-knowledge from homomorphic encryption. In S. Halevi and T. Rabin, editors, *Theory of Cryptography*, pages 41–59, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[24] S. Dupont, G. Ginis, M. Malacario, C. Porretti, N. Maunero, C. Ponsard, and P. Massonet. Incremental Common Criteria Certification Processes using DevSecOps Practices. In *IEEE European Symposium on Security and Privacy Workshops, EuroS&P 2021, Vienna, Austria, September 6-10, 2021*, pages 12–23. IEEE, 2021.

[25] D. Fett, K. Yasuda, and B. Campbell. Selective Disclosure for JWTs (SD-JWT). Internet-Draft draft-ietf-oauth-selective-disclosure-jwt-15, Internet Engineering Task Force, Jan. 2025. Work in Progress.

[26] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, Lecture Notes in Computer Science, pages 186–194. Springer, 1986.

[27] E. U. A. for Cybersecurity. Agreed cryptographic mechanisms, 2025.

[28] P. Fouque and D. Pointcheval. Threshold cryptosystems secure against chosen-ciphertext attacks. In C. Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 351–368. Springer, 2001.

[29] G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 33–62. Springer, 2018.

[30] G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 33–62, Cham, 2018. Springer International Publishing.

[31] T. E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory*, 31(4):469–472, 1985.

[32] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[33] M. Green. Secure blind decryption. In D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, editors, *Public Key Cryptography – PKC 2011*, pages 265–282, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[34] Z. Gui, K. G. Paterson, S. Patranabis, and B. Warinschi. Swissse: System-wide security for searchable symmetric encryption. *Proc. Priv. Enhancing Technol.*, 2024(1):549–581, 2024.

[35] U. Haböck and S. Krenn. Breaking and fixing anonymous credentials for the cloud. In Y. Mu, R. H. Deng, and X. Huang, editors, *Cryptology and Network Security*, pages 249–269, Cham, 2019. Springer International Publishing.

[36] H. Halpin. Nym credentials: Privacy-preserving decentralized identity with blockchains. In *2020 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 56–67. IEEE, 2020.

[37] ISO Central Secretary. Personal identification — ISO-compliant driving licence — Part 5: Mobile driving licence (mDL) application. Standard ISO/IEC 18013-5:2021, International Organization for Standardization, Geneva, CH, 2021.

[38] A. Kalu, B. C. Kus, P. Laud, K. L. Leung, N. Snetkov, and J. Vakarjuk. CACS: A cloud privacy-preserving attribute management system. In *Proceedings of the 18th International Conference on Availability, Reliability and Security, ARES 2023, Benevento, Italy, 29 August 2023- 1 September 2023*, pages 122:1–122:9. ACM, 2023.

[39] F. Karegar, D. Lindegren, J. S. Pettersson, and S. Fischer-Hübner. Assessments of a cloud-based data wallet for personal identity management. In *26th International Conference on Information Systems Development (ISD2017 Cyprus). Larnaca, Cyprus, September 6-8, 2017*, 2017.

[40] F. Karegar, C. Striecks, S. Krenn, F. Hörandner, T. Lorünser, and S. Fischer-Hübner. Opportunities and challenges of CREDENTIAL - towards a metadata-privacy respecting identity provider. In A. Lehmann, D. Whitehouse, S. Fischer-Hübner, L. Fritsch, and C. D. Raab, editors, *Privacy and Identity Management. Facing up to Next Steps - 11th IFIP WG 9.2, 9.5, 9.6/11.7, 11.4, 11.6/SIG 9.2.2 International Summer School, Karlstad, Sweden, August 21-26, 2016, Revised Selected Papers*, IFIP Advances in Information and Communication Technology, pages 76–91, 2016.

[41] J. Kilian and P. Rogaway. How to protect DES against exhaustive key search (an analysis of DESX). *J. Cryptol.*, 14(1):17–35, 2001.

[42] J. M. Kirss, P. Laud, N. Snetkov, and J. Vakarjuk. Server-supported decryption for mobile devices. In G. Lenzini and W. Meng, editors, *Security and Trust Management - 18th International Workshop, STM 2022, Copenhagen, Denmark, September 29, 2022, Proceedings*, volume 13867 of *Lecture Notes in Computer Science*, pages 71–81. Springer, 2022.

[43] A. Kostopoulos, E. Sfakianakis, I. Chochliouros, J. S. Pettersson, S. Krenn, W. Tesfay, A. Migliavacca, and F. Hörandner. Towards the adoption of secure cloud identity services. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, ARES '17, New York, NY, USA, 2017. Association for Computing Machinery.

[44] S. Krenn, T. Lorünser, A. Salzer, and C. Striecks. Towards attribute-based credentials in the cloud. In S. Capkun and S. S. M. Chow, editors, *Cryptology and Network Security*, pages 179–202, Cham, 2018. Springer International Publishing.

[45] K. Krips, N. Snetkov, J. Vakarjuk, and J. Willemson. Trust assumptions in voting systems. In S. K. Katsikas, H. Abie, S. Ranise, L. Verderame, E. Cambiaso, R. M. Ugarelli, I. Praça, W. Li, W. Meng, S. Furnell, B. Katt, S. Pirbhulal, A. Shukla, M. Ianni, M. D. Preda, K. R. Choo, M. P. Correia, A. Abhishta, G. Sileno, M. Alishahi, H. K. Kalutarage, and N. Yanai, editors, *Computer Security. ESORICS 2023 International Workshops - CPS4CIP, ADIoT, SecAssure, WASP, TAURIN, PriST-AI, and SECAI, The Hague, The Netherlands, September 25-29, 2023, Revised Selected Papers, Part II*, volume 14399 of *Lecture Notes in Computer Science*, pages 309–329. Springer, 2023.

[46] P. Laud, A. Pankova, and J. Vakarjuk. Privacy-preserving server-supported decryption. *CoRR*, abs/2410.19338, 2024.

[47] P. Laud, A. Pankova, and J. Vakarjuk. Privacy-preserving server-supported decryption . In *2025 IEEE 38th Computer Security Foundations Symposium (CSF)*, pages 48–63, Los Alamitos, CA, USA, June 2025. IEEE Computer Society.

[48] P. Laud and J. Vakarjuk. A comparison-based methodology for the security assurance of novel systems. In S. K. Katsikas, F. Cuppens, C. Kalloniatis, J. Mylopoulos, F. Pallas, J. Pohle, M. A. Sasse, H. Abie, S. Ranise, L. Verderame, E. Cambiaso, J. M. Vidal, M. A. S. Monge, M. Albanese, B. Katt, S. Pirbhulal, and A. Shukla, editors, *Computer Security. ESORICS 2022 International Workshops - CyberICPS 2022, SECPRE 2022, SPOSE 2022, CPS4CIP 2022, CDT&SECOMANE 2022, EIS 2022, and SecAssure 2022, Copenhagen, Denmark, September 26-30, 2022, Revised Selected Papers*, volume 13785 of *Lecture Notes in Computer Science*, pages 625–644. Springer, 2022.

[49] J. Lee, J. Choi, H. Oh, and J. Kim. Privacy-preserving identity management system. *IACR Cryptol. ePrint Arch.*, page 1459, 2021.

[50] R. T. Moreno, J. García-Rodríguez, J. B. Bernabé, and A. Skarmeta. A trusted approach for decentralised and privacy-preserving identity management. *IEEE Access*, 9:105788–105804, 2021.

[51] R. T. Moreno, J. G. Rodríguez, C. T. López, J. B. Bernabé, and A. F. Skarmeta. OLYMPUS: A distributed privacy-preserving identity management system. In *2020 Global Internet of Things Summit, GIoTS 2020, Dublin, Ireland, June 3, 2020*, pages 1–6. IEEE, 2020.

[52] National Computer Security Center. Rating Maintenance Phase Program Document Version 2. Rainbow Series, NCSC-TG-013 V2, 1995.

[53] Object Management Group. Business process model and notation (bpmn), version 2.0.2. Specification formal/2013-12-09, Object Management Group (OMG), Dec. 2013. Accessed 2026-03-10; official standard document.

[54] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[55] A. Pankova and J. Vakarjuk. Credential revocation assisted by a covertly corrupted server. Cryptology ePrint Archive, Paper 2025/1854, 2025.

[56] S. Pape. A Survey on Non-transferable Anonymous Credentials. In V. Matyás, S. Fischer-Hübner, D. Cvrcek, and P. Svenda, editors, *The Future of Identity in the Information Society - 4th IFIP WG 9.2, 9.6/11.6, 11.7/FIDIS International Summer School, Brno, Czech Republic, September 1-7, 2008, Revised Selected Papers*, volume 298 of *IFIP Advances in Information and Communication Technology*, pages 107–118. Springer, 2008.

[57] C. Paquin and G. Zaverucha. U-prove cryptographic specification v1.1 (revision 3), December 2013. Released under the Open Specification Promise (http://www.microsoft.com/openspecifications/en/us/programs/osp/default.aspx).

[58] Reuse of Evaluation Results and Evidence, Oct 26th 2002. Information Statement on behalf of the Common Criteria Recognition Arrangement Management Committee, Document no. 2002-08-009-002.

[59] M. Richter, M. Bertram, J. Seidensticker, and M. Margraf. Cryptographic requirements of verifiable credentials for digital identification documents. In *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1663–1668. IEEE, 2023.

[60] S. Ringers, E. R. Verheul, and J. Hoepman. An Efficient Self-blindable Attribute-Based Credential Scheme. In A. Kiayias, editor, *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers*, volume 10322 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2017.

[61] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[62] A. D. Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securely. In F. T. Leighton and M. T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 522–533. ACM, 1994.

[63] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *J. Cryptol.*, 15(2):75–96, 2002.

[64] A. D. Sinnhofer, W. Raschke, C. Steger, and C. Kreiner. Evaluation paradigm selection according to Common Criteria for an incremental product development. In S. Tverdyshev, editor, *International Workshop on MILS: Architecture and Assurance for Secure Systems, MILS@HiPEAC 2015, Amsterdam, The Netherlands, January 20, 2015*. Zenodo, 2015.

[65] A. D. Sinnhofer, W. Raschke, C. Steger, and C. Kreiner. Patterns for Common Criteria Certification. In C. Link and V. Eloranta, editors, *Proceedings of the 20th European Conference on Pattern Languages of Programs, EuroPLoP 2015, Kaufbeuren, Germany, July 8-12, 2015*, pages 33:1–33:15. ACM, 2015.

[66] N. Snetkov, J. Vakarjuk, and P. Laud. TOPCOAT: towards practical two-party crystals-dilithium. *Discov. Comput.*, 27(1):18, 2024.

[67] N. Snetkov, J. Vakarjuk, and P. Laud. Electrum: UC fail-stop server-supported signatures. In G. Yang, S. Liu, C. Su, A. Otsuka, and Z. Lian, editors, *Provable and Practical Security - 19th International Conference, ProvSec 2025, Yokohama, Japan, October 10-12, 2025, Proceedings*, volume 16172 of *Lecture Notes in Computer Science*, pages 88–107. Springer, 2025.

[68] N. Snetkov, J. Vakarjuk, and P. Laud. Universally composable server-supported signatures for smartphones. *IACR Commun. Cryptol.*, 3(1):13, 2026.

[69] N. Snetkov, J. Vakarjuk, and A. Pankova. Coppercloud: Blind server-supported RSA signatures. In S. D. C. di Vimercati and P. Samarati, editors, *Proceedings of the 23rd International Conference on Security and Cryptography, SECRYPT 2026, Porto, Portugal, July 16-18, 2026, Volume 1*, pages 419–428. SCITEPRESS, 2026.

[70] D. X. Song, D. A. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000*, pages 44–55. IEEE Computer Society, 2000.

[71] J. Vakarjuk and A. Pankova. Prettiness – privacy preserving attribute management system, 2026.

[72] J. Vakarjuk, A. Pankova, and A. Kalu. Comparison of private key protection techniques for digital identity wallets. In *2026 IEEE European Symposium on Security and Privacy Workshops*, pages 396–412, 2026.

[73] J. Vakarjuk, N. Snetkov, and P. Laud. Identifying obstacles of PQC migration in e-estonia. In C. Kwan, L. Lindström, D. Giovannelli, K. Podins, and D. Strucl, editors, *16th International Conference on Cyber Conflict: Over the Horizon, CyCon 2024, Tallinn, Estonia, May 28-31, 2024*, pages 63–81. IEEE, 2024.

[74] J. Vakarjuk, N. Snetkov, and J. Willemson. Dilizium: A two-party lattice-based signature scheme. *Entropy*, 23(8):989, 2021.

[75] J. Vakarjuk, N. Snetkov, and J. Willemson. Russian federal remote e-voting scheme of 2021 - protocol description and analysis. In D. Megías, R. D. Pietro, and J. García-Alfaro, editors, *EICC 2022: European Interdisciplinary Cybersecurity Conference, Barcelona, Spain, June 15 - 16, 2022*, pages 29–35. ACM, 2022.

[76] J. Vakarjuk, N. Snetkov, and J. Willemson. Comparing security levels of postal and internet voting. *Information Security Journal: A Global Perspective*, 0(0):1–21, 2024.

[77] F. Veseli, J. S. Olvera, T. Pulls, and K. Rannenberg. Engineering privacy by design: Lessons from the design and implementation of an identity wallet platform. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19, page 1475–1483, New York, NY, USA, 2019. Association for Computing Machinery.

[78] W3C Verifiable Credentials Working Group. Verifiable credentials data model v2.0, May 2025. W3C Recommendation, accessed 2026-03-09.

[79] A. C. Yao. Theory and application of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 80–91, 1982.

Acknowledgements

This research was conducted in cooperation with Cybernetica as part of my industrial PhD studies and has been supported by the Estonian Business and Innovation Agency and the Estonian Research Council under the grant number PRG1780.

Completing this PhD has been a long and complicated journey, especially on an emotional level. There were many moments of uncertainty, exhaustion, and self-doubt, but also moments of discovery, growth, and genuine excitement. Through all these challenges, I grew immensely as a cryptography researcher, learning not only how to approach difficult problems, but also how to persevere when the answers were far from clear. The journey demanded countless late nights, so many that, at times, I found myself seeing UC proofs even in my sleep. Despite the difficulties, I am deeply grateful for how much this experience has shaped me, both as a researcher and as a person.

I am sincerely grateful to my supervisor, Ahto Buldas, who introduced me to the fascinating world of cryptography and helped me begin this academic journey. I would also like to express my sincere gratitude to my second supervisor, Alisa Pankova, whose support was invaluable throughout my PhD journey. I am especially thankful for her careful guidance in writing and improving my research papers, as well as for the countless brainstorming sessions and inspiring exchanges of ideas. Her patience, encouragement, and willingness to discuss the challenging questions over and over again helped me develop as a researcher.

I would also like to thank my colleagues at Cybernetica for their support, valuable advice, and continuous encouragement. I am grateful for the inspiring discussions, shared knowledge, and motivation they gave me to keep moving forward.

I would like to express my deepest gratitude to my family for their love, patience, and support throughout my doctoral studies. Their encouragement gave me strength during the most difficult moments and helped me continue when the journey felt overwhelming. Above all, I am profoundly grateful to my husband. I could not have completed this PhD without his constant support, belief in me, and encouragement. During the moments when I doubted myself or felt ready to stop, he never let me to give up. He reminded me of how far I had come, encouraged me to keep moving forward, and stood beside me through every challenge. His patience, strength, and unconditional faith in me made this achievement possible, and I will always be grateful to have shared this journey with him.

To everyone who supported me along the way, thank you for believing in me and for being part of this journey. This thesis would not have been possible without you.

Abstract
Applying Privacy-Preserving Computations in Digital Identity

The increasing reliance on digital interactions has made the management of digital identities a critical task of modern society. A digital identity is composed of a collection of claims, or attributes, that serve to represent a person in the digital space. The massive amount of interactions involving digital identities raises significant privacy concerns regarding data leakage to issuers, relying parties, and storage providers. This work addresses the challenge of designing privacy-preserving identity management systems that satisfy both regulatory requirements, such as the eIDAS2 regulation and the European Digital Identity Wallet (EUDIW) framework, and well-established security requirements. The work focuses on developing a centralised system that utilizes advanced cryptographic primitives to protect user privacy while maintaining standards compliance and usability.

The primary motivation for this research comes from the limitations of existing identity management solutions. Many systems propose to store attributes locally on user devices, requiring users to manage cryptographic keys and credentials, which raises security concerns. Conversely, centralised Attribute Management Systems (AMS) offer better convenience and backup capabilities but introduce trust assumptions regarding the centralised server's ability to access user data. To bridge this gap, the thesis proposes several novel architectures and cryptographic schemes. The first contribution is CACS (Cloud Privacy-Preserving Attribute Management System), which allows users to store credentials in the cloud while preserving privacy against untrusted storage providers. CACS employs searchable symmetric encryption (SSE) and threshold decryption to ensure that the cloud provider learns neither the content of queries nor the attributes themselves.

Since threshold decryption alone still leaks information about which ciphertexts are being decrypted to the supporting server, the thesis further develops a privacy-preserving server-supported decryption scheme. This scheme enables a client to request decryption assistance from a server without revealing which ciphertext is being decrypted. The security of this protocol is formally defined in the Universal Composability (UC) framework and proven to be secure under a variation of the computational Diffie-Hellman problem. This construction is built from ElGamal key encapsulation, additionally using non-interactive zero-knowledge proofs, and designated-verifier zero-knowledge proofs. The work further proposes an improved attribute management system – PRETTINESS. This system enhances the privacy guarantees of CACS by reducing information leakage to the AMS server and utilising a revocation protocol that allows both users and issuers to revoke credentials without leaking information to the AMS server. This work defines an ideal functionality to model security requirements for a centralised identity management system and proves that the proposed system UC-realizes the defined functionality.

Furthermore, the thesis introduces a comparison-based methodology for the security assurance of novel systems. This methodology allows new cryptographic systems, such as those using threshold cryptography, to be compared against already certified standard systems (e.g., smartcard-based authentication) to ease the certification process. By modelling systems using Business Process Modelling Notation (BPMN) and analysing weaknesses, the methodology establishes that a novel system is at least as secure as the certified counterpart. As a concrete application of the proposed comparison-based security assurance methodology, this work formally analysed and compared private key protection methods, contrasting threshold cryptography and anonymous credentials with traditional hardware-based and remote HSM deployments. The analysis confirms that conventional systems are offering more straightforward certification paths but place trust in a single physical component.

Threshold cryptography, by contrast, distributes trust across the user device and server, enabling the implementation of additional properties like clone detection and resilience against device memory extraction. However, those systems introduce the need for server-side key management in HSMs. Potential improvements are also requiring blinding properties to prevent information leakage to the supporting server. Anonymous credential systems, while providing strong non-transferability, concentrate recovery risks in the master secret, requiring distributed storage and strict authentication to remain viable. The results of this analysis reinforce the broader conclusion that combining advanced privacy-preserving cryptography with formal security modelling enables the deployment of scalable, regulation-compliant identity wallets while maintaining interoperability.

The cryptographic schemes proposed in this work do not rely on heavy cryptography, making them practical for deployment on mobile devices and in resource-constrained environments. Performance evaluations indicate that the overhead introduced by privacy-preserving technologies, remains within acceptable limits for real-world applications. Additionally, the comparison methodology was successfully applied to authentication systems, demonstrating that a threshold cryptography-based system can be argued to be at least as secure as a smartcard-based system.

In conclusion, this work presents a methodological and technical framework for the development of digital identity management systems with privacy-preserving properties. It addresses the critical trade-off between standard compliance, user convenience, and privacy guarantees. By combining advanced cryptographic techniques with formal security definitions and a novel security assurance methodology, the thesis provides a foundation for the future development of secure and interoperable digital identity systems. The results contribute to the ongoing efforts in Estonia and the EU to implement secure digital infrastructure that respects user privacy, ensuring that technological advancements in cryptography can be safely integrated into regulatory frameworks.

Kokkuvõte
Privaatsust säilitavate tehnoloogiate kasutamine digitaalidentiteedis

Üha kasvav sõltuvus digitaalsuhtlusest on muutnud digitaalidentiteedi haldamise tänapäeva ühiskonna keskseks ülesandeks. Digitaalidentiteet koosneb hulgast omadustest, ehk atribuutidest, mis kirjeldavad ja esindavad inimest digitaalkeskkonnas. Digitaalidentiteediga seotud suhtluse suur maht tekitab märkimisväärseid privaatsusprobleeme, eelkõige seoses andmete võimaliku lekkega väljaandjatele, teenuseosutajatele ja talletusteenustele. Uurimistöö käsitleb selliste privaatsust säilitavate digitaalidentiteedi haldussüsteemide kavandamist, mis lisaks üldtunnustatud turvanõuetele vastavad ka regulatiivsetele nõuetele nagu näiteks eIDAS2 määrus ja Euroopa Liidu digiidentiteedirakenduse (EUDI Wallet) raamistik. Töö keskendub tsentraliseeritud süsteemi loomisele, kus kasutajate privaatsus tagatakse nüüdisaegsete krüptograafiliste primitiivide abil, säilitades samal ajal standarditele vastavuse ja kasutatavuse.

Uurimistöö peamine motivatsioon tuleneb olemasolevate digitaalidentiteedi halduslahenduste piirangutest. Paljud lahendused eeldavad, et atribuudid on salvestatud lokaalselt kasutaja seadmetes, nõudes kasutajatelt krüptograafiliste võtmete ja mandaatide haldamist, tekitades nii täiendavaid turvariske. Tsentraliseeritud atribuutide haldussüsteemid pakuvad paremat kasutusmugavust ja varundusvõimalusi, kuid eeldavad keskset usaldatud serverit, millel võib olla juurdepääs kasutaja andmetele. Selle lõhe ületamiseks pakub käesolev töö välja mitmeid uusi arhitektuure ja krüptograafilisi skeeme.

Töö esimene panus on CACS (Cloud Privacy-Preserving Attribute Management System), mis võimaldab kasutajatel salvestada mandaate pilvekeskkonda viisil, mis säilitab privaatsust ka ebausaldusväärse talletusteenuste korral. Süsteem kasutab otsingutoega sümmeetrilist krüpteerimissüsteemi (Searchable Symmetric Encryption) ning lävipõhist dekrüpteerimist, tagades, et pilveteenuse pakkuja ei saa teada ei päringute sisu ega kasutaja atribuute.

Kuna lävipõhine dekrüpteerimine võib lekitada informatsiooni dekrüpteeritavate krüptogrammide kohta, arendab käesolev töö edasi privaatsust säilitavat serveritoega dekrüpteerimisskeemi, võimaldamaks kliendil taotleda serverilt dekrüpteerimisel abi serverile konkreetse krüptogrammi kohta infot avaldamata. Skeemi turvalisus formaliseeritakse täieliku koosluskindluse raamistikus (Universal Composability Framework) ning tõestatakse eeldades Diffie-Hellmani arvutusprobleemi (computational Diffie- Hellman) ühe variatsiooni teostamatust. Konstruktsioon põhineb ElGamal'i võtmekapseldusmehhanismil ning kasutab täiendavalt mitteinteraktiivseid ja määratud verifitseerijaga nullteadmustõestusi.

Seejärel esitatakse täiustatud atribuutide haldussüsteem, mis arendab edasi CACS'i kontseptsiooni, vähendades veelgi infoleket AMS serverile ja realiseerib tühistamisprotokolli, mis võimaldab nii kasutajail kui väljaandjail mandaate tühistada ilma AMS serverile täiendavat teavet avaldamata. Töö defineerib ideaalse funktsionaalsuse tsentraliseeritud digitaalidentiteedi haldussüsteemide turvanõuete modelleerimiseks ning tõestab, et loodud süsteem teostab seda funktsionaalsust täieliku koosluskindluse raamistikus.

Lisaks esitatakse töös võrdlusel põhinev uute süsteemide turvalisuse hindamise metoodika, mis võimaldab võrrelda uusi krüptograafilisi lahendusi (näiteks lävikrüptograafial põhinevad süsteemid) juba sertifitseeritud standardlahendustega (näiteks kiipkaardil põhinev autentimine), lihtsustades nii sertifitseerimisprotsessi. Süsteeme äriprotsessimudeli ja -notatsiooni (BPMN) abil modelleerides ja nende nõrkusi analüüsides saab kindlaks teha, et uus süsteem on vähemalt sama turvaline kui juba sertifitseeritud süsteem. Metoodika praktilise rakendusena võrreldakse ja analüüsitakse töös privaatvõtmete kaitsemee-

todeid, võrreldes lävikrüptograafiat ja anonüümseid mandaate traditsiooniliste riistvarapõhiste lahendustega, sealhulgas füüsiliste kaugturvamoodulitega (remote HSM). Tulemused näitavad, et traditsioonilised süsteemid on lihtsamini sertifitseeritavad, kuid koondavad usaldust üksikutesse füüsilistesse komponentidesse. Lävikrüptograafia seevastu jaotab usaldust kasutaja seadme ja tugiserveri vahel, võimaldades täiendavaid omadusi, nagu kloonituvastus ja seadmemälu ekstraheerimiskindlus. Samas eeldavad need lahendused serveripoolset võtmehaldust turvalistes riistvarakeskkondades ning täiendavate privaatsustehnikate (nt pimendamine) rakendamist, et vältida infoleket tugiserverile. Anonüümsete mandaatide süsteemid pakuvad tugevat mitteülekantavust, kuid koondavad taastamisriskid ühte peasaladusse, mis nõuab hajutatud salvestust ja ranget autentimist.

Töös esitatud krüptograafilised skeemid ei tugine arvutus- ja ressurssimahukale krüptograafiale, mis teeb nad lihtsamini praktiliselt kasutatavaks ka mobiilseadmetes ja piiratud ressursiga keskkondades. Jõudlusanalüüsid näitavad, et privaatsust säilitavate mehhanismide lisakulu jääb praktiliste rakenduste kontekstis vastuvõetavatesse piiridesse. Samuti näidatakse, et lävikrüptograafial põhinevat autentimissüsteemi on võimalik põhjendatult käsitleda vähemalt sama turvalisena kui kiipkaardipõhised süsteemid.

Kokkuvõttes esitab käesolev töö metoodilise ja tehnilise raamistiku privaatsust säilitavate digitaalidentiteedi haldussüsteemide arendamiseks. Töö käsitleb tasakaalu standarditele vastavuse, kasutusmugavuse ja privaatsusgarantiide vahel. Kombineerides nüüdisaegseid krüptograafilisi meetodeid ja formaalseid turvadefinitsioone uudse turvalisuse võrdlusmetoodikaga, loob töö aluse turvaliste ja koostalitlusvõimeliste digitaalidentiteedi süsteemide edasiseks arendamiseks. Saadud tulemused toetavad Eesti ja Euroopa Liidu plaane turvalise ja privaatsust säilitava digitaalidentiteedirakenduse kujundamisel ning võimaldavad nüüdisaegsete krüptograafiliste lahenduste integreerimist regulatiivsetesse raamistikesse.

Appendix 1

Publication I

A. Kalu, B. C. Kus, P. Laud, K. L. Leung, N. Snetkov, and J. Vakarjuk. CACS: A cloud privacy-preserving attribute management system. In *Proceedings of the 18th International Conference on Availability, Reliability and Security, ARES 2023, Benevento, Italy, 29 August 2023- 1 September 2023*, pages 122:1–122:9. ACM, 2023

CACS: A Cloud Privacy-Preserving Attribute Management System

Aivo Kalu
Cybernetica AS
Tallinn, Estonia
aivo.kalu@cyber.ee

Burak Can Kus
Cybernetica AS
Tartu, Estonia
burak.can.kus@cyber.ee

Peeter Laud
Cybernetica AS
Tartu, Estonia
peeter.laud@cyber.ee

Kin Long Leung
Cybernetica AS
Tartu, Estonia

Nikita Snetkov
Cybernetica AS
Tallinn University of Technology
Tallinn, Estonia
nikita.snetkov@cyber.ee

Jelizaveta Vakarjuk
Cybernetica AS
Tallinn University of Technology
Tallinn, Estonia
jelizaveta.vakarjuk@cyber.ee

ABSTRACT

We present Centralized Attribute Collection Service (CACS), a system for storing credentials in the cloud, with satisfying privacy properties for users, and additional assurances for relying parties. The system deploys privacy-enhancing technologies to protect users' identities and credentials presented to relying parties. At the same time, the system can vouch for the trustfulness of issuers to relying parties. The system also allows users to obtain the access logs for their credentials, enabling them to compare them against their usage. The presentation of credentials to relying parties follows the standard protocols of Mobile Driving Licence (mDL, ISO 18013-5).

ACM Reference Format:
Aivo Kalu, Burak Can Kus, Peeter Laud, Kin Long Leung, Nikita Snetkov, and Jelizaveta Vakarjuk. 2023. CACS: A Cloud Privacy-Preserving Attribute Management System. In *The 18th International Conference on Availability, Reliability and Security (ARES 2023), August 29–September 01, 2023, Benevento, Italy*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3600160.3605022

1 INTRODUCTION

Verifiable credentials (VC) are statements about a *user*'s attributes, digitally signed by an *issuer*. An issuer can either create these attributes themselves (e.g. universities issuing diplomas, government licences, or fee collectors issuing proofs of payment), or can vouch for attributes' validity (e.g. the population registry stating the user's day of birth). Users may hold credentials from a large number of issuers, and present them, as needed or requested, to relying parties (RPs; e.g. service providers). RPs verify the signatures and check the content of the credentials against their access policies. There exist protocol sets going through the standardization [15, 28] that aim to give the user precise control over the exposure of their VCs to RPs.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ARES 2023, August 29–September 01, 2023, Benevento, Italy
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0772-8/23/08…$15.00
https://doi.org/10.1145/3600160.3605022

On the bright side, these data formats and protocols have the potential to give the user a great amount of sovereignty over their attributes. On the flip side, the user has to exert a lot of effort to obtain such control. The user must properly manage the collection and storage of their credentials, as well as their identifiers and the corresponding private keys.

The user could benefit from a more centralized service which provides management assistance. This service could store credentials obtained by the user from multiple issuers. Later, the service either could hand the necessary credentials over to the user while they communicate with an RP, or even participate in the handshake protocol itself by vouching for the existence of the specific credentials. This centralized service has to be transparent to RPs and issuers in terms of protocol flow. The service also has to be private — it should learn as little as possible about the credentials that are being presented.

The same service may also be useful for RPs who are unable to check the trustworthiness of all the different issuers themselves. Even while mediating this trust relationship, the central service should not be able to link different handshakes on the basis of the involved RPs or issuers, nor enable other parties to do the linking.

In this paper, we present CACS, a privacy-preserving centralized service that stores the credentials of users in the cloud, and facilitates their collection from issuers and their presentation to RPs. The Attribute Management Service (AMS) is the central component of the CACS system. It manages the credentials issued to users and allows RPs to verify them. RPs interact with users and the AMS through the standard data formats and protocols of *mobile Driver's License (mDL)* [15], allowing them to read the mobile credentials (*mdoc*) without changing their existing or upcoming mDL-supporting software components. mDL is a digital representation of the identification information contained in a physical driver's license or an identification card issued by the state that is stored on the user's mobile device [25]. The mDL holder, the person with the identity represented in the mDL, has full control over their data and can decide which information should be exchanged and with whom. When a verifier needs to obtain information about a person to grant access to one or more resources, a verifier initiates the process of sharing information by sending a request for the needed attributes. Next, the mDL holder receives a request on their device and decides whether to share the requested information. Finally, the requested information and accompanying data needed to prove the authenticity is sent to the verifier. This standards-compliance

is a major goal and a major constraint of CACS. In this paper, we describe and discuss which methods for enhancing users' privacy can still be applied under this constraint.

Compared to a "simple" mDL instance, CACS thus achieves greater convenience for users, as well as their greater control (and, arguably, privacy) over their attributes and credentials. The introduced central component, the AMS, takes over the management of decryption keys for the credentials and the maintenance of access logs. Considering that each user can have hundreds of attributes, introducing a centralised AMS server would help them manage attributes and outsource storage to a secure database. While this central component also gains access to the interaction and credential usage patterns of users, we consider this an acceptable price to pay for standards-compliance, because (1) the AMS server has a small footprint and similar tasks to certification authorities or cloud signature providers, permitting them to be audited against similar compliance statements [5], and (2) the small footprint of the AMS server makes its implementation on top of trusted execution environments, e.g. Intel SGX, feasible. Additionally, we consider backup and recovery to be an important property that is hard to achieve if attributes are stored locally on the user's device without having an AMS server component.

2 RELATED WORK

Krenn et al. [21] proposed a cloud attribute-based credential system using proxy re-encryption [3] and structure-preserving signatures [1]. An improved version of [21] was introduced by Haböck et al. [13], reducing the ambiguity of the original work but complicating the solution with blind signatures and additional zero-knowledge proofs. Later, a similar approach to build an attribute-based credential system was used in the CREDENTIAL project [1] [17, 18, 20, 30].

Android OS offers the *Identity Credential* API[2] that facilitates access to hardware-based credential store(s) in the device, allowing credentials obtained from issuers to be securely stored on the device and presented to relying parties. This API is used by Google's mDL implementation[3]. Our CACS system uses these APIs to implement the RP-to-AMS communication.

There exist a number of proposals for decentralized management of credentials [2, 14, 24] that may leverage blockchain technologies while still delivering privacy to users. The goals of such proposals diverge from those of the CACS system, where we aim to centralize the collection of attributes, management of keys, and maintenance of trust relationships. In addition, these proposals do not have the standards-compliance that we desire.

A big part of the contribution of this paper is the set of applied techniques for enhancing the privacy properties of accessing outsourced databases. One possible technique for such enhancement is searchable symmetric encryption (SSE) [6–10, 12, 22, 27]. It is a cryptographic primitive that allows users to outsource storing symmetrically encrypted data and later querying that encrypted data. There are various applications of SSE, including encrypted search on webmail systems [23], encrypted US national gun registries [16] and searchable audit logs [31].

Another possible privacy technology is threshold encryption [26]. This technology enables the decryption key to be shared among several parties such that a sufficiently large coalition of them must cooperate in order to decrypt a ciphertext. Depending on the number of decrypting parties in a system and their capabilities of computing and keyshare protection, different proposed schemes may be the most fitting, e.g. the password-initiated two-party protocol by Blazy et al. [4], or the password-protected two-party protocol by Kirss et al. [19].

3 PRELIMINARIES

3.1 Parties and processes

The main actors in the system are *issuers*, *users*, and *relying parties*. Their communication is facilitated by the *Attribute Management Service (AMS)*, a central service that helps issuers issue mDL-formatted attribute credentials to users; mediates the authentication of users to RPs; handles the creation of and removal of attribute records; manages the consent of users and the creation of wallet passes; and maintains the logs of credential issuance, use, and removal. Users communicate with the AMS server through its web portal. From the user side, this communication is managed by a special *wallet* application, through which the user authenticates themselves and authorises the AMS server to share their attributes with the RP.

The AMS provides credential management services to users, and therefore requires significant trust from them. However, the AMS does not have sufficient storage capacity for all users' credentials and log items. It has to outsource the storage to a cloud storage provider; this provider is no longer trusted by users. The *AMS database* is the cloud component that stores users' attributes. The *Log database* is the cloud component that stores logs of transactions involving the user's attributes, users' consent to request attributes from issuers, and consent to share attributes with the RP.

The main processes that the system together with the AMS has to support are the following:

- An issuer issuing an attribute to a user and uploading it to the AMS database for storage.
- A user presenting attributes to the RP. The presentation may happen over a variety of channels. In this paper, we describe it on the basis of the variant, where the mdoc presentation happens through face-to-face communication.
- A user reading the log of transactions performed with the attributes and credentials of this user.

The information from these processes should not reveal which RP the user is communicating with and what the content of their credentials to the cloud provider is.

3.2 Requirements

Considering the processes that have to be supported, the main functional and non-functional requirements for the CACS system and its components are the following.

Functional requirements.

- The AMS allows users to authenticate themselves on a portal using different identification services.

[1] https://credential.eu/
[2] https://developer.android.com/reference/android/security/identity/IdentityCredentialStore
[3] https://google.github.io/identity-credential/

- The AMS allows users to store attributes.
- The AMS allows users to revoke/invalidate their attributes.
- The AMS has the possibility to revoke/invalidate the user's attributes.
- The AMS allows releasing attributes to the RP if the RP's request passes validation; the RP has corresponding permissions and the user has consented to release his/her data.
- The AMS requests the user's consent to release data every time the user's attributes are requested by the RP.
- The AMS allows the user to disclose fewer attributes than are requested by the RP.
- Users have access to personal data that the AMS holds about them.
- All accesses of user's attributes are logged, and the user can look up the logs later through the AMS.

Non-functional requirements.

- The wallet application provides full control over the user's data.
- Identification, authentication, signature creation, and attributes sharing do not occur without the consent of the user.
- Any data element is encrypted at rest and during transit if applicable.
- Private keys are non-exportable and secure storage techniques are in place to enforce this.
- The system should not introduce significant performance overhead over the existing authentication and credential verification methods.

As mentioned above, we also want RPs to handle the credentials in a standards-compliant manner to interface with the CACS system without further changes from their side. The possible standards to target are either *mobile Driver's License* [15], or *Verifiable Credentials* [28]. As the first is significantly more comprehensive than the second [29], we have chosen mDL, its mdoc-formatted credentials, and its protocols for the RP to obtain and verify the credentials either from a user, or directly from the issuer of that credential.

4 ARCHITECTURE

4.1 Overall

The CACS system provides the AMS. The AMS has interfaces for issuers and for users' wallet applications. The latter is also a component that is provided by the CACS system. The wallet application enables the user to learn which attributes the RP wants to see, and to choose which attributes he consents to present to the RP. The wallet application also mediates the management of user's attributes stored by the AMS, and enables access to the log items associated with the user's attributes. The server that implements the business logic of the AMS is a trusted party in the CACS system — it is allowed to learn the user-RP interactions, as well as the attributes users present to RPs.

The AMS maintains two large databases, the *AMS database* storing users' attributes (together with their current status), and the append-only *Log database* storing the logs of accessing the attributes. These databases are untrusted components of the CACS system; the AMS uses privacy-enhancing technologies to access them. We assume that the cloud service provider hosting these databases follows the protocols described below — it is a reasonable assumption that large organizations do not deviate from the protocols, because getting caught would have large financial costs for them. However, the cloud service provider is curious and tries to gather information about users' and the RPs' behaviour from the messages it receives from the AMS.

Additionally, the AMS exposes the standards-compliant interface towards the RPs, where it appears as an issuer for them, supporting the online retrieval of credentials.

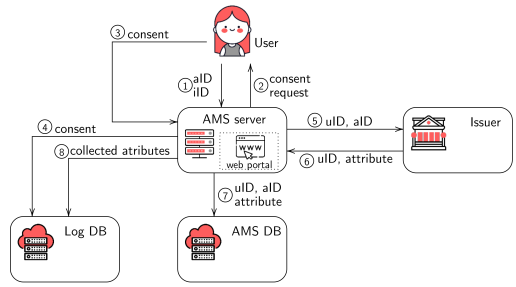


Figure 1: Issuing attributes

4.2 Detailed processes

We start by describing how the attributes are issued and added to the AMS database. The corresponding process is represented in Figure 1.

(1) The user authenticates to the AMS web portal and selects the attributes that should be requested via the corresponding attribute identifiers (aID). Additionally, the user selects Issuer (attribute source) identifiers (iID).
(2) The AMS server sends the consent request to the user in the form of an authentication or signing request to get the user's consent to query the chosen attributes.
(3) Based on their decision, the user performs an authentication or signing transaction. The AMS server receives the consent and verifies it.
(4) The AMS server saves the consent into the Log database.
(5) The AMS server creates an attribute collection request and sends it to the Issuer, providing the attribute identifier (aID) and the user identifier (uID).
(6) The issuer provides signed attributes.
(7) The AMS server verifies Issuer signature attributes and saves/updates the received attributes to the AMS database.
(8) The AMS server logs attribute collection by saving a new record in the Log database.
(9) The AMS server informs the user of attribute collection through the AMS web portal.

The presence of already established consent mechanisms between the user and the issuer allows for potential enhancements. An additional consent request can be added between Steps 5 and 6.

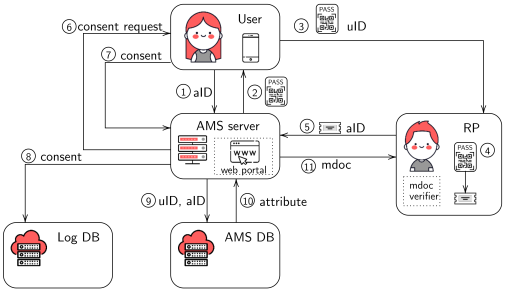


Figure 2: Presenting attributes

Once consent is obtained, the attribute can be transmitted to the AMS server. This enables the issuer to ask for consent from the user by leveraging a preexisting mechanism.

We note that the process of issuing credentials is not specified in mDL ISO standard [15]. All interfaces and messages in this process are specific to the CACS system.

Next, we describe the process of presenting attributes through the user's face-to-face interaction with the RP. The corresponding process is represented in Figure 2.

(1) The user authenticates to the AMS web portal and selects the attributes they want to share with the RP.
(2) The AMS generates a QR code which includes the server retrieval token. Then the AMS generates an Apple Wallet pass in PKPASS format with a QR code, which the user can download.
(3) The user downloads and saves the PKPASS file, opens it and presents the QR code to the RP.
(4) The RP uses the mdoc verifier to retrieve the server retrieval token by scanning the QR code.
(5) The RP requests the attributes from the AMS server by presenting the server retrieval token and the user identifier.
(6) The AMS server verifies the mdoc request and sends a consent request to the user in the form of authentication or signing.
(7) By authenticating or signing, the user gives consent to share data with the RP.
(8) The AMS server receives the consent, verifies it and saves it into the Log database.
(9) The AMS server requests the necessary attributes from the AMS database by providing the user identifier and attribute identifier.
(10) The AMS database sends the requested attributes to the AMS server.
(11) The AMS forms and signs an mdoc document that includes the user's chosen attributes. Next, the AMS server sends the mdoc response to the RP.
(12) The RP verifies the AMS server signature on the server retrieval mdoc response. The RP reads the mdoc data and performs verification.

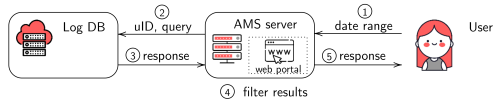


Figure 3: Searching for logs

In this process, standards compliance is important. The QR code generated in step (2) has the structure of the standard mDL device engagement structure. The request and response made in steps (5) and (11) are the standard mDL online credential verification query and its answer. However, we see how the steps in between are used to manage user consent.

The system, moreover, allows users to search through the logs in the Log database. These logs contain information about the consent the user has given to request and share their attributes. The process for searching logs is represented in Figure 3:

(1) The user authenticates to the AMS web portal and makes a request to view the logs connected to their transactions. The user presents the date range in which they wish to view the transactions.
(2) The AMS server requests the corresponding logs from the Log database by providing the user identifier.
(3) The AMS server receives all logs connected to the user.
(4) The AMS server filters the results based on the user-provided range.
(5) The AMS displays information about transactions to the user through the AMS web portal.

4.3 Key choices

When selecting appropriate cryptographic primitives for the system, we focus on the following parts:

- The AMS database;
- The log database;
- The interaction between the AMS server and the AMS database;
- The interaction between the AMS server and the Log database;

We want attributes to be stored in encrypted form in the AMS database and logs encrypted in the Log database. We consider the AMS database and the Log database to be cloud components and therefore it is crucial to protect the private information within them. That includes not only hiding contents of data while it is stored in the database, but also data that can be leaked through queries to the database. Since the AMS server forms an mdoc document that must contain user attributes in plaintext form (according to the standard), we cannot restrict the AMS server from being able to decrypt attributes. However, we want to add some additional privacy properties to hide the content of queries that the AMS server makes to the AMS database and the Log database. We are only changing the message flow between the AMS server and the AMS database, the AMS server and the user's device and AMS server and the Log database. We are not changing messages that are specified by mdoc format.

4.4 Application of searchable encryption

Searchable encryption schemes allow a client with limited memory to outsource storage of encrypted data to a server while being able to privately search through the encrypted data using keyword queries. Existing SSE schemes can be divided into two categories: static SSE [7, 10, 22] only supports search queries using keywords while dynamic SSE [6, 8, 12] supports additional update queries.

One of the basic constructions used for SSE is the *key-value store*. First, the client extracts keywords from the documents and prepares two tables. The first table is used to map keywords to the encrypted index of the document that contains this keyword. The second table maps the index of the document to the encrypted document. Next, the client uploads both tables to the server. Searching for the document using keywords happens in two steps: (1) the client performs a keyword query and receives the entry from the first table; (2) the client decrypts the index of the document and performs a document retrieval query [12].

One of the biggest challenges for SSE schemes is striking a balance between efficiency and privacy. In the context of SSE schemes, privacy refers to the leakage that occurs during the search/update queries. Most of the SSE schemes analyse leakage that occurs in the first phase, when the client performs a keyword query [11]. But the leakage that occurs during the document retrieval phase is ignored.

SWiSSSE is one of the recent dynamic SSE schemes that analyses the security of a system as a whole (taking into account both keyword and document retrieval queries) [12]. SWiSSSE uses some ORAM-style techniques to mitigate leakages in their SSE scheme. Because the dynamic version of SWiSSSE supports keyword searches and updates, both operations require only two rounds of communication between the client and the server. SWiSSSE is proven to be adaptively secure under system-wide security definitions with respect to different leakage profiles. A dynamic SWiSSSE construction offers stronger forward and backward privacy guarantees than other state-of-the-art SSE constructions, including those based on ORAM. Furthermore, SWiSSSE differs from ORAM-style SSE solutions by not requiring any write-backs during the online query process. Prior ORAM-style SSE schemes performed reads and writes during online query processing. However, SWiSSSE performs write-backs independently and periodically at pseudorandom timestamps.

We propose using SSE to store attributes in the cloud AMS database and perform search queries in a privacy-preserving manner. The description below is based on using SWiSSSE [12] as SSE. In this setup, we assume that the AMS database is an honest but curious party that follows the protocol but wants to learn some information about the attributes (and structure of the encrypted database) that it is not supposed to learn. By introducing SSE to the AMS database component, we ensure that the AMS database will not learn which attributes are requested by the AMS server. The AMS server is considered to be a trusted party as it is supposed to store cryptographic keys for SSE and a plaintext version of some of the fetched attributes temporarily before re-uploading them to the cloud.

Since SWiSSSE allows only for single-keyword queries, we are restricted in how we select keywords. We propose that keywords consist of concatenating the user identifier and attribute name (uID||aID). In the case that the AMS server needs to request several attributes, they perform several separate search queries.

When setting up an encrypted database for SSE, a database is padded with dummy entries. This is done to support insertion queries without the database noticing that new entry is added to the database. Thus, the AMS server will perform insertion queries by replacing fake documents with the real ones.

Let us assume that the SSE encrypted database is already set up on the AMS database side following the SWiSSSE setup phase. The AMS server holds a pair of SWiSSSE keys – one key is required for a pseudo random function (PRF) to store keywords and the second key is required for symmetric encryption of attributes.

User registration. When a new user is registered, they give consent to request a set of attributes. The AMS server requests the selected attributes from issuers and uploads them to the AMS database using a SWiSSSE insert query.

Searching for attributes. The only difference from the process described in the previous section is in Step 10. The AMS server now performs a SWiSSSE search query with the AMS database. The AMS server uses uID||aID as keywords to search for. The AMS server may need to perform multiple search queries if the user wants to share several attributes. Attribute search happens in two rounds:

(1) With the first round, the AMS server receives identifiers of the documents that contain the searched keyword.
(2) With the second round, the AMS server receives the documents themselves using the identifiers received in the first round.

Revocation of attributes. When the AMS server wants to revoke a specific attribute, the corresponding attribute is searched for using a SWiSSSE search query to the AMS database. The attribute is updated to contain the status *revoked* and is uploaded back to the AMS database as a dummy entry. We propose not to delete revoked attributes from the database entirely, but to convert them into dummy entries. This way we ensure that there will be enough space for the new attributes in the encrypted database.

Updating/adding new attributes. When the AMS receives a new attribute for the user, it performs a SWiSSSE insertion query with the AMS database. A new encrypted attribute replaces one of the dummy documents in the AMS database. Similarly, when the attribute value is updated, the old version of the attribute is updated to contain the status *revoked* (turning it into a dummy document). The new version of the attribute is added to the AMS database using a SWiSSSE insertion query.

Limitations. When setting up the encrypted database, the SWiSSSE scheme pads keyword occurrences with fake ones such that the minimum ratio between the padded keyword frequency and the original keyword frequency is two. The number of inserted fake documents equals the number of real documents. Therefore, the number of real documents can be doubled using insertion queries (by replacing fake documents with real ones). Additionally, the number of documents containing a certain keyword can be at most doubled without changing the way the keyword table is structured. Those parameters that identify the number of fake documents and padded keyword frequency can be changed depending on the application and its requirements. However, the number of fake documents will

still be limited and there will always be a limit on the number of new files that can be added to the encrypted database.

4.5 Application of threshold decryption

The other approach of protecting attribute storage would be to use a public key encryption scheme that supports threshold decryption. In a threshold decryption scheme, a private key is shared between n parties and only a subset of t (where $t \leq n$) parties can decrypt the ciphertext. Encryption happens without interaction, using a single public key. In our architecture, attributes will be stored encrypted in the AMS database, and when certain attributes should be shared with the RP, the AMS server and the user (wallet app) run the threshold decryption protocol. In this way, the user authorises the AMS server to share specific attributes by applying its share of the decryption key. Intuitively, with this solution we want to store attributes in encrypted form such that only the person to whom the attribute is issued can authorise decryption queries, by applying their share of the decryption key.

Using a threshold decryption scheme gives us the advantage of not storing private key entirely on the user's device or on the AMS server side. We split the trust by storing one share of the private key on the user's device and the second share on the AMS server. In order to decrypt ciphertext, both key shares should be applied. The encryption scheme that supports distributed decryption is constructed from ElGamal encryption scheme from Kirss et al [19].

User registration:

(1) The user's device (wallet app) runs the key generation protocol with the AMS server. As a result, the user device stores one share of the private key and the AMS server stores the second share. The user's share of the private key is stored protected by the user's PIN code.
(2) The AMS requests the user's attributes from the Issuer, and receives them encrypted (using the user's public key) and signed.
(3) The AMS stores encrypted attributes in the AMS database.

Searching for attributes. We describe the difference in Step 10 here.

(1) The AMS server requests encrypted attributes from the AMS database by uID and aID. The AMS server can request several attributes at a time.
(2) The AMS initiates a threshold decryption protocol with the user device.
(3) The user device applies its share of the private key and the AMS server finishes the decryption process by applying its share of the private key and obtains a plaintext version of the requested attributes.

Revocation of attributes. When the AMS server wants to revoke a specific attribute, the corresponding encrypted attribute is deleted from the AMS database.

Updating/adding new attributes. When the AMS server receives new attributes for the user (encrypted with the user's public key), they upload the corresponding ciphertext to the AMS database. When the attribute value is updated, the AMS receives the new ciphertext from the Issuer (encrypted with the user's public key), deletes the old version of the attribute from the AMS database and uploads the new one.

By using threshold decryption, we ensure that the AMS database does not learn the plaintext of stored attributes. Additionally, the AMS server cannot decrypt users' attributes without the user participating in the decryption process, which adds an additional layer of protection in the case when the AMS server wishes to share some of the attributes without user authorisation.

Limitations. One of the limitations of this approach is that the private key share that is stored on the user's device cannot be restored if the device is lost. In this scenario, the user would be required to generate a new key pair with the AMS server and the AMS server would need to request new attributes for the user encrypted under the newly generated public key.

4.6 Combined approach

In this section, we describe the combination of approaches outlined above and discuss its advantages and limitations.

Here we also assume that the AMS encrypted database has been already set up and the AMS is holding keys for the SWiSSSE scheme.

User registration:

(1) The AMS server runs the key generation protocol with the user's device.
(2) The AMS server requests the user's attributes from the Issuer, providing the Issuer with the user's public key.
(3) The issuer provides the requested attributes, signed and encrypted with the user's public key.
(4) The AMS server encrypts the received attributes and uploads them to the AMS encrypted database using a SWiSSSE insert query.

Searching for attributes:

(1) The AMS server performs SWiSSSE search queries with the AMS database and obtains attributes (that are still encrypted with the user's public key). If several attributes are required, the AMS server performs several search queries.
(2) The AMS server initiates a threshold decryption protocol with the user's device to decrypt all requested attributes.
(3) The user device applies its share of the private key. The AMS server finishes the decryption process by applying its share of the private key and obtains the plaintext version of the requested attributes.

Revocation of attributes. When the AMS server wants to revoke a specific attribute, the corresponding attribute is searched for using a SWiSSSE search query to the AMS database. Next, the AMS server initiates threshold decryption with the user's device to change the attribute status to *revoked*. Then, the attribute is encrypted with the user's public key and uploaded back to the AMS database as a dummy entry.

Updating/adding new attributes. When the AMS server receives a new attribute for the user (encrypted with the user's public key), it performs an SSE insertion query with the AMS database. Similarly, when the attribute value is updated, the AMS server initiates threshold decryption with the user's device to change the attribute status to *revoked* (turning it into a dummy document). The new version of the attribute is added to the AMS database using a SWiSSSE insertion query.

Discussion. With this approach, we achieve advantages of both schemes: the AMS database does not learn the search pattern or

the content of each query. Additionally, AMS server is no longer capable of decrypting the user's attributes without interacting with the user.

4.7 Protection of the Log database

For the Log database we propose to use the same solutions as was done for the AMS database. However, the usage of SSE would require creating several encrypted databases, as the amount of information that should be stored in the database will increase. We propose having several physically separated Log databases, each storing its own encrypted database. If we stored several encrypted databases within one Log database, the AMS server would need to indicate which database is being queried, leaking some information to the Log database.

SSE approach. When the AMS server is required to save a new entry to the Log database, it performs a SWiSSSE insertion query that replaces a dummy database entry with a new log. When the user submits a search request to the AMS server, presenting the date range they want to receive results in, the AMS server makes a SWiSSSE search query to the Log database, using the user identifier as a keyword. The AMS server decrypts all received results and sorts them based on the user-provided date range.

Threshold decryption approach. When the AMS server is required to save a new entry to the Log database, it extracts the date and encrypts the rest with the user's public key. It is more efficient to extract the date in advance so the AMS server would not need to run decryption of all the users logs, since each decryption process would require interaction between the user's device and the AMS server. Therefore, the AMS server uploads the ciphertext to the Log database, keeping the date field unencrypted. When the user submits a search request to the AMS server with the date range, the AMS server queries the Log database by the user identifier. The AMS server sorts encrypted logs based on the date range to identify which logs should be decrypted. Next, the AMS server initiates threshold decryption with the user's device.

Combined approach. To save a new entry in the Log database, the AMS server extracts the date and encrypts the rest with the user's public key. Next, the AMS server inserts this entry to the AMS database using a SWiSSSE insert query. When the user requests their logs within a certain time range, the AMS performs a SWiSSSE search query with the Log database. Next, the AMS server filters logs based on the time range provided by the user. The AMS server initiates a threshold decryption protocol with the user's device. Once all the logs are decrypted, the AMS server displays the results to the user.

5 ANALYSIS

5.1 Integrity

We note that CACS does not relax the integrity properties of mDL — all decisions to accept or reject user credentials are still made by RPs. The AMS may help RPs navigate the landscape of issuers, assuming a role similar to a *root CA* that can vouch for the trustworthiness of individual issuers.

5.2 Privacy

All the presented approaches meet the defined requirements. Moreover, they benefit from additional privacy properties, such as the AMS database not being able to learn the content of queries (for the SSE and the combined approaches) and the AMS not being able to decrypt attributes without interacting with the user's device (for the threshold and the combined approaches). For all approaches, keys that are stored on the AMS server side should be stored in an HSM to ensure that these cannot be extracted. For approaches relying on threshold decryption, the user's share of the private key should be stored protected with the user's PIN code.

All presented approaches have their advantages and limitations. The threshold decryption approach relies on public key cryptography, while searchable encryption is based only on symmetric primitives. Searchable encryption uses SHA-256 and AES as its main building blocks, while threshold decryption is based on the ElGamal encryption scheme and uses zero-knowledge proofs about private exponents. The combined approach, therefore, uses both sets of primitives.

In the searchable encryption approach, the AMS server stores the symmetric encryption key that can be used by the malicious AMS server to decrypt all attributes without the user's consent. By using threshold decryption, we can ensure that AMS server cannot decrypt attributes without user participation in the decryption process. By combining threshold decryption and SSE, we solve the problem of the AMS server being able to decrypt attributes without interacting with the user. The AMS server learns the plaintext version of attributes in any case when sharing those with the RP. However, we still can protect attributes from the AMS server before those attributes have been shared with the RP.

When using searchable encryption, an AMS attribute search query will always return some dummy attributes to hide the search pattern, meaning that it has a higher communication cost than the threshold decryption approach.

With all three approaches, the AMS database does not learn the plaintext attributes. However, searchable encryption and the combined approach also ensure that the AMS database does not learn which attributes are requested or the inner structure of the encrypted database. With threshold decryption, the AMS database additionally learns user identifiers as part of the search query made by the AMS server.

As with the threshold decryption approach, the combined approach has the problem of the user not being able to decrypt attributes when the device storing its share of the private key is lost. The SSE approach does not have this limitation as the AMS server is responsible for storing cryptographic keys that can be used to decrypt ciphertext attributes.

In Table 1 we present a comparison of approaches described in this work with the *plain model* that corresponds to the attributes being stored on the user's device as plaintext without a trusted AMS server that manages the attributes. The table focuses on the smaller part of the CACS system, mainly communication between: a) the user, the AMS server; b) the AMS server and the AMS database.

	Plain model	Searchable encryption	Threshold decryption	Combined approach
Trust	The user's device is trusted not to leak plaintext attributes.	The AMS server is trusted not to decrypt attributes without the user's consent and not to leak plaintext attributes. The user's device is trusted not to forge consent.	The AMS server is trusted not to leak plaintext attributes. The user's device is trusted not to forge consent.	The AMS server is trusted not to leak plaintext attributes. The user's device is trusted not to forge consent.
Leakage	Not applicable	Negligible leakage from the search queries to the AMS database.	The AMS database learns which ciphertexts are being decrypted.	Negligible leakage from the search queries to the AMS database.
Efficiency	Efficient since it does not require any cryptographic primitives.	Search is computationally efficient as it relies on symmetric primitives. Increased bandwidth due to dummy data presence.	Moderate computational efficiency since it relies on public-key cryptography. No bandwidth overhead. Increased number of communication rounds to decrypt several attributes.	Search is computationally efficient as it relies on symmetric primitives. Increased bandwidth due to dummy data presence. Increased number of communication rounds to decrypt several attributes.
Storage requirements	The user's device has to provide enough storage for attributes.	The AMS server should store some attributes in local memory. Storage overhead from dummy data on the AMS database side.	No storage overhead on the AMS database size. The user's device is required to only store share of the decryption key.	The AMS server should store some attributes in local memory. Storage overhead from dummy data on the AMS database side.
Difficulty for the user	The user is required to manage their attributes by themselves.	The user is not required to perform additional actions.	The user is required to enter their PIN to apply their share of decryption key.	The user is required to enter their PIN to apply their share of decryption key.
Difficulty for the issuer	The issuer should provide their own system to communicate with users to issue attributes directly.	Does not add constraints to the issuer.	If the user's device is lost, the issuer needs to provide new attributes encrypted with a new public key.	If the user's device is lost, the issuer needs to provide new attributes encrypted with a new public key.
Backup and recovery	The user's attributes are lost if the device is lost	A loss of device does not affect accessibility of the attributes.	The user's attributes cannot be decrypted if the device is lost.	The user's attributes cannot be decrypted if the device is lost.

Table 1: Comparison of approaches

5.3 Efficiency

Searchable encryption and threshold decryption come with their overhead. However, as reported in [12, 19], these overheads are in the range of tens to hundreds of milliseconds. Hence they will not significantly slow down the verification of credentials.

5.4 Standards-compliance

Making appropriate trade-offs, we have managed to make CACS support any RP that properly implements the credential presentation and parsing methods of mDL with the server retrieval method [15]. We have demonstrated the interoperability using Google's RP implementation.

6 CONCLUSION

We have proposed an attribute management system that strikes a reasonable compromise between a number of different requirements: integrity, ease-of-use, privacy, standards-compliance, and efficiency. The system can be transparently rolled out from the point-of-view of relying parties, and mostly also from the point of view of smaller and less capable attribute issuers that would normally outsource the signing of credentials containing the attributes they issue. It lessens the technical abilities that users need in order to take advantage of wallet ecosystems, while keeping their activities private from cloud service providers that offer storage for their credentials. It improves the capabilities of various parties to perform historical analyses of credential usage, in particular the users themselves, but also the maintainers of the system.

Small-scale pilots of the proposed system are planned to take place in 2023. Their results will inform the schedule of larger deployments. It remains to be seen how we can reduce trust in the AMS server by introducing additional cryptographic primitives to the system.

Acknowledgements. This research has been supported by Estonian Business and Innovation Agency and the Estonian Research Council under the grant number PRG1780. We would like to thank colleagues from SK ID Solutions for the help in designing the initial functional requirements of CACS. We would like to thank William Sendewicz for proofreading the final draft.

REFERENCES

[1] Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. 2011. Optimal Structure-Preserving Signatures in Asymmetric Bilinear Groups. In *Advances in Cryptology – CRYPTO 2011*, Phillip Rogaway (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 649–666.

[2] Amal Abid, Saoussen Cheikhrouhou, Slim Kallel, and Mohamed Jmaiel. 2022. NovidChain: Blockchain-based privacy-preserving platform for COVID-19 test/vaccine certificates. *Softw Pract Exp.* 52, 4 (2022), 841–867.

[3] Matt Blaze, Gerrit Bleumer, and Martin Strauss. 1998. Divertible protocols and atomic proxy cryptography. In *Advances in Cryptology — EUROCRYPT'98*, Kaisa Nyberg (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 127–144.

[4] Olivier Blazy, Laura Brouilhet, Céline Chevalier, Patrick Towa, Ida Tucker, and Damien Vergnaud. 2021. Hardware security without secure hardware: How to decrypt with a password and a server. *Theor. Comput. Sci.* 895 (2021), 178–211. https://doi.org/10.1016/j.tcs.2021.09.042

[5] CA/Browser Forum. 2023. Baseline Requirements Documents (SSL/TLS Server Certificates). https://cabforum.org/baseline-requirements-documents/. Accessed: 2023-06-21.

[6] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2014. Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, San Diego, California, USA, 1–16. https://www.ndss-symposium.org/ndss2014/dynamic-searchable-encryption-very-large-databases-data-structures-and-implementation

[7] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2013. Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I (Lecture Notes in Computer Science, Vol. 8042)*, Ran Canetti and Juan A. Garay (Eds.). Springer, Santa Barbara, CA, USA, 353–373. https://doi.org/10.1007/978-3-642-40041-4_20

[8] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. 2018. New Constructions for Forward and Backward Private Symmetric Searchable Encryption. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, Toronto, ON, Canada, 1038–1055. https://doi.org/10.1145/3243734.3243833

[9] Melissa Chase and Seny Kamara. 2010. Structured Encryption and Controlled Disclosure. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6477)*, Masayuki Abe (Ed.). Springer, Singapore, 577–594. https://doi.org/10.1007/978-3-642-17373-8_33

[10] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. 2006. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006*, Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati (Eds.). ACM, Alexandria, VA, USA, 79–88. https://doi.org/10.1145/1180405.1180417

[11] Zichen Gui, Kenneth G. Paterson, and Sikhar Patranabis. 2021. Rethinking Searchable Symmetric Encryption. Cryptology ePrint Archive, Paper 2021/879. https://eprint.iacr.org/2021/879 https://eprint.iacr.org/2021/879.

[12] Zichen Gui, Kenneth G. Paterson, Sikhar Patranabis, and Bogdan Warinschi. 2020. SWiSSSE: System-Wide Security for Searchable Symmetric Encryption. Cryptology ePrint Archive, Paper 2020/1328. https://eprint.iacr.org/2020/1328 https://eprint.iacr.org/2020/1328.

[13] Ulrich Haböck and Stephan Krenn. 2019. Breaking and Fixing Anonymous Credentials for the Cloud. In *Cryptology and Network Security*, Yi Mu, Robert H. Deng, and Xinyi Huang (Eds.). Springer International Publishing, Cham, 249–269.

[14] Harry Halpin. 2020. Nym Credentials: Privacy-Preserving Decentralized Identity with Blockchains. In *2020 Crypto Valley Conference on Blockchain Technology (CVCBT)*. IEEE, 56–67. https://doi.org/10.1109/CVCBT50464.2020.00010

[15] ISO Central Secretary. 2021. *Personal identification — ISO-compliant driving licence — Part 5: Mobile driving licence (mDL) application*. Standard ISO/IEC 18013-5:2021. International Organization for Standardization, Geneva, CH. https://www.iso.org/standard/69084.html

[16] Seny Kamara, Tarik Moataz, Andrew Park, and Lucy Qin. 2021. A Decentralized and Encrypted National Gun Registry. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 1520–1537. https://doi.org/10.1109/SP40001.2021.00072

[17] Farzaneh Karegar, Daniel Lindegren, John Sören Pettersson, and Simone Fischer-Hübner. 2017. Assessments of a cloud-based data wallet for personal identity management. In *26th International Conference on Information Systems Development (ISD2017 Cyprus). Larnaca, Cyprus, September 6-8, 2017*.

[18] Farzaneh Karegar, Christoph Striecks, Stephan Krenn, Felix Hörandner, Thomas Lorünser, and Simone Fischer-Hübner. 2016. *Opportunities and Challenges of CREDENTIAL*. Springer International Publishing, Cham, 76–91. https://doi.org/10.1007/978-3-319-55783-0_7

[19] Johanna Maria Kirss, Peeter Laud, Nikita Snetkov, and Jelizaveta Vakarjuk. 2023. Server-Supported Decryption for Mobile Devices. Cryptology ePrint Archive, Paper 2023/039. https://eprint.iacr.org/2023/039 https://eprint.iacr.org/2023/039.

[20] Alexandros Kostopoulos, Evangelos Sfakianakis, Ioannis Chochliouros, John Sören Pettersson, Stephan Krenn, Welderufael Tesfay, Andrea Migliavacca, and Felix Hörandner. 2017. Towards the Adoption of Secure Cloud Identity Services. In *Proceedings of the 12th International Conference on Availability, Reliability and Security* (Reggio Calabria, Italy) *(ARES '17)*. Association for Computing Machinery, New York, NY, USA, Article 90, 7 pages. https://doi.org/10.1145/3098954.3104061

[21] Stephan Krenn, Thomas Lorünser, Anja Salzer, and Christoph Striecks. 2018. Towards Attribute-Based Credentials in the Cloud. In *Cryptology and Network Security*, Srdjan Capkun and Sherman S. M. Chow (Eds.). Springer International Publishing, Cham, 179–202.

[22] Shangqi Lai, Sikhar Patranabis, Amin Sakzad, Joseph K. Liu, Debdeep Mukhopadhyay, Ron Steinfeld, Shifeng Sun, Dongxi Liu, and Cong Zuo. 2018. Result Pattern Hiding Searchable Encryption for Conjunctive Queries. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 745–762. https://doi.org/10.1145/3243734.3243753

[23] Tatsuya Midorikawa, Akihiro Tachikawa, and Akira Kanaoka. 2019. Helping Johnny to Search: Usable Encrypted Search on Webmail System. *J. Inf. Process.* 27 (2019), 763–772. https://doi.org/10.2197/ipsjjip.27.763

[24] Rafael Torres Moreno, Jesús García-Rodríguez, Jorge Bernal Bernabé, and Antonio Skarmeta. 2021. A Trusted Approach for Decentralised and Privacy-Preserving Identity Management. *IEEE Access* 9 (2021), 105788–105804. https://doi.org/10.1109/ACCESS.2021.3099837

[25] Secure Technology Alliance Identity Council. 2020. *The Mobile Driver's License (mDL) and Ecosystem*. Standard. Secure Technology Alliance. https://www.securetechalliance.org/wp-content/uploads/Mobile-Drivers-License-WP-FINAL-Update-March-2020-4.pdf

[26] Victor Shoup and Rosario Gennaro. 2002. Securing Threshold Cryptosystems against Chosen Ciphertext Attack. *J. Cryptol.* 15, 2 (2002), 75–96. https://doi.org/10.1007/s00145-001-0020-9

[27] Dawn Xiaodong Song, David A. Wagner, and Adrian Perrig. 2000. Practical Techniques for Searches on Encrypted Data. In *2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000*. IEEE Computer Society, 44–55. https://doi.org/10.1109/SECPRI.2000.848445

[28] Manu Sporny, Dave Longley, and David Chadwick. 2022. *Verifiable Credentials Data Model v1.1*. W3C Recommendation. World Wide Web Consortium. https://www.w3.org/TR/vc-data-model/

[29] UL Solutions. 2021. Verifiable Credentials and ISO/IEC 18013-5 Based Credentials: The competitive landscape of digital credentials. White paper, https://www.ul.com/insights/verifiable-credentials-and-isoiec-18013-5-based-credentials.

[30] Fatbardh Veseli, Jetzabel Serna Olvera, Tobias Pulls, and Kai Rannenberg. 2019. Engineering Privacy by Design: Lessons from the Design and Implementation of an Identity Wallet Platform. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing* (Limassol, Cyprus) *(SAC '19)*. Association for Computing Machinery, New York, NY, USA, 1475–1483. https://doi.org/10.1145/3297280.3297429

[31] Brent R. Waters, Dirk Balfanz, Glenn Durfee, and Diana K. Smetters. 2004. Building an Encrypted and Searchable Audit Log. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2004, San Diego, California, USA*. The Internet Society. https://www.ndss-symposium.org/ndss2004/building-encrypted-and-searchable-audit-log/

Appendix 2

Publication II

P. Laud, A. Pankova, and J. Vakarjuk. Privacy-preserving server-supported decryption . In *2025 IEEE 38th Computer Security Foundations Symposium (CSF)*, pages 48–63, Los Alamitos, CA, USA, June 2025. IEEE Computer Society

Privacy-preserving server-supported decryption

Peeter Laud
Cybernetica AS
Tartu, Estonia
peeter.laud@cyber.ee

Alisa Pankova
Cybernetica AS
Tartu, Estonia
alisa.pankova@cyber.ee

Jelizaveta Vakarjuk
Cybernetica AS
Tallinn University of Technology
Tallinn, Estonia
jelizaveta.vakarjuk@cyber.ee

***Abstract*—In this paper, we consider encryption systems with two-out-of-two threshold decryption, where one of the parties (the client) initiates the decryption and the other one (the server) assists. Existing threshold decryption schemes disclose to the server the ciphertext that is being decrypted. We give a construction, where the identity of the ciphertext is not leaked to the server, and the client's privacy is thus preserved. While showing the security of this construction, we run into the issue of defining the security of a scheme with blindly assisted decryption. We discuss previously proposed security definitions for similar cryptographic functionalities and argue why they do not capture the expected meaning of security. We propose an ideal functionality for the encryption with server-supported blind threshold decryption in the universal composability model, carefully balancing between the meaning of privacy, and the ability to implement it. We construct a protocol and show that it is a secure implementation of the proposed functionality in the random oracle model.**

***Index Terms*—threshold encryption, non-interactive zero knowledge, ElGamal**

I. Introduction

Threshold cryptography offers a set of techniques for management of private keys; it can help out if there is no single party sufficiently trusted and sufficiently in control of its computational environment, such that they could be allowed to know the whole private key. Both threshold signatures [1] and threshold decryption [2] are well-known cryptographic techniques. For signatures, there exist large-scale deployments in blockchain environments [3], [4], as well as for general electronic identity [5]. In the last example, threshold cryptography is used because a certain party does not have sufficient control over their computational environment. That party is a *user*, having a *smartphone* that stores their keyshare. If one wants to avoid secure hardware based solutions [6], the phone alone cannot offer sufficient protection for a private key.

We emphasize that in the use-cases related to the electronic identity, we are specifically interested in 2-out-of-2 secret sharing of the private key. Sharing is necessary, because otherwise the server alone would be able to use the private key. Having just two parties is natural in such system — the roles of the phone and the assisting server are clearly different, while any more parties (e.g. implementing the server in threshold manner) would reduce the usability of the system, would not significantly increase the security (the server is expected to securely manage the keyshare), and may even interfere with some of the mechanisms for detecting break-ins.

In the near future, a user's digital presence is constructed around the credentials that they have obtained from the various issuers and are able to present to the various relying parties (RP). There exist standards, e.g. for mobile Driver's License [7] by ISO or for verifiable credentials [8] by W3C that specify, how these credentials are formatted and through which protocols they are presented. There exist advanced legislative initiatives, e.g. the upcoming eIDAS 2.0 regulation of the European Union [9], [10] that will mandate the availability of *digital identity wallets* — smartphone applications that manage the user's credentials by following these standards, and implementing a certain architecture [11]. The architecture may, to a certain extent, protect the user's privacy through *selective disclosure*, releasing only user-approved parts of multi-part credentials to a RP [7], [12], but more capable privacy techniques, e.g. zero-knowledge proofs [13] are not part of the architecture.

The wallet app running on the smartphone has to manage the storage of credentials. They may be stored on- or off-device, but will definitely be encrypted, because their content may be sensitive. Before a presentation of the credential, it has to be decrypted, because the wallet app needs to know its content to be presented. The app and the phone have to have control over the decryption step. If one wants to avoid secure hardware based solutions, then the decryption has to be thresholdized, and the same 2-out-of-2 setting makes the most sense. In this case, whenever an RP asks the user to present a credential that the user has and the user agrees to present it, the phone sends the ciphertext encrypting that credential to the assisting server. The phone and the assisting server run the threshold decryption protocol, and (only) the phone learns the plaintext credential. After running the protocol(s) with the RP, the phone deletes the decrypted credential from its memory. Next time the presentation of a credential is requested, the phone and the assisting server again run the decryption protocol. Any threshold decryption protocol is in principle usable here, e.g. the original ElGamal-based construction of Shoup and Gennaro [2].

Unfortunately, with regular threshold decryption protocol, the server will learn how often one or another ciphertext is being decrypted. This is a potential violation of the user's privacy: if, over time, the server sees several requests to decrypt the same credential, then it may make some inferences on the user's access pattern towards different RPs. This paper proposes a solution for this problem: we construct a 2-out-of-

2 threshold encryption scheme, where the decryption protocol is privacy-preserving against the server. Aiming for generality, our scheme is secure against chosen-ciphertext attacks even when one of the decrypting parties (either the client or the server) is corrupted.

Definitions of security properties for such schemes are similar to those for the encryption schemes with *blind assisted decryption*. Unfortunately, we have found existing definitions to not correspond to our intuitive notions of security; the discussion of these definitions is the second main contribution of this paper. We follow that discussion by proposing our own security definition for the server-assisted blind decryption. While the previous definitions have been given in the game-based model, stating the security game(s) that an adversary attempts to win, our definition is given in the universal composability (UC) [14] framework where the desired security properties are covered by an ideal functionality describing how the protocol *should* work. As there are several distinct properties that we are aiming for, we would need several different game-based definitions, which make it more complicated to justify why exactly these properties are necessary and sufficient, while an ideal functionality provides a generic overview. In this way, we get a better understanding of the details of the security definition, hopefully giving us a better assurance that it is the right one. Still, we see that our definitions follow a fine line between the intuitive meaning of the security and privacy on one side, and the implementability on the other side; we explore some new ground in defining UC encryption, and the details become significant. We see that the blinding makes analysis of composability quite important in the malicious client case. Due to the blinding, the client can potentially combine several server responses (not necessarily successful decryptions) in such a way that the plaintext (or even several plaintexts at once) will not be learned until the last decryption, and some attacks by a malicious client (in particular, the ability to decrypt more ciphertexts than there have been sessions with the server initiated) could be missed in the stand-alone model.

In the construction of our threshold encryption scheme, the goal has been to show the feasibility, and the conceptual simplicity of presentation. Still, its performance is fully satisfactory for the use-case of verifiable credentials. It should also be simple to implement and deploy, as it uses only simple cryptographic primitives, in particular avoiding pairings.

This paper has the following structure. After reviewing related work in Sec. II, we will dive into the previous security definitions of blind decryption in Sec. III. Next, we describe our security definition by presenting corresponding ideal functionality in Sec. IV. In Sec. V, we present main building blocks of our protocols and give our construction of a privacy-preserving threshold encryption scheme. Finally, we prove in Sec. VI that our construction indeed satisfies our security definition.

II. Related work

Encryption schemes with threshold decryption [2] and with indistinguishability against chosen-ciphertext attacks (IND-CCA) were proposed shortly after IND-CCA secure asymmetric encryption schemes [15]. At present, threshold cryptography is a mature field, discussed in textbooks [16] and subject to standardization activities [17]. However, we note the dearth of the exploration of threshold decryption in the UC model, meaning that many details of our ideal functionality are novel.

Regarding the ideal functionalities for the usual public key encryption, several different approaches can be taken. One can design the ideal functionality so, that the adversary internally executes the encryption and decryption algorithms [18]. Alternatively, the adversary may give descriptions of encryption and decryption algorithms to the ideal functionality, with the latter invoking them locally [19]. In the former case, the adversary may cause malicious ciphertexts to decrypt to values that depend on the values to which legitimate ciphertexts are decrypted to [19].

Our target use case involves two non-equal parties. In this setting, Buldas et al. [5] have proposed a 2-out-of-2 sharing for *signing* keys, where one of the shares is stored in a central server and the other one in the user's smartphone, encrypted with only a PIN. The signing protocol employs measures against an adversary guessing that PIN, or making a copy of the memory of the smartphone. Lueks et al. [20] have proposed a system with the same kind of 2-out-of-2 sharing, where also the users' usage patterns are protected; the system also relies on blind signatures. For the threshold decryption, Kirss et al. [21] have proposed a protocol with the same measures for protecting the keyshare in the smartphone as [5]; together with the clone detection mechanisms of [22], it could offer a viable alternative to secure elements [6]. Unfortunately, they do not attempt to hide the ciphertext from the assisting server.

Our scheme offers privacy for one of the decrypting parties against the other decrypting party. In this sense, our scheme is an instance of *blind assisted decryption*. We are aware of two previous attempts to formalize the IND-CCA security of blind assisted decryption, and devise schemes satisfying these definitions. Green [23] proposed a scheme, where the client sends decryption requests to a server with the private key. Blazy et al. [24] proposed an encryption system where the decryption capability and authorization was shared among three parties with unequal roles, offering privacy against the "server" party. We discuss both schemes and corresponding security definitions in Sec. III.

We analyze the security of our scheme in the universal composability model with ROM [14]. We use a less common hardness assumption: the *one-more static computational Diffie-Hellman problem* [25], [26]. Still, the assumption is not controversial: it is known to hold in the Generic Group Model and be equivalent to the hardness of Discrete Logarithm in the Algebraic Group Model [27]. The one-more problems commonly occur in the security analysis of *blind signatures* [25], [28], hence one may even expect it to make an appearance for our blind primitive as well.

III. Security of blind assisted decryption

Blind assisted decryption has been previously considered by Green [23] and by Blazy et al. [24]. Both works have separately defined the blindness property, and the property of being secure against chosen ciphertext attacks. While the first property is not controversial, the proposed definitions for the latter cannot be considered fully satisfactory.

Blazy et al. consider a setting with three parties — *user*, *token*, and *server*. The token and the server have keyshares $sk_{\mathcal{T}}$ and $sk_{\mathcal{S}}$ of a 2-out-of-2 threshold decryption key, while the user and the server share a password pwd. The user initiates the decryption by activating the token with the ciphertext. The token then runs a protocol with the server. They define a password-protected Indistinguishability under Replayable Chosen-Ciphertext Attacks (P-IND-RCCA). Their multi-party (i.e. there are several users, tokens, and servers, each of those may be corrupted by the adversary) definition provides the adversary with a decryption oracle that performs decryption atomically (unless the submitted ciphertext decrypts to one of the two challenge plaintexts, as is the norm for RCCA [29]). When submitting these challenge plaintexts, the adversary specifies the identities of the user $\mathcal{U}^*$, token $\mathcal{T}^*$, and server $\mathcal{S}^*$ that it is attacking. The adversary also has access to the message-sending oracle, where it submits a message to one of the parties, and learns the message that this party replies with.

In order to model that no protection is offered if the adversary corrupts too many parties, the definition of P-IND-RCCA restricts, which parties the adversary may invoke. If $\mathcal{U}^*$ or $\mathcal{S}^*$ are corrupt (i.e. the adversary knows the password), then $\mathcal{T}^*$ may no longer be invoked. If $\mathcal{U}^*$ and $\mathcal{T}^*$ are corrupt (i.e. the adversary has the password and one of the keyshares) then $\mathcal{S}^*$ may no longer be invoked. Indeed, in such cases, ability to make queries to these parties might allow the adversary to decrypt the challenge ciphertext.

Such restrictions make P-IND-RCCA more akin to the "lunchtime" (CCA1) attack, when considering message-sending oracle. Also, some intuitively insecure encryption schemes satisfy P-IND-RCCA. Consider any P-IND-RCCA secure scheme, and modify it as follows:

1) Whenever the server $\mathcal{S}$ receives a message, it will check whether it has the format $(\mathsf{corrupt}, pwd_{\mathcal{U}}, sk_{\mathcal{T}})$, where $\mathcal{U}$ and $\mathcal{T}$ are the user and the token in the current decryption session, $pwd_{\mathcal{U}}$ is user's password, and $sk_{\mathcal{T}}$ is the keyshare of the token. The check can be done by encrypting a random plaintext and then running the decryption protocol in server's head. If the incoming message has such format, then respond with its own keyshare $sk_{\mathcal{S}}$.
2) *Alternatively*, when the token $\mathcal{T}$ receives the message $(\mathsf{corrupt}, sk_{\mathcal{S}})$, the "correctness" of which it can check in the same manner, then it responds with $sk_{\mathcal{T}}$.

We argue that the modified scheme (with either the first or the second change) should not be considered secure any more, because an adversary that has managed to take over one of the parties holding a secret share can easily extend this takeover to obtaining the other share. But the modified scheme still satisfies P-IND-RCCA, because such messages cannot be sent according to this definition. Indeed, an adversary that has corrupted the user and the token (giving it $pwd_{\mathcal{U}}$ and $sk_{\mathcal{T}}$) in the session being attacked, may no longer talk to the server. Similarly, after corrupting the server and learning $sk_{\mathcal{S}}$, the adversary may no longer talk to the token.

Green [23] considers a user $\mathcal{U}$ outsourcing decryption to a server $\mathcal{D}$. The server has the decryption key; the user, having a ciphertext, initiates the blind decryption protocol. The security of their scheme against chosen ciphertext attacks is captured by a "usual" IND-CCA2 definition, where the adversary can atomically invoke the "normal" decryption oracle, and by the *leak-freeness* definition. In this definition, a Distinguisher is asked to distinguish between a "real" and an "ideal" game. In the real game, an adversary chooses a ciphertext and the Distinguisher gets the user's view in the blind decryption protocol. Note that the execution of the blind decryption protocol is *atomic*, i.e. the adversary does not interfere with its run. In the ideal game, a simulator chooses a ciphertext, receives the corresponding plaintext, and simulates the trace. I.e. leak-freeness "ensures that an adversarial user gains no more information from the blind decryption protocol than they would from access to a standard decryption oracle" [23].

The absence of the blind decryption protocol in the IND-CCA2 definition and/or the atomicity requirement in the set-up of the "real" game allows some intuitively insecure schemes to satisfy Green's IND-CCA2 and leak-freeness definitions. Consider any scheme that satisfies them, and modify its blind decryption as follows:

- All messages from $\mathcal{U}$ to $\mathcal{D}$ have an additional bit b_{evil}.
- If $\mathcal{D}$ receives a message with $b_{\mathsf{evil}} = 0$, then it processes the rest of the message as in the original scheme.
- If $\mathcal{D}$ receives a message with $b_{\mathsf{evil}} = 1$, then it responds with the decryption key.
- $\mathcal{U}$ always sets $b_{\mathsf{evil}} = 0$.

Such scheme should definitely be considered as insecure. But it satisfies both IND-CCA2 (because blind decryption protocol is not invoked there) and leak-freeness (because the atomic execution of the blind decryption protocol means that no one sets $b_{\mathsf{evil}} = 1$).

IV. Ideal functionality for privacy-preserving server-assisted decryption

We consider asymmetric encryption schemes, where the decryption functionality is distributed between two parties — the *client*, and the *server* with different roles. An encryption scheme with the client-server decryption consists of the following sets, algorithms, and protocols, all parameterized with the security parameter λ.

- Sets of ciphertexts C, public keys PK, the client's private keys SK_{C}, and the server's private keys SK_{S}.
- Key-generation protocol $\langle \mathcal{KG}_{\mathrm{C}} | \mathcal{KG}_{\mathrm{S}} \rangle$, run by both parties. It returns $(\mathsf{sk}_C, \mathsf{pk}) \in \mathsf{SK}_{\mathrm{C}} \times \mathsf{PK}$ to the client, and $(\mathsf{sk}_S, \mathsf{pk}) \in \mathsf{SK}_{\mathrm{S}} \times \mathsf{PK}$ to the server.

- Encryption algorithm $\mathcal{E}nc$. It takes as input a public key $\mathsf{pk} \in \mathsf{PK}$, a plaintext $m \in \mathsf{M}$ and returns a ciphertext $c \in \mathsf{C}$.
- Decryption protocol $\langle \mathcal{DC}_{\mathrm{C}} | \mathcal{DC}_{\mathrm{S}} \rangle$, run by the client and the server. Client's inputs to $\mathcal{DC}_{\mathrm{C}}$ are $c \in \mathsf{C}$, $\mathsf{sk}_C \in \mathsf{SK}_{\mathrm{C}}$, and $\mathsf{pk} \in \mathsf{PK}$. Server's inputs to $\mathcal{DC}_{\mathrm{S}}$ are $\mathsf{sk}_S \in \mathsf{SK}_{\mathrm{S}}$ and $\mathsf{pk} \in \mathsf{PK}$. The protocol returns either $m \in \mathsf{M}$ or the failure notice $\perp$ to the client. It returns the success notice $\top$ or the failure notice $\perp$ to the server.

First of all, we define how such a scheme *should* work. Both functional and security requirements can be covered by an *ideal functionality* in the Universal Composability framework (UC) [14]. This framework has been widely used to capture security of distributed protocols. It allows to capture security properties of cryptographic schemes in the ideal/real process paradigm. In the real world, adversary $\mathcal{A}$ interacts with the real protocol π by means of corrupting different parties, gaining control over their inputs and outputs. In the ideal world, ideal functionality $\mathcal{F}$ defines an ideal process for the protocol π, intuitively describing how it should work. Ideal functionality can be viewed as a trusted party that receives inputs from all the parties, performs computation on these inputs, and returns to each party its output. It also interacts with an ideal adversary $\mathcal{A}_S$, from which it may receive certain commands and to which it may output some values that are explicitly allowed to be leaked. In both cases, parties receive inputs and return outputs to the environment $\mathcal{Z}$. A protocol π securely implements $\mathcal{F}$, if for any real-world adversary $\mathcal{A}$ there exists an ideal adversary $\mathcal{A}_S$, such that no environment $\mathcal{Z}$ can distinguish between interaction with π running in parallel with $\mathcal{A}$ and $\mathcal{F}$ running in parallel with $\mathcal{A}_S$.

The ideal functionality for our encryption scheme with two-party privacy-preserving decryption is given in Fig. 1. For simplicity, we define it only for a single public key (and the corresponding shared decryption functionality). Also for simplicity, we consider only static corruptions of parties; the identities of the corrupt parties are told to the $\mathcal{F}$ by the ideal adversary $\mathcal{A}_S$ at the beginning of the execution. The functionality $\mathcal{F}$ is given for an arbitrary number of parties $P_1, P_2, \ldots$, all of which can submit encryption requests. The decryption can be done jointly by parties P_1 (in the role of the client) and P_2 (in the role of the server).

The key generation (that only happens in the beginning) and encryption in $\mathcal{F}$ are similar to existing ideal functionalities for public-key encryption, where the public key pk is chosen by $\mathcal{A}_S$. Encryption is performed similarly to [18], where the adversary comes up with the ciphertext c for the plaintext m without actually seeing m, intuitively ensuring that c does not leak anything about m. Only if the encryptor party P_i is corrupted, will $\mathcal{A}_S$ learn the message m. The obtained pairs (m, c) are stored in a table T, allowing the decryption queries to be answered.

The encryption query does not contain a public key as an input. In practice, the environment could provide an honest encryptor with a public key pk' different from pk generated by $\mathcal{F}$, and expect a ciphertext encrypted with pk'. To cover this case, we would need to allow $\mathcal{F}$ deliver pk' to $\mathcal{A}_S$. This would introduce more details into the definition of $\mathcal{F}$ and the security proofs without being essential for the discussion of the key points of $\mathcal{F}$ and its secure implementations. We assume that a party (in the environment) only makes encryption queries once it has learned the correct pk.

Decryption in $\mathcal{F}$ corresponds to finding the plaintext m from a ciphertext c that was created using the key pk. It can only be initiated by P_1 and P_2, with P_1 providing the ciphertext.

If the client is honest, then the decryption is less straightforward than in the UC public key encryption, because neither of the two existing approaches appear to be fully satisfactory. If $\mathcal{A}_S$ provides the descriptions of the algorithms Enc and Dec to $\mathcal{F}$, then it may be difficult to build simulators in the Random Oracle Model (ROM), because the simulator probably needs to know the state of the oracles during decryption, as well as program the oracles at least during encryption. But if we task $\mathcal{A}_S$ to come up with the ciphertext values during encryption, and to decrypt unknown ciphertexts during decryption, then this goes against our intuition of "blinded decryption", where $\mathcal{A}_S$ should not learn the ciphertexts that an honest client wants to decrypt. We resolve this by making $\mathcal{A}_S$ run the encryption (programming the random oracles as necessary), and sending to $\mathcal{F}$ an *updated* description of Dec at each decryption. In this way, $\mathcal{A}_S$ does not learn the ciphertext that is being decrypted, but the current state of the random oracles can be included in the description of Dec and affect the result of decryption. Using this approach (and similarly to UC public key encryption), we need to be careful that the same ciphertext c will not decrypt to different values, which would violate correctness. For that reason, $\mathcal{F}$ adds to T all (m, c) pairs obtained during the decryption as well. If there will be two records (m, c) and (m', c) for $m \neq m'$, the decryption fails. In addition, since $\mathcal{A}_S$ cannot verify the correctness of the input ciphertext, $\mathcal{A}_S$ should first of all send to $\mathcal{F}$ a description of a function Verify which verifies the ciphertext, returning either true or false. $\mathcal{A}_S$ needs to learn this single bit of information, as in a real protocol the decryption would not start in this case.

If the client is corrupted (but the server is honest), then privacy of the ciphertext does not have to be protected; ciphertext can be given to $\mathcal{A}_S$. But we are now modeling threshold decryption, where a corrupted party is expected to learn the plaintext; we are not aware of any similar models in the literature. If a corrupted party is able to learn the plaintext m, then a simulator must also be able to learn it, in order to simulate that party's view. But if the ciphertext c was created using the encryption functionality of $\mathcal{F}$, then c is actually independent of m, and the only location containing m (beside the environment, which the simulator cannot depend on) is in the table T of $\mathcal{F}$. Hence we have to give $\mathcal{A}_S$ an ability, however minimal, to read that table. In the case of threshold decryption *without* blinding, it would be sufficient if $\mathcal{A}_S$ could query for the row (m, c), where c is the ciphertext that is being decrypted. For the blinded decryption, the adversary playing the corrupted client can actually start the decryption protocol for an arbitrary ciphertext, hence the row(s) queried by $\mathcal{A}_S$

Key generation
On message (KeyGen, sid) from both P_1 and P_2:

1) Send (KeyGen, sid) to $\mathcal{A}_S$ and wait for $(\mathsf{Key}, sid, \mathsf{pk})$ from $\mathcal{A}_S$.
2) If $\mathsf{pk} \neq \perp$, create an empty table T for storing encrypted messages, and set decryption counter $ctr_{\mathsf{dec}} \leftarrow 0$.
3) For $i \in \{1, 2\}$, if the party P_i is corrupted, wait for $(\mathsf{Output}, i, y_i)$ from $\mathcal{A}_S$. Otherwise, take $y_i = \mathsf{pk}$.
4) Output (Key, sid, y_1) to P_1 and (Key, sid, y_2) to P_2.

Encryption
On message $(\mathsf{Encrypt}, sid, m)$ from some party P_i:

1) If P_i is corrupted, send $(\mathsf{Encrypt}, sid, i, m)$ to $\mathcal{A}_S$. Otherwise, send $(\mathsf{Encrypt}, sid)$ to $\mathcal{A}_S$.
2) Wait for $(\mathsf{Encrypt}, sid, c)$ from $\mathcal{A}_S$ (the adversary chooses the ciphertext for the given plaintext).
3) Store the pair (m, c) in table T.
4) If the party P_i is corrupted, wait for (Output, i, y) from $\mathcal{A}_S$. Otherwise, take $y = c$.
5) Send $(\mathsf{Encrypted}, sid, y)$ to P_i.

Decryption procedure $Decrypt(\mathsf{Dec}, c)$ for a function $\mathsf{Dec} : \mathsf{C} \rightarrow \mathsf{M}$ and a ciphertext $c \in \mathsf{C}$:

- If there exists a pair (m, c) in T: (for correctness we need that c would decrypt to m)
  - If m is not unique (there are two different m for the same c), take $m' = \perp$.
  - If m is unique, take $m' = m$.
- Otherwise, compute $m' := \mathsf{Dec}(c)$, running it only a polynomial number of steps (the polynomial is a parameter of $\mathcal{F}$). Add (m', c) to T.
- Return m'.

Decryption (honest client)
On message $(\mathsf{Decrypt}, sid, c)$ from party P_1 and $(\mathsf{Decrypt}, sid)$ from party P_2:

1) Send $(\mathsf{Decrypt\text{-}init}, sid)$ to $\mathcal{A}_S$.
2) Upon receiving $(\mathsf{Decrypt\text{-}init}, sid, \mathsf{Verify})$ from $\mathcal{A}_S$, where $\mathsf{Verify} : \mathsf{C} \rightarrow \{\mathsf{true}, \mathsf{false}\}$, compute $b \leftarrow \mathsf{Verify}(c)$.
   - If $b = \mathsf{true}$, send $(\mathsf{Decrypt\text{-}good\text{-}c}, sid)$ to $\mathcal{A}_S$.
   - If $b = \mathsf{false}$ or $b = \perp$, send $(\mathsf{Decrypt\text{-}bad\text{-}c}, sid)$ to $\mathcal{A}_S$. Proceed to the point 4) with $y_1 = y_2 = \perp$.
3) The final output y_1 for P_1 depends on whether the server is corrupt.
   If the server is *honest*, the decryption can only be delayed:
   - Upon receiving $(\mathsf{Decrypt\text{-}complete}, sid, \mathsf{Dec})$ from $\mathcal{A}_S$, take $y_1 \leftarrow Decrypt(\mathsf{Dec}, c)$, and $y_2 \leftarrow \top$.

   If the server is *corrupted*, $\mathcal{A}_S$ tells whether decryption succeeded.
   - Upon receiving $(\mathsf{Decrypt\text{-}complete}, sid, \mathsf{Dec})$ from $\mathcal{A}_S$, take $y_1 \leftarrow Decrypt(\mathsf{Dec}, c)$.
   - Upon receiving $(\mathsf{Decrypt\text{-}fail}, sid)$ from $\mathcal{A}_S$, take $y_1 \leftarrow \perp$.
4) If the server is corrupted, wait for $(\mathsf{Output}, 2, y_2')$ from $\mathcal{A}_S$ and take $y_2 \leftarrow y_2'$.
5) Send $(\mathsf{Decrypted}, sid, y_1)$ to P_1 and $(\mathsf{Decrypted}, sid, y_2)$ to P_2.

Decryption (corrupted client)
On message $(\mathsf{Decrypt}, sid, c)$ from party P_1 and $(\mathsf{Decrypt}, sid)$ from party P_2:

1) Set $ctr_{\mathsf{dec}} \leftarrow ctr_{\mathsf{dec}} + 1$ and send $(\mathsf{Decrypt\text{-}init}, sid, c)$ to $\mathcal{A}_S$.
2) $\mathcal{A}_S$ tells whether decryption succeeded:
   - Upon receiving $(\mathsf{Decrypt\text{-}complete}, sid)$ from $\mathcal{A}_S$, take $y_2 \leftarrow \top$.
   - Upon receiving $(\mathsf{Decrypt\text{-}fail}, sid)$ from $\mathcal{A}_S$, take $y_2 \leftarrow \perp$.
3) Wait for $(\mathsf{Output}, 1, y_1)$ from $\mathcal{A}_S$.
4) Send $(\mathsf{Decrypted}, sid, y_1)$ to P_1 and $(\mathsf{Decrypted}, sid, y_2)$ to P_2.

At any point of time (corrupted client). On message $(\mathsf{Decrypt\text{-}msg}, sid, \mathsf{Dec}, c')$ from $\mathcal{A}_S$:

1) Compute $m = Decrypt(\mathsf{Dec}, c')$.
2) Set $ctr_{\mathsf{dec}} \leftarrow ctr_{\mathsf{dec}} - 1$. If $ctr_{\mathsf{dec}} < 0$, then stop.
3) Send $(\mathsf{Decrypted}, sid, m)$ to $\mathcal{A}_S$.

Fig. 1. Ideal functionality $\mathcal{F}$

.

can no longer be restricted like that. We should allow $\mathcal{A}_S$ to query for the row (m, c) for a ciphertext c its own choice. Moreover, we allow to make this query *after* the decryption session has ended, since the corrupted client may potentially be able to undo his own blinding in multiple ways after getting response from the server, thus being able to choose between decrypting several different ciphertexts later. Nevertheless, the adversary is only allowed to decrypt *at most one message per decryption query* coming from the environment.

We deliberately have omitted the case where both the server and the client are corrupted, as we do not aim to achieve any security guarantees in this case. The adversary would get full control over the key generation, encryption and decryption, and would be given all messages and ciphertexts that any party (including the honest ones) receives as an input.

Next, we discuss how the ideal functionality that we defined sidesteps the problems we identified with the previous security definitions. As a counterexample for P-IND-RCCA, we proposed a scheme where the adversary may obtain the secret key share of an honest party if it gets the secret key share of a corrupted party. If such a protocol were run in the UC model with either the client or the server corrupted, the adversary would be able to reconstruct the private key sk and decrypt the ciphertexts that have been encrypted with pk. The environment could distinguish the real protocol from $\mathcal{F}$ by decrypting (without involving $\mathcal{F}$) an encryption of a message m generated by $\mathcal{F}$, i.e. without letting $\mathcal{F}$ tell m to $\mathcal{A}_S$, getting a plaintext that does not depend on m.

Compared to the definition of Green, our execution of blinded decryption in $\mathcal{F}$ is not atomic, and the adversary has full control over the corrupted client, and thus could set up $b_{evil} = 1$. The environment would get sk and hence could distinguish the real protocol from $\mathcal{F}$ by decrypting (without involving $\mathcal{F}$) an encryption of a message m generated by $\mathcal{F}$, getting a plaintext that does not depend on m. Non-atomic execution of blinded decryption allows a corrupted client can decrypt a ciphertext of his own choice, which would not be possible using atomic execution. We explicitly let the corrupted client access a selected entry from the table T, which can be viewed as access to a standard decryption oracle. If the client is not corrupted, then the access to the decryption oracle is limited and controlled by the environment.

V. Secure implementation

While we have criticized the *definitions* of Green [23] and Blazy et al. [24], we may wonder whether their implementations, even though they are for different functionalities, are adaptable into a secure implementation of $\mathcal{F}$. We can immediately answer that question in negative for Green: his outsourced blinded decryption is not an instance of threshold decryption, and does not support the user independently finding out whether a ciphertext is valid. The latter is necessary for chosen-ciphertext security in threshold decryption schemes.

Blazy et al. [24] discuss whether their construction could be realizable in universally composable manner, and answer that question in negative. But that answer stems from the inability to simulate the adversary guessing the password of an honest party. It is possible that if the roles of *user* and *token* in their protocol were combined into one, then their construction would provide a secure implementation of $\mathcal{F}$.

However, the blind decryption step of Blazy et al. [24] uses Groth-Sahai proofs [30] to prove that a blinded ciphertext has been constructed from a ciphertext that was accompanied by proofs of validity. This means that their construction is complex, and heavily based on bilinear pairings, leading to both relatively heavy computations and to large message sizes.

We securely implement $\mathcal{F}$ under common cryptographic assumptions: hardness of discrete logarithm (DL), one-more CDH, and the existence of additively homomorphic encryption systems, for which the proofs of equality of plaintexts can be given. In this paper, we will instantiate the latter with Paillier encryption, which is IND-CPA (indistinguishability against chosen-plaintext attacks) secure under the Decisional Composite Residuosity Assumption (DCRA) [31]. Remarkably, our construction DVPS does not require bilinear pairings. Compared to [23], [24], a potential theoretical shortcoming of our construction is its use of the Random Oracle Model (ROM), but considering its prevalence in constructions employed in practice [32], [33], we do not see it as a weakness.

Our construction DVPS builds on top of the IND-CCA secure TDH1 cryptosystem of Gennaro and Shoup [2], adding to it ciphertext blinding and unblinding operations that are used when the client queries the server during decryption protocol. TDH1 builds upon the ElGamal key encapsulation mechanism (KEM), adding to its ciphertexts c the non-interactive zero-knowledge (NIZK) proofs π that someone (e.g. the entity encrypting the plaintext m) knows the randomness r used for the encryption. In DVPS, both c and the proof have to be blinded. To blind the proof, we need *malleable* NIZK proofs [34], but the known constructions are based on bilinear pairings. Fortunately, we do not need malleable NIZK in its full generality (as used by Blazy et al. [24]); we only need the server to be convinced by a proof that the client malleated. There exist malleable *designated verifier proofs* without pairings [35] that will be used in our construction.

Our ciphertexts will thus contain a proofs of knowledge of the randomness r, designated to be verified by the server. The client also needs to be convinced that r is known; we use "usual" NIZK proofs for that. Our ciphertexts also contain a third kind of proof, convincing the client that an honest server is going to accept the designated verifier proof. In [36], the DFN proofs of [35] have been found vulnerable against selective failure attacks that allow to find the secret of the verifier bit by bit. This can be mitigated by letting the server stop after a certain number of failed proofs and require a fresh key generation. The ability of the client to verify whether the server will accept the proof is needed to avoid attacks where an external encrypting party produces bad ciphertexts that will be rejected by the server and cause denial of service for an honest client. The details of this verification are given in App. B.

As next, we describe the cryptographic building blocks used

in our construction. The construction itself follows in Sec. V-B.

A. Preliminaries

We write $x_1, \ldots, x_n \leftarrow\!\$\ X$ to denote that the values $x_1, \ldots, x_n$ are uniformly, independently sampled from a set X. We also write $x \leftarrow\!\$\ \mathsf{X}(\ldots)$ to denote that x is returned by a stochastic computation X. We let $\mathbf{I}(n)$ denote the set $\{0, 1, \ldots, 2^n - 1\}$.

1) Hardness assumptions: Let $\mathbb{G}$ be a cyclic group of size p, with generator g. The *discrete logarithm problem* is to find $n \in \mathbb{Z}_p$, such that $g^n = h$, for a value $h \leftarrow\!\$\ \mathbb{G}$. The *decisional Diffie-Hellman (DDH) problem* is to distinguish tuples of the form (g, g^x, g^y, g^{xy}) (called *Diffie-Hellman tuples*) from the tuples of the form (g, g^x, g^y, g^z) for $x, y, z \leftarrow\!\$\ \mathbb{Z}_p$. The *computational Diffie-Hellman (CDH) problem* is, given (g, g^x, g^y) for $x \leftarrow\!\$\ \mathbb{Z}_p$ and $y \leftarrow\!\$\ \mathbb{Z}_p$, come up with g^{xy}.

The *one-more (static) computational Diffie-Hellman (CDH) problem* is, given (g, g^x) for $x \leftarrow\!\$\ \mathbb{Z}_p$, access to the oracle $(\cdot)^x$, and $h_0, \ldots, h_n \leftarrow\!\$\ \mathbb{G}$, come up with $y_0, \ldots, y_n$ satisfying $y_i = h_i^x$ while querying $(\cdot)^x$ at most n times. If $n = 0$, then we have the usual CDH problem. A problem is *hard* if all the efficient algorithms have at most negligible advantage (over a trivial algorithm) of solving it.

2) ElGamal KEM and encryption scheme: Key Encapsulation Mechanism consists of the key generation algorithm that outputs a pair of private and public key $(\mathsf{sk}, \mathsf{pk})$; encapsulation algorithm that takes as input a public key and outputs a shared secret ss and ciphertext c; decapsulation algorithm that on input of ciphertext c and private key sk outputs a shared secret ss. Fig. 2 presents ElGamal KEM.

Key Generation:	Encapsulation:	Decapsulation:
1 : $\mathsf{sk} \leftarrow\!\$\ \mathbb{Z}_p$	1 : $r \leftarrow\!\$\ \mathbb{Z}_p$	1 : $ss = c^{\mathsf{sk}}$
2 : $\mathsf{pk} = g^{\mathsf{sk}}$	2 : $ss = \mathsf{pk}^r$	2 : Return ss
3 : Return $(\mathsf{sk}, \mathsf{pk})$	3 : $c = g^r$	
	4 : Return (ss, c)	

Fig. 2. ElGamal KEM

IND-CPA security of ElGamal KEM is equivalent to the hardness of DDH in the used group $\mathbb{G}$. In *hashed ElGamal* KEM, the shared secret is $H'(\mathsf{pk}^r)$ for some hash function H' that we model as a random oracle; its security is equivalent to the hardness of CDH.

A KEM can be turned into a public-key encryption scheme by combining it with a data encapsulation mechanism (DEM). In random oracle model, we may imitate the SKE2 DEM [37] and define the encryption of the message m with the key k as $\mathsf{SE}(k, m) := (H'(k) \oplus m, H''(k, m))$ for some hash functions (random oracles) H' and H''. The respective decryption function $\mathsf{SD}(k, (c_1, c_2))$ computes $m \leftarrow H'(k) \oplus c_1$, checks that $H''(k, m) = c_2$, and outputs m.

3) Random oracles: In constructions and proofs in the *Random Oracle Model* (ROM) [38], all the parties are assumed to have an access to one or several random functions H, where the value of the function at each point is a random variable uniformly distributed over a fixed set, and the values at different points are independent of each other. These random functions may have different codomains, e.g. the set of bitstrings of a certain length, some group $\mathbb{G}$, etc., depending on the needs of the construction. The domain of a random oracle does not have to be fixed; anything encodable as a bitstring may be an input to it. In proofs, the simulator is in control of the output values of random oracles, defining them as it sees fit, as long as the distribution stays the same.

4) (Non-interactive) zero-knowledge proofs: A Σ-*protocol* for a binary relation R is a three-move protocol between two parties — prover and verifier — both knowing a value x, where the prover tries to convince the verifier that it knows some value w, such that $R(x, w)$ holds. The first message α is sent by the prover, followed by the verifier generating a fresh, independent random value β and sending it as the second message. After the prover has sent the third message γ, the verifier runs a check on $(x, \alpha, \beta, \gamma)$ and either accepts or rejects. A Σ-protocol must have *special honest-verifier zero-knowledge (ZK)*: given (x, β), one should be able to generate (α, γ) so, that the verifier cannot distinguish them from the real protocol runs. A Σ-protocol must also be *specially sound*: given $(x, \alpha, \beta_1, \beta_2, \gamma_1, \gamma_2)$ with $\beta_1 \neq \beta_2$, such that the verifier accepts both $(x, \alpha, \beta_1, \gamma_1)$ and $(x, \alpha, \beta_2, \gamma_2)$, it must be possible to find w.

A Σ-protocol for R is an (interactive) zero-knowledge proof, assuming the verifier does not deviate from the protocol. Random oracles can be used to turn Σ-protocols to non-interactive ZK (NIZK) proofs using the *Fiat-Shamir (FS) transform* [39]: instead of receiving β from the verifier, prover computes it as $\beta \leftarrow H(x, \alpha, ctx)$, where ctx describes the context in which the prover creates this proof. To verify the proof, the verifier recomputes β. In our construction, in some proofs it is easier to assume that the part of the proof is β, not α. The verifier then computes α that would satisfy the proof from x and ctx, and checks whether $\beta = H(x, \alpha, ctx)$.

A *Designated-Verifier* NIZK (DVNIZK) proof is a NIZK proof that can convince only a single verifier. Such proofs may be cheaper to use than publicly verifiable proofs, and they may have some additional properties. Damgård et al. [35] have introduced a method ("DFN proofs") that applies to such Σ-protocols where γ is computed as a linear function of β, turning them into DVNIZK proofs. In their method, the verifier generates a public-private key pair $(\mathsf{ek}, \mathsf{vk})$ for additively homomorphic encryption. It also selects a random β and computes $B \leftarrow\!\$\ \mathcal{E}_{\mathsf{ek}}(\beta)$. Public key ek and ciphertext B are given to the prover. The proof is (α, Γ), where α is the same as in the original Σ-protocol and Γ is an encryption of γ, computed using B and the homomorphic properties of the encryption scheme. Verifier can decrypt Γ and perform the verification as in the original Σ-protocol. It turns out the the DFN proofs have the malleability properties that we need.

For the publicly verifiable proofs of knowledge of the exponent r in an ElGamal ciphertext, we use Σ-protocols that are made non-interactive using the FS transform. As the construction of TDH1 [2] already observed, a simple

Schnorr proof of knowing r [33] seems to be insufficient for this purpose (unless we introduce an additional extractability assumption), because the simulator will not be able to decrypt certain ciphertexts [40]. A more complex proof is necessary. They start from a *DDH proof* $\mathsf{DHP}^H[r \mid^{u,v}_{g,h} | ctx]$ [41] — a NIZK proof that $\log_g u = \log_h v$ (i.e. it is not a proof *of knowledge*), given in *context* ctx, where $r \in \mathbb{Z}_p$ is that discrete logarithm and H is a hash function, modeled as a random oracle. Denote the checking procedure by $\mathsf{ChP}^H[\pi \mid^{u,v}_{g,h} | ctx]$. Both proof and verification procedure are given in Fig. 3.

$\mathsf{DHP}^H[r \mid^{u,v}_{g,h} | ctx]$:

1 : $s \leftarrow\$ \mathbb{Z}_p$
2 : $\alpha \leftarrow g^s, \quad \alpha' \leftarrow h^s$
3 : $\beta \leftarrow H(g, h, u, v, \alpha, \alpha', ctx) \in \mathbb{Z}_p$
4 : $\gamma \leftarrow s + r \cdot \beta$
5 : return $\pi \leftarrow (\beta, \gamma)$

$\mathsf{ChP}^H[\pi \mid^{u,v}_{g,h} | ctx]$:

1 : $\alpha \leftarrow g^\gamma / u^\beta$
2 : $\alpha' \leftarrow h^\gamma / v^\beta$
3 : **assert** $\beta = H(g, h, u, v, \alpha, \alpha', ctx) \in \mathbb{Z}_p$

Fig. 3. NIZK proof that $\log_g u = \log_h v$

From DDH proofs we get *simulatable proofs of knowledge of exponent* $\mathsf{KnE}^{H,\tilde{H}}_{\mathsf{d}}[r \mid^u_g | ctx]$, where $\mathsf{d} \in \{1, -1\}$ is needed for exponent extraction an in described in Sec. VI. These prove that someone knows the value $r = \log_g u$. The construction makes use of two hash functions, both modeled as random oracles, where H returns elements of $\mathbb{Z}_p$ and $\tilde{H}$ returns elements of $\mathbb{G}$. The checking procedure $\mathsf{ChE}^{H,\tilde{H}}_{\mathsf{d}}[\pi \mid^u_g | ctx]$ checks the underlying DDH proof. Both proof and verification procedure are given in Fig. 4.

$\mathsf{KnE}^{H,\tilde{H}}_{\mathsf{d}}[r \mid^u_g | ctx]$:

1 : $h \leftarrow \tilde{H}(g, u, ctx) \in \mathbb{G}$
2 : $v \leftarrow h^{r^{\mathsf{d}}}$
3 : if $\mathsf{d} = 1$, then $\pi \leftarrow \mathsf{DHP}^H[r \mid^{u,v}_{g,h} | ctx]$
4 : if $\mathsf{d} = -1$, then $\pi \leftarrow \mathsf{DHP}^H[r \mid^{u,h}_{g,v} | ctx]$
5 : return $\pi' \leftarrow (\pi, v)$

$\mathsf{ChE}^{H,\tilde{H}}_{\mathsf{d}}[\pi' \mid^u_g | ctx]$:

1 : $h \leftarrow \tilde{H}(g, u, ctx) \in \mathbb{G}$
2 : $(\pi, v) \leftarrow \pi'$
3 : if $\mathsf{d} = 1$, then **assert** $\mathsf{ChP}^H[\pi \mid^{u,v}_{g,h} | ctx]$
4 : if $\mathsf{d} = -1$, then **assert** $\mathsf{ChP}^H[\pi \mid^{u,h}_{g,v} | ctx]$

Fig. 4. Proof of knowledge of exponent in a group $\mathbb{G}$

B. Our construction

Our construction adds to the TDH1 scheme (with differently shared private key) the malleable DVNIZK proofs, and the proofs of these proofs being accepted by the server. TDH1 sets up a trapdoor with the proof of knowledge of the exponent r. Similarly, we need a number of trapdoors in the added proofs in order to be able to simulate a corrupted party.

Public parameters of DVPS contain the cyclic group $\mathbb{G}$ of size p with generator g. They also fix a homomorphic encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ for DFN proofs, where $\mathcal{K}$ generates a private and public key pair, $\mathcal{E}$ encrypts, and $\mathcal{D}$ decrypts. In this paper, that scheme will be the Paillier encryption scheme, working with moduli N of bit-length $\nu \in \mathbb{N}$. The public parameters moreover contain the statistical *soundness* and *privacy* parameters $\rho, \kappa \in \mathbb{N}$ for the computations over the integers (in the hidden-order group $\mathbb{Z}^*_N$). Typically, ρ and κ are between 80 and 256, and it is reasonable to assume that $\rho \leq \kappa \leq d$, where d is the bit-length of p. We need ν to be several times larger than ρ, κ, and d; this will be satisfied by the Paillier encryption (where N is a RSA modulus) and natural instantiations of $\mathbb{G}$ (as elliptic curve groups). Let $\mathbf{Cn}$ be the set of possible *random coins* used during encryption. Finally, the public parameters contain the definitions of several random functions, modeled as random oracles, introduced below. All the sub-routines that are used in our construction are listed in the Table I to ease the understanding of the protocol.

$\mathsf{SE}(k, m)$ and $\mathsf{SD}(k, c)$	encryption and decryption procedures of DEM for a key k, a plaintext m and a ciphertext c
$\mathsf{DHP}^H[r \mid^{u,v}_{g,h} \mid ctx]$ and $\mathsf{ChP}^H[\pi \mid^{u,v}_{g,h} \mid ctx]$	proof that $\log_g u = \log_h v$, given in *context* ctx, where $r \in \mathbb{Z}_p$ is that discrete logarithm
$\mathsf{KnE}^{H,\tilde{H}}_{\mathsf{d}}[r \mid^u_g \mid ctx]$ and $\mathsf{ChE}^{H,\tilde{H}}_{\mathsf{d}}[\pi \mid^u_g \mid ctx]$	proof generation and verification for the proof of knowledge of exponent $r = \log_g(u)$ in context ctx
$\mathsf{DVP}(r, r', \mathbf{r} \mid^{\alpha,\Gamma}_{g,u} \mid B)$ and $\mathsf{DVC}(\pi \mid^{\alpha,\Gamma}_{g,u} \mid B)$	proof generation and verification for the proofs of validity of DVNIZK proof (α, Γ) that uses randomness r, r' and $\mathbf{r}$ in the context of encrypted challenge B
$(\mathcal{K}, \mathcal{E}, \mathcal{D})$	Paillier key generation, encryption and decryption

TABLE I
SUB-ROUTINES USED IN OUR CONSTRUCTION

Key generation of DVPS is given in Fig. 5. During the key generation, the client and the server respectively select private key shares $\mathsf{sk}_1 \leftarrow\$ \mathbb{Z}_p$ and $\mathsf{sk}_2 \leftarrow\$ \mathbb{Z}_p$, compute public key shares $\mathsf{pk}_i = g^{\mathsf{sk}_i}$ (line 1), and exchange the latter values using hash commitments (lines 2–4). Beside pk_i, the client and server also exchange the proofs $\mathsf{KnE}^{H_0,\tilde{H}_0}_{-1}[\mathsf{sk}_i \mid^{\mathsf{pk}_i}_g]$ that they know their respective private keys (lines 5–7), this prevents a malicious party from choosing their public key share based on the share of the honest party without knowing corresponding exponent. They will verify the proofs (lines 8–9), and define combined public key $\mathsf{pk} = \mathsf{pk}^{\mathsf{sk}_i}_{3-i}$ (line 10), i.e. the private key is *multiplicatively*, not additively shared.

Having set up pk, the parties continue with setting up the public and private values for DVNIZK proofs. DVNIZK proofs are needed for the encrypting party to prove that they know the randomness used to generate ElGamal ciphertext. Moreover, chosen DVNIZK proofs allow client to blind the proof before sending it in the decryption query to the server

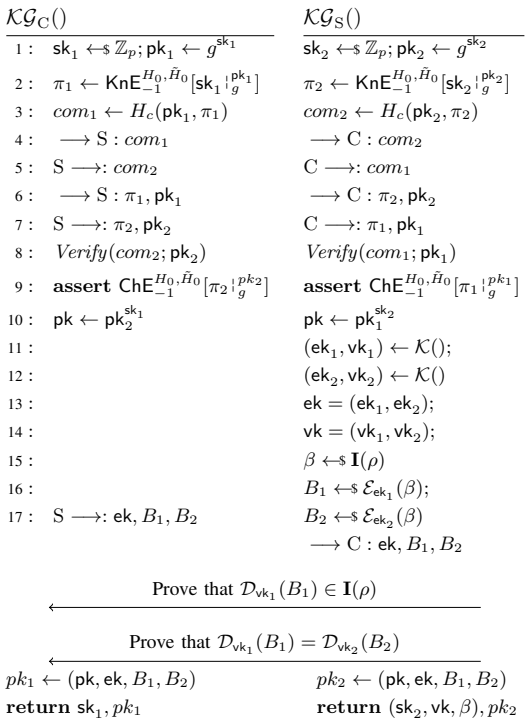


Fig. 5. Key generation for client and server in our construction

(who is acting as a designated verifier). The server generates two key pairs $(\mathsf{ek}_1, \mathsf{vk}_1)$ and $(\mathsf{ek}_2, \mathsf{vk}_2)$ for the homomorphic encryption scheme, where ek_i is the public key and vk_i the private key (line 11). The set-up also consists of public ciphertexts $B_1 \leftarrow\$ \mathcal{E}_{\mathsf{ek}_1}(\beta)$ and $B_2 \leftarrow\$ \mathcal{E}_{\mathsf{ek}_2}(\beta)$ (line 12), where $\beta \leftarrow\$ \mathbf{I}(\rho)$ is generated and kept secret by the server. The necessity of generating two instances comes from the security and we provide detailed explanation in Section VI. Server sends $\mathsf{ek}= (\mathsf{ek}_1, \mathsf{ek}_2)$ and B_1 and B_2 to the client. Finally, the set-up for DFN proofs consists of the server proving that B_1 indeed encrypts a value at most ρ bits long and proving that B_1 and B_2 encrypt the same value.

The public key is $(\mathsf{pk}, (\mathsf{ek}_1, \mathsf{ek}_2), B_1, B_2)$, allowing an encryptor to create the ElGamal ciphertext with the public key pk, and the two DVNIZK proofs of knowledge of the random exponent of the ciphertext, using (ek_1, B_1) and (ek_2, B_2) (the NIZK proof of knowledge of the exponent, as well as the proofs of the DVNIZK proofs being correct, are standard FS transforms of Σ-protocols). Client keeps sk_1 as his keyshare. Server keeps $(\mathsf{sk}_2, (\mathsf{vk}_1, \mathsf{vk}_2), \beta)$ as his keyshare. The both also keep $\mathsf{pk}_1, \mathsf{pk}_2$.

Encryption and **decryption** are given in Fig. 6. As the operations with B_1 and B_2 are mostly similar, we abbreviate the write-up by introducing the convention that the parameter ι ranges over $\{1, 2\}$ and the operations of defining values X_1 and X_2 are given only once by stating how X_ι is computed.

The encryption of the message m starts by computing its "standard" ElGamal ciphertext (u, c_2) (lines 1–3). In line 4–7, we compute the two DVNIZK proofs of the encryptor knowing r. Those proofs can only be verified by the server during the decryption process. The value α_1 is the first message of both proofs; it is computed as in a Schnorr proof of knowledge, raising the generator g to a random power r_1. The second message is the server's secret β, and the third message Γ_ι is an encryption of $\gamma = r_1 + \beta r$.

In line 8, we compute the (usual) NIZK proof that the encryptor knows r. The same proof, with the same trapdoor is present in TDH1. This proof can be verified by the client. Finally, in line 9 of $\mathcal{E}nc$ we compute the proofs π_ι ($\iota \in \{1, 2\}$) that an honest server will accept the designated verifier proofs (α_1, Γ_ι), while verifying them in the manner described in Sec. V-A. This proof is given using standard cryptographic techniques; we present it in App. B. No trapdoor is embedded in π_ι. The proof is required for the client not to engage in decryption of an incorrect ciphertext. If the proof does not verify on the client side, the client does not initiate protocol with the server. The final ciphertext is $[c_1 = (u, \alpha_1, \Gamma_1, \Gamma_2, \pi, \pi_1, \pi_2), c_2]$.

One has to be careful with the moduli: while the exponents of g are naturally taken modulo p, the computations under encryption are done modulo the RSA/Paillier modulus N. As N is much larger than p, we can just make sure that certain computations do not overflow. In particular, we want $\gamma = r_1 + \beta r$ be less than N. We also want r_1 to hide βr. Obtaining the perfect security may be hard, but we can get statistical security. The value βr is $\rho + d$ bits long. Hence we let r_1 to be $\rho + d + \kappa$ bits long (which must still be less than the size ν of N). This is similar to homomorphic commitments to integers [42], [43].

During the **decryption**, the client first verifies two of three proofs in lines 2–3. It will then construct the blinded encapsulation u' (line 6) by raising u to a random exponent z. Hence u' is independent of u. In lines 7–8, the client will then blind the proof (α_1, Γ_ι) of knowledge of $r = \log_g u$ to get the proof $(\alpha'_1, \Gamma'_\iota)$ of knowledge of $rz = \log_g u'$. $(\alpha_1^z, \Gamma_\iota^z)$ is such a proof, as long as the plaintext $z\gamma = \mathcal{D}_{\mathsf{vk}_\iota}(\Gamma_\iota^z)$ is not reduced modulo N. That reduction indeed does not happen, because $z\gamma$ is at most $\rho + 2d + \kappa \leq \nu$ bits long.

However, $z\gamma$ may leak something about z to the server, once the latter has decrypted Γ_ι^z and obtained it. Indeed, z is a factor of $z\gamma$. Hence we will further additively mask $(\alpha_1^z, \Gamma_\iota^z)$, using z'. We get statistical security if we let z' to be at last κ bits longer than $z\gamma$.

The client proceeds with generating proof π' that they know their private key in the context of blinded ciphertext that is being decrypted (line 9). This proof authenticates client to the server, demonstrating that they know the private key share. The clients sends u', π' and the malleated DVNIZK proofs to the server (line 10), who verifies received proofs (line 2–5 in $\mathcal{DC}_{\mathrm{S}}$). The rest of the protocol is straightforward, computing $\mathsf{pk}^r = (((u')^{\mathsf{sk}_2})^{\mathsf{sk}_1})^{1/z}$ from u'. The server does

$\mathcal{E}nc(m, \mathsf{pk}, \mathsf{ek}_1, \mathsf{ek}_2, B_1, B_2)$

1 : $r \leftarrow\!\!\$\ \mathbb{Z}_p$
2 : $u \leftarrow g^r$
3 : $c_2 \leftarrow \mathsf{SE}(\mathsf{pk}^r, m)$
4 : $r_1 \leftarrow\!\!\$\ \mathbf{I}(\rho + d + \kappa)$
5 : $\mathbf{r}_\iota \leftarrow\!\!\$\ \mathbf{Cn}$
6 : $\alpha_1 \leftarrow g^{r_1 \bmod p}$
7 : $\Gamma_\iota \leftarrow \mathcal{E}_{\mathsf{ek}_\iota}(r_1; \mathbf{r}_\iota) \cdot B_\iota^r$
8 : $\pi \leftarrow \mathsf{KnE}_1^{H_1, \tilde{H}_1}[r \restriction_g^u | \alpha_1, \Gamma_1, \Gamma_2]$
9 : $\pi_\iota \leftarrow \mathsf{DVP}(r, r_1, \mathbf{r}_\iota \restriction_{g,u}^{\alpha_1, \Gamma_\iota} | B_\iota)$
10 : $c_1 \leftarrow (u, \alpha_1, \Gamma_1, \Gamma_2, \pi, \pi_1, \pi_2)$
11 : **return** c_1, c_2

$\iota \in \{1, 2\}$

$\mathcal{DC}_{\mathrm{C}}(c_1, c_2, \mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}, \mathsf{ek}_1, \mathsf{ek}_2, B_1, B_2)$

1 : $(u, \alpha_1, \Gamma_1, \Gamma_2, \pi, \pi_1, \pi_2) \leftarrow c_1$
2 : **assert** $\mathsf{ChE}_1^{H_1, \tilde{H}_1}[\pi \restriction_g^u | \alpha_1, \Gamma_1, \Gamma_2]$
3 : **assert** $\mathsf{DVC}(\pi_\iota \restriction_{g,u}^{\alpha_1, \Gamma_\iota} | B_\iota)$
4 : $z \leftarrow\!\!\$\ \mathbb{Z}_p,\ z' \leftarrow\!\!\$\ \mathbf{I}(\rho + 2d + 2\kappa)$
5 : $\mathbf{r}'_\iota \leftarrow\!\!\$\ \mathbf{Cn}$
6 : $u' \leftarrow u^z$
7 : $\alpha'_1 \leftarrow \alpha_1^z \cdot g^{z' \bmod p}$
8 : $\Gamma'_\iota \leftarrow \Gamma_\iota^z \cdot \mathcal{E}_{\mathsf{ek}_\iota}(z'; \mathbf{r}'_\iota)$
9 : $\pi' \leftarrow\!\!\$\ \mathsf{KnE}_{-1}^{H_2, \tilde{H}_2}[\mathsf{sk}_1 \restriction_g^{\mathsf{pk}_1} | u', \alpha'_1, \Gamma'_1, \Gamma'_2]$
10 : $\longrightarrow \mathrm{S} : u', \alpha'_1, \Gamma'_1, \Gamma'_2, \pi'$
11 : $\mathrm{S} \longrightarrow : w, \pi''$
12 : **assert** $\mathsf{ChP}^{H_3}[\pi'' \restriction_{\mathsf{pk}_1, u'}^{\mathsf{pk}, w}]$
13 : **return** $\mathsf{SD}(w^{\mathsf{sk}_1 / z}, c_2)$

$\mathcal{DC}_{\mathrm{S}}(\mathsf{sk}_2, \mathsf{pk}_1, \mathsf{pk}, \mathsf{vk}_1, \mathsf{vk}_2, \beta)$

1 : $\mathrm{C} \longrightarrow : u', \alpha'_1, \Gamma'_1, \Gamma'_2, \pi'$
2 : $\gamma' \leftarrow \mathcal{D}_{\mathsf{vk}_1}(\Gamma'_1)$
3 : **assert** $\gamma' = \mathcal{D}_{\mathsf{vk}_2}(\Gamma'_2)$
4 : **assert** $g^{\gamma'} = \alpha'_1 \cdot (u')^\beta$
5 : **assert** $\mathsf{ChE}_{-1}^{H_2, \tilde{H}_2}[\pi' \restriction_g^{\mathsf{pk}_1} | u', \alpha'_1, \Gamma'_1, \Gamma'_2]$
6 : $w \leftarrow (u')^{\mathsf{sk}_2}$
7 : $\pi'' \leftarrow \mathsf{DHP}^{H_3}[\mathsf{sk}_2 \restriction_{\mathsf{pk}_1, u'}^{\mathsf{pk}, w}]$
8 : $\longrightarrow \mathrm{C} : w, \pi''$
9 : **return** $\top$

Fig. 6. Encryption and decryption in our construction

the first exponentiation in line 6 of $\mathcal{DC}_{\mathrm{S}}$ and sends it back to client (line 8), accompanying it with a proof (line 7) that it performed the exponentiation correctly. After verifying this proof (line 12 of $\mathcal{DC}_{\mathrm{C}}$) , the client performs the second and third exponentiation, and uses the obtained pk^r to find the plaintext m from c_2.

In the presented protocol, client contacts the server, which will then use its private key share. We do not include (neither here nor in the definition of $\mathcal{F}$ in Fig. 1) the details on how the server is *activated*. Indeed, if the server is always responsive, then an adversary with client's keyshare will be able to decrypt ciphertexts by masquerading as the client to the server. Formally, the activation of the server is decided in the environment. In practice [5], [21], client's keyshare is stored encrypted with a PIN, with guesses of the PIN impossible to verify without contacting the server. A mechanism for detecting the existence of several clones of keyshare is also used [22].

VI. Security proofs

Intuitively, our goal is to prove that any attack against the protocol DVPS of Fig. 5 and Fig. 6 can be converted into an attack against $\mathcal{F}$ of Fig. 1, where for the latter it is explicitly specified what the adversary is able to do. In particular, for a corrupted server, we prove that it cannot learn anything about the ciphertext of the user, and for a corrupted client, we prove that it can at most perform as may decryptions as there are decryption sessions with the server.

We begin by introducing an augmented version of the one-more CDH problem, whose hardness is a stronger assumption than hardness of the standard one-more CDH problem. In the *one-more (static) CDH problem with tests*, one is given (g, g^x) with $x \leftarrow\!\!\$\ \mathbb{Z}_p$, random $h_0, \ldots, h_n \in \mathbb{G}$, the oracle $(\cdot)^x$, and the oracle $\mathsf{Check}(\cdot)$, where $\mathsf{Check}(h)$ returns i such that $h = h_i^x$, and $\perp$ if there is no such i. The goal of the solver is to invoke $\mathsf{Check}(h_i^x)$ for all $i \in \{0, \ldots, n\}$, while making at most n calls to the oracle $(\cdot)^x$. The calls to Check-oracle are not restricted in number. While the Check-oracle may give the solver some extra power in the Standard Model, Bauer et al. [44, Thm. 5] show that nothing is gained in the Algebraic Group Model (AGM) [45].

Our scheme is instantiated with the simulatable proofs of knowledge of exponent $r = \log_g(u)$ (Fig. 4) that additionally allow the simulator to raise an arbitrary element $z \in \mathbb{G}$ of its choice to the power of r^{d} where $\mathsf{d} \in \{1, -1\}$. The simulator has to choose the value z at the time the adversary computes the corresponding proof. If the simulator wants to obtain $z^{r^{\mathsf{d}}}$, it will follow the instructions from Fig. 7. The quantity $z^{r^{\mathsf{d}}}$ can be computed as soon as the simulator gets the proof.

Raising $z \in \mathbb{G}$ to the power r^{d} for $\mathsf{d} \in \{1, -1\}$:

1) generate $t \leftarrow\!\!\$\ \mathbb{Z}_p$
2) program $\tilde{H}_i$ to return $h = z^{1/t}$ when the adversary queries it with g, u, ctx
3) at any point later, after obtaining the proof (π, v) from the adversary, compute $z^{r^{\mathsf{d}}} = v^t$. If the proof π is valid, then $v = h^{r^{\mathsf{d}}}$.

Fig. 7. Raising $z \in \mathbb{G}$ to the power of r^{d} for $\mathsf{d} \in \{1, -1\}$

Additionally, in proofs of security against the malicious server, the simulator may need to know the value β. Hence we give the proof of $\beta \in \mathbf{I}(\rho)$ in a manner that allows the simulator to extract it. The proof can be given using standard cryptographic techniques, we describe it in App. A-D. We believe that as key generation is a seldomly-performed operation, its high efficiency is relatively less important, as long as it can be performed in reasonable time. Moreover, in proofs of security against malicious client, the simulator may need to know the witnesses that the client has used in the creation of DFN proofs. While Damgård et al. [35] introduced

a *knowledge assumption* stating that such extraction is possible, we choose to add more components to the ciphertexts, allowing us to assume the hardness of DDH, one-more CDH and DCRA only. Thus, during the key generation the server also computes $B_2 \leftarrow\!\!\$\ \mathcal{E}_{\mathsf{ek}_2}(\beta)$. When the simulator is acting as the server, it will be able to make B_1 and B_2 contain different values $\beta_1 \neq \beta_2$, faking the plaintext equality proof. It will then be able to use the special soundness of the Σ-protocols to find the witness. Again, the proof can be given using standard cryptographic techniques, see App. A-B.

Theorem 1. The protocol set DVPS in Fig. 5 and Fig. 6 securely implements functionality $\mathcal{F}$ in presence of malicious static adversary under one-more CDH assumption with tests.

In this section, we give a proof sketch of Theorem 1. The full proof can be found in the full version of this paper [46].

In order to show universal composability security of our threshold decryption protocol, we construct a simulator $\mathcal{S}$ such that for each adversary $\mathcal{A}$ attacking the protocol, the environment $\mathcal{Z}$ cannot distinguish whether it is interacting with the real protocol and the adversary $\mathcal{A}$, or the ideal functionality $\mathcal{F}$ and the "ideal" adversary $\mathcal{A}_S$ (consisting of $\mathcal{A}$ and $\mathcal{S}$). We start with defining $\mathcal{S}$ and how it interacts with the ideal functionality and the adversary in case of different parties being corrupted.

Initialization:

We assume that the corruptions are known already in the initialization phase. There is a parameter n denoting the number of encryptions considered in this session. From an external challenger, $\mathcal{S}$ receives as an input $\mathsf{pk} = g^{\mathsf{sk}}$ (where the sk is only known to the challenger) and also the following:

- If the client is *corrupted*, $\mathcal{S}$ receives $u_0, \ldots, u_n$ for some parameter n, such that $(g, \mathsf{pk}, u_0, \ldots, u_n)$ is an instance of a one-more CDH problem. $\mathcal{S}$ also gets access to an oracle $(\cdot)^{\mathsf{sk}}$. Intuitively, if the number of decrypted ciphertexts exceeds the number of backdoor decryption queries allowed by $\mathcal{F}$, then the $\mathcal{S}$ finds solution to the one-more CDH problem.
- If the client is *honest*, $\mathcal{S}$ receives pk and u_0, such that (g, pk, u_0) is an instance of a CDH problem, i.e. one-more CDH problem with $n = 0$. $\mathcal{S}$ takes $u_1 = \cdots = u_n \leftarrow u_0$. Intuitively, breaking the IND-CCA property of the scheme would mean breaking the CDH problem.

In addition, in both cases, $\mathcal{S}$ gets access to the oracle $\mathsf{Check}(\cdot)$.

We proceed with arguing that the encryption and decryption performed by the ideal functionality interacting with the $\mathcal{S}$ is indistinguishable from the real protocol. If the environment is able to distinguish between the real protocol and the ideal protocol, then the simulator has enough information to solve one-more CDH problem that it received as an input.

The simulator prepares tables for storing the random oracle data, and in the background it runs processes that are constantly observing accesses of the adversary $\mathcal{A}$ and the environment $\mathcal{Z}$ to these oracles, programming them on demand to extract certain values as described further.

Key generation:

- Use random oracle H_c to extract the corrupted party's share of pk and the proof π from the commitment.
- Use knowledge extraction w.r.t. π to adjust the honest party's share of pk to the input pk and the share sk_i of the corrupted party (by applying the procedure of Fig. 7 to the random oracles H_0 and $\tilde{H}_0$).
- Simulate the proofs related to β. If the client is corrupted, $\mathcal{S}$ comes up with β_1 and β_2 such that $\beta_1 \neq \beta_2$ to be able to apply special soundness later. As the simulator has access to trapdoors, the proofs of equality can be simulated even if $\beta_1 \neq \beta_2$.

Encryption:

- Whenever $\mathcal{F}$ is used to encrypt a message, it asks for a ciphertext from $\mathcal{S}$, who constructs the ciphertext from the next challenge. In particular, it computes $u = (u_j)^r$ for a challenge u_j instead of taking $u = g^r$ for a known r. Also, it outputs a random c_2 instead of encrypting the true message (which it does not know) so that the random oracles of H' and H'' could be programmed later. All related proofs can be simulated.
- For any ciphertext encrypted outside of $\mathcal{F}$, a valid proof needs to be constructed for a later successful decryption, for which oracles H_1 and $\tilde{H}_1$ needs to be accessed. This allows $\mathcal{S}$ to prepare random oracles H_1 and $\tilde{H}_1$ to compute pk^r as in Fig. 7.

$\mathsf{Verify}^{H_1,\tilde{H}_1,B_1,B_2}(c_1, c_2, c_3)$

1 : $(u, \alpha_1, \Gamma_1, \Gamma_2, \pi, \pi_1, \pi_2) \leftarrow c_1$
2 : **assert** $\mathsf{ChE}_1^{H_1,\tilde{H}_1}[\pi \,|^{u}_{g}|\alpha_1, \Gamma_1, \Gamma_2]$
3 : **assert** $\mathsf{DVC}(\pi_\iota \,|^{\alpha_1,\Gamma_\iota}_{g,u}|B_\iota)$
4 : **return** true

Fig. 8. Ciphertext correctness verification function Verify (delegated by the simulator to $\mathcal{F}$). Here H_1 and $\tilde{H}_1$ are provided as tables.

$\mathsf{Dec}^{\mathsf{T}',H',H''}(c_1, c_2)$

1 : $(u, \ldots, \pi, \ldots) \leftarrow c_1$
2 : **if** $(m, ((u, \ldots), c_2))$ **in** T **then**
3 : **return** m
4 : $(\ldots, v) \leftarrow \pi$
5 : **get** (u, v, t) **from** T'
6 : $k \leftarrow v^t$
7 : $(c_2', c_2'') \leftarrow c_2$
8 : **assert** $c_2'' = H''(k, c_2')$
9 : compute $m \leftarrow H'(k) \oplus c_2'$
10 : **return** m

Fig. 9. Decryption function Dec (delegated by the simulator to $\mathcal{F}$). If only the proof part of c is different from some existing ciphertext c', take (m, c') from T. Otherwise, decrypt as in the real protocol. Although $\mathcal{F}$ does not know the secret key, it can compute $k = v^t$ using the table T' that contains all the entries (u, v, t) for which $\mathcal{S}$ has made precomputations of Fig. 7. It extracts the entry that corresponds to u and v (if there is no such entry, return $\perp$). Here H' and H'' are provided as tables.

Decryption (honest client):

- $\mathcal{F}$ waits for a verification function Verify from $\mathcal{S}$, who chooses it as the initial knowledge extraction and the DVNIZK verification on behalf of the client. As it needs to evaluate H_1 and $\tilde{H}_1$, the contents of these hash tables are delivered as a part of procedure Verify (Fig. 8). The quantities B_1 and B_2 are also delivered as a part of Verify.
- As $\mathcal{S}$ does not know the true ciphertext, interaction with the server is simulated for a ciphertext generated from a *random plaintext*. Due to the statistical blinding, the environment detects the difference with the negligible probability.
- $\mathcal{F}$ waits for a decryption function Dec from $\mathcal{S}$, who chooses it in such a way that $\mathcal{F}$ can combine the proof π with the data of the H_1 and $\tilde{H}_1$ to get pk^r. For this, $\mathcal{S}$ provides a table T' of all triples (u, v, t) obtained so far for H_1 and $\tilde{H}_1$ as in Fig. 7, so that $\mathcal{F}$ can take the one that corresponds to u and v. The function Dec (Fig. 9) takes into account that different ciphertexts *can* depend on each other as far as only the proof part is different, so it lets $\mathcal{F}$ take a record (m, c) from its internal table even if only u and c_2 match, but the proofs look different (as far as they are valid).
- For any ciphertext generated *inside* $\mathcal{F}$, the attacker could notice inconsistency of decryption only if it manages to construct a ciphertext $((u^*, \ldots), c_2^*)$ which would decrypt to something that is related to a message that is encrypted with $\mathsf{pk}^r = u^{\mathsf{sk}}$ where u is a CDH challenge, but $u^* \neq u$ or $c_2^* \neq c_2$. To construct such a ciphertext the attacker would need to solve CDH.

Decryption (corrupted client):

- The simulator gets from $\mathcal{A}$ a blinded ciphertext and needs to come up with a response w and a proof π'. The proof can be simulated, but the response should be valid for a valid ciphertext.
  - For messages generated outside of $\mathcal{F}$, using special soundness, $\mathcal{S}$ can compute $r' = \log_g(u')$ from a valid sigma-proof, which allows to compute $w = \mathsf{pk}_2^{r'}$.
  - For messages generated inside $\mathcal{F}$, it is impossible to extract the exponent, so $\mathcal{S}$ needs to compute $w = (u')^{\mathsf{sk}_2} = (u'^{1/\mathsf{sk}_1})^{\mathsf{sk}}$. For this, it needs to access the oracle $(\cdot)^{\mathsf{sk}}$. To obtain u'^{1/sk_1} from u', $\mathcal{S}$ uses Fig. 7 to prepare random oracles H_2 and $\tilde{H}_2$ to compute this value at the point when a valid proof is created (the value u' is a part of the context which is an input to $\tilde{H}_2$).
  - In general, homomorphic encryption allows to obtain a valid ciphertext as a linear combination of ideal ciphertexts. However, since $\mathcal{S}$ has used the same γ in Γ_1 and Γ_2 of the challenge ciphertexts, special soundness allows to extract the free term r', splitting $u' = u'_1 \cdot u'_2$ where $u' = g^{r'}$ and $u'_2 = \prod_j u_j^{z_j}$ is a linear combination of challenges. The simulator can compute $w = \mathsf{pk}_2^{r'} \cdot (u'_2)^{\mathsf{sk}}$, thus ensuring that $(\cdot)^{\mathsf{sk}}$ is only applied to the linear combinations of challenges and is not misused e.g. to break some proof.
- In any case, there will be no more oracle $(\cdot)^{\mathsf{sk}}$ called than there are decryptions approved by the server. At any point of time when the adversary attempts to actually decrypt a ciphertext (it may happen later, after the decryption has ended), the simulator has control over it as it controls H' and H'' that are valuated for a valid decryption. The simulator programs H' and H'' so that it would decrypt to the true message. If it is one of the previous ciphertexts by $\mathcal{F}$ (this fact can be verified using Check$(\cdot)$ oracle), the simulator can use the "backdoor" decryption of $\mathcal{F}$ to get the plaintext. If the adversary manages to obtain more plaintexts than there have been decryption sessions (and hence allowed "backdoor" decryptions), $\mathcal{S}$ has solved the one-more-CDH problem.

VII. Efficiency

The performance of the encryption scheme can most probably be improved, perhaps by relying on different hardness assumptions. But even currently, the performance is very satisfactory for the intended application — decrypting credentials before their use. We have implemented DVPS encryption and decryption in Python on top of the PyCryptodome cryptographic library, as well as the *python-paillier* library [47] for the Paillier encryption. We use the elliptic curve group P-256 as $\mathbb{G}$, and 3072-bit RSA modulus for Paillier. The running times on a laptop with an Intel® Core™ i5-10210U CPU and 16GB RAM are 1140 ms for $\mathcal{E}nc$, 563 ms for $\mathcal{DC}_{\mathrm{C}}$, and 153 ms for $\mathcal{DC}_{\mathrm{S}}$. The sizes of the messages sent from the client to the server and back are 11.5KB and 5.5KB, respectively.

We can count the size our ciphertexts as follows. Ignoring the symmetrically encrypted payload c_2 (Fig. 6), the component c_1 consists of two elements of $\mathbb{G}$ (each represented by $d+1$ bits), two Paillier ciphertexts, and three proofs π, π_1, π_2, where the first is different from the other two. The proof π consists of a group element and two numbers of length ρ and d, respectively. The proof π_ι (Fig. 10) consists of a ρ-bit hash value, two numbers of length $(\rho + d + \kappa)$ and $(2\rho + d + 2\kappa)$, respectively, and one value that can serve as the random coins for the homomorphic encryption. We have chosen the security parameter determining the privacy of the protocol as $\kappa = 128$. It is also reasonable to take the integrity parameter as $\rho = 80$. Moreover, the size of group elements is $d = 256$. Considering that Paillier ciphertexts are 6144 bits long and Paillier coins (which are elements of $\mathbb{Z}_N$ for the used RSA modulus N) are 3072 bits long, the total size of the ciphertext c_1 is 21714 bits.

Most of the size of c_1 is taken up by two Paillier ciphertexts and two Paillier coins. There exist additively homomorphic encryption schemes with smaller ciphertext sizes. The Castagnos-Laguillaumie (CL) encryption system [48] has ciphertexts of size 4166 bits at the same security level [49], with coins of ca. 1955 bits. The homomorphism is according to a smaller modulus, but sufficient for our purposes, where we assume that there is no modular reduction during the

operations inside the ciphertexts. Using CL encryption, our ciphertexts would be ca. 15500 bits in length. Similar sizes may be achievable with Joye-Libert cryptosystem [50].

We can compare these sizes with the ciphertext sizes in the schemes of Green [23] and Blazy et al. [24], both of which are pairing-based constructions. In the latter, the ciphertexts consist of 14 elements of the first source group, 15 elements of the second source group, and 4 elements of the target group. Assuming the use of the BLS12-381 curve [51], we estimate that the elements of these groups take ca. 384, $2 \cdot 384$, and $12 \cdot 384$ bits to represent, respectively. In this case, the size of the ciphertext is ca. 35328 bits. Green's scheme is given in the setting of *symmetric pairings*; the ciphertext consists of 25 elements of the source group and two numbers (used as exponents). We believe that in asymmetric setting, most components of the ciphertext can be elements of the first source group. With BLS12-381 curve, the size of the ciphertext should be in the range of 10–11 kilobits. As we discussed above, the goals of Green's scheme are different from ours.

VIII. Conclusions

We have discussed the existing security definitions for the blind assisted and/or threshold decryption, demonstrated their shortcomings, proposed new ones, and instantiated them with the DVPS scheme. The security of the scheme relies on well-known hardness assumptions, it combines existing building blocks in somewhat novel manner, and its security is proved using very recent techniques.

Our security definition is given in the universal composability model. We have used this model to be able to capture the details of of the security definition, capturing both the IND-CCA security and privacy notions. We have also given the first ever treatment of threshold decryption in the UC model, arguing what must be the interface that the ideal functionality offers to the simulator / the ideal adversary. We show how this interface changes when the decryption is privacy-preserving.

Acknowledgement

This research has been supported by Estonian Research Council, grant No. PRG1780.

References

[1] A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung, "How to share a function securely," in *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, ser. STOC '94. New York, NY, USA: Association for Computing Machinery, 1994, p. 522–533. [Online]. Available: https://doi.org/10.1145/195058.195405

[2] V. Shoup and R. Gennaro, "Securing threshold cryptosystems against chosen ciphertext attack," *J. Cryptol.*, vol. 15, no. 2, pp. 75–96, 2002. [Online]. Available: https://doi.org/10.1007/s00145-001-0020-9

[3] Y. Lindell and A. Nof, "Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM, 2018, pp. 1837–1854. [Online]. Available: https://doi.org/10.1145/3243734.3243788

[4] J. Doerner, Y. Kondi, E. Lee, and A. Shelat, "Threshold ecdsa from ecdsa assumptions: The multiparty case," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1051–1066.

[5] A. Buldas, A. Kalu, P. Laud, and M. Oruaas, "Server-supported rsa signatures for mobile devices," in *Computer Security – ESORICS 2017*, S. N. Foley, D. Gollmann, and E. Snekkenes, Eds. Cham: Springer International Publishing, 2017, pp. 315–333.

[6] M. Vauclair, "Secure element," in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg and S. Jajodia, Eds. Boston, MA: Springer US, 2011, pp. 1115–1116. [Online]. Available: https://doi.org/10.1007/978-1-4419-5906-5_303

[7] "Personal identification – ISO-compliant driving licence. Part 5: Mobile driving licence (mDL) application," ISO/IEC 18013-5:2021, 2021.

[8] M. Sporny, D. Longley, and D. Chadwick, "Verifiable Credentials Data Model v1.1," March 3rd 2022, https://www.w3.org/TR/vc-data-model/.

[9] "Proposal for a Regulation of the European Parliament and of the Council amending Regulation (EU) No 910/2014 as regards establishing a framework for a European Digital Identity (SEC(2021) 228 final) - (SWD(2021) 124 final) - (SWD(2021) 125 final)," 2021, https://digital-strategy.ec.europa.eu/en/library/trusted-and-secure-european-e-id-regulation.

[10] " Commission welcomes final agreement on EU Digital Identity Wallet," Press release, November 8th 2023, https://ec.europa.eu/commission/presscorner/detail/en/ip_23_5651.

[11] "The European Digital Identity Wallet," 2023, https://github.com/eu-digital-identity-wallet/eudi-doc-architecture-and-reference-framework.

[12] D. Fett, K. Yasuda, and B. Campbell, "Selective Disclosure for JWTs (SD-JWT)," Internet Engineering Task Force, Internet-Draft draft-ietf-oauth-selective-disclosure-jwt-08, Mar. 2024, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-oauth-selective-disclosure-jwt/08/

[13] S. Goldwasser, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract)," in *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, R. Sedgewick, Ed. ACM, 1985, pp. 291–304.

[14] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 2001, pp. 136–145.

[15] R. Cramer and V. Shoup, "A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack," in *Advances in Cryptology — CRYPTO '98*, H. Krawczyk, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 13–25.

[16] D. Boneh and V. Shoup, "A Graduate Course in Applied Cryptography," Jan. 2023, a book in preparation, v0.6.

[17] L. T. A. N. Brañdao, N. Mouha, and A. Vassilev, "Threshold Schemes for Cryptographic Primitives," National Institute of Standards and Technology (NIST), Tech. Rep. NISTIR 8214, Mar. 2019.

[18] R. Canetti, H. Krawczyk, and J. Nielsen, "Relaxing chosen-ciphertext security," Cryptology ePrint Archive, Paper 2003/174, 2003, https://eprint.iacr.org/2003/174. [Online]. Available: https://eprint.iacr.org/2003/174

[19] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," Cryptology ePrint Archive, Paper 2000/067, 2000, https://eprint.iacr.org/2000/067, version: 20051214:064128. [Online]. Available: https://eprint.iacr.org/2000/067

[20] W. Lueks, B. Hampiholi, G. Alpár, and C. Troncoso, "Tandem: Securing keys by using a central server while preserving privacy," *Proc. Priv. Enhancing Technol.*, vol. 2020, no. 3, pp. 327–355, 2020. [Online]. Available: https://doi.org/10.2478/popets-2020-0055

[21] J. M. Kirss, P. Laud, N. Snetkov, and J. Vakarjuk, "Server-supported decryption for mobile devices," in *Security and Trust Management - 18th International Workshop, STM 2022, Copenhagen, Denmark, September 29, 2022, Proceedings*, ser. Lecture Notes in Computer Science, G. Lenzini and W. Meng, Eds., vol. 13867. Springer, 2022, pp. 71–81. [Online]. Available: https://doi.org/10.1007/978-3-031-29504-1_4

[22] A. P. Sarr, "Cryptanalysis and improvement of smart-id's clone detection mechanism," Cryptology ePrint Archive, Paper 2019/1412, 2019, https://eprint.iacr.org/2019/1412. [Online]. Available: https://eprint.iacr.org/2019/1412

[23] M. Green, "Secure blind decryption," in *Public Key Cryptography – PKC 2011*, D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 265–282.

[24] O. Blazy, L. Brouilhet, C. Chevalier, P. Towa, I. Tucker, and D. Vergnaud, "Hardware security without secure hardware: How to decrypt with a password and a server," *Theor. Comput. Sci.*, vol. 895,

pp. 178–211, 2021. [Online]. Available: https://doi.org/10.1016/j.tcs.2021.09.042

[25] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*, ser. Lecture Notes in Computer Science, Y. Desmedt, Ed., vol. 2567. Springer, 2003, pp. 31–46. [Online]. Available: https://doi.org/10.1007/3-540-36288-6_3

[26] E. Bresson, J. Monnerat, and D. Vergnaud, "Separation results on the "one-more" computational problems," in *Topics in Cryptology – CT-RSA 2008*, T. Malkin, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 71–87.

[27] B. Bauer, G. Fuchsbauer, and A. Plouviez, "The one-more discrete logarithm assumption in the generic group model," in *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part IV*, ser. Lecture Notes in Computer Science, M. Tibouchi and H. Wang, Eds., vol. 13093. Springer, 2021, pp. 587–617. [Online]. Available: https://doi.org/10.1007/978-3-030-92068-5_20

[28] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko, "The one-more-rsa-inversion problems and the security of chaum's blind signature scheme," *J. Cryptol.*, vol. 16, no. 3, pp. 185–215, 2003. [Online]. Available: https://doi.org/10.1007/s00145-002-0120-1

[29] R. Canetti, H. Krawczyk, and J. B. Nielsen, "Relaxing chosen-ciphertext security," in *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, ser. Lecture Notes in Computer Science, D. Boneh, Ed., vol. 2729. Springer, 2003, pp. 565–582. [Online]. Available: https://doi.org/10.1007/978-3-540-45146-4_33

[30] J. Groth and A. Sahai, "Efficient non-interactive proof systems for bilinear groups," in *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, ser. Lecture Notes in Computer Science, N. P. Smart, Ed., vol. 4965. Springer, 2008, pp. 415–432. [Online]. Available: https://doi.org/10.1007/978-3-540-78967-3_24

[31] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology — EUROCRYPT '99*, J. Stern, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238.

[32] M. Bellare and P. Rogaway, "The exact security of digital signatures - how to sign with RSA and rabin," in *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, ser. Lecture Notes in Computer Science, U. M. Maurer, Ed., vol. 1070. Springer, 1996, pp. 399–416. [Online]. Available: https://doi.org/10.1007/3-540-68339-9_34

[33] C. P. Schnorr, "Efficient identification and signatures for smart cards," in *Advances in Cryptology — CRYPTO' 89 Proceedings*, G. Brassard, Ed. New York, NY: Springer New York, 1990, pp. 239–252.

[34] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn, "Malleable proof systems and applications," in *Advances in Cryptology – EUROCRYPT 2012*, D. Pointcheval and T. Johansson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 281–300.

[35] I. Damgård, N. Fazio, and A. Nicolosi, "Non-interactive zero-knowledge from homomorphic encryption," in *Theory of Cryptography*, S. Halevi and T. Rabin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 41–59.

[36] P. Chaidos and G. Couteau, "Efficient designated-verifier non-interactive zero-knowledge proofs of knowledge," Cryptology ePrint Archive, Paper 2017/1029, 2017, https://eprint.iacr.org/2017/1029. [Online]. Available: https://eprint.iacr.org/2017/1029

[37] R. Cramer and V. Shoup, "Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack," *SIAM J. Comput.*, vol. 33, no. 1, pp. 167–226, 2003. [Online]. Available: https://doi.org/10.1137/S0097539702403773

[38] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*, D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, Eds. ACM, 1993, pp. 62–73. [Online]. Available: https://doi.org/10.1145/168588.168596

[39] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology — CRYPTO' 86*, A. M. Odlyzko, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 186–194.

[40] D. Bernhard, M. Fischlin, and B. Warinschi, "On the hardness of proving cca-security of signed elgamal," in *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part I*, ser. Lecture Notes in Computer Science, C. Cheng, K. Chung, G. Persiano, and B. Yang, Eds., vol. 9614. Springer, 2016, pp. 47–69. [Online]. Available: https://doi.org/10.1007/978-3-662-49384-7_3

[41] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *Advances in Cryptology — CRYPTO' 92*, E. F. Brickell, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 89–105.

[42] I. Damgård and E. Fujisaki, "A statistically-hiding integer commitment scheme based on groups with hidden order," in *Advances in Cryptology — ASIACRYPT 2002*, Y. Zheng, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 125–142.

[43] I. Damgård and R. Thorbek, "Linear integer secret sharing and distributed exponentiation," in *Public Key Cryptography - PKC 2006*, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 75–90.

[44] B. Bauer, G. Fuchsbauer, and J. Loss, "A classification of computational assumptions in the algebraic group model," in *Advances in Cryptology – CRYPTO 2020*, D. Micciancio and T. Ristenpart, Eds. Cham: Springer International Publishing, 2020, pp. 121–151.

[45] G. Fuchsbauer, E. Kiltz, and J. Loss, "The algebraic group model and its applications," in *Advances in Cryptology – CRYPTO 2018*, H. Shacham and A. Boldyreva, Eds. Cham: Springer International Publishing, 2018, pp. 33–62.

[46] P. Laud, A. Pankova, and J. Vakarjuk, "Privacy-preserving server-supported decryption," 2024. [Online]. Available: https://arxiv.org/abs/2410.19338

[47] CSIRO's Data61, "Python Paillier Library," https://github.com/data61/python-paillier, 2013.

[48] G. Castagnos and F. Laguillaumie, "Linearly homomorphic encryption from DDH," in *Topics in Cryptology - CT-RSA 2015, The Cryptographer's Track at the RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings*, ser. Lecture Notes in Computer Science, K. Nyberg, Ed., vol. 9048. Springer, 2015, pp. 487–505. [Online]. Available: https://doi.org/10.1007/978-3-319-16715-2_26

[49] G. Castagnos, "Cryptography based on quadratic fields: cryptanalyses, primitives and protocols," Ph.D. dissertation, Université de Bordeaux, 2019.

[50] M. Joye and B. Libert, "Efficient cryptosystems from 2^k-th power residue symbols," in *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, ser. Lecture Notes in Computer Science, T. Johansson and P. Q. Nguyen, Eds., vol. 7881. Springer, 2013, pp. 76–92. [Online]. Available: https://doi.org/10.1007/978-3-642-38348-9_5

[51] S. Bowe, "BLS12-381: New zk-SNARK Elliptic Curve Construction," Electric Coin Co. blog post, March 11th 2017, https://electriccoin.co/blog/new-snark-curve/.

[52] Y. Lindell, A. Nof, and S. Ranellucci, "Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody," Cryptology ePrint Archive, Paper 2018/987, 2018, https://eprint.iacr.org/2018/987. [Online]. Available: https://eprint.iacr.org/2018/987

APPENDIX A
SUBPROTOCOLS FOR KEY GENERATION

A. Proving equality of Paillier encryption and Pedersen commitment

This proof is a subprotocol in protocols given below. It can be constructed from standard building blocks, as shown by Lindell et al. [52, Sec. 6.2.4]. In our setting, the server has presented a Paillier ciphertext $E = \mathcal{E}_{\mathsf{ek}}(x)$ and a Pedersen commitment $C = g^x h^z \in \mathbb{G}$ to the client, where the server knows Paillier private key vk, the value x, the exponent z, and

also the random coins used to create E. The value $g \in \mathbb{G}$ is the generator of $\mathbb{G}$, while h is another element of $\mathbb{G}$, such that neither the client nor the server know $\log_g h$.

We use this subprotocol unchanged from [52], and will not copy it here.

B. Simulatable proof of equality of plaintexts in Paillier encryptions

At some point, server has sent two Paillier ciphertexts E_1 and E_2 to the client, and wants to prove that $x_1 = x_2$, where $x_i = \mathcal{D}_{\mathsf{vk}_i}(E_i)$. In order to do this, the server creates a Pedersen commitments C_1, C_2 to x_1 and x_2 — it computes $C_i \leftarrow g^{x_i} h^{z_i}$, where $z_1, z_2 \leftarrow\$ \mathbb{Z}_p$. The server proves that the ciphertext E_i contains the same value as C_i (App. A-A) for $i \in \{1, 2\}$. The server also makes, and client verifies the proof $\pi \leftarrow \mathsf{KnE}_1[z \mid {}^{C}_{h}]$, where $z = z_1 - z_2$ and $C = C_1/C_2 = g^{x_1 - x_2} h^z$. The latter proof implies that server knows $\log_h C$, meaning that C must have been computed as $g^0 h^z$, because server does not know $\log_g h$.

This proof is simulatable, because the simulator knows $\log_g h$. It can thus produce π (or run the interactive version of it) even if C is not a commitment to 0.

C. Disjunction of Σ-protocols

Suppose there are Σ-protocols Σ_1, Σ_2 for two relations R_1 and R_2, and Prover and Verifier have an instance x_1, x_2 for each of them. Let Prover also have witness w_ι for one of them, i.e. $\iota \in \{1, 2\}$. Prover wants to show that it has this w_ι, but wants to disclose neither it nor the index ι to Verifier. Let the challenge messages in Σ_1 and Σ_2 belong to the same set, and let this set also have the structure of an Abelian group. The protocol for such disjunction is the following.

1) Prover generates the "challenge" $\beta_{3-\iota}$.
2) Using the zero-knowledge property of $\Sigma_{3-\iota}$, Prover generates $(\alpha_{3-\iota}, \gamma_{3-\iota})$, such that $(x_{3-\iota}, \alpha_{3-\iota}, \beta_{3-\iota}, \gamma_{3-\iota})$ would pass Verifier's final check in $\Sigma_{3-\iota}$.
3) Prover generates the message α_ι in the protocol Σ_ι, using x_ι, w_ι if necessary.
4) Prover sends (α_1, α_2) to Verifier.
5) Verifier generates the challenge β and sends it to Prover.
6) Prover defines $\beta_\iota \leftarrow \beta - \beta_{3-\iota}$.
7) Prover constructs the message γ_ι in protocol Σ_ι, using β_ι as the challenge.
8) Prover sends $(\beta_1, \beta_2, \gamma_1, \gamma_2)$ to Verifier.
9) Verifier performs the checks in both Σ_1 and Σ_2, using β_1, β_2 as the challenges. Also checks that $\beta_1 + \beta_2 = \beta$.

Such protocol is again a Σ-protocol. Both the zero-knowledge simulation, and extraction algorithms for the composed protocol can easily be constructed from the respective algorithms for the subprotocols.

D. Extractable range proof for the challenge

During the key generation of DVPS, the server has to prove to the client that the encrypted challenge $B = \mathcal{E}_{\mathsf{ek}}(\beta)$ has been correctly generated, meaning that $0 \le \beta < 2^\rho$. This proof, together with the necessary trapdoors can be constructed from standard building blocks. The proof requires a hash function H_{com}, mapping into the group $\mathbb{G}$, and modeled as a random oracle.

Let $h = H_{\mathsf{com}}(B, ctx)$, where ctx binds this range proof to the current key generation session. E.g. ctx may contain the public key pk that has already been constructed at that time. Using h as the public key in ElGamal encryption scheme, the server encrypts (in the exponent) each bit of β separately. I.e. if $\sum_{i=0}^{\rho-1} \beta_i \cdot 2^i = \beta$, then the server picks $r_0, \ldots, r_{\rho-1} \leftarrow\$ \mathbb{Z}_p$ and computes $c_{0,i} = g^{r_i}$, $c_{1,i} = g^{\beta_i} h^{r_i}$ for each i. The server sends all values $c_{0,0}, \ldots, c_{1,\rho-1}$ to the client. Both the server and the client define $C = \prod_{i=0}^{\rho-1} c_{1,i}^{2^i}$. In this way, C will be a Pedersen commitment to β, where the randomness is $z = \sum_{i=0}^{\rho-1} r_i \cdot 2^i$. Server proves to the client that the Paillier ciphertext B and Pedersen commitment C contain the same plaintext.

For each bit β_i, the server proves to the client that it, i.e. the plaintext for the ciphertext $(c_{0,i}, c_{1,i})$ is indeed a bit. Note that if $\beta_i = 0$, then $(g, h, c_{i,0}, c_{i,1})$ is a Diffie-Hellman tuple and the server could convince the client in this using the Σ-protocol for DDH proofs (Sec. V-A); the server knows the randomness r_i necessary to give the proof. Similarly, if $\beta_i = 1$, then $(g, h, c_{i,0}, c_{i,1})/g$ is a Diffie-Hellman tuple and the same protocol could be used again. In order to show that β_i is either 0 or 1, we combine these two Σ-protocols using the construction in App.A-C.

The simulator can decrypt the ElGamal ciphertexts $(c_{i,0}, c_{i,1})$ and find the bits β_i, because it can program H_{com}. Whenever H_{com} is invoked either by itself or by the adversary, the simulator will generate the answer as g^r, where $r \leftarrow\$ \mathbb{Z}_p$, and store r together with the argument(s) of H_{com}.

Appendix B
Proving that a DVNIZK proof is valid

Given a generator g of $\mathbb{G}$, a Paillier encryption key ek, a ciphertext B, and the values (u, α_1, Γ), suppose that the prover wants to convince the verifier that they have been constructed as in DVPS.$\mathcal{E}nc$ and the prover knows the used randomness $r = \log_g u \in \mathbf{I}(d)$, $r_1 = \log_g \alpha_1 \in \mathbf{I}(\rho + d + \kappa)$, and the random coins $\mathbf{r}$ for the encryption. The Σ-protocol for this is similar to proving the knowledge of exponent, where the prover constructs another similar triple (α_2, α_3, A) using fresh randomness $(r_2, r_3, \mathbf{r}')$ and sends it to the verifier, also sends a linear combination (with coefficients chosen by the verifier) of the original and fresh randomness to the verifier, who can then use the homomorphic properties of the operations to match the original triple against the fresh one. Instead of including α_2, α_3 and A in the proof, the prover instead includes β' and lets the verifier compute α_2, α_3 and A that would satisfy the equalities, followed by a check if β' indeed computed as their hash. We present the protocol in Fig. 10 as a NIZK proof, replacing verifier's choice of the coefficients of the linear combination with the invocation of a random oracle. The simulator is thus able to fake these proofs.

The proof uses the fact that Paillier's encryption is homomorphic with respect to the used coins. In Fig. 10, the

$\mathsf{DVP}(r, r_1, \mathbf{r} \mid^{\alpha_1,\Gamma}_{g,u} \mid B)$	$\mathsf{DVC}(\pi \mid^{\alpha_1,\Gamma}_{g,u} \mid B)$
$r_2 \leftarrow\!\$\ \mathbf{I}(\rho + d + \kappa)$	$(\beta', \gamma_2, \gamma_3, \gamma_{\mathbf{c}}) \leftarrow \pi$
$r_3 \leftarrow\!\$\ \mathbf{I}(2\rho + d + 2\kappa)$	$\alpha_2 \leftarrow g^{\gamma_2} / u^{\beta'}$
$\mathbf{r}' \leftarrow\!\$\ \mathbf{Cn}$	$\alpha_3 \leftarrow g^{\gamma_3} / \alpha_1^{\beta'}$
$\alpha_2 \leftarrow g^{r_2 \bmod p}$	$A \leftarrow \mathcal{E}_{\mathsf{ek}}(\gamma_3; \gamma_{\mathbf{c}}) \cdot B^{\gamma_2} / \Gamma^{\beta'}$
$\alpha_3 \leftarrow g^{r_3 \bmod p}$	$\mathbf{assert}\ \beta' = H(g, u, \alpha_1, \Gamma,$
$A \leftarrow \mathcal{E}_{\mathsf{ek}}(r_3; \mathbf{r}') \cdot B^{r_2}$	$\alpha_2, \alpha_3, A, B)$
$\beta' \leftarrow H(g, u, \alpha_1, \Gamma, \alpha_2, \alpha_3, A, B)$	
$\gamma_2 \leftarrow r_2 + \beta' \cdot r$	
$\gamma_3 \leftarrow r_3 + \beta' \cdot r_1$	
$\gamma_{\mathbf{c}} \leftarrow \mathbf{r}' \boxplus (\beta' \boxdot \mathbf{r})$	
$\pi \leftarrow (\beta', \gamma_2, \gamma_3, \gamma_{\mathbf{c}})$	
$\mathbf{return}\ \pi$	

Fig. 10. Proving the validity of a DVNIZK proof

operations $\boxdot$ and $\boxplus$ combine the coins. The value $\mathbf{r}$ is perfectly masked by $\mathbf{r}'$. The values r, r_1 however, are masked not perfectly, but only statistically, with random values having $\rho + \kappa$ more bits.

We show that the protocols of Fig. 10 prevent selective failure attacks on the underlying DFN proofs from [36]. In a DFN proof, the server verifies a linear combination $\gamma_1 = r_1 + r\beta$. The proof fails if $r_1 + r\beta \geq N$ where N is the size of the plaintext space of the homomorphic encryption scheme. The prover (in our case, the encrypting party) may take $r_1 = N - r \cdot b$ to check whether $\beta \geq b$ by observing whether the verifier (in our case, the server) rejects the proof. The verification procedure of Fig. 10 only depends on β through $B = \mathcal{E}_{\mathsf{ek}}(\beta)$ which does not leak β by properties of the homomorphic encryption scheme. However, we need to ensure that Fig. 10 does not miss selective failure attacks, and we show that either γ_2 or γ_3 will be too large so that they will rejected by the client. Let $\gamma_1 = r_1 + \beta r \geq N$. This means that either $r_1 \geq N/2$ or $\beta r \geq N/2$. In the first case, we have $\gamma_3 = r_3 + \beta' r_1 \geq \beta' r_1 \geq \beta' N/2$, which gives $\gamma_3 < N$ with probability at most $2^{-\rho}$ (since β' is not under attacker's control). In the second case, we have $\gamma_2 = r_2 + \beta' r \geq \beta' r \geq \beta' / \beta \cdot N/2 \geq \beta' \cdot N/2^{\rho+1}$. While such γ_2 can be smaller than N, it will still quite likely be larger than the largest possible γ_2 that can be obtained for a valid proof. A valid γ_2 by construction is strictly smaller than $2^{\rho+d+\kappa+1}$, while $N/2^{\rho+1} \geq 2^{2\rho+d+2\kappa}/2^{\rho+1} \geq 2^{\rho+d+\kappa}$, so the malicious γ_2 will be smaller than the allowed upper bound with probability $2^{-\rho}$ (again, since β' is not under attacker's control).

Appendix 3

Publication II

(Full version) P. Laud, A. Pankova, and J. Vakarjuk. Privacy-preserving server-supported decryption. *CoRR*, abs/2410.19338, 2024

Privacy-preserving server-supported decryption

Peeter Laud
Cybernetica AS
Tartu, Estonia
peeter.laud@cyber.ee

Alisa Pankova
Cybernetica AS
Tartu, Estonia
alisa.pankova@cyber.ee

Jelizaveta Vakarjuk
Cybernetica AS
Tallinn University of Technology
Tallinn, Estonia
jelizaveta.vakarjuk@cyber.ee

arXiv:2410.19338v1 [cs.CR] 25 Oct 2024

***Abstract*—In this paper, we consider encryption systems with two-out-of-two threshold decryption, where one of the parties (the client) initiates the decryption and the other one (the server) assists. Existing threshold decryption schemes disclose to the server the ciphertext that is being decrypted. We give a construction, where the identity of the ciphertext is not leaked to the server, and the client's privacy is thus preserved. While showing the security of this construction, we run into the issue of defining the security of a scheme with blindly assisted decryption. We discuss previously proposed security definitions for similar cryptographic functionalities and argue why they do not capture the expected meaning of security. We propose an ideal functionality for the encryption with server-supported blind threshold decryption in the universal composability model, carefully balancing between the meaning of privacy, and the ability to implement it. We construct a protocol and show that it is a secure implementation of the proposed functionality in the random oracle model.**

***Index Terms*—threshold encryption, non-interactive zero knowledge, ElGamal**

I. Introduction

Threshold cryptography offers a set of techniques for management of private keys; it can help out if there is no single party sufficiently trusted and sufficiently in control of its computational environment, such that they could be allowed to know the whole private key. Both threshold signatures [1] and threshold decryption [2] are well-known cryptographic techniques. For signatures, there exist large-scale deployments in blockchain environments [3], [4], as well as for general electronic identity [5]. In the last example, threshold cryptography is used because a certain party does not have sufficient control over their computational environment. That party is a *user*, having a *smartphone* that stores their keyshare. If one wants to avoid secure hardware based solutions [6], the phone alone cannot offer sufficient protection for a private key.

We emphasize that in the use-cases related to the electronic identity, we are specifically interested in 2-out-of-2 secret sharing of the private key. Sharing is necessary, because otherwise the server alone would be able to use the private key. Having just two parties is natural in such system — the roles of the phone and the assisting server are clearly different, while any more parties (e.g. implementing the server in threshold manner) would reduce the usability of the system, would not significantly increase the security (the server is expected to securely manage the keyshare), and may even interfere with some of the mechanisms for detecting break-ins.

In the near future, a user's digital presence is constructed around the credentials that they have obtained from the various issuers and are able to present to the various relying parties (RP). There exist standards, e.g. for mobile Driver's License [7] by ISO or for verifiable credentials [8] by W3C that specify, how these credentials are formatted and through which protocols they are presented. There exist advanced legistlative initiatives, e.g. the upcoming eIDAS 2.0 regulation of the European Union [9], [10] that will mandate the availability of *digital identity wallets* — smartphone applications that manage the user's credentials by following these standards, and implementing a certain architecture [11]. The architecture may, to a certain extent, protect the user's privacy through *selective disclosure*, releasing only user-approved parts of multi-part credentials to a RP [7], [12], but more capable privacy techniques, e.g. zero-knowledge proofs [13] are not part of the architecture.

The wallet app running on the smartphone has to manage the storage of credentials. They may be stored on- or off-device, but will definitely be encrypted, because their content may be sensitive. Before a presentation of the credential, it has to be decrypted, because the wallet app needs to know its content to be presented. The app and the phone have to have control over the decryption step. If one wants to avoid secure hardware based solutions, then the decryption has to be thresholdized, and the same 2-out-of-2 setting makes the most sense. In this case, whenever an RP asks the user to present a credential that the user has and the user agrees to present it, the phone sends the ciphertext encrypting that credential to the assisting server. The phone and the assisting server run the threshold decryption protocol, and (only) the phone learns the plaintext credential. After running the protocol(s) with the RP, the phone deletes the decrypted credential from its memory. Next time the presentation of a credential is requested, the phone and the assisting server again run the decryption protocol. Any threshold decryption protocol is in principle usable here, e.g. the original ElGamal-based construction of Shoup and Gennaro [2].

Unfortunately, with regular threshold decryption protocol, the server will learn how often one or another ciphertext is being decrypted. This is a potential violation of the user's privacy: if, over time, the server sees several requests to decrypt the same credential, then it may make some inferences on the user's access pattern towards different RPs. This paper proposes a solution for this problem: we construct a 2-out-of-

2 threshold encryption scheme, where the decryption protocol is privacy-preserving against the server. Aiming for generality, our scheme is secure against chosen-ciphertext attacks even when one of the decrypting parties (either the client or the server) is corrupted.

Definitions of security properties for such schemes are similar to those for the encryption schemes with *blind assisted decryption*. Unfortunately, we have found existing definitions to not correspond to our intuitive notions of security; the discussion of these definitions is the second main contribution of this paper. We follow that discussion by proposing our own security definition for the server-assisted blind decryption. While the previous definitions have been given in the game-based model, stating the security game(s) that an adversary attempts to win, our definition is given in the universal composability (UC) [14] framework where the desired security properties are covered by an ideal functionality describing how the protocol *should* work. As there are several distinct properties that we are aiming for, we would need several different game-based definitions, which make it more complicated to justify why exactly these properties are necessary and sufficient, while an ideal functionality provides a generic overview. In this way, we get a better understanding of the details of the security definition, hopefully giving us a better assurance that it is the right one. Still, we see that our definitions follow a fine line between the intuitive meaning of the security and privacy on one side, and the implementability on the other side; we explore some new ground in defining UC encryption, and the details become significant. We see that the blinding makes analysis of composability quite important in the malicious client case. Due to the blinding, the client can potentially combine several server responses (which are not necessarily successful decryption outputs) in such a way that the plaintext (or even several plaintexts at once) will not be learned until the last decryption, and some attacks by a malicious client (in particular, the ability to decrypt more ciphertexts than there have been sessions with the server initiated) could be missed in the stand-alone model.

In the construction of our threshold encryption scheme, the goal has been to show the feasibility, and the conceptual simplicity of presentation. Still, its performance is fully satisfactory for the use-case of verifiable credentials. It should also be simple to implement and deploy, as it uses only simple cryptographic primitives, in particular avoiding pairings.

This paper has the following structure. After reviewing related work in Sec. II, we will dive into the previous security definitions of blind decryption in Sec. III. Next, we describe our security definition by presenting corresponding ideal functionality in Sec. IV. In Sec. V, we present main building blocks of our protocols and give our construction of a privacy-preserving threshold encryption scheme. Finally, we prove in Sec. VI that our construction indeed satisfies our security definition.

II. Related work

Encryption schemes with threshold decryption [2] and with indistinguishability against chosen-ciphertext attacks (IND-CCA) were proposed shortly after IND-CCA secure asymmetric encryption schemes [15]. At present, threshold cryptography is a mature field, discussed in textbooks [16] and subject to standardization activities [17]. However, we note the dearth of the exploration of threshold decryption in the UC model, meaning that many details of our ideal functionality are novel.

Regarding the ideal functionalities for the usual public key encryption, several different approaches can be taken. One can design the ideal functionality so, that the adversary internally executes the encryption and decryption algorithms [18]. Alternatively, the adversary may give descriptions of encryption and decryption algorithms to the ideal functionality, with the latter invoking them locally [19]. In the former case, the adversary may cause malicious ciphertexts to decrypt to values that depend on the values to which legitimate ciphertexts are decrypted to [19].

Our target use case involves two non-equal parties. In this setting, Buldas et al. [5] have proposed a 2-out-of-2 sharing for *signing* keys, where one of the shares is stored in a central server and the other one in the user's smartphone, encrypted with only a PIN. The signing protocol employs measures against an adversary guessing that PIN, or making a copy of the memory of the smartphone. Lueks et al. [20] have proposed a system with the same kind of 2-out-of-2 sharing, where also the users' usage patterns are protected; the system also relies on blind signatures. For the threshold decryption, Kirss et al. [21] have proposed a protocol with the same measures for protecting the keyshare in the smartphone as [5]; together with the clone detection mechanisms of [22], it could offer a viable alternative to secure elements [6]. Unfortunately, they do not attempt to hide the ciphertext from the assisting server.

Our scheme offers privacy for one of the decrypting parties against the other decrypting party. In this sense, our scheme is an instance of *blind assisted decryption*. We are aware of two previous attempts to formalize the IND-CCA security of blind assisted decryption, and devise schemes satisfying these definitions. Green [23] proposed a scheme, where the client sends decryption requests to a server with the private key. Blazy et al. [24] proposed an encryption system where the decryption capability and authorization was shared among three parties with unequal roles, offering privacy against the "server" party. We discuss both schemes and corresponding security definitions in Sec. III.

We analyse the security of our scheme in the universal composability model with ROM [14]. We use a less common hardness assumption: the *one-more static computational Diffie-Hellman problem* [25], [26]. Still, the assumption is not controversial: it is known to hold in the Generic Group Model and be equivalent to the hardness of Discrete Logarithm in the Algebraic Group Model [27]. The one-more problems commonly occur in the security analysis of *blind signatures* [25], [28], hence one may even expect it to make an appearance for

our blind primitive as well.

III. Security of blind assisted decryption

Blind assisted decryption has been previously considered by Green [23] and by Blazy et al. [24]. Both works have separately defined the blindness property, and the property of being secure against chosen ciphertext attacks. While the first property is not controversial, the proposed definitions for the latter cannot be considered fully satisfactory.

Blazy et al. consider a setting with three parties — *user*, *token*, and *server*. The token and the server have keyshares $sk_{\mathcal{T}}$ and $sk_{\mathcal{S}}$ of a 2-out-of-2 threshold decryption key, while the user and the server share a password pwd. The user initiates the decryption by activating the token with the ciphertext. The token then runs a protocol with the server. They define a password-protected Indistinguishability under Replayable Chosen-Ciphertext Attacks (P-IND-RCCA). Their multi-party (i.e. there are several users, tokens, and servers, each of those may be corrupted by the adversary) definition provides the adversary with a decryption oracle that performs decryption atomically (unless the submitted ciphertext decrypts to one of the two challenge plaintexts, as is the norm for RCCA [29]). When submitting these challenge plaintexts, the adversary specifies the identities of the user $\mathcal{U}^*$, token $\mathcal{T}^*$, and server $\mathcal{S}^*$ that it is attacking. The adversary also has access to the message-sending oracle, where it submits a message to one of the parties, and learns the message that this party replies with.

In order to model that no protection is offered if the adversary corrupts too many parties, the definition of P-IND-RCCA restricts, which parties the adversary may invoke. If $\mathcal{U}^*$ or $\mathcal{S}^*$ are corrupt (i.e. the adversary knows the password), then $\mathcal{T}^*$ may no longer be invoked. If $\mathcal{U}^*$ and $\mathcal{T}^*$ are corrupt (i.e. the adversary has the password and one of the keyshares) then $\mathcal{S}^*$ may no longer be invoked. Indeed, in such cases, ability to make queries to these parties might allow the adversary to decrypt the challenge ciphertext.

Such restrictions make P-IND-RCCA more akin to the "lunchtime" (CCA1) attack, when considering message-sending oracle. Also, some intuitively insecure encryption schemes satisfy P-IND-RCCA. Consider any P-IND-RCCA secure scheme, and modify it as follows:

1) Whenever the server $\mathcal{S}$ receives a message, it will check whether it has the format $(\mathsf{corrupt}, pwd_{\mathcal{U}}, sk_{\mathcal{T}})$, where $\mathcal{U}$ and $\mathcal{T}$ are the user and the token in the current decryption session, $pwd_{\mathcal{U}}$ is user's password, and $sk_{\mathcal{T}}$ is the keyshare of the token. The check can be done by encrypting a random plaintext and then running the decryption protocol in server's head. If the incoming message has such format, then respond with its own keyshare $sk_{\mathcal{S}}$.
2) *Alternatively*, when the token $\mathcal{T}$ receives the message $(\mathsf{corrupt}, sk_{\mathcal{S}})$, the "correctness" of which it can check in the same manner, then it responds with $sk_{\mathcal{T}}$.

We argue that the modified scheme (with either the first or the second change) should not be considered secure any more, because an adversary that has managed to take over one of the parties holding a secret share can easily extend this takeover to obtaining the other share. But the modified scheme still satisfies P-IND-RCCA, because such messages cannot be sent according to this definition. Indeed, an adversary that has corrupted the user and the token (giving him $pwd_{\mathcal{U}}$ and $sk_{\mathcal{T}}$) in the session being attacked, may no longer talk to the server. Similarly, after corrupting the server and learning $sk_{\mathcal{S}}$, the adversary may no longer talk to the token.

Green [23] considers a user $\mathcal{U}$ outsourcing decryption to a server $\mathcal{D}$. The server has the decryption key; the user, having a ciphertext, initiates the blind decryption protocol. The security of their scheme against chosen ciphertext attacks is captured by a "usual" IND-CCA2 definition, where the adversary can atomically invoke the "normal" decryption oracle, and by the *leak-freeness* definition. In this definition, a Distinguisher is asked to distinguish between a "real" and an "ideal" game. In the real game, an adversary chooses a ciphertext and the Distinguisher gets the user's view in the blind decryption protocol. Note that the execution of the blind decryption protocol is *atomic*, i.e. the adversary does not interfere with its run. In the ideal game, a simulator chooses a ciphertext, receives the corresponding plaintext, and simulates the trace. I.e. leak-freeness "ensures that an adversarial user gains no more information from the blind decryption protocol than they would from access to a standard decryption oracle" [23].

The absence of the blind decryption protocol in the IND-CCA2 definition and/or the atomicity requirement in the set-up of the "real" game allows some intuitively insecure schemes to satisfy Green's IND-CCA2 and leak-freeness definitions. Consider any scheme that satisfies them, and modify its blind decryption as follows:

- All messages from $\mathcal{U}$ to $\mathcal{D}$ have an additional bit b_{evil}.
- If $\mathcal{D}$ receives a message with $b_{\mathsf{evil}} = 0$, then it processes the rest of the message as in the original scheme.
- If $\mathcal{D}$ receives a message with $b_{\mathsf{evil}} = 1$, then it responds with the decryption key.
- $\mathcal{U}$ always sets $b_{\mathsf{evil}} = 0$.

Such scheme should definitely be considered as insecure. But it satisfies both IND-CCA2 (because blind decryption protocol is not invoked there) and leak-freeness (because the atomic execution of the blind decryption protocol means that no one sets $b_{\mathsf{evil}} = 1$).

IV. Ideal functionality for privacy-preserving server-assisted decryption

We consider asymmetric encryption schemes, where the decryption functionality is distributed between two parties — the *client*, and the *server* with different roles. An encryption scheme with the client-server decryption consists of the following sets, algorithms, and protocols, all parameterized with the security parameter λ.

- Sets of ciphertexts C, public keys PK, the client's private keys SK_{C}, and the server's private keys SK_{S}.
- Key-generation protocol $\langle \mathcal{KG}_{\mathrm{C}} | \mathcal{KG}_{\mathrm{S}} \rangle$, run by both parties. It returns $(\mathsf{sk}_C, \mathsf{pk}) \in \mathsf{SK}_{\mathrm{C}} \times \mathsf{PK}$ to the client, and $(\mathsf{sk}_S, \mathsf{pk}) \in \mathsf{SK}_{\mathrm{S}} \times \mathsf{PK}$ to the server.

- Encryption algorithm $\mathcal{E}nc$. It takes as input a public key $\mathsf{pk} \in \mathsf{PK}$, a plaintext $m \in \mathsf{M}$ and returns a ciphertext $c \in \mathsf{C}$.
- Decryption protocol $\langle \mathcal{DC}_{\mathrm{C}} | \mathcal{DC}_{\mathrm{S}} \rangle$, run by the client and the server. Client's inputs to $\mathcal{DC}_{\mathrm{C}}$ are $c \in \mathsf{C}$, $\mathsf{sk}_C \in \mathsf{SK}_{\mathrm{C}}$, and $\mathsf{pk} \in \mathsf{PK}$. Server's inputs to $\mathcal{DC}_{\mathrm{S}}$ are $\mathsf{sk}_S \in \mathsf{SK}_{\mathrm{S}}$ and $\mathsf{pk} \in \mathsf{PK}$. The protocol returns either $m \in \mathsf{M}$ or the failure notice $\perp$ to the client. It returns the success notice $\top$ or the failure notice $\perp$ to the server.

First of all, we define how such a scheme *should* work. Both functional and security requirements can be covered by an *ideal functionality* in the Universal Composability framework (UC) [14]. This framework has been widely used to capture security of distributed protocols. It allows to capture security properties of cryptographic schemes in the ideal/real process paradigm. In the real world, adversary $\mathcal{A}$ interacts with the real protocol π by means of corrupting different parties, gaining control over their inputs and outputs. In the ideal world, ideal functionality $\mathcal{F}$ defines an ideal process for the protocol π, intuitively describing how it should work. Ideal functionality can be viewed as a trusted party that receives inputs from all the parties, performs computation on these inputs, and returns to each party its output. It also interacts with an ideal adversary $\mathcal{S}$, from which it may receive certain commands and to which it may output some values that are explicitly allowed to be leaked. In both cases, parties receive inputs and return outputs to the environment $\mathcal{Z}$. A protocol π securely implements $\mathcal{F}$, if for any real-world adversary $\mathcal{A}$ there exists an ideal adversary $\mathcal{S}$, such that no environment $\mathcal{Z}$ can distinguish between interaction with π running in parallel with $\mathcal{A}$ and $\mathcal{F}$ running in parallel with $\mathcal{S}$.

The ideal functionality for our encryption scheme with two-party privacy-preserving decryption is given in Fig. 1. For simplicity, we define it only for a single public key (and the corresponding shared decryption functionality). Also for simplicity, we consider only static corruptions of parties; the identities of the corrupt parties are told to the $\mathcal{F}$ by the ideal adversary $\mathcal{S}$ at the beginning of the execution. The functionality $\mathcal{F}$ is given for an arbitrary number of parties $P_1, P_2, \ldots$, all of which can submit encryption requests. The decryption can be done jointly by parties P_1 (in the role of the client) and P_2 (in the role of the server).

The key generation (that only happens in the beginning) and encryption in $\mathcal{F}$ are similar to existing ideal functionalities for public-key encryption, where the public key pk is chosen by $\mathcal{S}$. Encryption is performed similarly to [18], where the adversary comes up with the ciphertext c for the plaintext m without actually seeing m, intuitively ensuring that c does not leak anything about m. Only if the encryptor party P_i is corrupted, will $\mathcal{S}$ learn the message m. The obtained pairs (m, c) are stored in a table T, allowing the decryption queries to be answered.

The encryption query does not contain a public key as an input. In practice, the environment could provide an honest encryptor with a public key pk' different from pk generated by $\mathcal{F}$, and expect a ciphertext encrypted with pk'. To cover this case, we would need to allow $\mathcal{F}$ deliver pk' to $\mathcal{S}$. This would introduce more details into the definition of $\mathcal{F}$ and the security proofs without being essential for the discussion of the key points of $\mathcal{F}$ and its secure implementations. We assume that a party (in the environment) only makes encryption queries once he has learned the correct pk.

Decryption in $\mathcal{F}$ corresponds to finding the plaintext m from a ciphertext c that was created using the key pk. It can only be initiated by P_1 and P_2, with P_1 providing the ciphertext.

If the client is honest, then the decryption is less straightforward than in the UC public key encryption, because neither of the two existing approaches appear to be fully satisfactory. If $\mathcal{S}$ provides the descriptions of the algorithms Enc and Dec to $\mathcal{F}$, then it may be difficult to build simulators in the Random Oracle Model (ROM), because the simulator probably needs to know the state of the oracles during decryption, as well as program the oracles at least during encryption. But if we task $\mathcal{S}$ to come up with the ciphertext values during encryption, and to decrypt unknown ciphertexts during decryption, then this goes against our intuition of "blinded decryption", where $\mathcal{S}$ should not learn the ciphertexts that an honest client wants to decrypt. We resolve this by making $\mathcal{S}$ run the encryption (programming the random oracles as necessary), and sending to $\mathcal{F}$ an *updated* description of Dec at each decryption. In this way, $\mathcal{S}$ does not learn the ciphertext that is being decrypted, but the current state of the random oracles can be included in the description of Dec and affect the result of decryption. Using this approach (and similarly to UC public key encryption), we need to be careful that the same ciphertext c will not decrypt to different values, which would violate correctness. For that reason, $\mathcal{F}$ adds to T all (m, c) pairs obtained during the decryption as well. If there will be two records (m, c) and (m', c) for $m \neq m'$, the decryption fails. In addition, since $\mathcal{S}$ cannot verify the correctness of the input ciphertext, $\mathcal{S}$ should first of all send to $\mathcal{F}$ a description of a function Verify which verifies the ciphertext, returning either true or false. $\mathcal{S}$ needs to learn this single bit of information, as in a real protocol the decryption would not start in this case.

If the client is corrupted (but the server is honest), then privacy of the ciphertext does not have to be protected; ciphertext can be given to $\mathcal{S}$. But we are now modelling threshold decryption, where a corrupted party is expected to learn the plaintext; we are not aware of any similar models in the literature. If a corrupted party is able to learn the plaintext m, then a simulator must also be able to learn it, in order to simulate that party's view. But if the ciphertext c was created using the encryption functionality of $\mathcal{F}$, then c is actually independent of m, and the only location containing m (beside the environment, which the simulator cannot depend on) is in the table T of $\mathcal{F}$. Hence we have to give $\mathcal{S}$ an ability, however minimal, to read that table. In the case of threshold decryption *without* blinding, it would be sufficient if $\mathcal{S}$ could query for the row (m, c), where c is the ciphertext that is being decrypted. For the blinded decryption, the adversary playing the corrupted client can actually start the decryption protocol for an arbitrary ciphertext, hence the row(s) queried by $\mathcal{S}$ can

Key generation
On message (KeyGen, sid) from both P_1 and P_2:

1) Send (KeyGen, sid) to $\mathcal{S}$ and wait for $(\mathsf{Key}, sid, \mathsf{pk})$ from $\mathcal{S}$.
2) If $\mathsf{pk} \neq \perp$, create an empty table T for storing encrypted messages, and set decryption counter $ctr_{\mathsf{dec}} \leftarrow 0$.
3) For $i \in \{1, 2\}$, if the party P_i is corrupted, wait for $(\mathsf{Output}, i, y_i)$ from $\mathcal{A}$. Otherwise, take $y_i = \mathsf{pk}$.
4) Output (Key, sid, y_1) to P_1 and (Key, sid, y_2) to P_2.

Encryption
On message $(\mathsf{Encrypt}, sid, m)$ from some party P_i:

1) If P_i is corrupted, send $(\mathsf{Encrypt}, sid, i, m)$ to $\mathcal{S}$. Otherwise, send $(\mathsf{Encrypt}, sid)$ to $\mathcal{S}$.
2) Wait for $(\mathsf{Encrypt}, sid, c)$ from $\mathcal{S}$ (the adversary chooses the ciphertext for the given plaintext).
3) Store the pair (m, c) in table T.
4) If the party P_i is corrupted, wait for (Output, i, y) from $\mathcal{A}$. Otherwise, take $y = c$.
5) Send $(\mathsf{Encrypted}, sid, y)$ to P_i.

Decryption procedure $Decrypt(\mathsf{Dec}, c)$ for a function $\mathsf{Dec} : \mathsf{C} \to \mathsf{M}$ and a ciphertext $c \in \mathsf{C}$:

- If there exists a pair (m, c) in T: (for correctness we need that c would decrypt to m)
  - If m is not unique (there are two different m for the same c), take $m' = \perp$.
  - If m is unique, take $m' = m$.
- Otherwise, compute $m' := \mathsf{Dec}(c)$, running it only a polynomial number of steps (the polynomial is a parameter of $\mathcal{F}$). Add (m', c) to T.
- Return m'.

Decryption (honest client)
On message $(\mathsf{Decrypt}, sid, c)$ from party P_1 and $(\mathsf{Decrypt}, sid)$ from party P_2:

1) Send $(\mathsf{Decrypt\text{-}init}, sid)$ to $\mathcal{S}$.
2) Upon receiving $(\mathsf{Decrypt\text{-}init}, sid, \mathsf{Verify})$ from $\mathcal{S}$, where $\mathsf{Verify} : \mathsf{C} \to \{\mathsf{true}, \mathsf{false}\}$, compute $b \leftarrow \mathsf{Verify}(c)$.
   - If $b = \mathsf{true}$, send $(\mathsf{Decrypt\text{-}good\text{-}c}, sid)$ to $\mathcal{S}$.
   - If $b = \mathsf{false}$ or $b = \perp$, send $(\mathsf{Decrypt\text{-}bad\text{-}c}, sid)$ to $\mathcal{S}$. Proceed to the point 4) with $y_1 = y_2 = \perp$.
3) The final output y_1 for P_1 depends on whether the server is corrupt.
   If the server is *honest*, the decryption can only be delayed:
   - Upon receiving $(\mathsf{Decrypt\text{-}complete}, sid, \mathsf{Dec})$ from $\mathcal{S}$, take $y_1 \leftarrow Decrypt(\mathsf{Dec}, c)$, and $y_2 \leftarrow \top$.

   If the server is *corrupted*, $\mathcal{S}$ tells whether decryption succeeded.
   - Upon receiving $(\mathsf{Decrypt\text{-}complete}, sid, \mathsf{Dec})$ from $\mathcal{S}$, take $y_1 \leftarrow Decrypt(\mathsf{Dec}, c)$.
   - Upon receiving $(\mathsf{Decrypt\text{-}fail}, sid)$ from $\mathcal{S}$, take $y_1 \leftarrow \perp$.
4) If the server is corrupted, wait for $(\mathsf{Output}, 2, y_2')$ from $\mathcal{A}$ and take $y_2 \leftarrow y_2'$.
5) Send $(\mathsf{Decrypted}, sid, y_1)$ to P_1 and $(\mathsf{Decrypted}, sid, y_2)$ to P_2.

Decryption (corrupted client)
On message $(\mathsf{Decrypt}, sid, c)$ from party P_1 and $(\mathsf{Decrypt}, sid)$ from party P_2:

1) Set $ctr_{\mathsf{dec}} \leftarrow ctr_{\mathsf{dec}} + 1$ and send $(\mathsf{Decrypt\text{-}init}, sid, c)$ to $\mathcal{S}$.
2) $\mathcal{S}$ tells whether decryption succeeded:
   - Upon receiving $(\mathsf{Decrypt\text{-}complete}, sid)$ from $\mathcal{S}$, take $y_2 \leftarrow \top$.
   - Upon receiving $(\mathsf{Decrypt\text{-}fail}, sid)$ from $\mathcal{S}$, take $y_2 \leftarrow \perp$.
3) Wait for $(\mathsf{Output}, 1, y_1)$ from $\mathcal{A}$.
4) Send $(\mathsf{Decrypted}, sid, y_1)$ to P_1 and $(\mathsf{Decrypted}, sid, y_2)$ to P_2.

At any point of time (corrupted client). On message $(\mathsf{Decrypt\text{-}msg}, sid, \mathsf{Dec}, c')$ from $\mathcal{S}$:

1) Compute $m = Decrypt(\mathsf{Dec}, c')$.
2) Set $ctr_{\mathsf{dec}} \leftarrow ctr_{\mathsf{dec}} - 1$. If $ctr_{\mathsf{dec}} < 0$, then stop.
3) Send $(\mathsf{Decrypted}, sid, m)$ to $\mathcal{S}$.

Fig. 1. Ideal encryption functionality $\mathcal{F}$

.

no longer be restricted like that. We should allow $\mathcal{S}$ to query for the row (m, c) for a ciphertext c its own choice. Moreover, we allow to make this query *after* the decryption session has ended, since the corrupted client may potentially be able to undo his own blinding in multiple ways after getting response from the server, thus being able to choose between decrypting several different ciphertexts later. Nevertheless, the adversary is only allowed to decrypt *at most one message per decryption query* coming from the environment.

We deliberately have omitted the case where both the server and the client are corrupted, as we do not aim to achieve any security guarantees in this case. The adversary would get full control over the key generation, encryption and decrypton, and would be given all messages and ciphertexts that any party (including the honest ones) receives as an input.

Next, we discuss how the ideal functionality that we defined sidesteps the problems we identified with the previous security definitions. As a counterexample for P-IND-RCCA, we proposed a scheme where the adversary may obtain the secret key share of an honest party if he gets the secret key share of a corrupted party. If such a protocol were run in the UC model with either the client or the server corrupted, the adversary would be able to reconstruct the private key sk and decrypt the ciphertexts that have been encrypted with pk. The environment could distinguish the real protocol from $\mathcal{F}$ by decrypting (without involving $\mathcal{F}$) an encryption of a message m generated by $\mathcal{F}$, i.e. without letting $\mathcal{F}$ tell m to $\mathcal{S}$, getting a plaintext that does not depend on m.

Compared to the definition of Green, our execution of blinded decryption in $\mathcal{F}$ is not atomic, and the adversary has full control over the corrupted client, and thus could set up $b_{evil} = 1$. The environment would get sk and hence could distinguish the real protocol from $\mathcal{F}$ by decrypting (without involving $\mathcal{F}$) an encryption of a message m generated by $\mathcal{F}$, getting a plaintext that does not depend on m. Non-atomic execution of blinded decryption allows a corrupted client can decrypt a ciphertext of his own choice, which would not be possible using atomic execution. We explicitly let the corrupted client access a selected entry from the table T, which can be viewed as access to a standard decryption oracle. If the client is not corrupted, then the access to the decryption oracle is limited and controlled by the environment.

V. Secure implementation

While we have critisized the *definitions* of Green [23] and Blazy et al. [24], we may wonder whether their implementations, even though they are for different functionalities, are adaptable into a secure implementation of $\mathcal{F}$. We can immediately answer that question in negative for Green: his outsourced blinded decryption is not an instance of threshold decryption, and does not support the user independently finding out whether a ciphertext is valid. The latter is necessary for chosen-ciphertext security in threshold decryption schemes.

Blazy et al. [24] discuss whether their construction could be realizable in universally composable manner, and answer that question in negative. But that answer stems from the inability to simulate the adversary guessing the password of an honest party. It is possible that if the roles of *user* and *token* in their protocol were combined into one, then their construction would provide a secure implementation of $\mathcal{F}$.

However, the blind decryption step of Blazy et al. [24] uses Groth-Sahai proofs [30] to prove that a blinded ciphertext has been constructed from a ciphertext that was accompanied by proofs of validity. This means that their construction is complex, and heavily based on bilinear pairings, leading to both relatively heavy computations and to large message sizes.

We securely implement $\mathcal{F}$ under common cryptographic assumptions: hardness of discrete logarithm (DL), one-more CDH, and the existence of additively homomorphic encryption systems, for which the proofs of equality of plaintexts can be given. In this paper, we will instantiate the latter with Paillier encryption, which is IND-CPA (indistinguishability against chosen-plaintext attacks) secure under the Decisional Composite Residuosity Assumption (DCRA) [31]. Remarkably, our construction DVPS does not require bilinear pairings. Compared to [23], [24], a potential theoretical shortcoming of our construction is its use of the Random Oracle Model (ROM), but considering its prevalence in constructions employed in practice [32], [33], we do not see it as a weakness.

Our construction DVPS builds on top of the IND-CCA secure TDH1 cryptosystem of Gennaro and Shoup [2], adding to it ciphertext blinding and unblinding operations that are used when the client queries the server during decryption protocol. TDH1 builds upon the ElGamal key encapsulation mechanism (KEM), adding to its ciphertexts c the non-interactive zero-knowledge (NIZK) proofs π that someone (e.g. the entity encrypting the plaintext m) knows the randomness r used for the encryption. In DVPS, both c and the proof have to be blinded. To blind the proof, we need *malleable* NIZK proofs [34], but the known constructions are based on bilinear pairings. Fortunately, we do not need malleable NIZK in its full generality (as used by Blazy et al. [24]); we only need the server to be convinced by a proof that the client malleated. There exist malleable *designated verifier proofs* without pairings [35] that will be used in our construction.

Our ciphertexts will thus contain a proofs of knowledge of the randomness r, designated to be verified by the server. The client also needs to be convinced that r is known; we use "usual" NIZK proofs for that. Our ciphertexts also contain a third kind of proof, convincing the client that an honest server is going to accept the designated verifier proof. In [36], the DFN proofs of [35] have been found vulnerable against selective failure attacks that allow to find the secret of the verifier bit by bit. This can be mitigated by letting the server stop after a certain number of failed proofs and require a fresh key generation. The ability of the client to verify whether the server will accept the proof is needed to avoid attacks where an external encrypting party produces bad ciphertexts that will be rejected by the server and cause denial of service for an honest client. The details of this verification are given in App. B.

As next, we describe the cryptographic building blocks used

in our construction. The construction itself follows in Sec. V-B.

A. Preliminaries

We write $x_1, \ldots, x_n \leftarrow\!\$\ X$ to denote that the values $x_1, \ldots, x_n$ are uniformly, independently sampled from a set X. We also write $x \leftarrow\!\$\ \mathsf{X}(\ldots)$ to denote that x is returned by a stochastic computation X. We let $\mathbf{I}(n)$ denote the set $\{0, 1, \ldots, 2^n - 1\}$.

1) Hardness assumptions: Let $\mathbb{G}$ be a cyclic group of size p, with generator g. The *discrete logarithm problem* is to find $n \in \mathbb{Z}_p$, such that $g^n = h$, for a value $h \leftarrow\!\$\ \mathbb{G}$. The *decisional Diffie-Hellman (DDH) problem* is to distinguish tuples of the form (g, g^x, g^y, g^{xy}) (called *Diffie-Hellman tuples*) from the tuples of the form (g, g^x, g^y, g^z) for $x, y, z \leftarrow\!\$\ \mathbb{Z}_p$. The *computational Diffie-Hellman (CDH) problem* is, given (g, g^x, g^y) for $x \leftarrow\!\$\ \mathbb{Z}_p$ and $y \leftarrow\!\$\ \mathbb{Z}_p$, come up with g^{xy}.

The *one-more (static) computational Diffie-Hellman (CDH) problem* is, given (g, g^x) for $x \leftarrow\!\$\ \mathbb{Z}_p$, access to the oracle $(\cdot)^x$, and $h_0, \ldots, h_n \leftarrow\!\$\ \mathbb{G}$, come up with $y_0, \ldots, y_n$ satisfying $y_i = h_i^x$ while querying $(\cdot)^x$ at most n times. If $n = 0$, then we have the usual CDH problem. A problem is *hard* if all the efficient algorithms have at most negligible advantage (over a trivial algorithm) of solving it.

2) ElGamal KEM and encryption scheme: Key Encapsulation Mechanism consists of the key generation algorithm that outputs a pair of private and public key $(\mathsf{sk}, \mathsf{pk})$; encapsulation algorithm that takes as input a public key and outputs a shared secret ss and ciphertext c; decapsulation algorithm that on input of ciphertext c and private key sk outputs a shared secret ss. Figure 2 presents ElGamal KEM.

Key Generation:	Encapsulation:	Decapsulation:
1 : $\mathsf{sk} \leftarrow\!\$\ \mathbb{Z}_p$	1 : $r \leftarrow\!\$\ \mathbb{Z}_p$	1 : $ss = c^{\mathsf{sk}}$
2 : $\mathsf{pk} = g^{\mathsf{sk}}$	2 : $ss = \mathsf{pk}^r$	2 : Return ss
3 : Return $(\mathsf{sk}, \mathsf{pk})$	3 : $c = g^r$	
	4 : Return (ss, c)	

Fig. 2. ElGamal KEM

IND-CPA security of ElGamal KEM is equivalent to the hardness of DDH in the used group $\mathbb{G}$. In *hashed ElGamal* KEM, the shared secret is $H'(\mathsf{pk}^r)$ for some hash function H' that we model as a random oracle; its security is equivalent to the hardness of CDH.

A KEM can be turned into a public-key encryption scheme by combining it with a data encapsulation mechanism (DEM). In random oracle model, we may imitate the SKE2 DEM [37] and define the encryption of the message m with the key k as $\mathsf{SE}(k, m) := (H'(k) \oplus m, H''(k, m))$ for some hash functions (random oracles) H' and H''. The respective decryption function $\mathsf{SD}(k, (c_1, c_2))$ computes $m \leftarrow H'(k) \oplus c_1$, checks that $H''(k, m) = c_2$, and outputs m.

3) Random oracles: In constructions and proofs in the *Random Oracle Model* (ROM) [38], all the parties are assumed to have an access to one or several random functions H, where the value of the function at each point is a random variable uniformly distributed over a fixed set, and the values at different points are independent of each other. These random functions may have different codomains, e.g. the set of bitstrings of a certain length, some group $\mathbb{G}$, etc., depending on the needs of the construction. The domain of a random oracle does not have to be fixed; anything encodable as a bitstring may be an input to it. In proofs, the simulator is in control of the output values of random oracles, defining them as it sees fit, as long as the distribution stays the same.

4) (Non-interactive) zero-knowledge proofs: A Σ-*protocol* for a binary relation R is a three-move protocol between two parties — prover and verifier — both knowing a value x, where the prover tries to convince the verifier that he knows some value w, such that $R(x, w)$ holds. The first message α is sent by the prover, followed by the verifier generating a fresh, independent random value β and sending it as the second message. After the prover has sent the third message γ, the verifier runs a check on $(x, \alpha, \beta, \gamma)$ and either accepts or rejects. A Σ-protocol must have *special honest-verifier zero-knowledge (ZK)*: given (x, β), one should be able to generate (α, γ) so, that the verifier cannot distinguish them from the real protocol runs. A Σ-protocol must also be *specially sound*: given $(x, \alpha, \beta_1, \beta_2, \gamma_1, \gamma_2)$ with $\beta_1 \neq \beta_2$, such that the verifier accepts both $(x, \alpha, \beta_1, \gamma_1)$ and $(x, \alpha, \beta_2, \gamma_2)$, it must be possible to find w.

A Σ-protocol for R is an (interactive) zero-knowledge proof, assuming the verifier does not deviate from the protocol. Random oracles can be used to turn Σ-protocols to non-interactive ZK (NIZK) proofs using the *Fiat-Shamir (FS) transform* [39]: instead of receiving β from the verifier, prover himself computes it as $\beta \leftarrow H(x, \alpha, ctx)$, where ctx describes the context in which the prover creates this proof. To verify the proof, the verifier recomputes β. In our construction, in some proofs it is easier to assume that the part of the proof is β, not α. The verifier then computes α that would satisfy the proof from x and ctx, and checks whether $\beta = H(x, \alpha, ctx)$.

A *Designated-Verifier* NIZK (DVNIZK) proof is a NIZK proof that can convince only a single verifier. Such proofs may be cheaper to use than publicly verifiable proofs, and they may have some additional properties. Damgård et al. [35] have introduced a method ("DFN proofs") that applies to such Σ-protocols where γ is computed as a linear function of β, turning them into DVNIZK proofs. In their method, the verifier generates a public-private key pair $(\mathsf{ek}, \mathsf{vk})$ for additively homomorphic encryption. He also selects a random β and computes $B \leftarrow\!\$\ \mathcal{E}_{\mathsf{ek}}(\beta)$. Public key ek and ciphertext B are given to the prover. The proof is (α, Γ), where α is the same as in the original Σ-protocol and Γ is an encryption of γ, computed using B and the homomorphic properties of the encryption scheme. Verifier can decrypt Γ and perform the verification as in the original Σ-protocol. It turns out the the DFN proofs have the malleability properties that we need.

For the publicly verifiable proofs of knowledge of the exponent r in an ElGamal ciphertext, we use Σ-protocols that are made non-interactive using the FS transform. As the construction of TDH1 [2] already observed, a simple

Schnorr proof of knowing r [33] seems to be insufficient for this purpose (unless we introduce an additional extractability assumption), because the simulator will not be able to decrypt certain ciphertexts [40]. A more complex proof is necessary. They start from a *DDH proof* $\mathsf{DHP}^H[r \mid^{u,v}_{g,h} | ctx]$ [41] — a NIZK proof that $\log_g u = \log_h v$ (i.e. it is not a proof *of knowledge*), given in *context* ctx, where $r \in \mathbb{Z}_p$ is that discrete logarithm and H is a hash function, modeled as a random oracle. Denote the checking procedure by $\mathsf{ChP}^H[\pi \mid^{u,v}_{g,h} | ctx]$. Both proof and verification procedure are given in Figure 3.

$\mathsf{DHP}^H[r \mid^{u,v}_{g,h} | ctx]$:

1 : $s \leftarrow\$ \mathbb{Z}_p$
2 : $\alpha \leftarrow g^s, \quad \alpha' \leftarrow h^s$
3 : $\beta \leftarrow H(g, h, u, v, \alpha, \alpha', ctx) \in \mathbb{Z}_p$
4 : $\gamma \leftarrow s + r \cdot \beta$
5 : return $\pi \leftarrow (\beta, \gamma)$

$\mathsf{ChP}^H[\pi \mid^{u,v}_{g,h} | ctx]$:

1 : $\alpha \leftarrow g^\gamma / u^\beta$
2 : $\alpha' \leftarrow h^\gamma / v^\beta$
3 : **assert** $\beta = H(g, h, u, v, \alpha, \alpha', ctx) \in \mathbb{Z}_p$

Fig. 3. NIZK proof that $\log_g u = \log_h v$

From DDH proofs we get *simulatable proofs of knowledge of exponent* $\mathsf{KnE}^{H,\tilde{H}}_{\mathsf{d}}[r \mid^u_g | ctx]$, where $\mathsf{d} \in \{1, -1\}$ is needed for exponent extraction an in described in Sec. VI. These prove that someone knows the value $r = \log_g u$. The construction makes use of two hash functions, both modeled as random oracles, where H returns elements of $\mathbb{Z}_p$ and $\tilde{H}$ returns elements of $\mathbb{G}$. The checking procedure $\mathsf{ChE}^{H,\tilde{H}}_{\mathsf{d}}[\pi \mid^u_g | ctx]$ checks the underlying DDH proof. Both proof and verification procedure are given in Figure 4.

$\mathsf{KnE}^{H,\tilde{H}}_{\mathsf{d}}[r \mid^u_g | ctx]$:

1 : $h \leftarrow \tilde{H}(g, u, ctx) \in \mathbb{G}$
2 : $v \leftarrow h^{r^{\mathsf{d}}}$
3 : if $\mathsf{d} = 1$, then $\pi \leftarrow \mathsf{DHP}^H[r \mid^{u,v}_{g,h} | ctx]$
4 : if $\mathsf{d} = -1$, then $\pi \leftarrow \mathsf{DHP}^H[r \mid^{u,h}_{g,v} | ctx]$
5 : return $\pi' \leftarrow (\pi, v)$

$\mathsf{ChE}^{H,\tilde{H}}_{\mathsf{d}}[\pi' \mid^u_g | ctx]$:

1 : $h \leftarrow \tilde{H}(g, u, ctx) \in \mathbb{G}$
2 : $(\pi, v) \leftarrow \pi'$
3 : if $\mathsf{d} = 1$, then **assert** $\mathsf{ChP}^H[\pi \mid^{u,v}_{g,h} | ctx]$
4 : if $\mathsf{d} = -1$, then **assert** $\mathsf{ChP}^H[\pi \mid^{u,h}_{g,v} | ctx]$

Fig. 4. Proof of knowledge of exponent in a group $\mathbb{G}$

B. Our construction

Our construction adds to the TDH1 scheme (with differently shared private key) the malleable DVNIZK proofs, and the proofs of these proofs being accepted by the server. TDH1 sets up a trapdoor with the proof of knowledge of the exponent r. Similarly, we need a number of trapdoors in the added proofs in order to be able to simulate a corrupted party.

Public parameters of DVPS contain the cyclic group $\mathbb{G}$ of size p with generator g. They also fix a homomorphic encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ for DFN proofs, where $\mathcal{K}$ generates a private and public key pair, $\mathcal{E}$ encrypts, and $\mathcal{D}$ decrypts. In this paper, that scheme will be the Paillier encryption scheme, working with moduli N of bit-length $\nu \in \mathbb{N}$. The public parameters moreover contain the statistical *soundness* and *privacy* parameters $\rho, \kappa \in \mathbb{N}$ for the computations over the integers (in the hidden-order group $\mathbb{Z}^*_N$). Typically, ρ and κ are between 80 and 256, and it is reasonable to assume that $\rho \leq \kappa \leq d$, where d is the bit-length of p. We need ν to be several times larger than ρ, κ, and d; this will be satisfied by the Paillier encryption (where N is a RSA modulus) and natural instantiations of $\mathbb{G}$ (as elliptic curve groups). Let $\mathbf{Cn}$ be the set of possible *random coins* used during encryption. Finally, the public parameters contain the definitions of several random functions, modelled as random oracles, introduced below. All the sub-routines that are used in our construction are listed in the Table I to ease the understanding of the protocol.

$\mathsf{SE}(k, m)$ and $\mathsf{SD}(k, c)$	encryption and decryption procedures of DEM for a key k, a plaintext m and a ciphertext c
$\mathsf{DHP}^H[r \mid^{u,v}_{g,h} \mid ctx]$ and $\mathsf{ChP}^H[\pi \mid^{u,v}_{g,h} \mid ctx]$	proof that $\log_g u = \log_h v$, given in *context* ctx, where $r \in \mathbb{Z}_p$ is that discrete logarithm
$\mathsf{KnE}^{H,\tilde{H}}_{\mathsf{d}}[r \mid^u_g \mid ctx]$ and $\mathsf{ChE}^{H,\tilde{H}}_{\mathsf{d}}[\pi \mid^u_g \mid ctx]$	proof generation and verification for the proof of knowledge of exponent $r = \log_g(u)$ in context ctx
$\mathsf{DVP}(r, r', \mathbf{r} \mid^{\alpha,\Gamma}_{g,u} \mid B)$ and $\mathsf{DVC}(\pi \mid^{\alpha,\Gamma}_{g,u} \mid B)$	proof generation and verification for the proofs of validity of DVNIZK proof (α, Γ) that uses randomness r, r' and $\mathbf{r}$ in the context of encrypted challenge B
$(\mathcal{K}, \mathcal{E}, \mathcal{D})$	Paillier key generation, encryption and decryption

TABLE I
SUB-ROUTINES USED IN OUR CONSTRUCTION

Key generation of DVPS is given in Fig. 5. During the key generation, the client and the server respectively select private key shares $\mathsf{sk}_1 \leftarrow\$ \mathbb{Z}_p$ and $\mathsf{sk}_2 \leftarrow\$ \mathbb{Z}_p$, compute public key shares $\mathsf{pk}_i = g^{\mathsf{sk}_i}$ (line 1), and exchange the latter values using hash commitments (lines 2–4). Beside pk_i, the client and server also exchange the proofs $\mathsf{KnE}^{H_0,\tilde{H}_0}_{-1}[\mathsf{sk}_i \mid^{\mathsf{pk}_i}_g]$ that they know their respective private keys (lines 5–7), this prevents a malicious party from choosing their public key share based on the share of the honest party without knowing corresponding exponent. They will verify the proofs (lines 8–9), and define combined public key $\mathsf{pk} = \mathsf{pk}^{\mathsf{sk}_i}_{3-i}$ (line 10), i.e. the private key is *multiplicatively*, not additively shared.

Having set up pk, the parties continue with setting up the public and private values for DVNIZK proofs. DVNIZK proofs are needed for the encrypting party to prove that they know the randomness used to generate ElGamal ciphertext. Moreover, chosen DVNIZK proofs allow client to blind the proof before sending it in the decryption query to the server

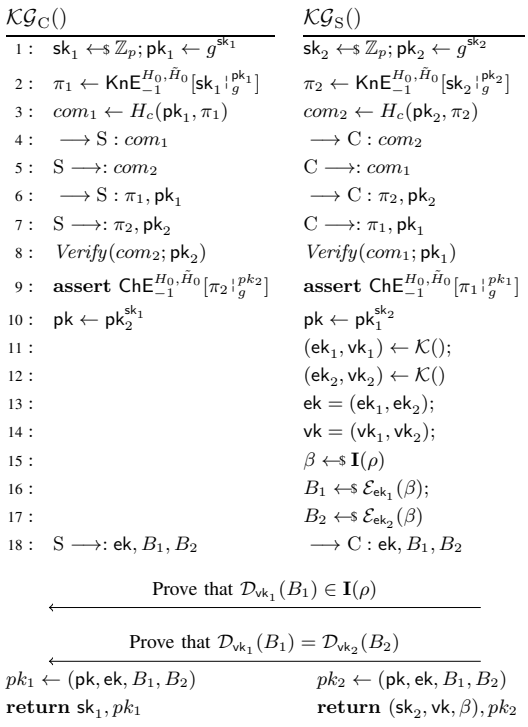


Fig. 5. Key generation for client and server in our construction

(who is acting as a designated verifier). The server generates two key pairs $(\mathsf{ek}_1, \mathsf{vk}_1)$ and $(\mathsf{ek}_2, \mathsf{vk}_2)$ for the homomorphic encryption scheme, where ek_i is the public key and vk_i the private key (line 11). The set-up also consists of public ciphertexts $B_1 \leftarrow\$ \mathcal{E}_{\mathsf{ek}_1}(\beta)$ and $B_2 \leftarrow\$ \mathcal{E}_{\mathsf{ek}_2}(\beta)$ (line 12), where $\beta \leftarrow\$ \mathbf{I}(\rho)$ is generated and kept secret by the server. The necessity of generating two instances comes from the security and we provide detailed explanation in Section VI. Server sends $\mathsf{ek}= (\mathsf{ek}_1, \mathsf{ek}_2)$ and B_1 and B_2 to the client. Finally, the set-up for DFN proofs consists of the server proving that B_1 indeed encrypts a value at most ρ bits long and proving that B_1 and B_2 encrypt the same value.

The public key is $(\mathsf{pk}, (\mathsf{ek}_1, \mathsf{ek}_2), B_1, B_2)$, allowing an encryptor to create the ElGamal ciphertext with the public key pk, and the two DVNIZK proofs of knowledge of the random exponent of the ciphertext, using (ek_1, B_1) and (ek_2, B_2) (the NIZK proof of knowledge of the exponent, as well as the proofs of the DVNIZK proofs being correct, are standard FS transforms of Σ-protocols). Client keeps sk_1 as his keyshare. Server keeps $(\mathsf{sk}_2, (\mathsf{vk}_1, \mathsf{vk}_2), \beta)$ as his keyshare. The both also keep $\mathsf{pk}_1, \mathsf{pk}_2$.

Encryption and **decryption** are given in Fig. 6. As the operations with B_1 and B_2 are mostly similar, we abbreviate the write-up by introducing the convention that the parameter ι ranges over $\{1, 2\}$ and the operations of defining values X_1 and X_2 are given only once by stating how X_ι is computed.

The encryption of the message m starts by computing its "standard" ElGamal ciphertext (u, c_2) (lines 1–3). In line 4–7, we compute the two DVNIZK proofs of the encryptor knowing r. Those proofs can only be verified by the server during the decryption process. The value α_1 is the first message of both proofs; it is computed as in a Schnorr proof of knowledge, raising the generator g to a random power r_1. The second message is the server's secret β, and the third message Γ_ι is an encryption of $\gamma = r_1 + \beta r$.

In line 8, we compute the (usual) NIZK proof that the encryptor knows r. The same proof, with the same trapdoor is present in TDH1. This proof can be verified by the client. Finally, in line 9 of $\mathcal{E}nc$ we compute the proofs π_ι ($\iota \in \{1, 2\}$) that an honest server will accept the designated verifier proofs (α_1, Γ_ι), while verifying them in the manner described in Sec. V-A. This proof is given using standard cryptographic techniques; we present it in App. B. No trapdoor is embedded in π_ι. The proof is required for the client not to engage in decryption of an incorrect ciphertext. If the proof does not verify on the client side, the client does not initiate protocol with the server. The final ciphertext is $[c_1 = (u, \alpha_1, \Gamma_1, \Gamma_2, \pi, \pi_1, \pi_2), c_2]$.

One has to be careful with the moduli: while the exponents of g are naturally taken modulo p, the computations under encryption are done modulo the RSA/Paillier modulus N. As N is much larger than p, we can just make sure that certain computations do not overflow. In particular, we want $\gamma = r_1 + \beta r$ be less than N. We also want r_1 to hide βr. Obtaining the perfect security may be hard, but we can get statistical security. The value βr is $\rho + d$ bits long. Hence we let r_1 to be $\rho + d + \kappa$ bits long (which must still be less than the size ν of N). This is similar to homomorphic commitments to integers [42], [43].

During the **decryption**, the client first verifies two of three proofs in lines 2–3. He will then construct the blinded encapsulation u' (line 6) by raising u to a random exponent z. Hence u' is independent of u. In lines 7–8, the client will then blind the proof (α_1, Γ_ι) of knowledge of $r = \log_g u$ to get the proof $(\alpha_1', \Gamma_\iota')$ of knowledge of $rz = \log_g u'$. $(\alpha_1^z, \Gamma_\iota^z)$ is such a proof, as long as the plaintext $z\gamma = \mathcal{D}_{\mathsf{vk}_\iota}(\Gamma_\iota^z)$ is not reduced modulo N. That reduction indeed does not happen, because $z\gamma$ is at most $\rho + 2d + \kappa \leq \nu$ bits long.

However, $z\gamma$ may leak something about z to the server, once the latter has decrypted Γ_ι^z and obtained it. Indeed, z is a factor of $z\gamma$. Hence we will further additively mask $(\alpha_1^z, \Gamma_\iota^z)$, using z'. We get statistical security if we let z' to be at last κ bits longer than $z\gamma$.

The client proceeds with generating proof π' that they know their private key in the context of blinded ciphertext that is being decrypted (line 9). This proof authenticates client to the server, demonstrating that they know the private key share. The clients sends u', π' and the malleated DVNIZK proofs to the server (line 10), who verifies received proofs (line 2–5 in $\mathcal{DC}_{\mathrm{S}}$). The rest of the protocol is straightforward, computing $\mathsf{pk}^r = (((u')^{\mathsf{sk}_2})^{\mathsf{sk}_1})^{1/z}$ from u'. The server does

$\mathcal{E}nc(m, \mathsf{pk}, \mathsf{ek}_1, \mathsf{ek}_2, B_1, B_2)$

1 : $r \leftarrow\!\!\$\ \mathbb{Z}_p$
2 : $u \leftarrow g^r$
3 : $c_2 \leftarrow \mathsf{SE}(\mathsf{pk}^r, m)$
4 : $r_1 \leftarrow\!\!\$\ \mathbf{I}(\rho + d + \kappa)$
5 : $\mathbf{r}_\iota \leftarrow\!\!\$\ \mathbf{Cn}$
6 : $\alpha_1 \leftarrow g^{r_1 \bmod p}$
7 : $\Gamma_\iota \leftarrow \mathcal{E}_{\mathsf{ek}_\iota}(r_1; \mathbf{r}_\iota) \cdot B_\iota^r$
8 : $\pi \leftarrow \mathsf{KnE}_1^{H_1, \tilde{H}_1}[r \upharpoonright_g^u | \alpha_1, \Gamma_1, \Gamma_2]$
9 : $\pi_\iota \leftarrow \mathsf{DVP}(r, r_1, \mathbf{r}_\iota \upharpoonright_{g,u}^{\alpha_1, \Gamma_\iota} | B_\iota)$
10 : $c_1 \leftarrow (u, \alpha_1, \Gamma_1, \Gamma_2, \pi, \pi_1, \pi_2)$
11 : **return** c_1, c_2

$\iota \in \{1, 2\}$

$\mathcal{DC}_{\mathrm{C}}(c_1, c_2, \mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}, \mathsf{ek}_1, \mathsf{ek}_2, B_1, B_2)$

1 : $(u, \alpha_1, \Gamma_1, \Gamma_2, \pi, \pi_1, \pi_2) \leftarrow c_1$
2 : **assert** $\mathsf{ChE}_1^{H_1, \tilde{H}_1}[\pi \upharpoonright_g^u | \alpha_1, \Gamma_1, \Gamma_2]$
3 : **assert** $\mathsf{DVC}(\pi_\iota \upharpoonright_{g,u}^{\alpha_1, \Gamma_\iota} | B_\iota)$
4 : $z \leftarrow\!\!\$\ \mathbb{Z}_p,\ z' \leftarrow\!\!\$\ \mathbf{I}(\rho + 2d + 2\kappa)$
5 : $\mathbf{r}'_\iota \leftarrow\!\!\$\ \mathbf{Cn}$
6 : $u' \leftarrow u^z$
7 : $\alpha'_1 \leftarrow \alpha_1^z \cdot g^{z' \bmod p}$
8 : $\Gamma'_\iota \leftarrow \Gamma_\iota^z \cdot \mathcal{E}_{\mathsf{ek}_\iota}(z'; \mathbf{r}'_\iota)$
9 : $\pi' \leftarrow\!\!\$\ \mathsf{KnE}_{-1}^{H_2, \tilde{H}_2}[\mathsf{sk}_1 \upharpoonright_g^{\mathsf{pk}_1} | u', \alpha'_1, \Gamma'_1, \Gamma'_2]$
10 : $\longrightarrow \mathrm{S} : u', \alpha'_1, \Gamma'_1, \Gamma'_2, \pi'$
11 : $\mathrm{S} \longrightarrow : w, \pi''$
12 : **assert** $\mathsf{ChP}^{H_3}[\pi'' \upharpoonright_{\mathsf{pk}_1, u'}^{\mathsf{pk}, w}]$
13 : **return** $\mathsf{SD}(w^{\mathsf{sk}_1 / z}, c_2)$

$\mathcal{DC}_{\mathrm{S}}(\mathsf{sk}_2, \mathsf{pk}_1, \mathsf{pk}, \mathsf{vk}_1, \mathsf{vk}_2, \beta)$

1 : $\mathrm{C} \longrightarrow : u', \alpha'_1, \Gamma'_1, \Gamma'_2, \pi'$
2 : $\gamma' \leftarrow \mathcal{D}_{\mathsf{vk}_1}(\Gamma'_1)$
3 : **assert** $\gamma' = \mathcal{D}_{\mathsf{vk}_2}(\Gamma'_2)$
4 : **assert** $g^{\gamma'} = \alpha'_1 \cdot (u')^\beta$
5 : **assert** $\mathsf{ChE}_{-1}^{H_2, \tilde{H}_2}[\pi' \upharpoonright_g^{\mathsf{pk}_1} | u', \alpha'_1, \Gamma'_1, \Gamma'_2]$
6 : $w \leftarrow (u')^{\mathsf{sk}_2}$
7 : $\pi'' \leftarrow \mathsf{DHP}^{H_3}[\mathsf{sk}_2 \upharpoonright_{\mathsf{pk}_1, u'}^{\mathsf{pk}, w}]$
8 : $\longrightarrow \mathrm{C} : w, \pi''$
9 : **return** $\top$

Fig. 6. Encryption and decryption in our construction

the first exponentiation in line 6 of $\mathcal{DC}_{\mathrm{S}}$ and sends it back to client (line 8), accompanying it with a proof (line 7) that he performed the exponentiation correctly. After verifying this proof (line 12 of $\mathcal{DC}_{\mathrm{C}}$) , the client performs the second and third exponentiation, and uses the obtained pk^r to find the plaintext m from c_2.

In the presented protocol, client contacts the server, which will then use its private key share. We do not include (neither here nor in the definition of $\mathcal{F}$ in Fig. 1) the details on how the server is *activated*. Indeed, if the server is always responsive, then an adversary with client's keyshare will be able to decrypt ciphertexts by masquerading as the client to the server. Formally, the activation of the server is decided in the environment. In practice [5], [21], client's keyshare is stored encrypted with a PIN, with guesses of the PIN impossible to verify without contacting the server. A mechanism for detecting the existence of several clones of keyshare is also used [22].

VI. Security proofs

We begin by introducing an augmented version of the one-more CDH problem, whose hardness is a stronger assumption than hardness of the standard one-more CDH problem. In the *one-more (static) CDH problem with tests*, one is given (g, g^x) with $x \leftarrow\!\!\$\ \mathbb{Z}_p$, random $h_0, \ldots, h_n \in \mathbb{G}$, the oracle $(\cdot)^x$, and the oracle $\mathsf{Check}(\cdot)$, where $\mathsf{Check}(h)$ returns i such that $h = h_i^x$, and $\perp$ if there is no such i. The goal of the solver is to invoke $\mathsf{Check}(h_i^x)$ for all $i \in \{0, \ldots, n\}$, while making at most n calls to the oracle $(\cdot)^x$. The calls to Check-oracle are not restricted in number. While the Check-oracle may give the solver some extra power in the Standard Model, Bauer et al. [44, Thm. 5] show that nothing is gained in the Algebraic Group Model (AGM) [45].

Our scheme is instantiated with the simulatable proofs of knowledge of exponent $r = \log_g(u)$ (Fig. 4) that additionally allow the simulator to raise an arbitrary element $z \in \mathbb{G}$ of its choice to the power of r^{d} where $\mathsf{d} \in \{1, -1\}$. The simulator has to choose the value z at the time the adversary computes the corresponding proof. If the simulator wants to obtain $z^{r^{\mathsf{d}}}$, it will follow the instructions from Figure 7. The quantity $z^{r^{\mathsf{d}}}$ can be computed as soon as the simulator gets the proof.

Raising $z \in \mathbb{G}$ to the power r^{d} for $\mathsf{d} \in \{1, -1\}$:

1) generate $t \leftarrow\!\!\$\ \mathbb{Z}_p$
2) program $\tilde{H}_i$ to return $h = z^{1/t}$ when the adversary queries it with g, u, ctx
3) at any point later, after obtaining the proof (π, v) from the adversary, compute $z^{r^{\mathsf{d}}} = v^t$. If the proof π is valid, then $v = h^{r^{\mathsf{d}}}$.

Fig. 7. Raising $z \in \mathbb{G}$ to the power of r^{d} for $\mathsf{d} \in \{1, -1\}$

Additionally, in proofs of security against the malicious server, the simulator may need to know the value β. Hence we give the proof of $\beta \in \mathbf{I}(\rho)$ in a manner that allows the simulator to extract it. The proof can be given using standard cryptographic techniques, we describe it in App. A-D. We believe that as key generation is a seldomly-performed operation, its high efficiency is relatively less important, as long as it can be performed in reasonable time. Moreover, in proofs of security against malicious client, the simulator may need to know the witnesses that the client has used in the creation of DFN proofs. While Damgård et al. [35] introduced a *knowledge assumption* stating that such extraction is possible, we choose to add more components to the ciphertexts, allowing us to assume the hardness of DDH, one-more CDH and DCRA only. Thus, during the key generation the server also computes $B_2 \leftarrow\!\!\$\ \mathcal{E}_{\mathsf{ek}_2}(\beta)$. When the simulator is acting as the server, it will be able to make B_1 and B_2 contain different values $\beta_1 \neq \beta_2$, faking the plaintext equality proof. It will then be able to use the special soundness of the Σ-protocols to

find the witness. Again, the proof can be given using standard cryptographic techniques, see App. A-B.

Theorem 1. The protocol set DVPS in Figures 5 and 6 securely implements functionality $\mathcal{F}$ in presence of malicious static adversary under one-more CDH assumption with tests.

In this section, we give a proof sketch of Theorem 1. The full proof can be found in App. C.

In order to show universal composability security of our threshold decryption protocol, we construct a simulator $\mathcal{S}$ such that for each adversary $\mathcal{A}$ attacking the protocol, the environment $\mathcal{Z}$ cannot distinguish whether it is interacting with the real protocol and the adversary $\mathcal{A}$, or the ideal functionality $\mathcal{F}$ and the "ideal" adversary (consisting of $\mathcal{A}$ and $\mathcal{S}$). We start with defining $\mathcal{S}$ and how it interacts with the ideal functionality and the adversary in case of different parties being corrupted.

Initialization:

We assume that the corruptions are known already in the initialization phase. There is a parameter n denoting the number of encryptions considered in this session. From an external challenger, $\mathcal{S}$ receives as an input $\mathsf{pk} = g^{\mathsf{sk}}$ (where the sk is only known to the challenger) and also the following:

- If the client is *corrupted*, $\mathcal{S}$ receives $u_0, \ldots, u_n$ for some parameter n, such that $(g, \mathsf{pk}, u_0, \ldots, u_n)$ is an instance of a one-more CDH problem. $\mathcal{S}$ also gets access to an oracle $(\cdot)^{\mathsf{sk}}$. Intuitively, if the number of decrypted ciphertexts exceeds the number of backdoor decryption queries allowed by $\mathcal{F}$, then the $\mathcal{S}$ finds solution to the one-more CDH problem.
- If the client is *honest*, $\mathcal{S}$ receives pk and u_0, so that (g, pk, u_0) is an instance of a CDH problem, i.e. one-more CDH problem with $n = 0$. $\mathcal{S}$ takes $u_1 = \cdots = u_n \leftarrow u_0$. Intuitively, breaking the IND-CCA property of the scheme would mean breaking the CDH problem.

In addition, in both cases, $\mathcal{S}$ gets access to the oracle $\mathsf{Check}(\cdot)$.

We proceed with arguing that the encryption and decryption performed by the ideal functionality interacting with the $\mathcal{S}$ is indistinguishable from the real protocol. If the environment is able to distinguish between the real protocol and the ideal protocol, then the simulator has enough information to solve one-more CDH problem that it received as an input.

The simulator prepares tables for storing the random oracle data, and in the background it runs processes that are constantly observing accesses of the adversary $\mathcal{A}$ and the environment $\mathcal{Z}$ to these oracles, programming them on demand to extract certain values as described further.

Key generation:

- Use random oracle H_c to extract the corrupted party's share of pk and the proof π from the commitment.
- Use knowledge extraction w.r.t. π to adjust the honest party's share of pk to the input pk and the share sk_i of the corrupted party (by applying the procedure of Fig. 7 to the random oracles H_0 and $\tilde{H}_0$).
- Simulate the proofs related to β. If the client is corrupted, $\mathcal{S}$ comes up with β_1 and β_2 such that $\beta_1 \neq \beta_2$ to be able to apply special soundness later. As the simulator has access to trapdoors, the proofs of equality can be simulated even if $\beta_1 \neq \beta_2$.

Encryption:

- Whenever $\mathcal{F}$ is used to encrypt a message, it asks for a ciphertext from $\mathcal{S}$, who constructs the ciphertext from the next challenge. In particular, it computes $u = (u_j)^r$ for a challenge u_j instead of taking $u = g^r$ for a known r. Also, it outputs a random c_2 instead of encrypting the true message (which it does not know) so that the random oracles of H' and H'' could be programmed later. All related proofs can be simulated.
- For any ciphertext encrypted outside of $\mathcal{F}$, a valid proof needs to be constructed for a later successful decryption, for which oracles H_1 and $\tilde{H}_1$ needs to be accessed. This allows $\mathcal{S}$ to prepare random oracles H_1 and $\tilde{H}_1$ to compute pk^r as in Fig. 7.

$\mathsf{Verify}^{H_1,\tilde{H}_1,B_1,B_2}(c_1, c_2, c_3)$

1 : $(u, \alpha_1, \Gamma_1, \Gamma_2, \pi, \pi_1, \pi_2) \leftarrow c_1$
2 : **assert** $\mathsf{ChE}_1^{H_1,\tilde{H}_1}[\pi \restriction_g^u | \alpha_1, \Gamma_1, \Gamma_2]$
3 : **assert** $\mathsf{DVC}(\pi_\iota \restriction_{g,u}^{\alpha_1,\Gamma_\iota} | B_\iota)$
4 : **return** true

Fig. 8. Ciphertext correctness verification function Verify (delegated by the simulator to $\mathcal{F}$). Here H_1 and $\tilde{H}_1$ are provided as tables.

$\mathsf{Dec}^{\mathsf{T}_1^{\mathsf{KNE}},H',H''}(c_1, c_2)$

1 : $(u, \ldots, \pi, \ldots) \leftarrow c_1$
2 : **if** $(m, ((u, \ldots), c_2))$ **in** T **then**
3 : **return** m
4 : $(\ldots, v) \leftarrow \pi$
5 : **get** (u, v, t) **from** $\mathsf{T}_1^{\mathsf{KNE}}$
6 : $k \leftarrow v^t$
7 : $(c_2', c_2'') \leftarrow c_2$
8 : **assert** $c_2'' = H''(k, c_2')$
9 : compute $m \leftarrow H'(k) \oplus c_2'$
10 : **return** m

Fig. 9. Decryption function Dec (delegated by the simulator to $\mathcal{F}$). If only the proof part of c is different from some existing ciphertext c', take (m, c') from T. Otherwise, decrypt as in the real protocol. Although $\mathcal{F}$ does not know the secret key, it can compute $k = v^t$ using the table $\mathsf{T}_1^{\mathsf{KNE}}$ that contains all the entries (u, v, t) for which $\mathcal{S}$ has made precomputations of Fig. 7. It extracts the entry that corresponds to u and v (if there is no such entry, return $\perp$). Here H' and H'' are provided as tables.

Decryption (honest client):

- $\mathcal{F}$ waits for a verification function Verify from $\mathcal{S}$, who chooses it as the initial knowledge extraction and the DVNIZK verification on behalf of the client. As it needs to evaluate H_1 and $\tilde{H}_1$, the contents of these hash tables are delivered as a part of procedure Verify (Fig. 8). The quantities B_1 and B_2 are also delivered as a part of Verify.
- As $\mathcal{S}$ does not know the true ciphertext, interaction with the server is simulated for a ciphertext generated from

a *random plaintext*. Due to the statistical blinding, the environment detects the difference with the negligible probability.

- $\mathcal{F}$ waits for a decryption function Dec from $\mathcal{S}$, who chooses it in such a way that $\mathcal{F}$ can combine the proof π with the data of the H_1 and $\tilde{H}_1$ to get pk^r. For this, $\mathcal{S}$ provides a table $\mathsf{T}_1^{\mathsf{KNE}}$ of all triples (u, v, t) obtained so far for H_1 and $\tilde{H}_1$ as in Fig. 7, so that $\mathcal{F}$ can take the one that corresponds to u and v. The function Dec (Fig. 9) takes into account that different ciphertexts *can* depend on each other as far as only the proof part is different, so it lets $\mathcal{F}$ take a record (m, c) from its internal table even if only u and c_2 match, but the proofs look different (as far as they are valid).
- For any ciphertext generated *inside* $\mathcal{F}$, the attacker could notice inconsistency of decryption only if it manages to construct a ciphertext $((u^*, \ldots), c_2^*)$ which would decrypt to something that is related to a message that is encrypted with $\mathsf{pk}^r = u^{\mathsf{sk}}$ where u is a CDH challenge, but $u^* \neq u$ or $c_2^* \neq c_2$. To construct such a ciphertext the attacker would need to solve CDH.

Decryption (corrupted client):

- The simulator gets from $\mathcal{A}$ a blinded ciphertext and needs to come up with a response w and a proof π'. The response should be valid for a valid ciphertext.
  - For messages generated outside of $\mathcal{F}$, using special soundness, $\mathcal{S}$ can compute $r' = \log_g(u')$ from a valid sigma-proof, which allows to compute $w = \mathsf{pk}_2^{r'}$.
  - For messages generated inside $\mathcal{F}$, it is impossible to extract the exponent, so $\mathcal{S}$ needs to compute $w = (u')^{\mathsf{sk}_2} = (u'^{1/\mathsf{sk}_1})^{\mathsf{sk}}$. For this, it needs to access the oracle $(\cdot)^{\mathsf{sk}}$. To obtain u'^{1/sk_1} from u', $\mathcal{S}$ uses Fig. 7 to prepare random oracles H_2 and $\tilde{H}_2$ to compute this value at the point when a valid proof is created (the value u' is a part of the context which is an input to $\tilde{H}_2$).
  - In general, homomorphic encryption allows to obtain a valid ciphertext as a linear combination of ideal ciphertexts. However, since $\mathcal{S}$ has used the same γ in Γ_1 and Γ_2 of the challenge ciphertexts, special soundness allows to extract the free term r', splitting $u' = u'_1 \cdot u'_2$ where $u' = g^{r'}$ and $u'_2 = \prod_j u_j^{z_j}$ is a linear combination of challenges. The simulator can compute $w = \mathsf{pk}_2^{r'} \cdot (u'_2)^{\mathsf{sk}}$, thus ensuring that $(\cdot)^{\mathsf{sk}}$ is only applied to the linear combinations of challenges and is not misused e.g. to break some proof.
- In any case, there will be no more oracle $(\cdot)^{\mathsf{sk}}$ called than there are decryptions approved by the server. At any point of time when the adversary attempts to actually decrypt a ciphertext (it may happen later, after the decryption has ended), the simulator has control over it as it controls H' and H'' that are valuated for a valid decryption. The simulator programs H' and H'' so that it would decrypt to the true message. If it is one of the previous ciphertexts by $\mathcal{F}$ (this fact can be verified using $\mathsf{Check}(\cdot)$ oracle), the simulator can use the "backdoor" decryption of $\mathcal{F}$ to get the plaintext. If the adversary manages to obtain more plaintexts than there have been decryption sessions (and hence allowed "backdoor" decryptions), $\mathcal{S}$ has solved the one-more-CDH problem.

VII. Efficiency

The performance of the encryption scheme can most probably be improved, perhaps by relying on different hardness assumptions. But even currently, the performance is very satisfactory for the intended application — decrypting credentials before their use. We have implemented DVPS encryption and decryption in Python on top of the PyCryptodome cryptographic library, as well as the *python-paillier* library [46] for the Paillier encryption. We use the elliptic curve group P-256 as $\mathbb{G}$, and 3072-bit RSA modulus for Paillier. The running times on a laptop with an Intel® Core™ i5-10210U CPU and 16GB RAM are 1140 ms for $\mathcal{E}nc$, 563 ms for $\mathcal{DC}_{\mathrm{C}}$, and 153 ms for $\mathcal{DC}_{\mathrm{S}}$. The sizes of the messages sent from the client to the server and back are 11.5KB and 5.5KB, respectively.

We can count the size our ciphertexts as follows. Ignoring the symmetrically encrypted payload c_2 (Fig. 6), the component c_1 consists of two elements of $\mathbb{G}$ (each represented by $d+1$ bits), two Paillier ciphertexts, and three proofs π, π_1, π_2, where the first is different from the other two. The proof π consists of a group element and two numbers of length ρ and d, respectively. The proof π_ι (Fig. 10) consists of a ρ-bit hash value, two numbers of length $(\rho + d + \kappa)$ and $(2\rho + d + 2\kappa)$, respectively, and one value that can serve as the random coins for the homomorphic encryption. We have chosen the security parameter determining the privacy of the protocol as $\kappa = 128$. It is also reasonable to take the integrity parameter as $\rho = 80$. Moreover, the size of group elements is $d = 256$. Considering that Paillier ciphertexts are 6144 bits long and Paillier coins (which are elements of $\mathbb{Z}_N$ for the used RSA modulus N) are 3072 bits long, the total size of the ciphertext c_1 is 21714 bits.

Most of the size of c_1 is taken up by two Paillier ciphertexts and two Paillier coins. There exist additively homomorphic encryption schemes with smaller ciphertext sizes. The Castagnos-Laguillaumie (CL) encryption system [47] has ciphertexts of size 4166 bits at the same security level [48], with coins of ca. 1955 bits. The homomorphism is according to a smaller modulus, but sufficient for our purposes, where we assume that there is no modular reduction during the operations inside the ciphertexts. Using CL encryption, our ciphertexts would be ca. 15500 bits in length. Similar sizes may be achievable with Joye-Libert cryptosystem [49].

We can compare these sizes with the ciphertext sizes in the schemes of Green [23] and Blazy et al. [24], both of which are pairing-based constructions. In the latter, the ciphertexts consist of 14 elements of the first source group, 15 elements of the second source group, and 4 elements of the target group. Assuming the use of the BLS12-381 curve [50], we estimate that the elements of these groups take ca. 384, $2 \cdot 384$, and $12 \cdot 384$ bits to represent, respectively. In this case, the size

of the ciphertext is ca. 35328 bits. Green's scheme is given in the setting of *symmetric pairings*; the ciphertext consists of 25 elements of the source group and two numbers (used as exponents). We believe that in asymmetric setting, most components of the ciphertext can be elements of the first source group. With BLS12-381 curve, the size of the ciphertext should be in the range of 10–11 kilobits. As we discussed above, the goals of Green's scheme are different from ours.

VIII. Conclusions

We have discussed the existing security definitions for the blind assisted and/or threshold decryption, demonstrated their shortcomings, proposed new ones, and instantiated them with the DVPS scheme. The security of the scheme relies on well-known hardness assumptions, it combines existing building blocks in somewhat novel manner, and its security is proved using very recent techniques.

Our security definition is given in the universal composability model. We have used this model to be able to capture the details of of the security definition, capturing both the IND-CCA security and privacy notions. We have also given the first ever treatment of threshold decryption in the UC model, arguing what must be the interface that the ideal functionality offers to the simulator / the ideal adversary. We show how this interface changes when the decryption is privacy-preserving.

Acknowledgement. This research has been supported by Estonian Research Council, grant No. PRG1780.

References

[1] A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung, "How to share a function securely," in *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, ser. STOC '94. New York, NY, USA: Association for Computing Machinery, 1994, p. 522–533. [Online]. Available: https://doi.org/10.1145/195058.195405

[2] V. Shoup and R. Gennaro, "Securing threshold cryptosystems against chosen ciphertext attack," *J. Cryptol.*, vol. 15, no. 2, pp. 75–96, 2002. [Online]. Available: https://doi.org/10.1007/s00145-001-0020-9

[3] Y. Lindell and A. Nof, "Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM, 2018, pp. 1837–1854. [Online]. Available: https://doi.org/10.1145/3243734.3243788

[4] J. Doerner, Y. Kondi, E. Lee, and A. Shelat, "Threshold ecdsa from ecdsa assumptions: The multiparty case," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1051–1066.

[5] A. Buldas, A. Kalu, P. Laud, and M. Oruaas, "Server-supported rsa signatures for mobile devices," in *Computer Security – ESORICS 2017*, S. N. Foley, D. Gollmann, and E. Snekkenes, Eds. Cham: Springer International Publishing, 2017, pp. 315–333.

[6] M. Vauclair, "Secure element," in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg and S. Jajodia, Eds. Boston, MA: Springer US, 2011, pp. 1115–1116. [Online]. Available: https://doi.org/10.1007/978-1-4419-5906-5_303

[7] "Personal identification – ISO-compliant driving licence. Part 5: Mobile driving licence (mDL) application," ISO/IEC 18013-5:2021, 2021.

[8] M. Sporny, D. Longley, and D. Chadwick, "Verifiable Credentials Data Model v1.1," March 3rd 2022, https://www.w3.org/TR/vc-data-model/.

[9] "Proposal for a Regulation of the European Parliament and of the Council amending Regulation (EU) No 910/2014 as regards establishing a framework for a European Digital Identity (SEC(2021) 228 final) - (SWD(2021) 124 final) - (SWD(2021) 125 final)," 2021, https://digital-strategy.ec.europa.eu/en/library/trusted-and-secure-european-e-id-regulation.

[10] " Commission welcomes final agreement on EU Digital Identity Wallet," Press release, November 8th 2023, https://ec.europa.eu/commission/presscorner/detail/en/ip_23_5651.

[11] "The European Digital Identity Wallet," 2023, https://github.com/eu-digital-identity-wallet/eudi-doc-architecture-and-reference-frame

[12] D. Fett, K. Yasuda, and B. Campbell, "Selective Disclosure for JWTs (SD-JWT)," Internet Engineering Task Force, Internet-Draft draft-ietf-oauth-selective-disclosure-jwt-08, Mar. 2024, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-oauth-selective-disclosure-jwt/08/

[13] S. Goldwasser, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract)," in *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, R. Sedgewick, Ed. ACM, 1985, pp. 291–304.

[14] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 2001, pp. 136–145.

[15] R. Cramer and V. Shoup, "A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack," in *Advances in Cryptology — CRYPTO '98*, H. Krawczyk, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 13–25.

[16] D. Boneh and V. Shoup, "A Graduate Course in Applied Cryptography," Jan. 2023, a book in preparation, v0.6.

[17] L. T. A. N. Brañdao, N. Mouha, and A. Vassilev, "Threshold Schemes for Cryptographic Primitives," National Institute of Standards and Technology (NIST), Tech. Rep. NISTIR 8214, Mar. 2019.

[18] R. Canetti, H. Krawczyk, and J. Nielsen, "Relaxing chosen-ciphertext security," Cryptology ePrint Archive, Paper 2003/174, 2003, https://eprint.iacr.org/2003/174. [Online]. Available: https://eprint.iacr.org/2003/174

[19] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," Cryptology ePrint Archive, Paper 2000/067, 2000, https://eprint.iacr.org/2000/067, version: 20051214:064128. [Online]. Available: https://eprint.iacr.org/2000/067

[20] W. Lueks, B. Hampiholi, G. Alpár, and C. Troncoso, "Tandem: Securing keys by using a central server while preserving privacy," *Proc. Priv. Enhancing Technol.*, vol. 2020, no. 3, pp. 327–355, 2020. [Online]. Available: https://doi.org/10.2478/popets-2020-0055

[21] J. M. Kirss, P. Laud, N. Snetkov, and J. Vakarjuk, "Server-supported decryption for mobile devices," in *Security and Trust Management - 18th International Workshop, STM 2022, Copenhagen, Denmark, September 29, 2022, Proceedings*, ser. Lecture Notes in Computer Science, G. Lenzini and W. Meng, Eds., vol. 13867. Springer, 2022, pp. 71–81. [Online]. Available: https://doi.org/10.1007/978-3-031-29504-1_4

[22] A. P. Sarr, "Cryptanalysis and improvement of smart-id's clone detection mechanism," Cryptology ePrint Archive, Paper 2019/1412, 2019, https://eprint.iacr.org/2019/1412. [Online]. Available: https://eprint.iacr.org/2019/1412

[23] M. Green, "Secure blind decryption," in *Public Key Cryptography – PKC 2011*, D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 265–282.

[24] O. Blazy, L. Brouilhet, C. Chevalier, P. Towa, I. Tucker, and D. Vergnaud, "Hardware security without secure hardware: How to decrypt with a password and a server," *Theor. Comput. Sci.*, vol. 895, pp. 178–211, 2021. [Online]. Available: https://doi.org/10.1016/j.tcs.2021.09.042

[25] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*, ser. Lecture Notes in Computer Science, Y. Desmedt, Ed., vol. 2567. Springer, 2003, pp. 31–46. [Online]. Available: https://doi.org/10.1007/3-540-36288-6_3

[26] E. Bresson, J. Monnerat, and D. Vergnaud, "Separation results on the "one-more" computational problems," in *Topics in Cryptology – CT-RSA 2008*, T. Malkin, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 71–87.

[27] B. Bauer, G. Fuchsbauer, and A. Plouviez, "The one-more discrete logarithm assumption in the generic group model," in *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part IV*, ser. Lecture Notes in Computer Science, M. Tibouchi and H. Wang,

Eds., vol. 13093. Springer, 2021, pp. 587–617. [Online]. Available: https://doi.org/10.1007/978-3-030-92068-5_20
[28] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko, "The one-more-rsa-inversion problems and the security of chaum's blind signature scheme," *J. Cryptol.*, vol. 16, no. 3, pp. 185–215, 2003. [Online]. Available: https://doi.org/10.1007/s00145-002-0120-1
[29] R. Canetti, H. Krawczyk, and J. B. Nielsen, "Relaxing chosen-ciphertext security," in *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, ser. Lecture Notes in Computer Science, D. Boneh, Ed., vol. 2729. Springer, 2003, pp. 565–582. [Online]. Available: https://doi.org/10.1007/978-3-540-45146-4_33
[30] J. Groth and A. Sahai, "Efficient non-interactive proof systems for bilinear groups," in *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, ser. Lecture Notes in Computer Science, N. P. Smart, Ed., vol. 4965. Springer, 2008, pp. 415–432. [Online]. Available: https://doi.org/10.1007/978-3-540-78967-3_24
[31] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology — EUROCRYPT '99*, J. Stern, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238.
[32] M. Bellare and P. Rogaway, "The exact security of digital signatures - how to sign with RSA and rabin," in *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, ser. Lecture Notes in Computer Science, U. M. Maurer, Ed., vol. 1070. Springer, 1996, pp. 399–416. [Online]. Available: https://doi.org/10.1007/3-540-68339-9_34
[33] C. P. Schnorr, "Efficient identification and signatures for smart cards," in *Advances in Cryptology — CRYPTO' 89 Proceedings*, G. Brassard, Ed. New York, NY: Springer New York, 1990, pp. 239–252.
[34] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn, "Malleable proof systems and applications," in *Advances in Cryptology – EUROCRYPT 2012*, D. Pointcheval and T. Johansson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 281–300.
[35] I. Damgård, N. Fazio, and A. Nicolosi, "Non-interactive zero-knowledge from homomorphic encryption," in *Theory of Cryptography*, S. Halevi and T. Rabin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 41–59.
[36] P. Chaidos and G. Couteau, "Efficient designated-verifier non-interactive zero-knowledge proofs of knowledge," Cryptology ePrint Archive, Paper 2017/1029, 2017, https://eprint.iacr.org/2017/1029. [Online]. Available: https://eprint.iacr.org/2017/1029
[37] R. Cramer and V. Shoup, "Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack," *SIAM J. Comput.*, vol. 33, no. 1, pp. 167–226, 2003. [Online]. Available: https://doi.org/10.1137/S0097539702403773
[38] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*, D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, Eds. ACM, 1993, pp. 62–73. [Online]. Available: https://doi.org/10.1145/168588.168596
[39] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology — CRYPTO' 86*, A. M. Odlyzko, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 186–194.
[40] D. Bernhard, M. Fischlin, and B. Warinschi, "On the hardness of proving cca-security of signed elgamal," in *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part I*, ser. Lecture Notes in Computer Science, C. Cheng, K. Chung, G. Persiano, and B. Yang, Eds., vol. 9614. Springer, 2016, pp. 47–69. [Online]. Available: https://doi.org/10.1007/978-3-662-49384-7_3
[41] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *Advances in Cryptology — CRYPTO' 92*, E. F. Brickell, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 89–105.
[42] I. Damgård and E. Fujisaki, "A statistically-hiding integer commitment scheme based on groups with hidden order," in *Advances in Cryptology — ASIACRYPT 2002*, Y. Zheng, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 125–142.
[43] I. Damgård and R. Thorbek, "Linear integer secret sharing and distributed exponentiation," in *Public Key Cryptography - PKC 2006*, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 75–90.
[44] B. Bauer, G. Fuchsbauer, and J. Loss, "A classification of computational assumptions in the algebraic group model," in *Advances in Cryptology – CRYPTO 2020*, D. Micciancio and T. Ristenpart, Eds. Cham: Springer International Publishing, 2020, pp. 121–151.
[45] G. Fuchsbauer, E. Kiltz, and J. Loss, "The algebraic group model and its applications," in *Advances in Cryptology – CRYPTO 2018*, H. Shacham and A. Boldyreva, Eds. Cham: Springer International Publishing, 2018, pp. 33–62.
[46] CSIRO's Data61, "Python Paillier Library," https://github.com/data61/python-paillier, 2013.
[47] G. Castagnos and F. Laguillaumie, "Linearly homomorphic encryption from DDH," in *Topics in Cryptology - CT-RSA 2015, The Cryptographer's Track at the RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings*, ser. Lecture Notes in Computer Science, K. Nyberg, Ed., vol. 9048. Springer, 2015, pp. 487–505. [Online]. Available: https://doi.org/10.1007/978-3-319-16715-2_26
[48] G. Castagnos, "Cryptography based on quadratic fields: cryptanalyses, primitives and protocols," Ph.D. dissertation, Université de Bordeaux, 2019.
[49] M. Joye and B. Libert, "Efficient cryptosystems from 2^k-th power residue symbols," in *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, ser. Lecture Notes in Computer Science, T. Johansson and P. Q. Nguyen, Eds., vol. 7881. Springer, 2013, pp. 76–92. [Online]. Available: https://doi.org/10.1007/978-3-642-38348-9_5
[50] S. Bowe, "BLS12-381: New zk-SNARK Elliptic Curve Construction," Electric Coin Co. blog post, March 11th 2017, https://electriccoin.co/blog/new-snark-curve/.
[51] Y. Lindell, A. Nof, and S. Ranellucci, "Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody," Cryptology ePrint Archive, Paper 2018/987, 2018, https://eprint.iacr.org/2018/987. [Online]. Available: https://eprint.iacr.org/2018/987

APPENDIX A
SUBPROTOCOLS FOR KEY GENERATION

A. Proving equality of Paillier encryption and Pedersen commitment

This proof is a subprotocol in protocols given below. It can be constructed from standard building blocks, as shown by Lindell et al. [51, Sec. 6.2.4]. In our setting, the server has presented a Paillier ciphertext $E = \mathcal{E}_{\mathsf{ek}}(x)$ and a Pedersen commitment $C = g^x h^z \in \mathbb{G}$ to the client, where the server knows Paillier private key vk, the value x, the exponent z, and also the random coins used to create E. The value $g \in \mathbb{G}$ is the generator of $\mathbb{G}$, while h is another element of $\mathbb{G}$, such that neither the client nor the server know $\log_g h$.

We use this subprotocol unchanged from [51], and will not copy it here.

B. Simulatable proof of equality of plaintexts in Paillier encryptions

At some point, server has sent two Paillier ciphertexts E_1 and E_2 to the client, and wants to prove him that $x_1 = x_2$, where $x_i = \mathcal{D}_{\mathsf{vk}_i}(E_i)$. In order to do this, the server creates a Pedersen commitments C_1, C_2 to x_1 and x_2 — it computes $C_i \leftarrow g^{x_i} h^{z_i}$, where $z_1, z_2 \leftarrow\!\!\$\ \mathbb{Z}_p$. The server proves that the ciphertext E_i contains the same value as C_i (App. A-A) for $i \in \{1, 2\}$. The server also makes, and client verifies the proof $\pi \leftarrow \mathsf{KnE}_1[z \mid \overset{C}{h}]$, where $z = z_1 - z_2$ and $C = C_1/C_2 = g^{x_1 - x_2} h^z$. The latter proof implies that server knows $\log_h C$,

meaning that C must have been computed as $g^0 h^z$, because server does not know $\log_g h$.

This proof is simulatable, because the simulator knows $\log_g h$. It can thus produce π (or run the interactive version of it) even if C is not a commitment to 0.

C. Disjunction of Σ-protocols

Suppose there are Σ-protocols Σ_1, Σ_2 for two relations R_1 and R_2, and Prover and Verifier have an instance x_1, x_2 for each of them. Let Prover also have witness w_ι for one of them, i.e. $\iota \in \{1, 2\}$. Prover wants to show that it has this w_ι, but wants to disclose neither it nor the index ι to Verifier. Let the challenge messages in Σ_1 and Σ_2 belong to the same set, and let this set also have the structure of an Abelian group. The protocol for such disjunction is the following.

1) Prover generates the "challenge" $\beta_{3-\iota}$.
2) Using the zero-knowledge property of $\Sigma_{3-\iota}$, Prover generates $(\alpha_{3-\iota}, \gamma_{3-\iota})$, such that $(x_{3-\iota}, \alpha_{3-\iota}, \beta_{3-\iota}, \gamma_{3-\iota})$ would pass Verifier's final check in $\Sigma_{3-\iota}$.
3) Prover generates the message α_ι in the protocol Σ_ι, using x_ι, w_ι if necessary.
4) Prover sends (α_1, α_2) to Verifier.
5) Verifier generates the challenge β and sends it to Prover.
6) Prover defines $\beta_\iota \leftarrow \beta - \beta_{3-\iota}$.
7) Prover constructs the message γ_ι in protocol Σ_ι, using β_ι as the challenge.
8) Prover sends $(\beta_1, \beta_2, \gamma_1, \gamma_2)$ to Verifier.
9) Verifier performs the checks in both Σ_1 and Σ_2, using β_1, β_2 as the challenges. Also checks that $\beta_1 + \beta_2 = \beta$.

Such protocol is again a Σ-protocol. Both the zero-knowledge simulation, and extraction algorithms for the composed protocol can easily be constructed from the respective algorithms for the subprotocols.

D. Extractable range proof for the challenge

During the key generation of DVPS, the server has to prove to the client that the encrypted challenge $B = \mathcal{E}_{\mathsf{ek}}(\beta)$ has been correctly generated, meaning that $0 \le \beta < 2^\rho$. This proof, together with the necessary trapdoors can be constructed from standard building blocks. The proof requires a hash function H_{com}, mapping into the group $\mathbb{G}$, and modelled as a random oracle.

Let $h = H_{\mathsf{com}}(B, ctx)$, where ctx binds this range proof to the current key generation session. E.g. ctx may contain the public key pk that has already been constructed at that time. Using h as the public key in ElGamal encryption scheme, the server encrypts (in the exponent) each bit of β separately. I.e. if $\sum_{i=0}^{\rho-1} \beta_i \cdot 2^i = \beta$, then the server picks $r_0, \ldots, r_{\rho-1} \leftarrow\!\$\ \mathbb{Z}_p$ and computes $c_{0,i} = g^{r_i}$, $c_{1,i} = g^{\beta_i} h^{r_i}$ for each i. The server sends all values $c_{0,0}, \ldots, c_{1,\rho-1}$ to the client. Both the server and the client define $C = \prod_{i=0}^{\rho-1} c_{1,i}^{2^i}$. In this way, C will be a Pedersen commitment to β, where the randomness is $z = \sum_{i=0}^{\rho-1} r_i \cdot 2^i$. Server proves to the client that the Paillier ciphertext B and Pedersen commitment C contain the same plaintext.

$\mathsf{DVP}(r, r_1, \mathbf{r} \mid^{\alpha_1, \Gamma}_{g,u} \mid B)$	$\mathsf{DVC}(\pi \mid^{\alpha_1, \Gamma}_{g,u} \mid B)$
$r_2 \leftarrow\!\$\ \mathbf{I}(\rho + d + \kappa)$	$(\beta', \gamma_2, \gamma_3, \gamma_{\mathbf{c}}) \leftarrow \pi$
$r_3 \leftarrow\!\$\ \mathbf{I}(2\rho + d + 2\kappa)$	$\alpha_2 \leftarrow g^{\gamma_2} / u^{\beta'}$
$\mathbf{r}' \leftarrow\!\$\ \mathbf{Cn}$	$\alpha_3 \leftarrow g^{\gamma_3} / \alpha_1^{\beta'}$
$\alpha_2 \leftarrow g^{r_2 \bmod p}$	$A \leftarrow \mathcal{E}_{\mathsf{ek}}(\gamma_3; \gamma_{\mathbf{c}}) \cdot B^{\gamma_2} / \Gamma^{\beta'}$
$\alpha_3 \leftarrow g^{r_3 \bmod p}$	**assert** $\beta' = H(g, u, \alpha_1, \Gamma,$
$A \leftarrow \mathcal{E}_{\mathsf{ek}}(r_3; \mathbf{r}') \cdot B^{r_2}$	$\alpha_2, \alpha_3, A, B)$
$\beta' \leftarrow H(g, u, \alpha_1, \Gamma, \alpha_2, \alpha_3, A, B)$	
$\gamma_2 \leftarrow r_2 + \beta' \cdot r$	
$\gamma_3 \leftarrow r_3 + \beta' \cdot r_1$	
$\gamma_{\mathbf{c}} \leftarrow \mathbf{r}' \boxplus (\beta' \boxdot \mathbf{r})$	
$\pi \leftarrow (\beta', \gamma_2, \gamma_3, \gamma_{\mathbf{c}})$	
return π	

Fig. 10. Proving the validity of a DVNIZK proof

For each bit β_i, the server proves to the client that it, i.e. the plaintext for the ciphertext $(c_{0,i}, c_{1,i})$ is indeed a bit. Note that if $\beta_i = 0$, then $(g, h, c_{i,0}, c_{i,1})$ is a Diffie-Hellman tuple and the server could convince the client in this using the Σ-protocol for DDH proofs (Sec. V-A); the server knows the randomness r_i necessary to give the proof. Similarly, if $\beta_i = 1$, then $(g, h, c_{i,0}, c_{i,1})/g$ is a Diffie-Hellman tuple and the same protocol could be used again. In order to show that β_i is either 0 or 1, we combine these two Σ-protocols using the construction in App.A-C.

The simulator can decrypt the ElGamal ciphertexts $(c_{i,0}, c_{i,1})$ and find the bits β_i, because it can program H_{com}. Whenever H_{com} is invoked either by itself or by the adversary, the simulator will generate the answer as g^r, where $r \leftarrow\!\$\ \mathbb{Z}_p$, and store r together with the argument(s) of H_{com}.

Appendix B
Proving that a DVNIZK proof is valid

Given a generator g of $\mathbb{G}$, a Paillier encryption key ek, a ciphertext B, and the values (u, α_1, Γ), suppose that the prover wants to convince the verifier that they have been constructed as in DVPS.$\mathcal{E}nc$ and the prover knows the used randomness $r = \log_g u \in \mathbf{I}(d)$, $r_1 = \log_g \alpha_1 \in \mathbf{I}(\rho + d + \kappa)$, and the random coins $\mathbf{r}$ for the encryption. The Σ-protocol for this is similar to proving the knowledge of exponent, where the prover constructs another similar triple (α_2, α_3, A) using fresh randomness $(r_2, r_3, \mathbf{r}')$ and sends it to the verifier, also sends a linear combination (with coefficients chosen by the verifier) of the original and fresh randomness to the verifier, who can then use the homomorphic properties of the operations to match the original triple against the fresh one. Instead of including α_2, α_3 and A in the proof, the prover instead includes β' and lets the verifier compute α_2, α_3 and A that would satisfy the equalities, followed by a check if β' indeed computed as their hash. We present the protocol in Fig. 10 as a NIZK proof, replacing verifier's choice of the coefficients of the linear combination with the invocation of a random oracle. The simulator is thus able to fake these proofs.

The proof uses the fact that Paillier's encryption is homomorphic with respect to the used coins. In Fig. 10, the operations $\boxdot$ and $\boxplus$ combine the coins. The value $\mathbf{r}$ is perfectly masked by $\mathbf{r}'$. The values r, r_1 however, are masked not perfectly, but only statistically, with random values having $\rho + \kappa$ more bits.

We show that the protocols of Fig. 10 prevent selective failure attacks on the underlying DFN proofs from [36]. In a DFN proof, the server verifies a linear combination $\gamma_1 = r_1 + r\beta$. The proof fails if $r_1 + r\beta \geq N$ where N is the size of the plaintext space of the homomorphic encryption scheme. The prover (in our case, the encrypting party) may take $r_1 = N - r \cdot b$ to check whether $\beta \geq b$ by observing whether the verifier (in our case, the server) rejects the proof. The verification procedure of Fig. 10 only depends on β through $B = \mathcal{E}_{\mathsf{ek}}(\beta)$ which does not leak β by properties of the homomorphic encryption scheme. However, we need to ensure that Fig. 10 does not miss selective failure attacks, and we show that either γ_2 or γ_3 will be too large so that they will rejected by the client. Let $\gamma_1 = r_1 + \beta r \geq N$. This means that either $r_1 \geq N/2$ or $\beta r \geq N/2$. In the first case, we have $\gamma_3 = r_3 + \beta' r_1 \geq \beta' r_1 \geq \beta' N/2$, which gives $\gamma_3 < N$ with probability at most $2^{-\rho}$ (since β' is not under attacker's control). In the second case, we have $\gamma_2 = r_2 + \beta' r \geq \beta' r \geq \beta'/\beta \cdot N/2 \geq \beta' \cdot N/2^{\rho+1}$. While such γ_2 can be smaller than N, it will still quite likely be larger than the largest possible γ_2 that can be obtained for a valid proof. A valid γ_2 by construction is strictly smaller than $2^{\rho+d+\kappa+1}$, while $N/2^{\rho+1} \geq 2^{2\rho+d+2\kappa}/2^{\rho+1} \geq 2^{\rho+d+\kappa}$, so the malicious γ_2 will be smaller than the allowed upper bound with probability $2^{-\rho}$ (again, since β' is not under attacker's control).

Appendix C
Full proof of Theorem 1

In order prove security of our threshold decryption protocol in the UC model, we construct a simulator $\mathcal{S}$ such that for each adversary $\mathcal{A}$ attacking the protocol, the environment $\mathcal{Z}$ cannot distinguish whether it is interacting with the real protocol and the adversary $\mathcal{A}$, or the ideal functionality $\mathcal{F}$ and the "ideal" adversary (consisting of $\mathcal{A}$ and $\mathcal{S}$). Details of $\mathcal{S}$ are presented in this section.

We start with defining $\mathcal{S}$ and how it interacts with the ideal functionality and the adversary in case of different parties being corrupted. $\mathcal{S}$ receives an one-more CDH challenge as input. We proceed with arguing that encryption and decryption performed by the ideal functionality interacting with $\mathcal{S}$ is indistinguishable from the real protocol. If the environment is able to distinguish between real protocol and the ideal protocol, then the simulator has enough information to solve one-more CDH problem. Intuitively, we need one-more CDH problem only in the corrupted client case to ensure that the number of decrypted ciphertexts does not exceed the number of backdoor decryption queries allowed by $\mathcal{F}$.

We assume that the corruptions are known in advance, and $\mathcal{S}$ gets some data from an external challenger. The challenger generates $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p$, $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$, and $r_i \leftarrow\!\!\$\ \mathbb{Z}_p$, $u_i \leftarrow\!\!\$\ g^{r_i}$ for $i \in \{0, \ldots, n\}$ for a parameter n.

- If the client is *corrupted*, $\mathcal{S}$ receives pk and $u_0, \ldots, u_n$, such that $(g, \mathsf{pk}, u_0, \ldots, u_n)$ is an instance of a one-more CDH problem. $\mathcal{S}$ also gets access to an oracle $(\cdot)^{\mathsf{sk}}$.
- If the client is *honest*, $\mathcal{S}$ receives pk and u_0, so that (g, pk, u_0) is an instance of a CDH problem, i.e. one-more CDH problem with $n = 0$. $\mathcal{S}$ takes $u_1 = \cdots = u_n \leftarrow u_0$.

In addition, in both cases, $\mathcal{S}$ gets access to the oracle $\mathsf{Check}(\cdot)$.

Throughout the proof, we will for simplicity assume the following:

- *Programming a random oracle always succeeds*. In reality, programming a random oracle can fail if the value has already been set in the oracle hash table during one of the previous queries by the simulator or adversary. However, this happens with the negligible probability over all the previously made queries to the corresponding oracles. Random oracle programming happens during the simulation of zero-knowledge proofs, where input values, not known to both parties, are random elements of group $\mathbb{G}$. Programming oracle used for commitments H_c also involves random group element. Finally, there is programming of oracles H' and H'' whose input depends on pk^r for $r \leftarrow\!\!\$\ \mathbb{Z}_p$. Therefore, there is only a negligible probability of input collisions.
- *The adversary cannot create fake proofs of knowledge*. The probability of this happening is negligible due to properties of the underlying constructions. Due to special honest-verifier zero-knowledge, adversary cannot distinguish simulated protocol messages from the real ones. Due to the special soundness, if adversary creates two accepting transcripts for the same witness, simulator will be able to extract it.

The simulation of random oracles that are used in the protocol is defined in Figure 11.

Simulation of knowledge extraction

Oracle $H_i(x), i \in \{0, \ldots, 3, c\}$:

1 : If the mapping $\{x \mapsto h\}$ is not in hash table T_i:
2 : Generate random $h \leftarrow\!\!\$\ range(H_i)$
3 : Add $\{x \mapsto h\}$ to T_i
4 : Return $\mathsf{T}_i(x)$

Oracle $\tilde{H}_i(x), i \in \{0, \ldots, 3\}$:

1 : If the mapping $\{x \mapsto (h, t)\}$ is not in hash table T_i
2 : Generate random $t \leftarrow\!\!\$\ \mathbb{Z}_p$
3 : $h \leftarrow g^t$
4 : Add $\{x \mapsto (h, t)\}$ to $\tilde{\mathsf{T}}_i$
5 : Return $fst(\tilde{\mathsf{T}}_i(x))$

The simulation of H' and H'' is analogous to H_i.

Fig. 11. Simulation of random oracles.

In Figure 7, we have described how knowledge extraction proofs can be used to raise an element z to a power of r^{d} for $\mathsf{d} \in \{-1, 1\}$. Since the proofs are not necessarily generated during the execution of $\mathcal{F}$ (e.g. they can be generated in advance), we need to describe separately how $\mathcal{S}$ acts at the moments when either $\mathcal{A}$ or $\mathcal{Z}$ accesses the corresponding random oracles. We have three instances of knowledge extraction, indexed by $0, 1, 2$, each using its own random oracles H_i and $\tilde{H}_i$, and $\mathsf{d}_i \in \{-1, 1\}$. For each instance, the simulator maintains a table $\mathsf{T}_i^{\mathsf{KNE}}$ to store records (u, v, t) such that $v^t = z^{r^{\mathsf{d}}}$ for z of its choice (which depends on i) is expected if there is a valid proof of $v = h^{r^{\mathsf{d}}}$. For $i \in \{0, 1, 2\}$, we define the routine $\mathsf{RO}(i)$ as follows:

At any point of time, whenever either $\mathcal{A}$ or $\mathcal{Z}$ accesses the oracle $\tilde{H}_i$ with some input (g, u, ctx), if $i \in \{0, 1\}$, take $z = \mathsf{pk}$. If $i = 2$, take $z = u'/g^{r'}$ for $(u', \alpha_1', \Gamma_1', \Gamma_2') \leftarrow ctx$ and $r' = (\mathcal{D}_{\mathsf{vk}_1}(\Gamma_1') - \mathcal{D}_{\mathsf{vk}_2}(\Gamma_2'))/(\beta_1 - \beta_2)$. Follow instructions from Fig. 7 to prepare $\tilde{H}_i$ for computing $z^{\log_g(u)^{\mathsf{d}_i}}$.

At any point of time, whenever either $\mathcal{A}$ or $\mathcal{Z}$ accesses the oracle H_i with an input of the form $(g, h, u, v, \ldots)$that corresponds to a previously called $h = \tilde{H}_i(g, u, ctx)$, take the t that was generated as in Fig. 7 and add (u, v, t) to $\mathsf{T}_i^{\mathsf{KNE}}$. At this point, the simulator does not know yet whether $v^t = z^{\log_g(u)^{\mathsf{d}_i}}$ (it depends on the correctness of v), but it can be verified after getting the corresponding proof π.

When initialized, the simulator creates empty tables $\mathsf{T}_i^{\mathsf{KNE}}$ and starts running $\mathsf{RO}(0)$, and $\mathsf{RO}(1)$. If the client is corrupted, then as soon as the key generation succeeds, $\mathcal{S}$ also starts running $\mathsf{RO}(2)$ (note that, as far as $\mathcal{S}$ has not generated β_1, β_2, vk_1, vk_2, the inputs to $\tilde{H}_2$ can be ignored as generating a valid proof w.r.t. these secret values is not possible). We further describe how the simulator $\mathcal{S}$ handles and responds to the commands from $\mathcal{F}$.

Key generation (client and server are both honest)

If the server and the client are both honest, the adversary may only interrupt the protocol run. The simulator generates all the exchanged messages himself in such a way that the challenge pk is used as a part of the public key.

Key generation.

Upon receiving message (KeyGen, sid) from $\mathcal{F}$:

- Simulate (inside $\mathcal{S}$) messages for both client and server, setting pk to the value that was input to the simulator, allowing $\mathcal{A}$ to stop communication.
- If $\mathcal{A}$ stops communication, set $pk = \perp$.
- Otherwise, set $pk = (\mathsf{pk}, \mathsf{ek}, B_1, B_2)$, where (ek, B_1, B_2) come from the simulated protocol.
- Send (Key, sid, pk) to $\mathcal{F}$.

Key generation (either server or client is corrupted)

Simulation of key generation is similar for the cases where either client or server is corrupted. Therefore, we present this simulation jointly here. Let $i \in \{1, 2\}$ be the index of the honest party (the index of the corrupted party will be $3 - i$).

Key generation.

Upon receiving message (KeyGen, sid) from $\mathcal{F}$:

- generate a random bit string $com_i \leftarrow\$\ range(H_c)$ send it to the adversary as commitment on pk_i
- once com_{3-i} is received from adversary, search hash table to get pk_{3-i} and π_{3-i}.
- Verify the proof $\mathsf{ChE}_{-1}^{H_0, \tilde{H}_0}[\pi_{3-i} \,|\, g^{\mathsf{pk}_{3-i}}]$. If it verifies, then $\mathcal{S}$ takes $(\ldots, v) \leftarrow \pi_{3-i}$ and looks for an entry $(\mathsf{pk}_{3-i}, v, t) \in \mathsf{T}_0^{\mathsf{KNE}}$, and takes $\mathsf{pk}_i = v^t = \mathsf{pk}^{\mathsf{sk}_{3-i}^{-1}}$.
- program random oracle H_c such that $H_c(\mathsf{pk}_i) = com_i$
- simulate proof of knowledge of sk_i, following the instructions from Figure 12.
- send proof π_i and key share pk_i to the adversary
- upon receiving $(\mathsf{pk}_{3-i}, \pi_{3-i})$, verify the commitment opening $H_c(\mathsf{pk}_{3-i}\pi_{3-i}) = com_{3-i}$. If commitment opening does not verify, set $pk = \perp$.
- If client is honest proceed as follows:
  - upon receiving (ek, B_1, B_2) and the additional proofs from the adversary, verify them
  - if proofs verify, extract β from proofs, and set $pk = (\mathsf{pk}, \mathsf{ek}, B_1, B_2)$. If proofs do not verify, set $pk = \perp$.
- If server is honest proceed as follows:
  - generate $(\mathsf{ek}_1, \mathsf{vk}_1) \leftarrow \mathcal{K}()$, $(\mathsf{ek}_2, \mathsf{vk}_2) \leftarrow \mathcal{K}()$
  - sample $\beta_1, \beta_2 \leftarrow\$\ \mathbf{I}(\rho)$ s.t. $\beta_1 \neq \beta_2$ and compute $B_1 \leftarrow \mathcal{E}_{\mathsf{ek}_1}(\beta_1), B_2 \leftarrow \mathcal{E}_{\mathsf{ek}_2}(\beta_2)$
  - send (ek, B_1, B_2) with corresponding simulated proofs to the adversary; take $pk = (\mathsf{pk}, \mathsf{ek}, B_1, B_2)$.
- Send (Key, sid, pk) to $\mathcal{F}$.
- For $i \in \{1, 2\}$, wait for the ouptut y_i of a corrupted party P_i from $\mathcal{A}$. Send $(\mathsf{Output}, i, y_i)$ to $\mathcal{F}$.

Simulating $\mathsf{KnE}_{-1}^{H_0, \tilde{H}_0}[\mathsf{sk}_i \,|\, g^{\mathsf{pk}_i}]$

1 : $\gamma_\pi, \beta_\pi, t' \leftarrow\$\ \mathbb{Z}_p$
2 : Program RO $\tilde{H}$ as $h' = \tilde{T}(g, pk_i, ctx) = g^{t'}$
3 : $v \leftarrow pk_i^{t'}$
4 : $\alpha_\pi := g^{\gamma_\pi}/pk_i^{\beta_\pi}, \alpha_\pi' := (v')^{\gamma_\pi}/(h')^{\beta_\pi}$
5 : Program RO H as $\beta_\pi = T(g, h', pk_i, v, \alpha_\pi, \alpha_\pi', ctx)$
6 : Return $(\beta_\pi, \gamma_\pi, v)$

Fig. 12. Simulating proof of knowledge of secret key share

After key generation phase has been simulated, $\mathcal{S}$ sets a counter $ctr_{\mathsf{chal}} \leftarrow 0$, defining the index of the challenge $u_{ctr_{\mathsf{chal}}}$ that will be used in the next encrypiton.

Encryption (any corruptions)

We assume that the key generation has already been simulated, and $\mathcal{S}$ has come up with a public key $pk = (\mathsf{pk}, \mathsf{ek}, B_1, B_2)$. The simulator maintains a local table $\mathsf{T}_{\mathsf{i,r,c}}$, storing a record (i, r, c) for a challenge index i, the randomness r, and the ciphertext c for each ciphertext c generated for $\mathcal{F}$.

Encryption

Upon receiving message $(\mathsf{Encrypt}, sid)$ from $\mathcal{F}$, compute $c \leftarrow \mathcal{E}nc_{\mathcal{S}}(j, \mathsf{pk}, \mathsf{ek}, B_1, B_2)$ for $j = ctr_{\mathsf{chal}}$ as in Fig. 14. Set $ctr_{\mathsf{chal}} \leftarrow ctr_{\mathsf{chal}} + 1$. Send message $(\mathsf{Encrypt}, sid, c)$ to $\mathcal{F}$.

Upon receiving message $(\mathsf{Encrypt}, sid, i, m)$, simulate input m for P_i in the real protocol and wait for y from $\mathcal{A}$. Send message $(\mathsf{Encrypt}, sid, y)$ to $\mathcal{F}$.

Simulating $\mathsf{KnE}_1^{H_1,\tilde{H}_1}[r\,|\,{}^{u}_{g}|ctx]$

$\gamma_\pi, \beta_\pi, t' \leftarrow\$ \mathbb{Z}_p$
Program RO $\tilde{H}$ as $h' = \tilde{T}(g, u, ctx) = g^{t'}$
$v \leftarrow u^{t'}$
$\alpha_\pi \leftarrow g^{\gamma_\pi}/u^{\beta_\pi}, \alpha'_\pi := (h')^{\gamma_\pi}/v^{\beta_\pi}$
Program RO H as $\beta_\pi = T(g, h', u, v, \alpha_\pi, \alpha'_\pi, ctx)$
Return $(\beta_\pi, \gamma_\pi, v)$

Simulating $\mathsf{DVP}(r, r_\iota\,|\,{}^{\alpha_1,\Gamma_\iota}_{g,u}|B_\iota)$

$\beta'_\iota, \gamma_{(\pi_\iota,2)}, \gamma_{(\pi_\iota,3)} \leftarrow\$ \mathbb{Z}_p, \gamma_{\mathbf{c}} \leftarrow\$ \mathbf{Cn}$
$\alpha_2 \leftarrow g^{\gamma_{(\pi_\iota,2)}}/u^{\beta'_\iota}, \alpha_3 \leftarrow g^{\gamma_{(\pi_\iota,3)}}/\alpha_1^{\beta'_\iota}$
$A \leftarrow (\mathcal{E}_{\mathsf{ek}_\iota}(\gamma_3; \gamma_{\mathbf{c}}) \cdot B_\iota^{\gamma_2})/\Gamma_\iota^{\beta'_\iota}$
Program RO H as $\beta'_\iota = T(g, u, \alpha_1, \Gamma_\iota, \alpha_2, \alpha_3, A, B_\iota)$
Return $(\beta'_\iota, \gamma_2, \gamma_3, \gamma_{\mathbf{c}})$

Fig. 13. Simulators for zero-knowledge proofs of encryption

$\mathcal{E}nc_{\mathcal{S}}(i, \mathsf{pk}, \mathsf{ek}, B_1, B_2)$

$r \leftarrow\$ \mathbb{Z}_p$
$u \leftarrow u_i^r$ where u_i is a challenge
$\mathbf{r}_\iota \leftarrow\$ \mathbf{Cn}$
$\gamma \leftarrow\$ \mathbf{I}(\rho + d + \kappa)$
$\alpha_1 \leftarrow g^\gamma/u^{\beta_1}$ if the client is honest, sample $\beta_1 \leftarrow\$ \mathbf{I}(\rho)$
$\Gamma_\iota \leftarrow \mathcal{E}_{\mathsf{ek}_\iota}(\gamma; \mathbf{r}_\iota)$
Simulate π, π_ι as in Fig. 13
$c_1 \leftarrow (u, \alpha_1, \Gamma_1, \Gamma_2, \pi, \pi_1, \pi_2)$
$c_{21} \leftarrow\$ range(H')$
$c_{22} \leftarrow\$ range(H'')$
$c \leftarrow (c_1, (c_{21}, c_{22}))$
Store the record (i, r, c) in a table $\mathsf{T}_{\mathsf{i,r,c}}$
return c

Fig. 14. Encryption function $\mathcal{E}nc_{\mathcal{S}}$ as computed by the simulator. It simulates real protocol encryption $\mathcal{E}nc$ (Fig. 6) to get c_1. The challenge u_i is embedded into the ciphertext, and all proofs are simulated. The parts of code that are the same as in $\mathcal{E}nc$ are colored gray. The simulator samples random c_{21} and c_{22} so that the RO of H' and H'' satisfying $c_{21} = H'(pk^r) \oplus m$ and $c_{22} = H''(pk^r, c_{21})$ could be programmed later. The simulator remembers which challenge and which randomness was used for which ciphertext.

Decryption (honest client)
Decryption. Upon receiving $(\mathsf{Decrypt\text{-}init}, sid)$ from $\mathcal{F}$:

- Send a message $(\mathsf{Decrypt\text{-}init}, sid, \mathsf{Verify})$ to $\mathcal{F}$, where the algorithm Verify is defined as on Figure 8.
- Upon receiving $(\mathsf{Decrypt\text{-}bad\text{-}c,sid})$ from $\mathcal{F}$, stop the simulation on behalf of the client before any interaction has started. This corresponds to failed proof check. If the server is corrupted, wait for the server output y_2 from $\mathcal{A}$ and send $(\mathsf{Output}, 2, y_2)$ to $\mathcal{F}$.
- Upon receiving $(\mathsf{Decrypt\text{-}good\text{-}c,sid})$ from $\mathcal{F}$:
  If the server is *honest*:
  - Proceed with simulating protocol messages, allowing $\mathcal{A}$ to interrupt communication.
  - When the protocol simulation finishes, send $(\mathsf{Decrypt\text{-}complete}, sid, \mathsf{Dec})$ to $\mathcal{F}$ with Dec defined as in Fig. 9.

  If the server is *corrupted*:
  - Compute $c \leftarrow \mathcal{E}nc(\mu, \mathsf{pk}, \mathsf{ek}, B_1, B_2)$ for some fixed messge $\mu \in \mathsf{M}$.
  - Sample blinding values for c and generate the proof π' according to $\mathcal{DC}_{\mathrm{C}}$ of Fig. 6 (note that secret key is not needed for these steps), getting a blinded ciphertext c'. Send c' to $\mathcal{A}$.
  - Upon receiving (w, π'') from $\mathcal{A}$ verify π''.
    If it verifies, send $(\mathsf{Decrypt\text{-}complete}, sid, \mathsf{Dec})$ to $\mathcal{F}$ with Dec defined as in Fig. 9. Otherwise, send $(\mathsf{Decrypt\text{-}fail}, sid)$ to $\mathcal{F}$. Ignore w.
  - Wait for the server output y_2 from $\mathcal{A}$ and send $(\mathsf{Output}, 2, y_2)$ to $\mathcal{F}$.

Since the simulator encrypts a fixed message μ in the simulated protocol, we need that the adversary would not be able to distinguish if a random ciphertext is being decrypted or it corresponds to a blinded version of some ciphertext c that might have been seen previously by the adversary. This refers to privacy property provided for the decrypted ciphertext, and since we have statistical blinding, this probability is negligible.

Decryption (corrupted client)
Decryption Upon receiving $(\mathsf{Decrypt\text{-}init}, sid, c')$ from $\mathcal{F}$:

- send c' to the adversary;
- upon receiving $(u', \alpha'_1, \Gamma'_1, \Gamma'_2, \pi')$ from the adversary, if π' does not verify, send $(\mathsf{Decrypt\text{-}fail}, sid)$ to $\mathcal{F}$;
- If π' verifies, compute $\gamma_1 \leftarrow \mathcal{D}_{\mathsf{vk}_1}(\Gamma'_1)$ and $\gamma_2 \leftarrow \mathcal{D}_{\mathsf{vk}_2}(\Gamma'_2)$. Extract $r' = (\gamma_1 - \gamma_2)/(\beta_1 - \beta_2)$.
- If $g^{r'} = u'$ then compute $w = \mathsf{pk}_2^{r'}$.
- Otherwise, compute $u'' = u'/g^{r'}$ and $\gamma' = \gamma_1 - \beta_1 r'$ (note that $\gamma' = \gamma_2 - \beta_2 r'$ due to choice of r'). Verify $\gamma' = \alpha'_1 \cdot (u'')^{\beta_1}$. If this check fails, send $(\mathsf{Decrypt\text{-}fail}, sid)$ to $\mathcal{F}$. If this check passes, then the client has come up with $\Gamma'_2 = \mathcal{E}_{\mathsf{ek}_2}(\log_g(\alpha'_1) + \log_g(u'') \cdot \beta_1 + \log_g(u'/u'') \cdot \beta_2)$ for the provided α'_1 and u'. Unless $u'' = 1$, the client needs quantities of the form $\mathcal{E}_{\mathsf{ek}_2}(r_j \cdot \beta_1)$, which can only be taken from challenge ciphertexts where $r_j = \log_g(u_j)$ for a challenge u_j, so $u'' = \prod_j u_j^{z_j}$ for such u_j and some linear coefficients z_j.
- As the proof $\pi' = (\pi, v)$ has verified, there is $(\mathsf{pk}_1, v, t) \in \mathsf{T}_2^{\mathsf{KNE}}$ such that $v^t = u'/g^{r'} = u''$;
- Call the oracle $(\cdot)^{\mathsf{sk}}$ at the point $w' := v^t$, getting $w'' = ((u'')^{1/sk_1})^{\mathsf{sk}} = (u'')^{\mathsf{sk}_2}$. Take $w = w'' \cdot \mathsf{pk}_2^{r'}$. At this point, the oracle $(\cdot)^{\mathsf{sk}}$ has only been used on linear combinations (with coefficients known to the adversary) on challenge ciphertexts, so the result cannot be used to break any assumptions, e.g. fake some proofs.
- simulate proof π'' following the instructions of Figure 15 and send (w, π'') to the adversary;
- Send $(\mathsf{Decrypt\text{-}complete}, sid)$ to $\mathcal{F}$.
- Upon receiving the final output y_1 from $\mathcal{A}$, send $(\mathsf{Output}, 1, y_1)$ to $\mathcal{F}$.

Offline decryption

Before proving correctness of the simulations for encryption and decryption, we need to describe how the simulator resolves adversary's attempts to decrypt a ciphertext c offline, i.e. without involving $\mathcal{F}$. If the client is corrupted, then in addition to $\mathsf{T}_{i,r,c}$ the simulator maintains a local table $\mathsf{T}_{\mathsf{m,c}}$ storing records (m, c) for message-ciphertext pairs it already knows.

At any point of time whenever either $\mathcal{A}$ or $\mathcal{Z}$ queries H' at the point x or H'' at the point (x, y) for any y, if $H'(x)$ has not been programmed yet:

- Iterate over records (i, r_i, c_i) of the table $\mathsf{T}_{\mathsf{i,r,c}}$.
- For each r_i, take $x' \leftarrow x^{1/r_i}$, and call $\mathsf{Check}(x')$. This will tell whether x is a solution to some component of the one-more CDH problem, i.e. corresponds to a ciphertext generated by $\mathcal{S}$ for $\mathcal{F}$.
- If the check succeeds for some i, returning j, then x is a solution to the CDH problem (g, pk, u_j).
  - If the client is *honest*, then $u_j = u_0$ for all j, and the simulator has solved the CDH problem (g, pk, u_0).
  - If the client is *corrupted*, then $\mathcal{S}$ may not have solved one-more CDH problem yet. If this is the case, $\mathcal{S}$ proceeds as follows:
    - let $(c_{i1}, (c_{i2}, c_{i3})) \leftarrow c_i$;
    - if $\mathsf{T}_{\mathsf{m,c}}$ does not contain a record (m, c_i), then send $(\mathsf{Decrypt\text{-}msg}, sid, \bot, c_i)$ to $\mathcal{F}$, wait for response $(\mathsf{Decrypted}, sid, m)$, and add (m, c_i) to $\mathsf{T}_{\mathsf{m,c}}$;
    - program random oracle H' s.t. $H'(x) = m \oplus c_{i2}$;
    - program random oracle H'' s.t. $H''(x, c_{i2}) = c_{i3}$.

Simulating $\mathsf{DHP}^{H_3}[\mathsf{sk}_2 \,|\, {}^{\mathsf{pk},w}_{\mathsf{pk}_1,u'}]$

1 : $\gamma_{\pi''}, \beta_{\pi''} \leftarrow\!\$\, \mathbb{Z}_p$
2 : $\alpha_{\pi''} \leftarrow pk_1^{\gamma_{\pi''}}/pk^{\beta_{\pi''}}, \alpha'_{\pi''} \leftarrow (u')^{\gamma_{\pi''}}/(w')^{\beta_{\pi''}}$
3 : Program RO H $\beta_{\pi''} = T(pk_2, pk, w, u', \alpha_{\pi''}, \alpha'_{\pi''})$
4 : Return $(\beta_{\pi''}, \gamma_{\pi''})$

Fig. 15. Simulating proof of knowledge of secret key share for server decryption

Proofs of indistinguishability

We prove that encryption and decryption performed by $\mathcal{F}$ output to the receiving party the values that would be expected in a real protocol.

Encryption. The ciphertext $c' \leftarrow \mathcal{E}nc_{\mathcal{S}}(i, \mathsf{pk}, \mathsf{ek}, B_1, B_2)$ output by $\mathcal{F}$ is indistinguishable from a real ciphertext $c \leftarrow \mathcal{E}nc(m, \mathsf{pk}, \mathsf{ek}, B_1, B_2)$ as far as $H'(u_i^{r\cdot\mathsf{sk}}) \oplus m$ is indistinguishable from a random element of $range(H')$, and $H''(u_i^{r\cdot\mathsf{sk}}, c_2)$ from a random element of $range(H'')$. This distinguishing is possible in the following cases:

- The hash function H' (or H'') does not return a random element of $range(H')$ (or $range(H'')$). We assume that H' and H'' act as random oracles.
- The adversary could obtain $u_i^{r\cdot\mathsf{sk}}$. In this case, $\mathcal{S}$ would receive $u_i^{r\cdot\mathsf{sk}}$ at the point where the adversary queries $H'(x)$ or $H''(x, y)$ for $x = u_i^{r\cdot\mathsf{sk}}$. Acting according to the instructions for the offline decryption, he would either solve one instance of the one-more CDH problem, or program H' and H'' so that c decrypts to m.

If the client is *honest*, then solving one instance of one-more CDH problem is enough to answer the challenge. If the client is *corrupted*, since $\mathcal{S}$ is allowed to get at most ctr_{dec} messages m from $\mathcal{F}$, we need to show that ctr_{dec} is sufficiently large.

W.l.o.g. suppose that the adversary manages to obtain $u_1^{r_1\cdot\mathsf{sk}}, \ldots, u_\ell^{r_\ell\cdot\mathsf{sk}}$. We show that $\ell \leq n_d$ where n_d is the number of decryption sessions initiated so far. In each decryption session, $\mathcal{S}$ accesses the oracle $(\cdot)^{\mathsf{sk}}$ at most once, and there are no other accesses to $(\cdot)^{\mathsf{sk}}$, so $n = n_d$. For ℓ different challenges $u_i^{r_i\cdot\mathsf{sk}}$, there will be needed ℓ queries to $\mathcal{F}$ to get m. If $\ell > n = n_d$, then $\mathcal{S}$ has solved one-more CDH problem of size n_d. If $\ell \leq n = n_d$, then ctr_{dec} stays non-negative.

Decryption. The message m resulting from decryption of c by $\mathcal{F}$ should be indistinguishable from what would be expected in the real protocol. This is relevant only for the honest client case, as the output of a corrupted client is determined by the adversary. There are different cases for c:

- If c was output to the environment by $\mathcal{F}$ as a response to an encryption query, then $\mathcal{F}$ has a valid pair (m, c) in the table T and does not need to apply Dec algorithm.
- If c was generated externally, then it is being decrypted using algorithm Dec which computes pk^r as v^t for an entry (u, v, t) of the table $\mathsf{T}_1^{\mathsf{KNE}}$ for u and π that are a part of c. This record is always there since Dec is applied only in the cases where Verify has been successful, which means that the corresponding proofs have been generated and oracles H_1, $\tilde{H}_1$ accessed. If c depends on some ciphertext c' generated for a message m' using $\mathcal{F}$, the adversary could notice that c' does not depend on m'. We show that c cannot be related to c' unless either $c = c'$, or the only part that differs from c are the components of c_1 that are different from u, which means that Dec would decrypt c using T.

Suppose that the adversary is able to get a ciphertext $c = (c_1, (c_2, c_3))$ acceptable for decryption by $\mathcal{F}$. Let $c' = (c'_1, (c'_2, c'_3))$ be any ciphertext generated using $\mathcal{F}$.

1) Suppose the adversary has constructed π itself. The other components of c may depend on c'. However, since pk^r can be extracted from π, the ciphertext c could be decrypted *without* using $\mathcal{F}$, and we proved in the previous point indistinguishability for c' in this case.
2) Suppose the adversary has taken the proof π from c'. Since π has been generated w.r.t. $u_i^{r_i}$, α_1, Γ_ι, the adversary would need to keep these components the same. Due to the assertion $c'_3 = H''(u_i^{r_i\cdot\mathsf{sk}}, c'_2)$ that an honest client would perform in the real protocol before outputting m, the adversary needs to compute $H''(u_i^{r_i\cdot\mathsf{sk}}, c_2)$ to take $c_2 \neq c'_2$. This would require access to H'' with $u_i^{r_i\cdot\mathsf{sk}}$, which would allow $\mathcal{S}$ to solve CDH. If $c_1 = c'_1$ and $c_2 = c'_2$, then also $c_3 = c'_3$.

Appendix 4

Publication III

J. Vakarjuk and A. Pankova. Prettiness – privacy preserving attribute management system, 2026

arXiv:2603.23221v1 [cs.CR] 24 Mar 2026

PRETTINESS - Privacy pResErving aTTrIbute maNagEment SyStem

Jelizaveta Vakarjuk[1,2] and Alisa Pankova[1]
{jelizaveta.vakarjuk, alisa.pankova}@cyber.ee

[1] Cybernetica AS, Mäealuse 2/1, 12618 Tallinn, Estonia
[2] Tallinn University of Technology, Akadeemia tee 15a, 12618, Tallinn, Estonia

Abstract. Recent advancements in the European regulatory frameworks (eIDAS2, GDPR) as well as increasing awareness of user privacy needs, have fostered a growing demand for identity management solutions that prioritise privacy. Attribute-based credential systems offer a solution, allowing users to prove ownership of their attributes to relying parties while selectively sharing specific attributes. While many systems have been proposed, and some systems have practical implementations, these solutions often store attributes locally on user devices.
In this work, we propose an attribute management system, where the user attributes are stored in encrypted form by the attribute management service (AMS). AMS helps to offload the storage from the user's mobile device and manage all the operations with credentials, including revocation. Our system is built by carefully balancing between the centralised and decentralised identity system approaches. The system relies on AMS, at the same time offering privacy against AMS, and ensuring integrity even if the AMS is corrupted. We prove security of the proposed system in the universal composability model.

Keywords: attribute-based credentials, privacy, universal composability

1 Introduction

With the changes in the regulatory framework of eIDAS2, there is an increasing demand in the research of identity management solutions with privacy guarantees. eIDAS2 defines the new European Digital Identity (EUDI) Wallet concept that aims to provide a user with the way to manage and present their credentials to the relying parties (RP), while trying to ensure interoperability of solutions between the EU member states. The regulation [1] lists security and privacy requirements for wallets, mentioning *selective disclosure of data*, *generating pseudonyms and storing them encrypted*, *accessing a log of all transactions* (Article 5a, §4) as well as *not allowing any party, after the issuance of the attestation of attributes, to obtain data that allows transactions or user behaviour to be tracked, linked or correlated unless explicitly authorised by the user* and *enabling privacy preserving techniques which ensure unlinkability* (Article 5a, §16). The currently proposed solution is defined in The Architecture

and Reference Framework (ARF, version 2.7.3) [2]. Additionally, the ARF defines wallet functionalities, including ability of creating backup of a list of user's attributes, attestations, and configurations as well as revocation by the issuer. In our view, the currently proposed solution defined in The Architecture and Reference Framework (ARF), is lacking some of the functionality and security properties. Similar concerns were raised in the Cryptographers' Feedback on the EU Digital Identity's ARF (version 1.4.0) [5]. However, we look at it from the different angle, trying to balance between the practical usability of the system and providing reasonable security and privacy guarantees.

According to the system description, the user is responsible for managing and storing the attributes themselves on their (mobile) device. However, to ensure a safe backup of credential data, it is beneficial to have an Attribute Management Service (AMS) that would be responsible for storing user's credentials. Such a service would also help with the process of issuing new credentials and revoking old ones. In this work, we introduce PRETTINESS – Privacy pResErving aTTrIbute maNagEment SyStem. In design of Prettiness, we are trying to find the balance between the centralised and decentralised identity management systems. We ensure that the user is in charge of their credentials, allowing selective disclosure and revocation by the user. At the same time, a centralised AMS server helps user with the storage and management of those credentials. Additionally, while details of logging user transactions are omitted from the description of our protocol, this functionality can be added to the system by saving user's signatures that are created to authorise each transaction with their credential. One more detail that we omit from the current definition of the protocol is the use of pseudonyms. However, we note that this functionality can be added to the system through the user identifiers uID being used as pseudonyms.

In this work, we use revocation functionality that works through AMS and allows RP to verify the validity of presented credential without the need to request any additional information from the issuer, which is important to ensure that presentations cannot be tracked by the issuer. On the other hand, the AMS does not learn which credential is being revoked or presented. We allow two types of credential revocation: by the user and by the issuer, where revocation by the user does not require communicating with the credential issuer.

In our system, the user should be able to decrypt stored credentials and generate signatures. We ensure the protection of the cryptographic keys that need to be stored on the user's mobile device by using server-supported threshold cryptography for signing and decryption, sharing the secret key between the user device and a supporting server. The same AMS can take the role of supporting server, storing the second key share and assisting in blind server-supported decryption and signing.

We describe the resulting system in terms of an ideal functionality, construct a protocol that implements that functionality, and provide a security proof in the universal composability [11] model.

This paper is organized as follows. The background and related work is given in Sec. 2. The building blocks that will be used by the proposed systems are

described in Sec. 3. The system itself is presented in Sec. 4. The efficiency of the system is evaluated in Sec. 5. The results are concluded in Sec. 6.

2 Background

2.1 Related work

There are several well-known attribute-based credential (ABC) schemes that maintain privacy of users, allowing users to prove possession of credentials without disclosing any additional information. Among these schemes are Idemix [9] and U-Prove [23]. Idemix is provably unlinkable and unforgeable, and its security relies on the strong RSA problem. There are several real-world implementations of Idemix as PrimeLife[3], and Yivi app with IRMA backend system[4]. U-Prove is more efficient compared to Idemix, as it can be implemented using elliptic curves, but it does not provide unlinkability. There is a number of other attribute-based systems that utilise bilinear pairings [10,26,4], where performing operations with pairings is more expensive than performing exponentiations on an elliptic curve.

Additionally, there is a line of works focusing on the cloud-based ABC systems, where a dedicated cloud service helps to present the users' credentials to the relying party. This approach is useful when there are efficiency constraints on the user's device. In [20] authors have proposed a scheme, where the computations have been outsourced to a semi-trusted cloud, and the main underlying cryptographic construction is proxy re-encryption. Subsequent work [16] focus on improvements of the proposed scheme and mitigating possible attacks.

The Request For Comments (RFC) [14] defines a mechanism that allows the user to selectively disclose attributes from the JSON Web Token (SD-JWT) for the presentation. The idea behind the solution is to create hash commitments to the attribute values that can be later opened by the user by providing the attribute value itself with the salt used to create commitments. Additionally, the RFC defines Key Binding JWT, where the user's signature over SD-JWT is included. This signature servers as proof of possession of the user's private key to the verifier of the credential presentation. More precisely, the user is required to sign an issuer-signed JWT and corresponding disclosures selected for the presentation to the verifier. The verifier uses the user's public key contained in SD-JWT to verify this signature. This format is referenced in the EUDI Wallet ARF [2] as one of a few suitable standardised formats for releasing electronic attestations of attributes. In this work, we follow this approach as it describes an option of selectively disclosing attributes.

Another approach referenced in the EUDI Wallet ARF is mobile driver's license (mDL) [18,27]. It is a digital representation of physical driving license, stored in the user's mobile device. The format of mobile credential allows to use it for various types of user attributes, not only as a driving license. Similarly to JWT, it uses hash commitments with random salt to allow selective disclosure of

[3] http://primelife.ercim.eu/about/factsheet, last visited December 2025
[4] https://docs.yivi.app/what-is-yivi/, last visited December 2025

attributes. A particular attribute management system CACS (Centralized Attribute Collection Service) [19] supports interaction of users, issuers, and relying parties, following mDL standards.

In this work, we follow the main idea of [19], letting a centralized system help to manage user's credentials. However, while [19] assumes a trusted server that is allowed to learn the user's data, in this work we are protecting the user's data from the server, improving the way credentials are stored and handled by the server. We also provide an integration of revocation mechanism, which has been missing in [19]. The usage of centralised server is motivated by the following reasons:

- EUDIW ARF defines the need in backup of a list of the user's attributes, attestations, and configurations. Thus, the attributes should be stored in a backup server by default.
- The server can assist in managing revocations.
- We need a centralised server for server-supported decryption and signing. This approach ensures that:
  - The adversary cannot decrypt attributes even if the adversary copies the mobile device's memory in between decryption sessions;
  - If the memory of the mobile device gets leaked during the decryption session, a special mechanism of the server-supported signature schemes (clone detection) allows to decrypt attributes for at most one presentation before the server stops to communicate.

2.2 System requirements

The requirements that we pose on our system are mainly motivated by [25] and [5]. In [5] (2024), Carsten Baum et al. have given a feedback to the European Digital Identity ARF regulation, pointing out important properties that a system should achieve by law. The survey by Maximilian Richter et al. [25] (2023) is more technical, and it lists some properties of a system that are classified as "Essential" and "Important", and references to possible solutions that allow to achieve these properties.

Let us first go through the *essential* properties of [25]. We mark with + properties that we address in Prettiness system, and with − that we do not.

+ *Completeness.* A valid credential will be accepted.
+ *Soundness.* An invalid credential will be rejected.
+ *Unforgeability.* A valid credential can only be created by the corresponding issuer. This can be achieved by letting the issuer sign a credential.
+ *Freshness of the presentation.* Every presentation has to be created anew for every verification. This can be achieved by letting the verifier produce a challenge that the user should respond to (e.g. in the proof of key ownership).
+ *Impossibility to duplicate credentials.* No one (except the issuer) may create a valid copy of a credential. This includes the ability of one user to present the same credential from multiple devices. In this work, we propose to use a

threshold key signature scheme with a clone detection mechanism [6] which gives a designated semi-trusted server the ability to detect that the signing protocol was initiated from different devices.
We note that the problem of *non-transferability* of the credentials is deeper, as we eventually cannot prevent a user from sharing their device with a friend. Using all-or-nothing approach [9] or one-time credentials that are not easy to obtain [8] would be possible, but we do not discuss such possibilities in this work, and leave these as optional additional mechanisms.

\+ *Revocation* is an essential feature that is difficult to implement efficiently while preserving privacy. An alternative approach to a separate revocation functionality is to set short validity time for the credentials, requiring credentials to be frequently re-generated. However, we believe that introducing separate revocation functionality is essential for the usability of the attribute management system in practice. While the user should be responsible for managing and storing their attributes, ARF also requires that the issuer should be able to revoke their issued credential at any time. Assume that the user has a driver licence issued and stored in the system. After an event requiring the police to revoke driver's licence occurs, police should be able to revoke the user's driver's licence by contacting the issuer.

Let us now go through non-essential, but *important* properties of [25].

\+ *Selective disclosure.* The user should be able to present only certain attributes of a credential. In the survey of [24], the methods for this are classified to hash-based, signature-based, and zero-knowledge based. In this work, we use a kind of hash-based disclosure.

\- *Unlinkability of presentations.* The verifier should not be able to correlate several presentations of the same credential. In this work, we are considering the setting where the relying party *should* learn the user's identity, so we are not aiming for this property. This is in accordance with (Article 5a, §16), as in this case the user *explicitly* authorises linkability. We see our system in the context of legal transactions by officially recognised issuers, like presenting driving/fishing licence, or getting access to a building. We do not aim to use it in anonymous transactions (like shop purchases).

\- *Repudiation.* RP should not be able to prove the authenticity of verified statements to a third party. In this work, we are not aiming for this property. In contrary, we believe that we may need non-repudiation for later accountability of transactions (although we leave it out of scope of this paper).

2.3 Participants and Threat model

In this section, we discuss the participants and the threat model for our system, including constraints on what a corrupted party should be allowed to see or do, motivated by the list of requirements on credential management systems [25].

User is the party to whom the attributes (and, hence, credentials) belong. A user can use their mobile device to get credentials from an issuer, present them

to a relying party (RP), and revoke credentials previously issued to them. User can be *dynamically* corrupted, and *corruption level* c_{U} may increase. There are the following corruption models for the user:

- $c_{\mathsf{U}} = 0$ (no corruption)
- Starting from $c_{\mathsf{U}} = 0$ (no corruption), the corruption level may rise to $c_{\mathsf{U}} = 1$ (partial corruption, after the key generation). This can advance further, moving to $c_{\mathsf{U}} = 2$ (partial corruption *and* learning user's PIN)
- $c_{\mathsf{U}} = 3$ (full corruption, before key generation)

The motivation behind introducing $c_{\mathsf{U}} = 1$ is that we want to ensure privacy of the user data even if its mobile device memory gets leaked (e.g. the device gets malware), but the secret key stored encrypted under the user's PIN is not leaked. For $c_{\mathsf{U}} = 2$, the attacker learns all the device memory, including the the secret key. This can happen if adversary guessed correct user's PIN. The final case, $c_{\mathsf{U}} = 3$ means that adversary has fully corrupted the user's device, including access to memory and ability to issue requests. In this case, the attacker learns everything, and there are no privacy guarantees for the user.

The adversary can corrupt the user before key generation, reaching level $c_{\mathsf{U}} = 3$. Otherwise, $c_{\mathsf{U}} = 0$ is assumed during the setup phase, and the adversary may increase the corruption levels later. The adversary can clone device memory, reaching level $c_{\mathsf{U}} = 1$ (the main focus of security analysis). If the adversary guesses PIN, it proceeds to $c_{\mathsf{U}} = 2$, and there are no privacy guarantees for the user anymore.

The reason for analysing our system in such corruption model is that static corruption of the user (from the beginning of the protocol) does not allow us to rely on specific properties introduced by server-supported signing protocol. To make use of the user's key protection mechanisms, we need to assume that the user has not been corrupted during the key generation and adversary can copy device's memory.

Attribute management service (AMS) is responsible for helping the user to manage storage and interactions with their credentials. AMS server can be *statically* corrupted. In particular, we assume a *covertly* [3] (explained in more details in Sec. 2.4) corrupted server, which may deviate from the prescribed routines, but will not do it if the probability of being caught is sufficiently large. Such an assumption is reasonable for AMS established by an institution which cares for its reputation.

AMS server is allowed to see credential issuance requests submitted by the user, including the issuer identifier (as AMS is responsible for establishing connection). It is also allowed to see the *identifiers* of requested attribute, as these likely depend on the issuer identity anyway. However, AMS server should not learn the *values* of attributes that are a part of issued credential.

If AMS has not seen the credential yet (e.g. from an issuer or a RP), it should not be able to learn when that credential has been revoked. Also, it should not be able to track which credentials have been presented, and to which RP (unless RP is also corrupted).

Issuer is responsible for issuing credentials to the user, signed with the issuer's private key. Additionally, the issuer has authority to submit credential revocation requests to the AMS for the law enforcement purposes. Issuer can be *statically* corrupted. The issuer should not be able to track presentations (unless colluded with RP) and revocations by the user (unless colluded with AMS) of its credentials.

Relying Party (RP) is a party requesting to see certain user's attributes to authenticate user or to give access to certain services. RP can be *statically* corrupted. From presentation, RP should not be able to learn anything except the attributes than the user has explicitly shown to it. However, we do not aim for unlinkability of presentations, as this can be tracked through the issuer's signature. Differently from the anonymous credentials of Idemix [9], we are assuming standard signatures like RSA and ECDSA (for compatibility with current standards), which cannot be updated for every presentation without the issuer. Nevertheless, we see the possibility of unlinkability using proofs of ECDSA signatures in zero-knowledge as in [15]. This addition is out of scope of this paper. Similarly to the EUDIW defined in ARF [2], our solutions supports methods proposed in ARF (section 7.4.3.5.2) for dealing with RP linkability. For the *once-only attestations*, user would need to run issuing protocol several times to obtain batch of the credentials (for the same attributes). Then each credential is used only once. For the *limited-time attestations*, issuer will need to include additional entry to the credential, defining the validity period of credential. However, even with this method we still allow to use long-term attestations with revocation. For the *rotating-batch attestations*, user would need to run issuing protocol several times to obtain batch of the credentials. During the presentation, user would randomly choose which credential from the batch to present. This method does not restrict credential to be used only once. In this case, random selection of credential for presentation and resetting the batch happens in the environment. Finally, *Per-Relying Party attestations* method can be achieved through parametrising $\mathcal{F}_{\mathsf{ideal}}$ with RP. Meaning that the user will use different instance of $\mathcal{F}_{\mathsf{ideal}}$ for each RP.

2.4 Covert Adversary

In this work, we will make some assumption on the corruption of the AMS server. Since AMS is assumed to be a reputable entity, one option would be to assume a honest-but-curious server, which will not deviate from prescribed protocols. However, in practice it could be a too strong assumption, as even a reputable entity could be capable of silent cheating, which would not be detected, but could potentially harm user privacy. To ensure that such silent attacks are not possible, we make use of the notion of a *covert* adversary, proposed by Aumann and Lindell [3]. Intuitively, a covertly corrupted party will not cheat if it will be caught with a certain non-negligible probability.

In this work, we strengthen this assumption. We assume that the AMS will not cheat if it will be *possible* to prove the fact of cheating to a third party. In particular, while AMS will be technically able to conceal the fact of revocation

from the verifier, the revoker and the verifier together will be able to detect the problem and prove to a third party (e.g. external judge) that AMS has cheated.

2.5 Universal Composability

Universal composability (UC) model [11] follows the real-ideal world paradigm. In the real protocol execution, honest parties run the protocol π over a network controlled by an adversary $\mathcal{A}$, who can also corrupt some parties by sending a special corruption message directly to the party. In the ideal world, honest parties are viewed as *dummy parties* who forward their inputs directly to the ideal functionality $\mathcal{F}$ that internally executes the main task of the protocol, allowing the adversary to send certain corruption messages to a special interface of $\mathcal{F}$, whose effect on the outcome of $\mathcal{F}$ is specified explicitly. In hybrid model, ideal functionalities can in turn be used by parties of the real protocol, and the adversary may communicate with them directly via adversary interface.

There is an environment $\mathcal{Z}$, representing the whole world outside of the protocol, which provides inputs to the parties and receives corresponding outputs. The protocol π securely realises ideal functionality $\mathcal{F}$ if for every real adversary $\mathcal{A}$ attacking the protocol there exists an ideal adversary (simulator) $\mathcal{S}$, such that no environment $\mathcal{Z}$ can distinguish whether it is interacting with the protocol π and real adversary $\mathcal{A}$ or with ideal functionality $\mathcal{F}$ and simulator $\mathcal{S}$. Intuitively, any attack on the environment that is possible in the real world, is also possible in the ideal world.

3 Building Blocks

3.1 Signing functionality

For the digital signatures that are needed to sign issued attributes, we propose to use any of the standardised signature schemes like ECDSA, RSA. For the security analysis, we use digital signature ideal functionality $\mathcal{F}_{\mathsf{sign}}$ proposed by Canetti [12], depicted in Figure 1. In contrast to the regular signing functionalities, it does not have the key generation interface. This functionality binds produced signatures to the identity of the signer (issuer). Intuitively, in the real world, it represents sending signatures with *certificates* that bind the verification process to the identity of signer, not the public key. In this functionality, the signature σ is generated by the adversary. For signatures generated outside of the ideal functionality by a corrupted signer, there are no verification guarantees, and the adversary may decide the outcome of verification, with the only requirement that the outcome of verification should be consistent with previous verifications.

Moreover, we use blind threshold signatures on the user's side to protect the storage of private key material. We use functionality $\mathcal{F}_{\mathsf{tsign}}$ described in [28] for the two-party threshold blind signature scheme with clone detection (functionality is presented in Figures 17, 18 in App. A). Since the scheme assumes the use-case where one share of the private key is stored on the user's mobile device

Signature Generation. Upon receiving $(\mathsf{sign}, \mathsf{sid}, m)$ from S, send $(\mathsf{sign}, \mathsf{sid}, m)$ to the $\mathcal{A}$. Upon receiving $(\mathsf{signature}, \mathsf{sid}, m, \sigma)$ from the $\mathcal{A}$, verify that no entry $(m, \sigma, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\mathsf{signature}, \mathsf{sid}, m, \sigma)$ to S, and record the entry $(m, \sigma, 1)$.
Signature Verification. Upon receiving a value $(\mathsf{verify}, \mathsf{sid}, m, \sigma)$ from some party P, send $(\mathsf{verify}, \mathsf{sid}, m, \sigma)$ to the $\mathcal{A}$. Upon receiving $(\mathsf{verified}, \mathsf{sid}, m, \phi)$ from the $\mathcal{A}$, do:

- If $(m, \sigma, 1)$ is recorded then set $f = 1$.
- Else, if the signer is not corrupted, and no entry $(m, \sigma', 1)$ for any σ' is recorded, then set f = 0 and record the entry $(m, \sigma, 0)$.
- Else, if there is an entry (m, σ, f') recorded, then set $f = f'$.
- Else, set $f = \phi$, and record the entry (m, σ', ϕ).

Output $(\mathsf{verified}, \mathsf{sid}, m, f)$ to P.

Fig. 1. The signing functionality $\mathcal{F}_{\mathsf{sign}}$

(client), client's share of the private key in storage is protected with the knowledge factor (PIN code) required from the user for each signing session. The other share is stored on the supporting server side, and we assume that AMS takes this role. The usage of *blind* threshold signature ensures that, for the signatures created for the verifiable presentation, supporting server does not learn which credential is being presented. In this scheme, the message is blinded only from the supporting server. We note that the blindness requirement is only important for the presentation phase. For the signatures that are used to initialise other requests, AMS server is still expected to see and verify resulting (unblinded) signature.

The functionality $\mathcal{F}_{\mathsf{tsign}}$ lists the following routines – Key generation, Signing, Verification, Corruption. Additionally, $\mathcal{F}_{\mathsf{tsign}}$ has some triggers that are used internally by functionality and not accessible to the environment.

Key generation is initiated by both client and server. $\mathcal{F}_{\mathsf{tsign}}$ outputs public key pk, which is allowed to be chosen by adversary.

Signing is initiated by both client and server. Message to be signed is input by the client, and the server does not know the message. $\mathcal{F}_{\mathsf{tsign}}$ verifies correctness of PIN code provided by the user and calls clone detection mechanism. If both verification checks are successful, $\mathcal{F}_{\mathsf{tsign}}$ outputs signature σ (only to the client), that is allowed to be generated by adversary.

Verification can be queried by any party to verify the correctness of the signature. Verification procedure is similar to the one presented in regular signing functionality $\mathcal{F}_{\mathsf{sign}}$. A corrupted client cannot produce more valid signatures than the number of signing sessions approved by the server.

Corruption allows adversary to corrupt parties throughout the protocol. [28] defines several corruption levels for the client and server components. Corruption levels for the sever are trivial consisting of just two options $\mathsf{c_S} = 0$ (uncorrupted)

or $\mathsf{c}_\mathrm{S} = 1$ (corrupted – adversary is being in full control of the server). Corruption of the client consists of more levels:

- $\mathsf{c}_\mathrm{P} = 0$ –uncorrupted
- $\mathsf{c}_\mathrm{P} = 1$ –adversary learned phone's memory *between* the signing sessions (encrypted memory). It means that all the sensitive values that are recomputed for signing are deleted from memory, and only the values in encrypted form or less sensitive values can be cloned by adversary.
- $\mathsf{c}_\mathrm{P} = 2$ –adversary learned phone's memory *during* a signing session (unencrypted memory). It means that adversary can access the values that are recomputed for signing, including private key share.
- $\mathsf{c}_\mathrm{P} = 3$ –adversary has full control over the phone. It means that adversary learns all the updated values in the memory and can initiate interactions.

$\mathcal{F}_{\mathsf{tsign}}$ is internally keeping track of the corruption levels of parties, and the source of the last signing query to support additional security properties (clone detection).

corruption level	**explanation**	**consequences**	**how to achieve?**
$\mathsf{c}_\mathrm{P} = 0$	not corrupted	nothing is leaked to adversary	-
$\mathsf{c}_\mathrm{P} = 1$	memory between signing sessions (encrypted memory) is leaked	adversary learns values stored in device's memory, including *encrypted* share of the private key	clone memory of the device after key generation, in between signing sessions
$\mathsf{c}_\mathrm{P} = 2$	memory during the signing session (unencrypted memory) is leaked	adversary learns values stored in device's memory as well as values generated during signing session, including *unencrypted* share of the private key	clone memory of the device during signing process **OR** guess the user's PIN after learning encrypted memory
$\mathsf{c}_\mathrm{P} = 3$	device is fully corrupted	adversary gains full access to the device, including learning its memory and being able to initialise some commands	get full access to the device, including during the key generation phase

Table 1. Corruption levels

Since in Prettiness, we only assume static corruptions after the key generation phase, we do not need to use all the previously defined corruption levels for client. When client gets corrupted after key generation, the corruption level is set to $\mathsf{c}_\mathrm{P} = 1$. During the interaction with protocol adversary may guess client's PIN, which will increase corruption level to $\mathsf{c}_\mathrm{P} = 2$. Since we do not allow adaptive corruptions, adversary cannot get to level $\mathsf{c}_\mathrm{P} = 3$, which would mean gaining full control over the user's device. However, adversary can corrupt user to level $\mathsf{c}_\mathrm{P} = 3$ before the key generation, but in this case we cannot provide any security guarantees for the user.

3.2 Threshold decryption

For decryption of the data stored at the AMS server, we propose to use privacy-preserving server-supported decryption from [21]. The scheme ensures privacy against the server, meaning that when the client initiates decryption session, it sends a blinded ciphertext so the server is not able to learn which ciphertext exactly is being decrypted. We use functionality $\mathcal{F}_{\mathsf{tdecrypt}}$ defined in [21] (presented in Figures 15 and Figure 16 in App. A).

Key generation is initiated by the client and the server, its functionality is similar to functionalities for public-key encryption, defined by Canetti et al. [13], where the public key pk is chosen by the adversary.

Encryption can be queried by any party, providing a plaintext message m as input. The ciphertext c is created by adversary, who does not receive plaintext m as an input. Pair (m, c) is saved to the internal table T.

Decryption functionality has more nuanced details, but the main idea for $\mathcal{F}_{\mathsf{tdecrypt}}$ is to find the plaintext m from a ciphertext c that was created using the key pk. This functionality is initiated by both parties, client and server with the client being the one who provides the ciphertext c as input. A corrupted client cannot decrypt more ciphertexts than the number of decryptions approved by the server.

Additionally, we introduce a new interface to the functionality $\mathcal{F}_{\mathsf{tdecrypt}}$ – verification of decryption. After receiving signed encrypted credential and corresponding disclosure, RP can use it to verify that the disclosure indeed corresponds to the ciphertext from signed credential.

Verify decryption requires inputs from the user and some other verifier party (in our case, the RP). Given user consent, $\mathcal{F}_{\mathsf{tdecrypt}}$ tells to the verifier whether the pair (m, c) provided as input has been saved in table T of previously encrypted ciphertexts.

We propose to extend threshold decryption protocol from [21] with this interface as follows. In the protocol of [21] that UC-realises $\mathcal{F}_{\mathsf{tdecrypt}}$, the ciphertext is of the form $c = (\mathsf{SE}(k, m), H'(k, m))$ for some hash function (random oracle) H' where $\mathsf{SE}(k, m)$ is symmetric encryption of the message m with the key k, the second component $H'(k, m)$ makes the decryption unique, and the single-use key k is derived through KEM (key exchange mechanism) between the client and the server.

1. Together with the ciphertext c, the user reveals the message m and the single-use KEM key k to decrypt c.
2. The verifier parses c as (c_1, c_2) and performs the following checks:
   - check that $m = \mathsf{SD}(k, c_1)$ (symmetric decryption of c_1 with k).
   - check that $H''(k, m') = c_2$ (note that this check does not allow the user to reveal a different k and make c open to a different message).

3.3 Revocation mechanism

For revocation of credentials, we use functionality of [22]. This functionality allows to extend the credential with a special *revocation token* in such a way that

only a designated revoking party will be able to revoke that token. A credential that is linked to a revoked token is also considered revoked. We will assign the revoker role to *both* the user and the issuer, creating *two* distinct revocation tokens for one credential. This ensures that the issuer will not learn that the user has revoked the credential issued by that issuer. We use functionality $\mathcal{F}_{\mathsf{rev}}$ defined in [22] (presented in Figures 13 and 14 in App. A).

In this functionality, we have the server S (in practice, the AMS server) who maintains the set X^- of revoked tokens, the revoker R (the issuer or the user) who gets the issued token and may revoke it later, the user U who has the right to query token revocation status, and the verifier V (in practice the RP) who learns x and whether $x \in X^-$.

Initialisation. The adversary chooses the universal set $\mathcal{X}$ of tokens. The size of $\mathcal{X}$ is given as a parameter to $\mathcal{F}_{\mathsf{rev}}$, so it is not allowed to be too small.

Issuing. $\mathcal{F}_{\mathsf{rev}}$ samples a fresh token $x \leftarrow\!\!\$\ \mathcal{X}$ that it outputs to the user and to the revoker. In our case, the revoker party can be the user or the issuer. The issuer may then include x into a credential as a special attribute (this will happen in the embedding protocol, outside of $\mathcal{F}_{\mathsf{rev}}$).

Revocation. A party R may revoke x if $\mathcal{F}_{\mathsf{rev}}$ has previously declared R as revoker of x. A revoked token is added to the set X^- of revoked tokens.

User notification. The user U may ask from $\mathcal{F}_{\mathsf{rev}}$ whether a token, that belongs to the user U, has already been revoked. Such a notification would be called just before verification, to ensure freshness of revocation status.

Verifier notification. $\mathcal{F}_{\mathsf{rev}}$ remembers the current state of revocation database for future reference, without leaking anything to V. Such a notification could be triggered just before verification, but having it as a separate routine can be more practical.

Verification. The user presents a token x. The verifier learns x and whether x has been revoked at the moment when V made its last *verifier notification* request.

Cheating detection. The functionality $\mathcal{F}_{\mathsf{rev}}$ proposes interfaces for cheating detection, which allow the revoker and the verifier together to figure out whether any revocation by that revoker has been silenced from the verifier. This interface is important, assuming that the covertly corrupted server will not cheat if it will be accountable for cheating. We formalise this assumption in Definition 1.

Definition 1 (covertly corrupted AMS). *We say that AMS is* covertly *corrupted if the adversary will not interact with the protocol in a way that allows an honest revoker R and a honest verifier V to make cheating detection interface of $\mathcal{F}_{\mathsf{rev}}$ report AMS as cheater.*

4 Attribute management system

4.1 Ideal functionality

We start with defining system functional and security requirements in the form of ideal functionality. The ideal functionality $\mathcal{F}_{\mathsf{ideal}}$ is inspired by the previous

works [7,17] with some adjustments to fit design principles and requirements, following the model defined in Section 2.3. We can view $\mathcal{F}_{\mathsf{ideal}}$ as an ideal credential storage, which stores issued credentials, and also allows to revoke them, and present to third parties. Let us use ST to denote the storage, which in practice will be realised by a single AMS server. Ideal functionality provides the following interfaces – Initialisation, Issue, Present, Revoke, Verify, Corrupt. Every call to such an interface includes a session identifier sid, which is needed to separate calls from each other. It does not contain any private data by itself, but leaking sid to $\mathcal{A}$ allows it to match the session against a certain call from the environment.

Initialisation is used to initialise tables $\mathsf{T}_a, \mathsf{T}_r, \mathsf{T}_p, \mathsf{T}_v$, for storing issued attributes, revocation information, credential presentations and verification information correspondingly. Additionally, $\mathsf{c_U} = 0$ gets initialised, storing information about the corruption level of user U, as in Section 2.3. Next, we let adversary define Present and Ver functions. Present generates credential presentation for the presentation phase, and Ver is the corresponding verification function. We need these functions to be able to output a particular presentation to $\mathcal{Z}$ to be shown later to a verifier, who can verify it using $\mathcal{F}_{\mathsf{ideal}}$.

Silent request is initiated by adversary at any point of time trying to query Issue, Revoke or Present interfaces on behalf of corrupted initiator (user or issuer). For a user, $\mathcal{F}_{\mathsf{ideal}}$ firstly calls the authentication sub-routine which verifies user corruption level. If *user is not fully corrupted* ($\mathsf{c_U} < 2$)*, adversary is not allowed to make those requests* to the $\mathcal{F}_{\mathsf{ideal}}$. If the user is fully corrupted ($\mathsf{c_U} \geq 2$), $\mathcal{F}_{\mathsf{ideal}}$ proceeds with the requested interface using input provided by the adversary. Finally, $\mathcal{F}_{\mathsf{ideal}}$ gives the user's output to the adversary. This way, even though the adversary will see all the inputs and outputs of partially corrupted users, it cannot initiate any routines on their behalf.

Issue is initiated by the user query containing information about the requested attribute(s). If nobody is corrupted, adversary just gets notification that the communication is happening, and is able to interrupt the communication. We are not hiding issuer's identity and the issued $\overrightarrow{\mathsf{aID}}$, from a corrupted storage (ST). This is a functional requirement: to make querying by credential identifier cID usable in practice, we need that it would be self-explanatory, and we also need these values for the transaction log. Actually, since in our model, ST knows the issuer identifier, it will likely anyway be able to guess which kinds of attributes have been requested by the user.

The attribute request is sent to the issuer and, if not interrupted by the adversary, the issued credential consisting of attributes $\overrightarrow{\mathsf{atr}}$ is recorded to the table T_a. Every record in table T_a serves as an issued credential. While credential identifier cID is chosen by the adversary, $\mathcal{F}_{\mathsf{ideal}}$ checks that it is unique for an honest user and an honest storage. After attributes are received from the issuer, *corrupted storage learns only* cID*, but not the values of attributes.*

Revoke is initialised either by the user or issuer query containing identifier cID of the credential to be revoked. $\mathcal{F}_{\mathsf{ideal}}$ verifies that a credential with cID exists. The adversary learns cID provided by an honest P if either (1) the user of cID is corrupted and hence is able to trace revocation, or (2) ST and some

RP to which cID has been presented before, are both corrupted (in that case, adversary already learned the link between the credential and corresponding revocation token). Otherwise, $\mathcal{F}_{\mathsf{ideal}}$ just notifies adversary that the revocation process has been initiated. If the revoking party is an issuer, $\mathcal{F}_{\mathsf{ideal}}$ also leaks to corrupted storage the identity of the user, owner of the revoked credential. In practice, this is needed purely for logging purposes, so that AMS could show to the user only those revocation logs which concern that user.

After receiving approval from the adversary, $\mathcal{F}_{\mathsf{ideal}}$ records that cID has been revoked by user or issuer in table T_r and confirms successful revocation to the revoker.

Present is initiated by the user query containing information about the credential cID and attribute(s) $\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}$ that need to be presented. First of all, $\mathcal{F}_{\mathsf{ideal}}$ verifies whether the requested credential is in the table T_a. $\mathcal{F}_{\mathsf{ideal}}$ sends only the user identifier and the number of attributes to the adversary, even if AMS and the issuer are corrupted. If presentation is approved by the adversary, $\mathcal{F}_{\mathsf{ideal}}$ verifies that credential has not been revoked, by consulting table T_r. If the check passes, $\mathcal{F}_{\mathsf{ideal}}$ generates presentation for the requested credential using $\mathsf{Present}$ function. The obtained credential presentation is output to the user.

Verify is initiated by the user and RP. If the AMS server (with or without the issuer of presented credential) is corrupted, adversary does not learn neither RP identifier nor credential identifier. It helps to capture the fact that *AMS server does not learn RP identifier or which credential is being presented by the user or which attributes are being disclosed during the presentation.* $\mathcal{F}_{\mathsf{ideal}}$ verifies that presented credential has not been revoked, and has been recorded by the issuer. Note that, as far as no revocations took place between a presentation and a revocation, both checks always verify for the presentations generated through the $\mathcal{F}_{\mathsf{ideal}}$.

Corrupt allows adversary to submit corruption requests identifying which party is being corrupted. User corruption is handled as described in Sec. 2.3.

Properties of $\mathcal{F}_{\mathsf{ideal}}$. In Table 2, we summarize which values are leaked to the adversary depending on the corruption and the queried interface of $\mathcal{F}_{\mathsf{ideal}}$. During the issuing, supporting server learns attribute values only if it colludes with issuer. Otherwise, it learns only credential identifier and attribute identifiers. During the revocation, AMS server learns identifier of revoked credential only if it colludes with the RP to which this credential has been presented before. During the presentation, AMS server learns disclosed attributes only if it colludes with RP. The only coalition that can learn all the attributes from presented credential consists of both the issuer and the RP, since the issuer has all the information available about the user's attributes, and the RP sees when the credential is being presented. Only AMS and the RP together may trace revocation status of presented credentials.

We claim that functionality $\mathcal{F}_{\mathsf{ideal}}$ satisfies the following properties, formally stated and proved in below.

Initialisation. On input $(\mathsf{setup}, \mathsf{sid})$ from $\mathcal{A}$:

- Create empty tables $\mathsf{T}_a, \mathsf{T}_r, \mathsf{T}_p, \mathsf{T}_v$, set flag $\mathsf{c}_\mathsf{U} = 0$.
- Send $(\mathsf{setup\text{-}ok}, \mathsf{sid})$ to $\mathcal{A}$ and wait for $(\mathsf{setup\text{-}ok}, \mathsf{sid}, \mathsf{Ver}, \mathsf{Present})$ from $\mathcal{A}$.

Corrupted inputs. At any time when $(\mathsf{routine}, \mathsf{sid}, x)$ is called from $\mathcal{Z}$ by a party $P \in \mathcal{C}$, send $(\mathsf{input}, P, \mathsf{routine}, \mathsf{sid}, x)$ to $\mathcal{A}$. Wait for $(\mathsf{input}, P, \mathsf{routine}, \mathsf{sid}, x')$ from $\mathcal{A}$ and run routine with input x' instead of x.
Silent request. At any time, on input $(\mathsf{invoke}, \mathsf{routine}, \mathsf{sid}, P, \mathsf{inputs})$ from $\mathcal{A}$:

- If $\mathsf{ST} \in \mathcal{C}$, or $P = \mathsf{I}$, authentication is not required.
- Otherwise $\mathsf{authenticate}(\mathsf{sid}, \mathsf{U})$, proceed if successful.

Wait for $(\mathsf{input}, P, \mathsf{routine}, \mathsf{sid}, x)$ from $\mathcal{A}$ and follow routine interface x as input of party P. Return output of routine to $\mathcal{A}$.
Issue. On input $(\mathsf{issue\text{-}req}, \mathsf{sid}, \overrightarrow{\mathsf{aID}}, \mathsf{I})$ from U:

- If $\mathsf{I} \in \mathcal{C}$ or $\mathsf{ST} \in \mathcal{C}$ send $(\mathsf{issue\text{-}req}, \mathsf{sid}, \mathsf{U}, \mathsf{I}, \overrightarrow{\mathsf{aID}})$ to $\mathcal{A}$.// I and ST learn $\overrightarrow{\mathsf{aID}}$
  Otherwise, send $(\mathsf{issue\text{-}req}, \mathsf{sid}, \mathsf{U}, \mathsf{I}, |\overrightarrow{\mathsf{aID}}|)$ to $\mathcal{A}$.
- Wait for $(\mathsf{issue\text{-}ok}, \mathsf{sid}, \mathsf{cID})$ from $\mathcal{A}$. Check that cID is unique.
  Send $(\mathsf{issue\text{-}req}, \mathsf{sid}, \mathsf{U}, \mathsf{cID}, \overrightarrow{\mathsf{aID}})$ to I.
- Upon receiving $(\mathsf{issue\text{-}atr}, \mathsf{sid}, \overrightarrow{\mathsf{atr}})$ from I:
  - If $\mathsf{U} \in \mathcal{C}$ send message $(\mathsf{issue\text{-}atr}, \mathsf{sid}, \overrightarrow{\mathsf{atr}})$ to $\mathcal{A}$.// U learns $\overrightarrow{\mathsf{atr}}$
  - Otherwise, send message $(\mathsf{issue\text{-}atr}, \mathsf{sid})$ to $\mathcal{A}$.
- Wait for $(\mathsf{atr\text{-}ok}, \mathsf{sid})$ response from $\mathcal{A}$.
- Record $(\mathsf{U}, \mathsf{I}, \mathsf{cID}, \overrightarrow{\mathsf{aID}}, \overrightarrow{\mathsf{atr}})$ to table T_a. Send $(\mathsf{issue\text{-}resp}, \mathsf{sid}, \mathsf{cID})$ to U and I.

Get all credential identifiers. On input $(\mathsf{getcids\text{-}req}, \mathsf{sid})$ from U:

- Send $(\mathsf{getcids\text{-}req}, \mathsf{sid}, \mathsf{U})$ to $\mathcal{A}$. Wait for $(\mathsf{getcids\text{-}ok}, \mathsf{sid})$ from $\mathcal{A}$
- Let $\overrightarrow{\mathsf{cID}}$ be all identifiers stored in T_a. Output $(\mathsf{getcids\text{-}resp}, \mathsf{sid}, \overrightarrow{\mathsf{cID}})$ to U.

Revoke. On input $(\mathsf{revoke\text{-}req}, \mathsf{sid}, \mathsf{cID})$ from P:

- Find an entry $(\mathsf{U}, \mathsf{I}, \mathsf{cID}, \_, \_) \in \mathsf{T}_a$, s.t. $P \in \{\mathsf{I}, \mathsf{U}\}$.
  If there is no such entry, send $(\mathsf{revoke\text{-}resp}, \mathsf{sid}, \bot)$ to the party P.
- Produce leakage to the adversary:
  - If $\mathsf{ST} \in \mathcal{C}$ and $(\mathsf{RP}, \mathsf{cID}) \in \mathsf{T}_p$ for $\mathsf{RP} \in \mathcal{C}$, send $(\mathsf{revoke\text{-}req}, \mathsf{sid}, P, \mathsf{cID})$ to $\mathcal{A}$.
    // credential has been presented to the corrupted RP before
  - If $\mathsf{U} \in \mathcal{C}$ send $(\mathsf{revoke\text{-}req}, \mathsf{sid}, P, \mathsf{cID})$ to $\mathcal{A}$.
    // corrupted user learns if their credential is revoked
  - Otherwise, send $(\mathsf{revoke\text{-}req}, \mathsf{sid}, P, \mathsf{U})$ to $\mathcal{A}$.
- Wait for $(\mathsf{revoke\text{-}ok}, \mathsf{sid})$ response from $\mathcal{A}$.
- If there is no entry $(\mathsf{U}, \mathsf{cID}, P)$ in table T_r, add $(\mathsf{U}, \mathsf{cID}, P)$ to the table T_r.
- Send $(\mathsf{revoke\text{-}resp}, \mathsf{sid}, \mathsf{ok})$ to the party P.

Fig. 2. Ideal functionality $\mathcal{F}_{\mathsf{ideal}}$ (Part 1)

Present. On input $(\mathsf{present\text{-}req}, \mathsf{sid}, \mathsf{cID}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}})$ from U:
// values in $\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}$ are identifying attributes that will be disclosed to the RP

- Find an entry $\mathsf{cred} = (\mathsf{U}, \mathsf{I}, \mathsf{cID}, \overrightarrow{\mathsf{aID}}, \overrightarrow{\mathsf{atr}}) \in \mathsf{T}_a$ with $\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}} \subseteq \overrightarrow{\mathsf{aID}}$.
  If there is no such entry, send $(\mathsf{present\text{-}resp}, \mathsf{sid}, \perp)$ to U.
- Send $(\mathsf{present\text{-}req}, \mathsf{sid}, \mathsf{U}, |\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}|)$ to $\mathcal{A}$. Wait for $(\mathsf{present\text{-}ok}, \mathsf{sid})$ from $\mathcal{A}$.
- If $(\mathsf{U}, \mathsf{cID}, \_) \in \mathsf{T}_r$, send $(\mathsf{present\text{-}resp}, \mathsf{sid}, \perp)$ to U.
- Generate $\mathsf{pres} = \mathsf{Present}(\mathsf{cred}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}})$.
  If $\mathsf{Ver}(\mathsf{cred}, \mathsf{pres}) = 1$, add $(\mathsf{cred}, \mathsf{pres}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, 1)$ to T_v.
  Otherwise, send $(\mathsf{present\text{-}resp}, \mathsf{sid}, \perp)$ to U
- Send $(\mathsf{present\text{-}resp}, \mathsf{sid}, \mathsf{pres})$ to U.

Verify. On input $(\mathsf{verify\text{-}req}, \mathsf{sid}, \mathsf{pres}, \mathsf{RP})$ from U:

- If $\mathsf{U} \notin \mathcal{C}$, find $(\mathsf{cred} = (\mathsf{U}, \_, \mathsf{cID}, \_, \_), \mathsf{pres}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, 1) \in \mathsf{T}_v$. Add $(\mathsf{RP}, \mathsf{cID})$ to T_p.
  // honest users only input presentations previously produced by $\mathcal{F}_{\mathsf{ideal}}$
  If $\mathsf{U} \in \mathcal{C}$, find $\mathsf{cred} = (\_, \_, \mathsf{cID}, \_, \_) \in \mathsf{T}_a$ s.t. $\mathsf{Ver}(\mathsf{cred}, \mathsf{pres}) = 1$ and $(\_, \mathsf{cID}, \_) \notin \mathsf{T}_r$.
  // presentation of non-issued attributes cannot be forged
  // revoked credentials cannot be presented
  In both cases, if there is no such entry, send $(\mathsf{verify\text{-}resp}, \mathsf{sid}, \perp)$ to U and RP.
- Produce leakage to the adversary.
  - If $\mathsf{RP} \in \mathcal{C}$ and $\mathsf{ST} \in \mathcal{C}$, send $(\mathsf{verify\text{-}req}, \mathsf{sid}, \mathsf{U}, \mathsf{cID}, \mathsf{pres}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, S_{ID})$ to $\mathcal{A}$,
    where $S_{ID} \leftarrow \{\mathsf{sid} \mid (\mathsf{revoke\text{-}req}, \mathsf{sid}, \mathsf{U}, \mathsf{cID})$was input to $\mathcal{F}_{\mathsf{ideal}}\}$.
  - Otherwise, if $\mathsf{RP} \in \mathcal{C}$, send $(\mathsf{verify\text{-}req}, \mathsf{sid}, \mathsf{U}, \mathsf{cID}, \mathsf{pres}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}})$ to $\mathcal{A}$.
  - Otherwise, send $(\mathsf{verify\text{-}req}, \mathsf{sid}, \mathsf{U})$ to $\mathcal{A}$.
- Wait for $(\mathsf{verify\text{-}ok}, \mathsf{sid}, \mathsf{ch})$ from $\mathcal{A}$. If $\mathsf{RP} \notin \mathcal{C}$, check $\mathsf{ch} \in \mathcal{CH}$.
- Set $\mathsf{pres}' \leftarrow (\mathsf{pres}, \mathsf{ch})$. Set $\mathcal{CH} \leftarrow \mathcal{CH} \setminus \{\mathsf{ch}\}$.
- Send $(\mathsf{verify\text{-}resp}, \mathsf{sid}, \mathsf{pres}', v)$ to RP and to U.

Corrupt.

- // corruption in the beginning of the protocol
  On input $(\mathsf{corrupt\text{-}party}, P, \mathsf{sid})$ from $\mathcal{A}$ where $P \in \{\mathsf{I}, \mathsf{RP}, \mathsf{ST}, \mathsf{U}\}$,
  set $P \in \mathcal{C}$ (if $P = \mathsf{U}$, set $\mathsf{c_U} = 3$) and send $(\mathsf{corrupted}, P, \mathsf{sid})$ to $\mathcal{A}$.
- // corruption after initialisation for user
  On input $(\mathsf{corrupt\text{-}party}, \mathsf{U}, \mathsf{sid})$ from $\mathcal{A}$:
  - if $\mathsf{c_U} = 0$, set $\mathsf{c_U} = 1$ and $\mathsf{U} \in \mathcal{C}$;
  - if $\mathsf{c_U} = 1$, set $\mathsf{c_U} = 2$ and send $(\mathsf{full\text{-}leakage}, \mathsf{sid}, \mathsf{U}, (\mathsf{T}_a^{\mathsf{U}}, \mathsf{T}_r^{\mathsf{U}}))$ to $\mathcal{A}$, where T^{U} is projection of table T to the user U.

$\mathsf{authenticate}(\mathsf{sid}, \mathsf{U})$:

- Send $(\mathsf{auth}, \mathsf{sid})$ to $\mathcal{A}$ and wait for $(\mathsf{auth\text{-}ok}, \mathsf{sid})$.
- If $\mathsf{c_U} \in \{0, 1\}$, ignore this query.
  // with partial corruption, no queries allowed
- If $\mathsf{c_U} \in \{2, 3\}$, proceed with the protocol.

Fig. 3. Ideal functionality $\mathcal{F}_{\mathsf{ideal}}$ (Part 2)

	Issue			Revoke	Present		
corrupted parties	cID	$\overrightarrow{\mathsf{aID}}$	$\overrightarrow{\mathsf{atr}}$	cID	$\overrightarrow{\mathsf{aID}}$	$\overrightarrow{\mathsf{atr}}$	cID
issuer	yes	yes	yes	no	no	no	no
AMS	yes	yes	no	no	only $\lvert\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}\rvert$	no	no
RP	no	no	no	no	only $\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}$	only $\overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}$	yes
issuer + RP	yes	yes	yes	no	yes* (via cID)	yes* (via cID)	yes
AMS + issuer	yes	yes	yes	no	only $\lvert\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}\rvert$	no	no
AMS + RP	yes	yes	no	yes*	yes* (via cID)	only $\overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}$	yes
AMS + issuer + RP	yes	yes	yes	yes	yes	yes	yes

Table 2. User data disclosed to corrupted parties, where cID and $\overrightarrow{\mathsf{aID}}$ are credential and attribute identifiers, $\overrightarrow{\mathsf{atr}}$ are attribute values, $\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}$ are the attributes used in a presentation, and $\overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}$ is the disclosure of attribute values coming from that presentation. Disclosures marked by * are possible only for the given coalition of corrupted parties, and are *not* derived from smaller coalitions. Highlighted disclosures (yes) are intentional and needed for functionality.

- The user and the AMS server cannot convince RP of possession of attributes that have not been issued to the user using issuing interface of $\mathcal{F}_{\mathsf{ideal}}$.
- The user cannot convince RP of possession of the credential that has been revoked by the user themselves or by the issuer of that credential.
- An adversary who has only partially corrupted the user cannot make issuing, presentation, or revocation requests.

The first two items are covered by Theorem 1. The third item is covered by Theorem 2.

Theorem 1. *An adversary $\mathcal{A}$ and an environment $\mathcal{Z}$ running in parallel with $\mathcal{F}_{ideal}$, are not able to make a relying party accept an output* $(\mathsf{verify\text{-}resp}, \mathsf{sid}, (\mathsf{pres}, \mathsf{ch}), 1)$ *if there is no* $\mathsf{cred} = (\_, \_, \mathsf{cID}, \_, \_) \in \mathsf{T}_a$ *that would satisfy* $\mathsf{Ver}(\mathsf{cred}, \mathsf{pres})$ *and* $(\_, \mathsf{cID}, \_) \notin \mathsf{T}_r$.

Proof. Let us assume an adversary $\mathcal{A}$ produced pres, such that a relying party received an output $(\mathsf{verify\text{-}resp}, \mathsf{sid}, (\mathsf{pres}, \mathsf{ch}), \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, 1)$, but there is no $\mathsf{cred} = (\_, \_, \mathsf{cID}, \_, \_) \in \mathsf{T}_a$ s.t. $\mathsf{Ver}(\mathsf{cred}, \mathsf{pres}) = 1$ and $(\_, \mathsf{cID}, \_) \notin \mathsf{T}_r$.

If $U \in \mathcal{C}$, then the checks $\mathsf{cred} = (\_, \_, \mathsf{cID}, \_, \_) \in \mathsf{T}_a$, $\mathsf{Ver}(\mathsf{cred}, \mathsf{pres}) = 1$, and $(\_, \mathsf{cID}, \_) \notin \mathsf{T}_r$ are performed explicitly.

If $U \notin \mathcal{C}$, then pres was produced as a response to the presentation command by $\mathcal{F}_{\mathsf{ideal}}$. This implies that functionality has verified that there exists $\mathsf{cred} \in \mathsf{T}_a$ such that $(\_, \mathsf{cID}, \_) \notin \mathsf{T}_r$ before creating pres. Passing subsequent verification check implies that $\mathsf{Ver}(\mathsf{cred}, \mathsf{pres}) = 1$. Altogether, this contradicts the assumption.

Theorem 2. *An adversary $\mathcal{A}$ and an environment $\mathcal{Z}$ running in parallel with $\mathcal{F}_{ideal}$, corrupting user for at most $\mathsf{c}_\mathsf{U} = 1$, are not able to issue requests (issuing, user revocation, presentation) to the ideal functionality.*

Proof. Assume that $\mathcal{A}$, who partially corrupted user ($\mathsf{c_U} = 1$) makes request on behalf of the user to the issuing, user revocation or presentation interface. Each of the interfaces firstly calls authenticate sub-routine with the user identifier U as input. To proceed with the interface instructions, it should hold that either the user is fully corrupted by the adversary ($c_\mathsf{U} = 2$), or the interface has been called by the honest user, not the adversary, which contradicts the assumption.

4.2 The protocol set

In this section, we present a set of protocols of the proposed attribute management system. The AMS server uses $\mathcal{F}_{\mathsf{rev}}$ to manage revocations. It also serves as the supporting server for threshold decryption ($\mathcal{F}_{\mathsf{tdecrypt}}$) and signing ($\mathcal{F}_{\mathsf{tsign}}$).

The notation used in this section is given in Table 3. The real protocol routines will need to internally store some additional values, which are not a part of input/output interface. For clarity, we highlight such values.

Notation	Explanation
aID	attribute identifier
$\overrightarrow{\mathsf{aID}} = (\mathsf{aID}_1, \ldots, \mathsf{aID}_n)$	list of attribute identifiers
atr	attribute value
c_{atr}	encrypted attribute value
$\overrightarrow{\mathsf{atr}} = (\mathsf{atr}_1, \ldots, \mathsf{atr}_n)$	list of attribute values
cID	credential identifier
$\overrightarrow{\mathsf{cID}} = (\mathsf{cID}_1, \ldots, \mathsf{cID}_n)$	list of credential identifiers
$\mathsf{cred} = \langle(\mathsf{aID}_1, c_{\mathsf{atr}_1}), \ldots (\mathsf{aID}_n, c_{\mathsf{atr}_n})\rangle$	credential
pres	credential presentation
uID	user identifier
IID	issuer identifier
$\mathcal{F}(cmd, \mathsf{sid}, x)$	run the subroutine cmd of $\mathcal{F}$ with sid and input x

Table 3. Notation for Prettiness system

Initialisation Parties initialise building block functionalities as well as perform generation of keypairs as depicted on Figure 4. There is an instance of $\mathcal{F}_{\mathsf{sign}}^{\mathsf{IID}}$ for an issuer identified by IID, and of $\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}$, $\mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}$ for a user identified by uID. Technically, the issuers inform to the verifiers their public keys, and each user generates keys for signing and decryption, shared with the supporting server, getting a PIN that it can use to access the signing functionality later. Revocation functionality $\mathcal{F}_{\mathsf{rev}}$ is initialised by AMS for all users altogether.

User authentication. After the initialisation, each query to the AMS server by the user starts from authentication. For this, user creates a threshold signature σ_{req}, using $\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}$. Threshold signature generation using $\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}$ already includes

$\mathsf{Initialise_U}$	$\mathsf{Initialise_{ST}}$
// get input from environment	// generate signing key pair
$\mathcal{Z} \longrightarrow$: (initialise-req, sid);	$\mathsf{pk} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}(\mathsf{keygen}, \mathsf{sid}, T_0)$
// generate signing key pair	// generate decryption key pair
$\mathsf{pk} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}(\mathsf{keygen}, \mathsf{sid}, L, T_0, \mathsf{PIN})$	$\mathsf{pk_s} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{KeyGen}, \mathsf{sid})$
// generate decryption key pair	
$\mathsf{pk_c} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{KeyGen}, \mathsf{sid})$	

Fig. 4. Initialisation protocol (User, ST)

user authentication to the server. Technically, going into details of the protocol implementing $\mathcal{F}_{\mathsf{tsign}}$, the user proves ownership of their secret key share, as well as a special randomness, which serves as a challenge for the next authentication. Obtained signature serves as a consent from the user to perform certain actions with their data. The usage of threshold signatures instead of regular signatures is important to ensure user's key protection. Since the user's private key is stored on the mobile device, usage of threshold signatures adds level of protection to the private keys. Since the signature also servers as a consent to perform certain actions with the user's credentials, this signature should be verifiable for audit purposes, hence the security of the corresponding private key is crucial.

Issuing The protocol is presented in Figure 5. Upon successful user authentication, AMS server forwards attribute request to the issuer. The request contains a fresh identifier id_{cID} for the credential. To avoid repetition attacks by a corrupted AMS, the issuer keeps all received signatures and tracks their repetitions.

AMS provides to the user a link that they can use to directly communicate with the issuer IID (i.e. links IID to a communication channel of I), which is needed to generate revocation tokens using $\mathcal{F}_{\mathsf{rev}}$. There will be two such tokens, one for the user, and one for the issuer. First, the user generates a token rev_U whose revoker will be the user. Then, the user and the issuer jointly generate a token rev_I, whose revoker will be the issuer.

For each requested attribute aID_i, the issuer comes up with a value atr_i. The issuer creates the credential by encrypting each atr_i under the user's public key of $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$, including revocation tokens rev_I and rev_U. Next, the issuer signs the credential using $\mathcal{F}^{\mathsf{IID}}_{\mathsf{sign}}$ and forwards it to the AMS server. AMS server verifies the issuer signature and forwards issued credential to the user for approval. User decrypts attributes from the credential, verifies that the attributes are not malformed (i.e. that the value atr_i belongs to the set of valid attribute values $\mathcal{V}_{\mathsf{aID}_i}$), that revocation token data is correct, and the correctness of issuer signature. If all the verification steps succeed, AMS server saves the issued credential with identifier $\mathsf{cID} = (id_{\mathsf{cID}}, \mathsf{summary}(\mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID}))$, where id_{cID} ensures uniqueness of cID, and function summary generates an intuitive short description of the cre-

dential. The user includes credential data into a small checksum digest that it maintains locally.

Get all cID-s The protocol of Figure 7 allows to fetch all the previously issued credential identifiers $\overrightarrow{\mathsf{cID}}$, as well as all the stored credentials $\overrightarrow{\mathsf{cred}}$. While user has already received all this data during the issuing phase, the server storage can be used as a backup. Verification of the received data against the previously stored checksum ensures integrity against corrupted AMS. While issuer's signature already protects integrity of a credential, the checksum prevents AMS from being silent about the existence of some credential. At this step, the user only downloads *encrypted* credentials, and will need access to the decryption functionality to get their contents.

Since we want the server to serve as a backup, we need to be sure that user has all relevant data to further *use* the credentials. Besides raw credential data, the user also needs the following protocol-specific data:

- The ciphertexts of $\mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}$ should be decryptable, and the signatures of $\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}$ generatable. We assume that the user has not lost their secret keys corresponding to these functionalities.
- To keep integrity against the corrupted server, the user needs to keep the small checksum digest. As this digest is not sensitive (it it is computed from encrypted credentials), it can be copied to any other device.
- The revocation protocol of [22] that realises functionality $\mathcal{F}_{\mathsf{rev}}$ assumes that the user knows a small secret r that corresponds to the token x, which is needed for both revocation and presentation. Technically, r can also be stored on the server in form of a ciphertext c_r, which can (but does not have to) be included into the credential. This additional overhead has been included into the benchmarks of Sec. 5.
- The user, acting as a revoker, should be able to prove that AMS has ignored their revocation. In the revocation protocol of [22], the user should be able to present a confirmation signature of AMS. Such signatures cannot be stored by AMS. Here we assume that the number of such emergency-revoked credentials is not large for a single user, so they can afford to store them elsewhere.

Revocation Revocation can be initiated either by the user (Figure 8) who wants to revoke their credential, or by the issuer (Figure 9). Issuer revocation is initiated by the issuer's signed revocation request, which confirms that it is an authorised revocation. Both parties revoke cID by revoking the corresponding revocation token (rev_U or rev_I) using $\mathcal{F}_{\mathsf{rev}}$.

Revocation procedure may result in a double revocation of the same credential, by the user and by the issuer. During verification, as RP does the check for rev_I and rev_U separately, this could leak to RP the fact that some credential has been rejected twice. Moreover, if the token has been revoked once, RP could

$\mathsf{Issue}_\mathsf{U}$	$\mathsf{Issue}_\mathsf{ST}$
// get input from environment	
$\mathcal{Z} \longrightarrow: (\mathsf{issue\text{-}req}, \mathsf{sid}, \overrightarrow{\mathsf{aID}}, \mathsf{IID})$;	
$id_{\mathsf{cID}} \leftarrow\!\!\$\ \mathcal{X}_{\mathsf{id}}$;	
// establish connection with the server	// wait for user connection
$\longrightarrow \mathsf{ST} : \mathsf{tag}_{issue}, \mathsf{sid}, \mathsf{uID}$;	$U \longrightarrow: \mathsf{tag}_{issue}, \mathsf{sid}, \mathsf{uID}$;
// generate request message	
$m = (\mathsf{tag}_{issue}, id_{\mathsf{cID}}, \mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID})$;	
// generate threshold signature	// generate and verify threshold signature
$\sigma_{\mathsf{req}} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}(\mathsf{sign}, \mathsf{sid}, m, \mathsf{PIN})$;	$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}(\mathsf{sign}, \mathsf{sid})$;
$\longrightarrow \mathsf{ST} : \sigma_{\mathsf{req}}, m$;	$\mathsf{U} \longrightarrow: \sigma_{\mathsf{req}}, m = (\mathsf{tag}_{issue}, id_{\mathsf{cID}}, \mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID})$;
	$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}(\mathsf{verify}, \mathsf{sid}, m, \sigma_{\mathsf{req}})$;
// Get connected to the issuer I	// authenticate I as IID to U
$\mathsf{ST} \longrightarrow: I, \mathsf{IID}$;	$\longrightarrow \mathsf{U} : I, \mathsf{IID}$;
// locally generate a revocation token for the user	// authenticate U as uID to I
$\mathsf{rev}_U \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{issue}, U, (\mathsf{sid}, 0))$;	$\longrightarrow \mathsf{I} : U, id_{\mathsf{cID}}, \mathsf{uID}, \overrightarrow{\mathsf{aID}}, \sigma_{\mathsf{req}}$;
$\longrightarrow I : \mathsf{rev}_U$;	// get encrypted credential from I
// jointly generate revocation token for the issuer	$\mathsf{I} \longrightarrow: (\mathsf{cred}, \sigma_{\mathsf{cred}})$;
$\mathsf{rev}_I \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{issue}, I, (\mathsf{sid}, 1))$;	$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{IID}}_{\mathsf{sign}}(\mathsf{verify}, \mathsf{sid}, \mathsf{cred}, \sigma_{\mathsf{cred}})$;
// get encrypted credential from AMS	// send encrypted credential to U
$\mathsf{ST} \longrightarrow: (\mathsf{cred}, \sigma_{\mathsf{cred}})$;	$\longrightarrow \mathsf{U} : (\mathsf{cred}, \sigma_{\mathsf{cred}})$;
$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{IID}}_{\mathsf{sign}}(\mathsf{verify}, \mathsf{sid}, \mathsf{cred}, \sigma_{\mathsf{cred}})$;	
// decrypt and verify content	// participate in threshold decryption
$(\mathsf{rev}'_I, \mathsf{rev}'_U) \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Decrypt}, \mathsf{sid}, c_{\mathsf{rev}})$;	$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Decrypt}, \mathsf{sid})$;
$\mathbf{assert}\ (\mathsf{rev}_I, \mathsf{rev}_U) = (\mathsf{rev}'_I, \mathsf{rev}'_U)$;	
$\mathbf{for}\ \mathsf{aID}_i \in \overrightarrow{\mathsf{aID}}\ \mathbf{do}$:	$\mathbf{for}\ \mathsf{aID}_i \in \overrightarrow{\mathsf{aID}}\ \mathbf{do}$:
$\quad \mathsf{atr}_i \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Decrypt}, (\mathsf{sid}, i), c_{\mathsf{atr}_i})$;	$\quad ok \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Decrypt}, (\mathsf{sid}, i))$;
$\quad \mathbf{assert}\ \mathsf{atr}_i \in \mathcal{V}_{\mathsf{aID}_i}$;	
$\mathbf{assert}\ \mathsf{atr}_0 = \mathsf{pk}_{\mathsf{uID}}$;	
$\mathsf{cID} \leftarrow (\mathsf{id}_{\mathsf{cID}}, \mathsf{summary}(\mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID}))$;	$\mathsf{cID} \leftarrow (\mathsf{id}_{\mathsf{cID}}, \mathsf{summary}(\mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID}))$;
$h_{\mathsf{cred}} \leftarrow H(h_{\mathsf{cred}}, (\mathsf{cID}, \mathsf{cred}, \sigma_{\mathsf{cred}}))$;	$\mathsf{DB}_\mathsf{S}[\mathsf{uID}, \mathsf{cID}] \leftarrow (\mathsf{cred}, \sigma_{\mathsf{cred}})$;
$\longrightarrow \mathcal{Z} : (\mathsf{issue\text{-}resp}, \mathsf{sid}, \mathsf{cID})$;	

Fig. 5. Issuing protocol (User, ST)

$\mathsf{Issue_I}$

// receive an issue request from the server

$\mathsf{ST} \longrightarrow: \mathsf{sid}, U, id_{\mathsf{cID}}, \mathsf{uID}, \overrightarrow{\mathsf{aID}}, \sigma_{\mathsf{req}}$;

// verify the request

$ok \leftarrow \mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}(\mathsf{verify}, \mathsf{sid}, m = (\mathsf{tag}_{issue}, id_{\mathsf{cID}}, \mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID}), \sigma_{\mathsf{req}})$

// verify freshness of the credential identifier

$\mathsf{cID} \leftarrow (id_{\mathsf{cID}}, \mathsf{summary}(\mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID}))$; **assert** $\mathsf{cID} \notin \overrightarrow{\mathsf{cID}}$;

$\overrightarrow{\mathsf{cID}} \leftarrow \overrightarrow{\mathsf{cID}} \cup \mathsf{cID}$

// generate revocation tokens for the user and the issuer

$U \longrightarrow: \mathsf{rev}_U$;

$\mathsf{rev}_I \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{issue}, U, (\mathsf{sid}, 1))$;

// request attribute values from the environment

$\longrightarrow \mathcal{Z} : (\mathsf{issue\text{-}req}, \mathsf{sid}, \mathsf{uID}, \mathsf{cID}, \overrightarrow{\mathsf{aID}})$;

$\mathcal{Z} \longrightarrow: (\mathsf{issue\text{-}atr}, \mathsf{sid}, \overrightarrow{\mathsf{atr}})$;

// verify attribute values obtained from the environment

assert $(\mathsf{atr}_1 \in \mathcal{V}_{\mathsf{aID}_1}, \ldots, \mathsf{atr}_n \in \mathcal{V}_{\mathsf{aID}_n})$;

// encrypt revocation tokens and the user public key

$c_{\mathsf{rev}} \leftarrow \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Encrypt}, \mathsf{sid}, (\mathsf{rev}_I, \mathsf{rev}_U))$;

$\mathsf{atr}_0 \leftarrow \mathsf{pk}_{\mathsf{uID}}$;

$c_{\mathsf{atr}_0} \leftarrow \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Encrypt}, \mathsf{sid}, \mathsf{atr}_0)$;

// encrypt attribute values

for $\mathsf{aID}_i \in \overrightarrow{\mathsf{aID}}$ **do**

  $c_{\mathsf{atr}_i} \leftarrow \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Encrypt}, (\mathsf{sid}, i), \mathsf{atr}_i)$

// compose the credential

$\mathsf{cred} = \langle(\mathsf{aID}_0, c_{\mathsf{atr}_0}), (\mathsf{aID}_1, c_{\mathsf{atr}_1}), \ldots (\mathsf{aID}_n, c_{\mathsf{atr}_n}), (\mathsf{aID}_{\mathsf{rev}}, c_{\mathsf{rev}})\rangle$

$\sigma_{\mathsf{cred}} \leftarrow \mathcal{F}_{\mathsf{sign}}^{\mathsf{IID}}(\mathsf{sign}, \mathsf{sid}, \mathsf{cred})$;

$\longrightarrow \mathsf{ST} : (\mathsf{cred}, \sigma_{\mathsf{cred}})$;

// remember revocation token

$\mathsf{DB_I}[\mathsf{cID}] = \mathsf{rev}_I$;

$\longrightarrow \mathcal{Z} : (\mathsf{issue\text{-}resp}, \mathsf{sid}, \mathsf{cID})$;

Fig. 6. Issuing protocol (Issuer)

$\mathsf{GetCIDs_U}$	$\mathsf{GetCIDs_{ST}}$
// get input from environment	
$\mathcal{Z} \longrightarrow: (\mathsf{getsids\text{-}req}, \mathsf{sid})$;	
// establish connection with the server	// wait for user connection
$\longrightarrow \mathsf{ST} : \mathsf{tag}_{cids}, \mathsf{sid}, \mathsf{uID}$;	$U \longrightarrow: \mathsf{tag}_{cids}, \mathsf{sid}, \mathsf{uID}$;
// generate request message	
$m = (\mathsf{tag}_{cids}, \mathsf{uID})$;	
// generate threshold signature	// generate threshold signature
$\sigma_{\mathsf{req}} \leftarrow \mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}(\mathsf{sign}, \mathsf{sid}, m, \mathsf{PIN})$	$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}(\mathsf{sign}, \mathsf{sid})$
$\longrightarrow S : \sigma_{\mathsf{req}}, m$;	$U \longrightarrow: \sigma_{\mathsf{req}}, m = (\mathsf{tag}_{cids}, \mathsf{uID})$;
	$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}(\mathsf{verify}, \mathsf{sid}, m, \sigma_{\mathsf{req}})$;
// get credential data from AMS	// send credential data to the user
$S \longrightarrow: \overrightarrow{\mathsf{cID}}, \overrightarrow{\mathsf{cred}}$	$(\overrightarrow{\mathsf{cID}}, \overrightarrow{\mathsf{cred}}) \leftarrow \mathsf{DB_S}[\mathsf{uID}]$;
// check integrity of credential data	$\longrightarrow U : \overrightarrow{\mathsf{cID}}, \overrightarrow{\mathsf{cred}}$;
assert $h_{\mathsf{cred}} = H(\ldots H(0, (\mathsf{cID}_1, \mathsf{cred}_1)) \ldots, (\mathsf{cID}_n, \mathsf{cred}_n))$;	
$\mathsf{DB_U}[\overrightarrow{\mathsf{cID}}] \leftarrow \overrightarrow{\mathsf{cred}}$;	
$\longrightarrow \mathcal{Z} : (\mathsf{getcids\text{-}resp}, \mathsf{sid}, \overrightarrow{\mathsf{cID}})$;	

Fig. 7. Protocol for downloading credential backup from the server

learn whether it has been revoked by the user or by the issuer. As far as the AMS is honest, since the user will call revocation notification just before the final verification, this leakage is only possible if the server collaborates with a corrupted RP.

Presentation The protocol of Figure 10 allows the user to generate presentation of credential with identifier cID and selectively disclosed attributes $\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}$. First of all, the user looks for a credential cred identified by cID. User proceeds with authentication, followed by decrypting revocation token pair $\mathsf{rev} = (\mathsf{rev}_I, \mathsf{rev}_U)$ and the chosen attributes indexed by the identifiers $\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}$ using threshold decryption of $\mathcal{F}_{\mathsf{tdecrypt}}$. The user calls $\mathcal{F}_{\mathsf{rev}}$ to verify that revocation tokens rev_I and rev_U have not been revoked. The attributes are decrypted even if the credential is revoked, to prevent leaking the fact of revocation to the server. The obtained credential presentation pres contains only the disclosed attributes. All auxiliary data that will be needed to verify its validity will be provided in the verification phase.

Verification by relying party The protocol is presented on Figure 12. Before verification, RP receives updated state of the revocation functionality using notification interface of $\mathcal{F}_{\mathsf{rev}}$. If it is done immediately before the presentation, we

$\mathsf{Revoke}_{\mathsf{U}}$	$\mathsf{Revoke}_{\mathsf{ST}}$
// get input from environment	
$\mathcal{Z} \longrightarrow$: (revoke-req, sid, cID);	
// select credential to revoke	
$(\mathsf{cred}, \sigma_{\mathsf{cred}}) \leftarrow \mathsf{DB}_{\mathrm{U}}[\mathsf{cID}]$;	
// establish connection with the server	// wait for user connection
$\longrightarrow \mathsf{ST} : \mathsf{tag}_{rev}, \mathsf{sid}, \mathsf{uID}$;	$U \longrightarrow$: $\mathsf{tag}_{rev}, \mathsf{sid}, \mathsf{uID}$;
// generate request message	
$c_{\mathsf{cID}} \leftarrow \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Encrypt}, sid, \mathsf{cID})$;	
$m = (\mathsf{tag}_{rev}, \mathsf{uID}, c_{\mathsf{cID}})$;	
// generate threshold signature	// generate and verify threshold signature
$\sigma_{\mathsf{req}} \leftarrow \mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}(\mathsf{sign}, \mathsf{sid}, m, \mathsf{PIN})$	$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}(\mathsf{Sign}, \mathsf{sid})$
$\longrightarrow S : \sigma_{\mathsf{req}}, m$;	$U \longrightarrow$: $\sigma_{\mathsf{req}}, m = (\mathsf{tag}_{rev}, \mathsf{uID}, c_{\mathsf{cID}})$;
	$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}(\mathsf{verify}, \mathsf{sid}, m, \sigma_{\mathsf{req}})$;
// decrypt revocation token data	// participate in threshold decryption
$(\mathsf{rev}_I, \mathsf{rev}_U) \leftarrow \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Decrypt}, \mathsf{sid}, c_{\mathsf{rev}})$;	$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Decrypt}, \mathsf{sid})$;
// revoke the selected token	// confirm revocation
$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{revoke}, \mathsf{sid}, \mathsf{rev}_U)$;	$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{revoke}, \mathsf{sid})$;
$\longrightarrow \mathcal{Z}$: (revoke-resp, sid, ok);	

Fig. 8. Revocation protocol initiated by user

$\mathsf{Revoke_I}$	$\mathsf{Revoke_{ST}}$
// get input from environment	
$\mathcal{Z} \longrightarrow: (\mathsf{revoke\text{-}req}, \mathsf{sid}, \mathsf{cID})$;	
// select credential to revoke	
$\mathsf{rev}_I \leftarrow \mathsf{DB_I}[\mathsf{cID}]$;	
// establish connection with the server	// wait for issuer connection
$\longrightarrow \mathsf{ST} : \mathsf{tag}_{rev}, \mathsf{sid}, \mathsf{IID}$;	$I \longrightarrow: \mathsf{tag}_{rev}, \mathsf{sid}, \mathsf{IID}$;
// extract uID from cID	
$\mathsf{uID} \leftarrow \mathsf{cID}$;	
// generate request message	
$c_{\mathsf{cID}} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Encrypt}, sid, \mathsf{cID})$;	
$m = (\mathsf{tag}_{rev}, \mathsf{IID}, c_{\mathsf{cID}})$;	
$\sigma_{\mathsf{req}} \leftarrow \mathcal{F}^{\mathsf{IID}}_{\mathsf{sign}}(\mathsf{sign}, \mathsf{sid}, m)$;	
$\longrightarrow \mathsf{ST} : (m, \sigma_{\mathsf{req}})$;	$\mathsf{I} \longrightarrow: (m{=}\ (\mathsf{tag}_{rev}, \mathsf{IID}, c_{\mathsf{cID}}), \sigma_{\mathsf{req}})$
	$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{IID}}_{\mathsf{sign}}(\mathsf{verify}, \mathsf{sid}, m, \sigma_{\mathsf{req}})$
// revoke the selected token	// confirm revocation
$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{revoke}, \mathsf{sid}, \mathsf{rev}_I)$;	$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{revoke}, \mathsf{sid})$;
$\longrightarrow \mathcal{Z} : (\mathsf{revoke\text{-}resp}, \mathsf{sid}, \mathsf{ok})$;	

Fig. 9. Revocation protocol initiated by issuer

$\mathsf{Present}_\mathsf{U}$	$\mathsf{Present}_\mathsf{ST}$
// get input from environment	
$\mathcal{Z} \longrightarrow: (\mathsf{present\text{-}req}, \mathsf{sid}, \mathsf{cID}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}})$;	
// select credential to present	
$(\mathsf{cred}, \sigma_{\mathsf{cred}}) \leftarrow \mathsf{DB}_\mathrm{U}[\mathsf{cID}]$;	
assert $\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}} \subseteq \overrightarrow{\mathsf{aID}}$;	
// establish connection with the server	// wait for user connection
$\longrightarrow \mathsf{ST} : \mathsf{tag}_{pres}, \mathsf{sid}, \mathsf{uID}$;	$U \longrightarrow: \mathsf{tag}_{pres}, \mathsf{sid}, \mathsf{uID}$;
// generate request message	
$c_{\mathsf{cID}} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Encrypt}, sid, (\mathsf{cID}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}))$;	
$m = (\mathsf{tag}_{pres}, \mathsf{uID}, \lvert\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}\rvert, c_{\mathsf{cID}})$;	
// generate threshold signature	// generate and verify threshold signature
$\sigma_{\mathsf{req}} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}(\mathsf{sign}, \mathsf{sid}, m, \mathsf{PIN})$	$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}(\mathsf{sign}, \mathsf{sid})$
$\longrightarrow \mathsf{ST} : \sigma_{\mathsf{req}}, m$;	$\mathsf{U} \longrightarrow: \sigma_{\mathsf{req}}, m = (\mathsf{tag}_{pres}, \mathsf{uID}, n, c_{\mathsf{cID}})$;
	$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}(\mathsf{verify}, \mathsf{sid}, m, \sigma_{\mathsf{req}})$;
// check revocation status	// participate in subroutine calls
$(\mathsf{rev}_I, \mathsf{rev}_U) \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Decrypt}, \mathsf{sid}, c_{\mathsf{rev}})$;	$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Decrypt}, \mathsf{sid})$;
$b_I \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{unotify}, \mathsf{sid}, \mathsf{rev}_I)$;	$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{unotify}, \mathsf{sid})$;
$b_U \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{unotify}, \mathsf{sid}, \mathsf{rev}_U)$;	$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{unotify}, \mathsf{sid})$;
// decrypt disclosed attributes	// participate in threshold decryption
for $\mathsf{aID}_i \in \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}$:	**for** $i \in 1..n$ **do**
$\overrightarrow{\mathsf{atr}}_{\mathsf{dscl}} \leftarrow \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}} \cup \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Decrypt}, (\mathsf{sid}, i), c_{\mathsf{aID}_i})$;	$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Decrypt}, (\mathsf{sid}, i))$;
assert $\neg(b_I \vee b_U)$;	
$\longrightarrow \mathcal{Z} : (\mathsf{present\text{-}resp}, \mathsf{sid}, \mathsf{pres} = \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}})$;	

Fig. 10. Presentation with selective disclosure

cannot guarantee the unlinkability of presentation to the RP by AMS server, so it should be run independently. RP generates fresh challenge and sends it to the user for signature. This step is required to guarantee freshness of the presentation. User generates signature using $\mathcal{F}_{\mathsf{tsign}}$, and blindness property ensures that the AMS server does not learn anything about credential presentation, except the fact that the user has presented some credential to some party. We include cred and σ_{cred} to the message that gets signed, to achieve structure more similar to SD-JWT presentation protocol. RP receives credential presentation from the user and verifies the user and the issuer signatures. It uses verification interface of $\mathcal{F}_{\mathsf{rev}}$ to ensure that the credential is not revoked. Then it verifies whether the selectively disclosed attributes are indeed contained in the signed credential.

Using threshold signing for the challenge means that the supporting server should be online upon verification. If we want to make verification possible offline, we will need to use ordinary signature like $\mathcal{F}_{\mathsf{sign}}$, which means weaker guarantees for the RP if the user device gets stolen together with the secret key, or cloned. If server-supported signature is used, then the actual user may contact the server to stop generating signatures in case of stealth, and clone detection protects against silent cloning of the user device.

Selective disclosure is achieved by decrypting only the values of selected $\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}$. For each such aID, user calls verification interface of $\mathcal{F}_{\mathsf{tdecrypt}}$ to prove to RP that the presented plaintext value indeed corresponds to the ciphertext from the signed credential cred. Figure 11 presents comparison of the proposed solution with the selective disclosure for JWT specification with the trivial solution of protecting credentials in the storage. The main difference is that in the case of SD-JWT for the presentation, the user needs to decrypt all the attributes. Therefore, if the device memory leaks to the adversary at this point, all the attribute plaintext values will be disclosed. In our solution, only the attributes that are disclosed for the specific presentation are decrypted and can potentially leak.

Logging. We assume that the server logs all request messages along with the time at which these requests have been made. The user can then get an overview of transactions performed to their data. The list of logs and information that the user can extract from these is given in Table 4.

In case of corrupted AMS, even if the user compares the log against some digest for integrity, the AMS may simply refuse to return the log to the user, so logging is useful assuming an *honest-but-curious* server. Moreover, if the issuer is corrupted, then there are no guarantees that valid revoked cID has been recorded. While accountability of AMS could potentially be enforced by requiring its signature on the digest, and the ciphertext c_{cID} of the issuer could contain information about the actual performed revocation (accompanied with a proof-of-encryption), these enhancements would require more details, and are out of scope of this paper.

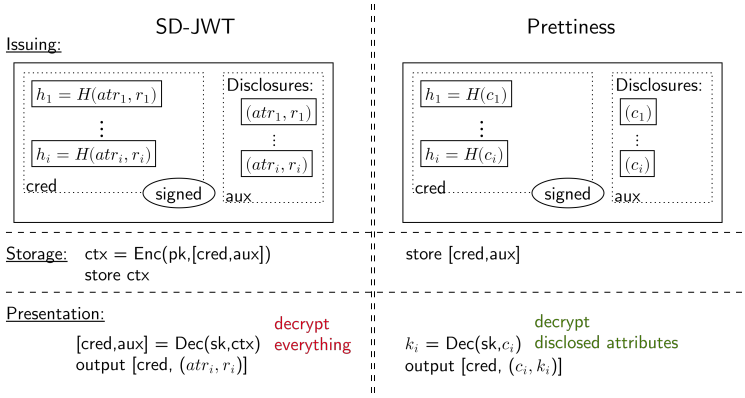


Fig. 11. Comparison of Prettiness approach with Selective Disclosures for JWT

Protocol	**Logged record**	**What can be read by user**
Issue	$(\mathsf{tag}_{issue}, id_{\mathsf{cID}}, \mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID})$	issued cID
Get cIDs	$(\mathsf{tag}_{cids}, \mathsf{uID})$	fact of credential backup request
Revoke(user)	$(\mathsf{tag}_{rev}, \mathsf{uID}, c_{\mathsf{cID}})$	revoked $\mathsf{cID} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Decrypt}, \mathsf{sid}, c_{\mathsf{cID}})$
Revoke(issuer)	$(\mathsf{tag}_{rev}, \mathsf{IID}, c_{\mathsf{cID}})$	revoker IID and $\mathsf{cID} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Decrypt}, \mathsf{sid}, c_{\mathsf{cID}})$
Present	$(\mathsf{tag}_{pres}, \mathsf{uID}, n, c_{\mathsf{cID}})$	presented $\mathsf{cID} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Decrypt}, \mathsf{sid}, c_{\mathsf{cID}})$

Table 4. Logged records and their interpretation.

Cheating detection. Revocation functionality $\mathcal{F}_{\mathsf{rev}}$ allows AMS to conceal the fact of revocation from a party P (a user or a verifier). In this case, P and R together can detect cheating and prove it to the third party. However, some event needs to trigger such a cheating detection in practice. This depends on the impact that cheating may impose.

Cheating to a user. If AMS silences some revocation from the user, it may result in a rejected legitimate verification. The user may consult the issuer to ask whether the issuer has revoked the credential. If it is indeed so, the user and the issuer together may prove the fact of cheating to a third party. If the credential has not been revoked, then the verifier has chosen not to accept the credential, but it is a legitimate decision of the verifier (in the scope of $\mathcal{F}_{\mathsf{ideal}}$).

Cheating to a verifier. If AMS silences some revocation from the verifier, it may result in an accepted illegitimate verification. The verifier and the issuer may resolve this problem, but the problem must be detected in first place. In practice, the verifier and the issuer could periodically run verification (with issuer acting as a presenter) on randomly selected tokens held by the issuers whom the verifier accepts, to detect cheating with some non-negligible probability.

$\mathsf{Verify}_{\mathsf{RP}}$

// get input from the environment

$\mathcal{Z} \longrightarrow$: $(\mathsf{verify\text{-}req}, \mathsf{sid})$;

// get last revocation data (can be done in advance)

$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{vnotify}, \mathsf{sid})$;

// wait for user connection

$U \longrightarrow$: $\mathsf{tag}_{verify}, \mathsf{sid}, \mathsf{uID}$;

// generate a challenge for the user

$\mathsf{ch} \leftarrow\!\!\$\ \mathcal{CH}$;

$\longrightarrow U : \langle \mathsf{ch} \rangle$;

// get response from the user

$U \longrightarrow$: $\langle \mathsf{uID}, \mathsf{cred}, \sigma_{\mathsf{cred}}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}, \mathsf{IID}, \mathsf{rev}, \sigma_{\mathsf{pres}} \rangle$

// verify user and issuer signatures

$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{sign}}^{\mathsf{IID}}(\mathsf{verify}, \mathsf{sid}, \mathsf{cred}, \sigma_{\mathsf{cred}})$

$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}(\mathsf{verify}, \mathsf{sid}, (\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}, \mathsf{cred}, \sigma_{\mathsf{cred}}, \mathsf{ch}), \sigma_{\mathsf{pres}})$

// verify revocation status

assert $\mathcal{F}_{\mathsf{rev}}(\mathsf{verify}, \mathsf{sid}) = (\mathsf{rev}_I, \mathsf{true})$;

assert $\mathcal{F}_{\mathsf{rev}}(\mathsf{verify}, \mathsf{sid}) = (\mathsf{rev}_U, \mathsf{true})$;

// verify credential data

$\langle (\mathsf{aID}_0, c_0), \ldots (\mathsf{aID}_n, c_n), (\mathsf{aID}_{\mathsf{rev}}, c_{\mathsf{rev}}) \rangle \leftarrow \mathsf{cred}$;

$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Verify}, \mathsf{sid}, (c_{\mathsf{rev}}, \mathsf{rev}))$;

$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Verify}, \mathsf{sid}, (c_0, \mathsf{pk}_{\mathsf{uID}}))$;

for $(\mathsf{aID}_i, \mathsf{atr}_i) \in (\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}})$:

  $\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Verify}, \mathsf{sid}, (c_{\mathsf{aID}_i}, \mathsf{atr}_i))$;

$\longrightarrow \mathcal{Z} : (\mathsf{verify\text{-}resp}, (\overrightarrow{\mathsf{aID}}, \overrightarrow{\mathsf{atr}}, \mathsf{IID}, \mathsf{ch}), 1)$;

$\mathsf{Verify}_{\mathsf{U}}$

// get input from the environment

$\mathcal{Z} \longrightarrow$: $(\mathsf{verify\text{-}req}, \mathsf{sid}, \mathsf{cID}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \mathsf{pres} = \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}, \mathsf{RP})$;

// cred, σ_{cred}, rev, IID come from presentation of cID

// Establish connection with RP

$\longrightarrow \mathsf{RP} : \mathsf{tag}_{verify}, \mathsf{sid}, \mathsf{uID}$;

// Get the challenge of RP

$\mathsf{RP} \longrightarrow$: $\langle \mathsf{ch} \rangle$

$m \leftarrow (\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}, \mathsf{IID}, \mathsf{cred}, \sigma_{\mathsf{cred}}, \mathsf{ch})$

// Establish connection with AMS

$\longrightarrow \mathsf{ST} : \mathsf{tag}_{verify}, \mathsf{sid}, \mathsf{uID}$;

$\sigma_{\mathsf{pres}} \leftarrow \mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}(\mathsf{sign}, \mathsf{sid}, m, \mathsf{PIN})$

$\longrightarrow \mathsf{RP} : \langle \mathsf{uID}, \mathsf{cred}, \sigma_{\mathsf{cred}}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}, \mathsf{IID}, \mathsf{rev}, \sigma_{\mathsf{pres}} \rangle$;

// prove that credential is not revoked

$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{verify}, \mathsf{sid}, \mathsf{rev}_I)$;

$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{verify}, \mathsf{sid}, \mathsf{rev}_U)$;

// participate in proofs of encryption

$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Verify}, \mathsf{sid}, (c_{\mathsf{rev}}, \mathsf{rev}))$;

$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Verify}, \mathsf{sid}, (c_0, \mathsf{pk}_{\mathsf{uID}}))$;

for $(\mathsf{aID}_i, \mathsf{atr}_i) \in (\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}})$:

  $\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Verify}, \mathsf{sid}, (c_{\mathsf{aID}_i}, \mathsf{atr}_i))$;

$\longrightarrow \mathcal{Z} : (\mathsf{verify\text{-}resp}, (\overrightarrow{\mathsf{aID}}, \overrightarrow{\mathsf{atr}}, \mathsf{IID}, \mathsf{ch}), 1)$;

$\mathsf{Verify}_{\mathsf{ST}}$

// wait for user connection

$U \longrightarrow$: $\mathsf{tag}_{verify}, \mathsf{sid}, \mathsf{uID}$;

// participate in generating threshold signature

$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}(\mathsf{sign}, \mathsf{sid})$

Fig. 12. Verification of credential presentation. Here $\mathsf{rev} = (\mathsf{rev}_I, \mathsf{rev}_U)$

Expiration. Expired records need to be periodically removed (or archived). Let Δ be an overestimated time after which any attribute will "definitely expire". For both the credential and the revocation tables, the server removes an entry as soon as the time Δ passes after it has been added to the corresponding table.

4.3 Security of the proposed protocol

Theorem 3. *Assuming a* covertly *corrupted AMS (Definition 1), the protocol set $\pi_{PRETTINESS}$ of Sec. 4 UC-realizes $\mathcal{F}_{ideal}$ in $\mathcal{F}_{sign}^{\mathsf{IID}}, \mathcal{F}_{tsign}^{\mathsf{uID}}, \mathcal{F}_{tdecrypt}^{\mathsf{uID}}, \mathcal{F}_{rev}$- hybrid model.*

The full security proof is presented in supplementary material. In this section, we present the intuition behind the simulation, considering the most interesting corruptions.

For the proof, we define a simulator $\mathcal{S}$ mediating communication between the real adversary $\mathcal{A}$ and the ideal functionality $\mathcal{F}_{\mathsf{ideal}}$, trying to generate messages for $\mathcal{A}$ and $\mathcal{F}_{\mathsf{ideal}}$ in such a way that $\mathcal{Z}$ gets convinced that it is communicating with the real protocol $\pi_{\mathsf{PRETTINESS}}$. The simulator plays roles of honest parties in $\pi_{\mathsf{PRETTINESS}}$, generating messages for corrupted parties, which are controlled by $\mathcal{A}$. In addition, $\mathcal{S}$ interacts with $\mathcal{F}_{\mathsf{ideal}}$ to synchronise with its inputs and outputs. This shows that any attack of $\mathcal{A}$ on $\pi_{\mathsf{PRETTINESS}}$ can be converted (by the simulator) to an analogous attack on $\mathcal{F}_{\mathsf{ideal}}$. In particular, all leakages of $\pi_{\mathsf{PRETTINESS}}$ to the adversary are computable from the "intensional" leakage of $\mathcal{F}_{\mathsf{ideal}}$ and the data that $\mathcal{A}$ has provided for corrupted parties. We describe how $\mathcal{S}$ behaves in each routine of $\mathcal{F}_{\mathsf{ideal}}$.

Initialisation: $\mathcal{S}$ defines the function $\mathsf{Present}$ for $\mathcal{F}_{\mathsf{ideal}}$, used to generate credential presentation. This function takes the attribute values $\overrightarrow{\mathsf{atr}}$ that correspond to cID and $\overrightarrow{\mathsf{aID}}$ from the table T_a of issued attributes, as is done in the real protocol. $\mathcal{S}$ defines Ver that is used to verify credential presentation by checking that $\overrightarrow{\mathsf{atr}}$ have indeed been stored in T_a for given cID and $\overrightarrow{\mathsf{aID}}$. $\mathcal{S}$ also initializes all functionalities $\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}$, $\mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}$ (a separate instance for each user of the system), $\mathcal{F}_{\mathsf{rev}}$ (one instance for the entire system) and $\mathcal{F}_{\mathsf{sign}}^{\mathsf{IID}}$ (a separate instance for each issuer in the system).

Issuing: $\mathcal{S}$ ensures that the credential contents are only added to T_a if it has indeed been requested by the (honest) user and issued by the (honest) issuer. On behalf of an honest issuer, the simulator treats credential request from a malicious user as successful only if it could indeed come up with some valid attribute values $\overrightarrow{\mathsf{atr}}$ for that user, and if generation of revocation token for that user has succeeded, so a credential will only be sent to the user if the issuer has possibility to revoke it later. On behalf of an honest user, before accepting credential from a malicious issuer, simulator verifies that (1) it contains attribute from the valid set of the attributes, (2) that issued credential contains revocation token rev generated by the $\mathcal{S}$ and (3) that credential contains valid signature. This ensures that successful issuing will be followed by successful presentation and revocation. Finally, all credential data except the party identities and the

attribute identifiers $\overrightarrow{\mathsf{aID}}$ (which are explicitly leaked by $\mathcal{F}_{\mathsf{ideal}}$) is encrypted, and can be simulated to a corrupted AMS.

Whenever $\mathcal{F}_{\mathsf{ideal}}$ adds a credential record cID to T_a, the simulator adds a corresponding entry (with additional protocol-specific data, like signatures and revocation tokens) to its local table $\mathsf{T}_{\mathsf{issue}}$.

Get all CID-s: On behalf of honest AMS, $\mathcal{S}$ simulates to a corrupted user all the cID-s and credentials that have been issued to the corrupted user so far. All such credentials have already been simulated to $\mathcal{A}$ in the issuing phase, and recorded to $\mathsf{T}_{\mathsf{issue}}$. On behalf of honest user, $\mathcal{S}$ verifies the data provided by corrupted AMS against the digest representing the data that the AMS should have shown, which ensures consistency with the output of $\mathcal{F}_{\mathsf{ideal}}$.

Revocation: On behalf of honest AMS, $\mathcal{S}$ collaborates with the corrupted revoker, letting $\mathcal{A}$ choose the token to revoke. It is not possible to illegitimately revoke a credential cID of an honest user without having rights on its revocation, which have previously been given by $\mathcal{F}_{\mathsf{rev}}$ only to the user and the issuer of cID. On behalf of an honest revoker, unless the user of the revoked credential or the RP is corrupted (in these cases, $\mathcal{S}$ gets cID from $\mathcal{F}_{\mathsf{ideal}}$ and can use the same token rev that was issued for cID), $\mathcal{S}$ takes a random revocation token $\mathsf{rev} \leftarrow \mathcal{X}$ (which will not coincide with any other token with probability $1/|\mathcal{X}|$), and simulates calls of $\mathcal{F}_{\mathsf{rev}}$ with that token. Adversary will not notice the difference until the details of revocation gets leaked, which happens either upon reaching user corruption level $c_{\mathsf{U}} = 2$, or during presentation to a corrupted RP. In both cases, $\mathcal{F}_{\mathsf{ideal}}$ will provide all relevant revocation data to $\mathcal{S}$, so that it can add all relevant tokens rev to X^- of $\mathcal{F}_{\mathsf{rev}}$.

Whenever $\mathcal{F}_{\mathsf{ideal}}$ adds a credential record cID to T_r, the simulator has added a corresponding revocation token rev to its local set X^- of revoked tokens, unless cID has not been leaked to $\mathcal{S}$ yet. In the latter case, the user of cID is an honest party, and it will anyway not call verification of $\mathcal{F}_{\mathsf{rev}}$ with a revoked token.

Presentation: On behalf of honest AMS, $\mathcal{S}$ collaborates with the corrupted user, letting $\mathcal{A}$ choose the ciphertexts of their choice for decryption. On behalf of honest user, $\mathcal{S}$ simulates presentation for a randomly chosen credential cID' from $\mathsf{T}_{\mathsf{issue}}$, which is consistent with T_a of $\mathcal{F}_{\mathsf{ideal}}$. If there is no such cID', presentation trivially fails, as there have been no credentials issued to that user. At this point, $\mathcal{A}$ only learns $|\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}|$, i.e. the number of threshold decryptions. $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$ does not leak which ciphertexts exactly have been decrypted. The simulator does not learn whether presentation of an honest user has eventually succeeded, but this fact does not need to be simulated to $\mathcal{A}$.

Verification: If the user is honest and RP is corrupted, $\mathcal{S}$ gets disclosed presentation data from $\mathcal{F}_{\mathsf{ideal}}$, which is enough to construct all messages for RP, using additional data from $\mathsf{T}_{\mathsf{issue}}$. Obtained cID may be different from cID' previously chosen for the presentation phase, but the simulation has not depended on cID' anyway.

If the user is corrupted with level $c_{\mathsf{U}} \geq 2$ (with $c_{\mathsf{U}} = 1$, corruptions are passive), it may show a revoked or invalid credential to an honest RP. $\mathcal{S}$ detects it using its internal tables for revocation, signing and decryption for that user. If

$c_{\mathsf{U}} = 3$, all these tables are valid for that user from the beginning. Upon $c_{\mathsf{U}} = 2$ corruption, $\mathcal{F}_{\mathsf{ideal}}$ leaks internal records of that user to $\mathcal{S}$, allowing it to update its tables accordingly. Hence, these tables are filled with data consistent with the internal tables of $\mathcal{F}_{\mathsf{ideal}}$.

If both RP and ST are corrupted, then the adversary expects to learn the sessions on which the presented credential cID has been revoked (if such a session exists). In this case, $\mathcal{F}_{\mathsf{ideal}}$ sends a corresponding index of the revocation table to $\mathcal{S}$, from which it can refer to the session id of that revocation.

5 Efficiency

We have estimated how well Prettiness would perform in practice. First of all, we collected benchmarks for implementations of certain protocols UC-realising $\mathcal{F}_{\mathsf{rev}}$, $\mathcal{F}_{\mathsf{tdecrypt}}$, $\mathcal{F}_{\mathsf{tsign}}$, and $\mathcal{F}_{\mathsf{sign}}$. We run the benchmarks on Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz, 4 cores, 2 threads per core, and 15GB RAM.

5.1 Benchmarks for digest function.

We assume that the local database digest is computed using SHA256 hash function. We measured its computation times using `openssl speed sha256`, getting values below $7 \cdot 10^{-9}$ seconds per byte. While this small number is very small, the total time to compute the digest of the entire database grows with the number of user credenitals.

5.2 Benchmarks for $\mathcal{F}_{\mathsf{sign}}$ and $\mathcal{F}_{\mathsf{tsign}}$.

We assume that $\mathcal{F}_{\mathsf{sign}}$ is realised with an ordinary 2048-bit RSA signature. We measured the signature generation and verification times using `openssl speed rsa`, getting values below 0.001 seconds. This small number has very little impact on the overall efficiency.

The efficiency of a protocol UC-realizing $\mathcal{F}_{\mathsf{tsign}}$ has been taken from [6], where it is based on RSA signature. Since protocol presented in [28] just adds blinding to the protocol presented in [6], the benchmarks will be similar. Signing operation itself takes tens of milliseconds, which we for safety round up to 0.1 seconds. According to [6], the most time overhead comes from the network delay to transmit the signature share from the client to the server, which we do not include into the timing benchmark, instead giving a separate benchmark for communication. The signature itself is an ordinary RSA signature, verified in a usual manner.

The communication has been estimated instantiating [6] with 2048-bit RSA and 128-bit randomness. The communication consists of a message $\langle y, m, r, r' \rangle$ that the client sends to the server, and $\langle s, m \rangle$ that the server sends back. Here y and s are 2048-bit RSA signatures, and $r, r' \in \{0,1\}^{\eta}$. This gives $2 \cdot (2048 + 128 + |m|_b) = 4352 + 2 \cdot |m|_b$ bits, or $544 + 2 \cdot |m|_B$ bytes, where $|m|_B$ is the message length in bytes.

	Local comp. time (s)	**Comm. (bytes)**	**Output size (bytes)**
SHA256(per byte)	0.00000000682	–	32
$\mathcal{F}_{\mathsf{sign}}(\mathsf{sign})$	0.001	–	256
$\mathcal{F}_{\mathsf{sign}}(\mathsf{verify})$	0.001	–	–
$\mathcal{F}_{\mathsf{tsign}}(\mathsf{sign})$	0.1	$544 + \lvert\mathsf{msg}\rvert$	256
$\mathcal{F}_{\mathsf{tsign}}(\mathsf{verify})$	0.001	–	–
$\mathcal{F}_{\mathsf{tdecrypt}}(\mathsf{encrypt})$	0.282	–	$2618 + \lvert\mathsf{msg}\rvert$
$\mathcal{F}_{\mathsf{tdecrypt}}(\mathsf{decrypt})$	0.362	106	–
$\mathcal{F}_{\mathsf{tdecrypt}}(\mathsf{verify})$	0.00000131	32	–

Table 5. Benchmarks for signing and threshold decryption protocols. Here the *output size* is the signature size for signing routines, and ciphertext size for encryption routine.

Local computation times (ms)

m	0	100	10,000	1,000,000
$\mathcal{F}_{\mathsf{rev}}(\mathsf{issue})$	0.00447	–	–	–
$\mathcal{F}_{\mathsf{rev}}(\mathsf{revoke})$	0.108	16.5	23.8	743
$\mathcal{F}_{\mathsf{rev}}(\mathsf{unotify})$	0.081	0.137	5.57	673
$\mathcal{F}_{\mathsf{rev}}(\mathsf{vnotify})$	0.075	0.137	6.52	716
$\mathcal{F}_{\mathsf{rev}}(\mathsf{verify})$	$\leq$2300	–	–	–

Communication (bytes)

0	100	10,000	1,000,000
48	–	–	–
160	–	–	–
114	3,314	320,114	32,000,114
114	3,314	320,114	32,000,114
162	–	–	–

Table 6. Benchmarks for different routines of revocation protocol. Notation "–" represent numbers that are the same as for $m = 0$.

5.3 Benchmarks for $\mathcal{F}_{\mathsf{tdecrypt}}$.

Protocols for $\mathcal{F}_{\mathsf{tdecrypt}}$ have been re-implemented in Go, which gives better results compared to the Python implementation of [21], and provides a more precise estimation of sizes of different components. The underlying hash function is SHA-256. We got ca 282ms for encryption, 362ms for decryption, and 106B of total decryption communication.

5.4 Benchmarks for $\mathcal{F}_{\mathsf{rev}}$.

Efficiency of a protocol for $\mathcal{F}_{\mathsf{rev}}$ has been taken from [22]. The issuing and verification routines of this protocol do not depend on the total number of revoked tokens. However, the efficiency of notifications (and slightly of the revocations) also depends on the total number of revoked elements. We reproduce some benchmarks In Table 6. We will assume $m = 1,000,000$ revoked tokens in this work, but we also reproduce the results for some smaller values of m to demonstrate the efficiency in the beginning, when not so many credentials have been revoked yet.

5.5 Estimating efficiency of the full Prettiness protocol

Benchmarks of the building block protocols can be found in Table 5 and Table 6. The times are measured for local computations only, without the network

	Issue	**Get Creds**	**Revoke(I)**	**Revoke(U)**	**Present**	**DB update**	**Verify**
comm (B)	90,716	3,906,474	3,355	6,688	7,590	32,000,114	122,170
time (s)	8.114	0.127	1.027	1.487	5.71	0.716	4.702

Table 7. Summarised efficiency of the full system for $N = 100$ credentials per user, $n = 10$ attributes per credential, $m = 10^6$ revoked tokens.

overhead, which is measured separately. Based on benchmarks of the building blocks, we estimated theoretical efficiency of the full system. In this estimate, we do *not* include the overhead of establishing communication channel (TLS) between parties, as well as the actual sizes of network packets moving inside TLS connection.

Since efficiency depends a lot on the actual number of credentials and their sizes, we considered a particular use case with $N = 100$ credentials per user, and $n = 10$ attributes per credential, and $m = 1,000,000$ total revocations. In practice, such an m can be an overestimate if there would be not too many urgent revocations of active credentials (for which expiration time has not passed).

The summary of results is given in Table 7. As the user and verifier notification requires a large download of the revocation database, which can be done in advance, we treat is as a separate routine **DB update** in our estimate, which has more or less the same efficiency for the user and the verifier. More detailed benchmarks, explaining how exactly the times have been estimated based on the protocol descriptions, can be found in the supplementary material.

Let us discuss the bottlenecks of the full protocol efficiency.

Issuing. The main overhead of issuing comes from letting the user decrypt the attributes to check their contents. Formally, we need this step to avoid selective failure attacks, where a further presentation of a malformed credential to RP surprisingly fails.

Get all Credentials. The communication overhead is caused by downloading the full backup from the server. Although the user could download only *some* of the credentials, this would violate user privacy, giving to the server a hint of *which* credential is going to be presented. We find this communication feasible if the number of credentials that have not been expired yet is not too large, as in the benchmarked $N = 100$ case.

DB Update. In this case, communication overhead comes from the necessity to download the data for m revoked tokens. Assuming that RP updates the status of the revocation database periodically (e.g. once per hour), and it is not urgent, then a download of 32MB of data per update would not be an issue. It is more complicated for the user who may be located in a place with bad network connection. We propose that, as our protocol anyway does not hide from AMS the identity of the user of a revoked credential (to make logging easier), the user only needs to download m_U of the m entries of the revocation database, where

m_U is the number of revocations that concern the user U, which makes download size much smaller.

Authenticating the revoker party allows to protect against denial-of-service attacks, where a revoker maliciously attempts to add to the revocation database too many (potentially non-existent) credentials. AMS could limit the number of allowed revocations for same user/issuer. E.g. a user is not allowed to revoke more credentials that have been issued *to* it, and an issuer is not allowed to revoke more credentials that have been issued *by* it.

6 Conclusion

This work proposes attribute management system, where the user is in control over their credentials. Credentials are stored on the supporting server side. To ensure secure storage of cryptographic material on the user's device, system relies on threshold cryptography (used for user's signatures and attribute decryption functionalities). Credentials can be revoked by both user and the issuer. We construct an ideal functionality that captures security guarantees provided by the system and prove that proposed protocol set securely realises this functionality. Finally, we provide an efficiency estimate of our system. Future work includes post-quantum version of the proposed system as well as possible optimisations.

References

1. Regulation (EU) 2024/1183 of the European Parliament and of the Council of 11 April 2024 amending Regulation (EU) No 910/2014 as regards establishing the European Digital Identity Framework, 2024.
2. The European Digital Identity Wallet Architecture and Reference Framework V2.7.3 , 2025.
3. Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology*, 23(2):281–343, 2010.
4. Amira Barki, Solenn Brunet, Nicolas Desmoulins, and Jacques Traoré. Improved algebraic macs and practical keyed-verification anonymous credentials. In Roberto Avanzi and Howard M. Heys, editors, *Selected Areas in Cryptography - SAC 2016 - 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers*, volume 10532 of *Lecture Notes in Computer Science*, pages 360–380. Springer, 2016.
5. Carsten Baum, Olivier Blazy, Jan Camenisch, Jaap-Henk Hoepman, Eysa Lee, Anja Lehmann, Anna Lysyanskaya, René Mayrhofer, Hart Montgomery, Ngoc Khanh Nguyen, et al. Cryptographers' feedback on the eu digital identity's arf. *Tech. Rep.*, 2024.
6. Ahto Buldas, Aivo Kalu, Peeter Laud, and Mart Oruaas. Server-supported rsa signatures for mobile devices. In *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I 22*, pages 315–333. Springer, 2017.
7. Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. Practical UC-Secure Delegatable Credentials with Attributes and Their Application to Blockchain. In

Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 683–699. ACM, 2017.

8. Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 201–210, 2006.
9. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer, 2001.
10. Jan Camenisch and Anna Lysyanskaya. Signature Schemes and Anonymous Credentials from Bilinear Maps. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer, 2004.
11. Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.
12. Ran Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, page 219. IEEE Computer Society, 2004.
13. Ran Canetti, Shai Halevi, and Jonathan Katz. Adaptively-secure, non-interactive public-key encryption. In Joe Kilian, editor, *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, volume 3378 of *Lecture Notes in Computer Science*, pages 150–168. Springer, 2005.
14. Daniel Fett, Kristina Yasuda, and Brian Campbell. Selective Disclosure for JSON Web Tokens. RFC 9901, November 2025.
15. Matteo Frigo et al. Anonymous credentials from ecdsa. *Cryptology ePrint Archive*, 2024.
16. Ulrich Haböck and Stephan Krenn. Breaking and fixing anonymous credentials for the cloud. In Yi Mu, Robert H. Deng, and Xinyi Huang, editors, *Cryptology and Network Security - 18th International Conference, CANS 2019, Fuzhou, China, October 25-27, 2019, Proceedings*, volume 11829 of *Lecture Notes in Computer Science*, pages 249–269. Springer, 2019.
17. Julia Hesse, Nitin Singh, and Alessandro Sorniotti. How to bind anonymous credentials to humans. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3047–3064, Anaheim, CA, August 2023. USENIX Association.
18. ISO Central Secretary. Personal identification — ISO-compliant driving licence — Part 5: Mobile driving licence (mDL) application. Standard ISO/IEC 18013-5:2021, International Organization for Standardization, Geneva, CH, 2021.
19. Aivo Kalu, Burak Can Kus, Peeter Laud, Kin Long Leung, Nikita Snetkov, and Jelizaveta Vakarjuk. Cacs: A cloud privacy-preserving attribute management system. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*, pages 1–9, 2023.

20. Stephan Krenn, Thomas Lorünser, Anja Salzer, and Christoph Striecks. Towards Attribute-Based Credentials in the Cloud. In Srdjan Capkun and Sherman S. M. Chow, editors, *Cryptology and Network Security - 16th International Conference, CANS 2017, Hong Kong, China, November 30 - December 2, 2017, Revised Selected Papers*, volume 11261 of *Lecture Notes in Computer Science*, pages 179–202. Springer, 2017.
21. Peeter Laud, Alisa Pankova, and Jelizaveta Vakarjuk. Privacy-preserving server-supported decryption. In *2025 IEEE 38th Computer Security Foundations Symposium (CSF)*, pages 127–142. IEEE, 2025.
22. Alisa Pankova and Jelizaveta Vakarjuk. Credential revocation assisted by a covertly corrupted server. Cryptology ePrint Archive, Paper 2025/1854, 2025.
23. Christian Paquin and Greg Zaverucha. U-prove cryptographic specification v1.1 (revision 3), December 2013. Released under the Open Specification Promise (http://www.microsoft.com/openspecifications/en/us/programs/osp/default.aspx).
24. Šeila Bećirović Ramić, Ehlimana Cogo, Irfan Prazina, Emir Cogo, Muhamed Turkanović, Razija Turčinhodžić Mulahasanović, and Saša Mrdović. Selective disclosure in digital credentials: A review. *ICT Express*, 2024.
25. Maximilian Richter, Magdalena Bertram, Jasper Seidensticker, and Marian Margraf. Cryptographic requirements of verifiable credentials for digital identification documents. In *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1663–1668. IEEE, 2023.
26. Sietse Ringers, Eric R. Verheul, and Jaap-Henk Hoepman. An Efficient Self-blindable Attribute-Based Credential Scheme. In Aggelos Kiayias, editor, *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers*, volume 10322 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2017.
27. Secure Technology Alliance Identity Council. The Mobile Driver's License (mDL) and Ecosystem. Standard, Secure Technology Alliance, 2020.
28. Nikita Snetkov, Mihkel Jaas Karu, Jelizaveta Vakarjuk, and Alisa Pankova. Coppercloud: Blind server-supported RSA signatures. Cryptology ePrint Archive, Paper 2025/1824, 2025.

A Supporting ideal functionalities

Figures 13 and 14 define revocation functionality $\mathcal{F}_{\mathsf{rev}}$. The leakages of $\mathcal{F}_{\mathsf{rev}}$ for a particular revocation protocol of [22] are listed in Table 8.

Figures 17,18,19 define blind server-supported signing functionality $\mathcal{F}_{\mathsf{tsign}}$. Figures 15,16 define blind server-supported decryption $\mathcal{F}_{\mathsf{tdecrypt}}$.

$\mathcal{F}_{\mathsf{rev}}$ is parametrised by the total number of tokens N.
If any assertion or parsing inside $\mathcal{F}_{\mathsf{rev}}$ fails, it outputs $\perp$ to all involved parties.

Initialisation. On input $(\mathsf{init}, \mathcal{X})$ from $\mathcal{S}$:

- Check that $|\mathcal{X}| \geq N$. Store $\mathcal{X}$.
- Initialise an empty map of revoked tokens $X^- : \mathcal{S}_{ID} \to (\mathcal{X}, \mathcal{P})$.
- For all $x \in \mathcal{X}$, initialise empty sets $\mathcal{R}_x$ and $\mathcal{U}_x$.

Issuing. On input $(\mathsf{issue}, \mathsf{sid}, R)$ from party U and $(\mathsf{issue}, \mathsf{sid}, U)$ from party R:

1. Let $L = \mathcal{L}^{\mathcal{C}}_{issue}(U, R, x)$ for $x \leftarrow\!\!\$\ \mathcal{X}$. Send $(\mathsf{issue\text{-}req}, \mathsf{sid}, L)$ to $\mathcal{S}$.
2. Receive $(\mathsf{issue\text{-}req\text{-}ok}, \mathsf{sid}, \overline{x})$ from $\mathcal{S}$. If $U \in \mathcal{C}$, take $x \leftarrow \overline{x}$.
3. Assert that there is no $U' \notin \mathcal{C}$ such that $U' \in \mathcal{U}_x$.
   Update $\mathcal{U}_x \leftarrow \mathcal{U}_x \cup \{U\}$, $\mathcal{R}_x \leftarrow \mathcal{R}_x \cup \{R\}$, $\mathcal{X} \leftarrow \mathcal{X} \setminus \{x\}$.
4. Output $(\mathsf{issue\text{-}resp\text{-}u}, \mathsf{sid}, x)$ to U. Output $(\mathsf{issue\text{-}resp\text{-}r}, \mathsf{sid}, x)$ to R.

Revocation. On input $(\mathsf{revoke}, \mathsf{sid}, x)$ from a party R, and $(\mathsf{revoke}, \mathsf{sid})$ from S:

1. Let $L = \mathcal{L}^{\mathcal{C}}_{revoke}(R, x)$. Send $(\mathsf{revoke\text{-}req}, \mathsf{sid}, L)$ to $\mathcal{S}$.
2. Receive $(\mathsf{revoke\text{-}req\text{-}ok}, \mathsf{sid})$ from $\mathcal{S}$.
3. If $R \in \mathcal{R}_x$ or $\exists R' \in \mathcal{R}_x \cap \mathcal{C}$, assign $X^-[\mathsf{sid}] = (x, R)$.
4. Output $(\mathsf{revoke\text{-}resp\text{-}s}, \mathsf{sid}, \mathsf{ok})$ to S. Output $(\mathsf{revoke\text{-}resp\text{-}r}, \mathsf{sid}, \mathsf{ok})$ to R.

User notification. On input $(\mathsf{unotify}, \mathsf{sid}, x)$ from U and $(\mathsf{unotify}, \mathsf{sid})$ from S:

1. Let $L = \mathcal{L}^{\mathcal{C}}_{unotify}(U, x)$. Let $X \leftarrow \emptyset$. Send $(\mathsf{unotify\text{-}req}, \mathsf{sid}, L)$ to $\mathcal{S}$.
2. Receive $(\mathsf{unotify\text{-}req\text{-}ok}, \mathsf{sid}, S_{ID})$ from $\mathcal{S}$.
   If $S \in \mathcal{C}$, take $X \leftarrow \{x \mid (x, R) \in X^-[S_{ID}], \mathsf{brokenCons}(U, R) = \mathsf{true}\}$.
3. Let $b = \mathsf{true}$ iff $U \in \mathcal{U}_x$ and $x \in X^- \setminus X$.
4. Output $(\mathsf{unotify\text{-}resp\text{-}s}, \mathsf{sid}, \mathsf{ok})$ to S. Output $(\mathsf{unotify\text{-}resp\text{-}u}, \mathsf{sid}, b)$ to U.

Verifier notification. On input $(\mathsf{vnotify}, \mathsf{sid})$ from V and $(\mathsf{vnotify}, \mathsf{sid})$ from S:

1. Let $L = \mathcal{L}^{\mathcal{C}}_{vnotify}(V)$. Let $X \leftarrow \emptyset$. Send $(\mathsf{vnotify\text{-}req}, \mathsf{sid}, L)$ to $\mathcal{S}$.
2. Receive $(\mathsf{vnotify\text{-}req\text{-}ok}, \mathsf{sid}, S_{ID})$ from $\mathcal{S}$.
   If $S \in \mathcal{C}$, take $X \leftarrow \{x \mid (x, R) \in X^-[S_{ID}], \mathsf{brokenCons}(V, R) = \mathsf{true}\}$.
3. Assign $X^-_V \leftarrow X^- \setminus X$.
4. Output $(\mathsf{vnotify\text{-}resp\text{-}s}, \mathsf{sid}, \mathsf{ok})$ to S. Output $(\mathsf{vnotify\text{-}resp\text{-}v}, \mathsf{sid}, \mathsf{ok})$ to V.

Verification. On input $(\mathsf{verify}, \mathsf{sid}, x, V)$ from U, and $(\mathsf{verify}, \mathsf{sid})$ from V:

1. Let $L = \mathcal{L}^{\mathcal{C}}_{verify}(U, V, x)$. Send $(\mathsf{verify\text{-}req}, \mathsf{sid}, L)$ to $\mathcal{S}$.
2. Receive $(\mathsf{verify\text{-}req\text{-}ok}, \mathsf{sid})$ from $\mathcal{S}$.
3. Assert $x \notin X^-_V$.
4. Output $(\mathsf{verify\text{-}resp\text{-}v}, \mathsf{sid}, x)$ to V. Output $(\mathsf{verify\text{-}resp\text{-}u}, \mathsf{sid}, \mathsf{ok})$ to U.

Fig. 13. Ideal functionality $\mathcal{F}_{\mathsf{rev}}$ (Part 1). For each subroutine f, the leakage function $\mathcal{L}^{\mathcal{C}}_f$ outputs the values leaked to the adversary assuming the set $\mathcal{C}$ of corrupted parties, and the particular leakages are listed in Table 8. Gray colour denotes commands which are never called assuming a covertly corrupted server.

Routine	Leakage	Value	Comments
Issue	$\mathcal{L}^{\emptyset}_{issue}(U,R,x)$	U,R	Do not hide the party identities.
	$\mathcal{L}^{\{R\}}_{issue}(U,R,x)$	x	The revoker learns x
	$\mathcal{L}^{\{U\}}_{issue}(U,R,x)$	x	The user learns x
Revoke	$\mathcal{L}^{\emptyset}_{revoke}(R,x)$	R	Do not hide the identity of the revoker R.
		$(R \in \mathcal{R}_x)$ $\wedge (x \notin X^-)$	Leak whether the request is legitimate and x has not been revoked yet.
	$\mathcal{L}^{\{S,V\}}_{revoke}(R,x)$	$\{x\} \cap \mathsf{pres}(V)$	The server learns x shown to corrupted V.
	$\mathcal{L}^{\mathcal{U}_x}_{revoke}(R,x)$	x	The user of x learns x
Unotify	$\mathcal{L}^{\emptyset}_{unotify}(U,x)$	U	Do not hide the identity of the user U.
		$U \in \mathcal{U}_x$	Leak the fact of illegitimate request.
Vnotify	$\mathcal{L}^{\emptyset}_{vnotify}(V)$	V	Do not hide the identity of the verifier V.
Verify	$\mathcal{L}^{\emptyset}_{verify}(U,V,x)$	$x \notin \mathsf{notif}(U)$	Leak whether the user has previously called a sufficiently fresh notification for x.
	$\mathcal{L}^{\{V\}}_{verify}(U,V,x)$	x	The verifier learns x.
	$\mathcal{L}^{\{S,V\}}_{verify}(U,V,x)$	$\mathsf{rev\_sids}(x)$	Server and verifier learn when x was revoked.

Table 8. Leakages of $\mathcal{F}_{\mathsf{rev}}$. The set $\mathsf{pres}(V)$ is the set of tokens x such that $(\mathsf{verify}, \mathsf{sid}, x, V)$ has been input to $\mathcal{F}_{\mathsf{rev}}$. The set $\mathsf{notif}(U)$ is the set of tokens x such that $\mathcal{F}_{\mathsf{rev}}$ has returned $(\mathsf{unotify\text{-}resp\text{-}u}, \mathsf{sid}, \mathsf{true})$ on the *latest* input $(\mathsf{unotify}, \mathsf{sid}, x)$ from the user U, which happened *after* $\mathcal{F}_{\mathsf{rev}}$ sent its last $(\mathsf{vnotify\text{-}resp\text{-}v}, \mathsf{sid}, \mathsf{ok})$ to V. The set $\mathsf{rev\_sids}(x)$ is the set of session identifiers sid such that $(\mathsf{revoke}, \mathsf{sid}, x)$ has been input to $\mathcal{F}_{\mathsf{rev}}$ before.

B Detailed benchmarks of Prettiness protocols

The detailed benchmarks for the full system can be found in Table 9 (communication) and Table 10 (time). For each routine, we collected the total communication between all parties, including communication induced by used subfunctionalities. The total time overhead, summed up over all involved parties, is considered for the most time-consuming operations, which are (significantly) slower than the others. For clarity, we report time overheads of the all used building block functionalities even if these are negligible.

In benchmarks, we assume that the token randomness (hidden into the details of protocol implementing $\mathcal{F}_{\mathsf{rev}}$) is generated by the user in advance, and is stored in encrypted form as a part of credential, to ensure full backup. This ciphertext corresponds to the value c_{rev_r} in the benchmarks.

C UC proof for the full Prettiness system

To prove security of the proposed system, we define a simulator $\mathcal{S}$ mediating communication between the $\mathcal{A}$ and the ideal functionality $\mathcal{F}_{\mathsf{ideal}}$, trying to convince $\mathcal{A}$ that it is communicating with the real protocol $\pi_{\mathsf{PRETTINESS}}$. Description of the simulator is split into different phases where each phase has separate subdescription for the different corruption models. The involved party that is not

Issue	
$\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}(\mathsf{sign\ and\ verify})$	7638
$\mathsf{ST} \rightarrow U : I, \mathsf{IID}$	128
$\mathsf{ST} \rightarrow I : U, id_{\mathsf{cID}}, \mathsf{uID}, \overrightarrow{\mathsf{aID}}, \sigma_{\mathsf{req}}, c_{\mathsf{rev},r}$	3994
$U \rightarrow I : \mathsf{rev}_U$	32
$\mathcal{F}_{\mathsf{rev}}(\mathsf{issue})$	48
$I \rightarrow \mathsf{ST} : \mathsf{cred}, \sigma_{\mathsf{cred}}$	38802
$\mathsf{ST} \rightarrow U : \mathsf{cred}, \sigma_{\mathsf{cred}}$	38802
$(n+2) \times \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Dec})$	1272
Total:	90716

Get all Creds	
$\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}(\mathsf{sign\ and\ verify})$	674
$\mathsf{ST} \rightarrow U : data$	3905800
Total:	3906474

Revocation by Issuer	
$I \rightarrow ST : m, \sigma_{\mathsf{req}}$	3195
$\mathcal{F}_{\mathsf{rev}}(\mathsf{revoke})$	160
Total:	3355

Revocation by User	
$\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}(\mathsf{sign\ and\ verify})$	6422
$\mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Dec})$	106
$\mathcal{F}_{\mathsf{rev}}(\mathsf{revoke})$	160
Total:	6688

Present	
$\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}(\mathsf{sign\ and\ verify})$	6424
$(n+1) \times \mathcal{F}_{\mathsf{tdecrypt}}(\mathsf{Dec})$	1166
Total:	7590

DB update	
$\mathcal{F}_{\mathsf{rev}}((\mathsf{v/u})\mathsf{notify})$	32000114
Total:	32000114

Verify	
$RP \rightarrow U : \mathsf{ch}$	32
$\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}(\mathsf{sign})$	80900
$U \rightarrow RP : data$	40530
$2 \times \mathcal{F}_{\mathsf{rev}}(\mathsf{verify})$	324
$(n+2) \times \mathcal{F}_{\mathsf{tdecrypt}}(\mathsf{Verify})$	384
Total:	122170

Table 9. Communication (in bytes) for the full system for $N = 100$ credentials per user, $n = 10$ attributes per credential, $m = 10^6$ revoked tokens.

Issue		
U	Generate randomness of rev in advance, encrypt it using $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Enc})$	0.282
U, ST	$\mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}$(sign and verify)	0.1
I	$\mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}$(verify)	0.001
I	$(n+2) \times \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Enc})$	3.384
I	$\mathcal{F}^{\mathsf{IID}}_{\mathsf{sign}}$(sign)	0.001
U, I	$\mathcal{F}_{\mathsf{rev}}$(issue)	0.0
ST	$\mathcal{F}^{\mathsf{IID}}_{\mathsf{sign}}$(verify)	0.001
U	$\mathcal{F}^{\mathsf{IID}}_{\mathsf{sign}}$(verify)	0.001
U	update h_{cred}	0.0003
U, ST	$(n+2) \times \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Dec})$	4.344
Total:		8.1143

Get all Creds		
U, ST	$\mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}$(sign and verify)	0.1
U	compute h_{cred}	0.0266
Total:		0.1266

Revocation by Issuer		
I	$\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Enc})$	0.282
I	$\mathcal{F}^{\mathsf{IID}}_{\mathsf{sign}}$(sign)	0.001
ST	$\mathcal{F}^{\mathsf{IID}}_{\mathsf{sign}}$(verify)	0.001
I, ST	$\mathcal{F}_{\mathsf{rev}}$(revoke)	0.743
Total:		1.027

Revocation by User		
U	$\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Enc})$	0.282
U, ST	$\mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}$(sign and verify)	0.1
U, ST	$\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Dec})$	0.362
U, ST	$\mathcal{F}_{\mathsf{rev}}$(revoke)	0.743
Total:		1.487

Present		
U	$\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Enc})$	0.282
U, ST	$\mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}$(sign and verify)	0.1
U, ST	$(n+1) \times \mathcal{F}_{\mathsf{tdecrypt}}(\mathsf{Dec})$	3.982
U, ST	$2 \times \mathcal{F}_{\mathsf{rev}}$(unotify)	1.346
Total:		5.71

DB update		
U/RP	$\mathcal{F}_{\mathsf{rev}}$((v/u)notify)	0.716
Total:		0.716

Verify		
U, ST	$\mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}$(sign)	0.1
RP	$\mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}$(verify)	0.001
RP	$\mathcal{F}^{\mathsf{IID}}_{\mathsf{sign}}$(verify)	0.001
U, RP	$2 \times \mathcal{F}_{\mathsf{rev}}$(verify)	4.6
U, RP	$(n+2) \times \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$(Verify)	0.0
Total:		4.702

Table 10. Time (in seconds) for the full system for $N = 100$ credentials per user, $n = 10$ attributes per credential, $m = 10^6$ revoked tokens.

Break consistency. On input $(\mathsf{brokenCons}, (V, R))$ from $\mathcal{S}$, if $V \notin \mathcal{C}$,$R \notin \mathcal{C}$, $S \in \mathcal{C}$, set $\mathsf{brokenCons}(V, R) = \mathsf{true}$.
Detect cheating. Upon receiving $(\mathsf{cheat\text{-}detect}, \mathsf{sid}, \mathsf{sid}_P)$ from $P \in \{U, V\}$, $(\mathsf{cheat\text{-}detect}, \mathsf{sid}, \mathsf{sid}_R)$ from R, and $(\mathsf{cheat\text{-}detect}, \mathsf{sid})$ from S and J:

1. If at least one involved party P, R, S, J is corrupted:
   - Send $(\mathsf{dectect\text{-}cheat\text{-}req}, \mathsf{sid}, \mathsf{sid}_P, \mathsf{sid}_R)$ to $\mathcal{S}$.
   - Wait for $(\mathsf{dectect\text{-}cheat\text{-}ok}, \mathsf{sid}, \mathcal{C}')$ from $\mathcal{S}$ with $\mathcal{C}' \subseteq \mathcal{C}$.

   Otherwise, set $\mathcal{C}' = \emptyset$.
2. If $\mathsf{brokenCons}(V, R) = \mathsf{true}$, output $(\mathsf{verdict}, \mathsf{sid}, \mathsf{blame}(\{S\} \cup \mathcal{C}'))$ to J. Otherwise, output $(\mathsf{verdict}, \mathsf{sid}, \mathsf{blame}(\mathcal{C}'))$ to J.

Unfairness. Whenever a message of the form $(response, \mathsf{sid}, y)$ for $response \neq \mathsf{verdict}$ is to be output to an honest party $P \neq S$, send $(response, \mathsf{sid}, y)$ to $\mathcal{S}$. Wait for $(response\text{-}\mathsf{ok}, \mathsf{sid})$ from $\mathcal{S}$ and send $(response, \mathsf{sid}, y)$ to P.

Fig. 14. Ideal functionality $\mathcal{F}_{\mathsf{rev}}$ (Part 2, adversary interfaces and accountability).

listed among the "malicious" parties is treated as honest, and is being simulated by $\mathcal{S}$. We only consider "interesting" corruptions, e.g. we do not describe simulation of a corrupted issuer alone if cheating is only possible in collaboration with a corrupted storage. We do not describe the case where all involved parties are corrupted, as there is nothing to be simulated in the real protocol, and their interactions with the sub-functionalities affect only corrupted parties. We also do not describe the case where all involved parties are honest, as the simulator would only need to simulate the notifications of different steps to $\mathcal{A}$, and simulations of these are already covered by the cases where that party is honest.

The protocol $\pi_{\mathsf{PRETTINESS}}$ is using functionality $\mathcal{F}_{\mathsf{rev}}$ which is parametrized by leakage functions $\mathcal{L}_f^{\mathcal{C}}$ for a subroutine f and the set of corrupted parties $\mathcal{C}$. We write $L = \mathcal{L}_f^{\mathcal{C}}(x)$ to denote leakage L that will be shown to the adversary, computed based on the internal state and the inputs x of simulated $\mathcal{F}_{\mathsf{rev}}$. Throughout the proof, we write $\mathcal{F}(cmd, \mathsf{sid}, x)$ to denote running the subroutine cmd of $\mathcal{F}$ with sid and input x.

For the purpose of this paper, we allow adversary to corrupt user either before key generation, reaching level $c_{\mathsf{U}} = 3$ or after the key generation phase, reaching level $c_{\mathsf{U}} = 1$ (and subsequently $c_{\mathsf{U}} = 2$). In the latter case, we prove that RP is protected from malicious user. Adversary can increase its level of corruption while making silent requests to the functionality. Processing of each silent request begins with the call to authentication sub-routine and if it results in successful authentication, corruption level can increase.

C.1 Initialisation

In this phase, initialisation of all used sub-functionalities is simulated. $\mathcal{S}$ also initializes $\mathcal{F}_{\mathsf{ideal}}$ using the ideal adversary's interface. We omit simulation of ini-

Key generation
On message (KeyGen, sid) from both client and server:

1. Send (KeyGen, sid) to $\mathcal{S}$ and wait for $(\mathsf{Key}, sid, \mathsf{pk})$ from $\mathcal{S}$.
2. If $\mathsf{pk} \neq \perp$, create an empty table T for storing encrypted messages, and set decryption counter $ctr_{\mathsf{dec}} \leftarrow 0$.
3. If the party $P \in \{\mathsf{client}, \mathsf{server}\}$ is corrupted, wait for $(\mathsf{Output}, i, y_i)$ from $\mathcal{A}$ for $i \in \{\mathsf{c},\mathsf{s}\}$. Otherwise, take $y_i = \mathsf{pk}$.
4. Output $(\mathsf{Key}, sid, y_{\mathsf{c}})$ to client and $(\mathsf{Key}, sid, y_{\mathsf{s}})$ to server.

Encryption
On message $(\mathsf{Encrypt}, sid, m)$ from some party P:

1. If P is corrupted, send $(\mathsf{Encrypt}, sid, P, m)$ to $\mathcal{S}$. Otherwise, send $(\mathsf{Encrypt}, sid)$ to $\mathcal{S}$.
2. Wait for $(\mathsf{Encrypt}, sid, c)$ from $\mathcal{S}$ (the adversary chooses the ciphertext for the given plaintext).
3. Store the pair (m, c) in table T.
4. If the party P is corrupted, wait for (Output, P, y) from $\mathcal{A}$. Otherwise, take $y = c$.
5. Send $(\mathsf{Encrypted}, sid, y)$ to P.

Decryption procedure *Decrypt*(Dec, c) for a function $\mathsf{Dec} : \mathsf{C} \rightarrow \mathsf{M}$ and a ciphertext $c \in \mathsf{C}$:

- If there exists a pair (m, c) in T: (for correctness we need that c would decrypt to m)
  - If m is not unique (there are two different m for the same c), take $m' = \perp$.
  - If m is unique, take $m' = m$.
- Otherwise, compute $m' := \mathsf{Dec}(c)$, running it only a polynomial number of steps (the polynomial is a parameter of $\mathcal{F}_{\mathsf{tdecrypt}}$). Add (m', c) to T.
- Return m'.

Fig. 15. Ideal encryption functionality $\mathcal{F}_{\mathsf{tdecrypt}}$ from [21] (1/2)

Decryption (honest client)
On message $(\mathsf{Decrypt}, sid, c)$ from client and $(\mathsf{Decrypt}, sid)$ from party server:

1. Send $(\mathsf{Decrypt\text{-}init}, sid)$ to $\mathcal{S}$.
2. Upon receiving $(\mathsf{Decrypt\text{-}init}, sid, \mathsf{Ver})$ from $\mathcal{S}$, where $\mathsf{Ver} : \mathsf{C} \rightarrow \{\mathsf{true}, \mathsf{false}\}$, compute $b \leftarrow \mathsf{Ver}(c)$.
   - If $b = \mathsf{true}$, send $(\mathsf{Decrypt\text{-}good\text{-}c}, sid)$ to $\mathcal{S}$.
   - If $b = \mathsf{false}$ or $b = \perp$, send $(\mathsf{Decrypt\text{-}bad\text{-}c}, sid)$ to $\mathcal{S}$. Proceed to the point 4) with $y_\mathsf{c} = y_\mathsf{c} = \perp$.
3. The final output y_c for client depends on whether the server is corrupt. If the server is *honest*, the decryption can only be delayed:
   - Upon receiving $(\mathsf{Decrypt\text{-}complete}, sid, \mathsf{Dec})$ from $\mathcal{S}$, take $y_\mathsf{c} \leftarrow Decrypt(\mathsf{Dec}, c)$, and $y_\mathsf{s} \leftarrow \top$.

   If the server is *corrupted*, $\mathcal{S}$ tells whether decryption succeeded.
   - Upon receiving $(\mathsf{Decrypt\text{-}complete}, sid, \mathsf{Dec})$ from $\mathcal{S}$, take $y_\mathsf{c} \leftarrow Decrypt(\mathsf{Dec}, c)$.
   - Upon receiving $(\mathsf{Decrypt\text{-}fail}, sid)$ from $\mathcal{S}$, take $y_\mathsf{s} \leftarrow \perp$.
4. If the server is corrupted, wait for $(\mathsf{Output}, 2, y'_\mathsf{s})$ from $\mathcal{A}$ and take $y_\mathsf{s} \leftarrow y'_\mathsf{s}$.
5. Send $(\mathsf{Decrypted}, sid, y_\mathsf{c})$ to client and $(\mathsf{Decrypted}, sid, y_\mathsf{s})$ to server.

Decryption (corrupted client)
On message $(\mathsf{Decrypt}, sid, c)$ from client and $(\mathsf{Decrypt}, sid)$ from server:

1. Set $ctr_\mathsf{dec} \leftarrow ctr_\mathsf{dec} + 1$ and send $(\mathsf{Decrypt\text{-}init}, sid, c)$ to $\mathcal{S}$.
2. $\mathcal{S}$ tells whether decryption succeeded:
   - Upon receiving $(\mathsf{Decrypt\text{-}complete}, sid)$ from $\mathcal{S}$, take $y_\mathsf{s} \leftarrow \top$.
   - Upon receiving $(\mathsf{Decrypt\text{-}fail}, sid)$ from $\mathcal{S}$, take $y_\mathsf{s} \leftarrow \perp$.
3. Wait for $(\mathsf{Output}, 1, y_\mathsf{c})$ from $\mathcal{A}$.
4. Send $(\mathsf{Decrypted}, sid, y_\mathsf{c})$ to client and $(\mathsf{Decrypted}, sid, y_\mathsf{s})$ to server.

Silent decryption (corrupted client). On message $(\mathsf{Decrypt\text{-}msg}, sid, \mathsf{Dec}, c')$ from $\mathcal{S}$:

1. Compute $m = Decrypt(\mathsf{Dec}, c')$.
2. Set $ctr_\mathsf{dec} \leftarrow ctr_\mathsf{dec} - 1$. If $ctr_\mathsf{dec} < 0$, then stop.
3. Send $(\mathsf{Decrypted}, sid, m)$ to $\mathcal{S}$.

Verify decryption. On input $(\mathsf{Verify}, \mathsf{sid}, (c, m))$ from the client and from the verifier:

1. If there exists a pair (m, c) in T:
   - If m is not unique (there are two different m for the same c), output $(\mathsf{Verify\text{-}fail}, \mathsf{sid})$ to the verifier.
   - If m is unique, output $(\mathsf{Verify\text{-}success}, \mathsf{sid})$ to the verifier.
2. Otherwise, output $(\mathsf{Verify\text{-}fail}, \mathsf{sid})$ to the verifier.

Fig. 16. Ideal encryption functionality $\mathcal{F}_{\mathsf{tdecrypt}}$ from [21] (2/2), extended with verification interface.

Initialization: On input (init) from $\mathcal{S}$, set $\mathsf{c_S} \leftarrow \mathsf{c_C} \leftarrow T \leftarrow 0$; $b_{\mathrm{lq}} \leftarrow b_{\mathrm{OK}} \leftarrow b_{\mathrm{sk}} \leftarrow 1$; $\mathsf{ctr}_s \leftarrow 0$;$\mathsf{Ver} \leftarrow \perp$. Send (init) to $\mathcal{S}$.
Key Generation: On input $(\mathsf{keygen}, \mathsf{sid}, L, T_0, \mathsf{PIN})$ from client and $\overline{(\mathsf{keygen}, \mathsf{sid}, T_0)}$ from server, if Ver is already recorded or $\mathsf{PIN} \notin \{0, \ldots, L-1\}$, ignore this query. Otherwise:

- Send (keygen-init) to adversary $\mathcal{S}$.
- Upon receiving $(\mathsf{key}, \mathsf{pk})$ from $\mathcal{S}$, where pk is public key, store $(\mathsf{pk}, L, \mathsf{PIN}, T_0)$ and send $(\mathsf{key}, \mathsf{pk})$ to both client and server.

Corrupt server: On input (corrupt-server) from $\mathcal{S}$: set $\mathsf{c_S} \leftarrow 1$ and send $\overline{(\mathsf{corrupt\text{-}server})}$ to server.
Corrupt client: On input $(\mathsf{corrupt\text{-}client}, \ell)$ from $\mathcal{S}$, where $\ell \in \{1, 2, 3\}$:

- If $\mathsf{c_C} > \ell$, ignore this query.
- Set $b_{\mathrm{lq}} \leftarrow \perp$. If $\mathsf{pk} = \perp$, set $\mathsf{c_C} \leftarrow 3$. Otherwise, set $\mathsf{c_C} \leftarrow \ell$. Send $(\mathsf{corrupt\text{-}client}, \ell)$ to client.

Derived values: $b_{\mathrm{KG}} \equiv [\mathsf{pk} \neq \perp]$.
Trigger: If $\mathsf{c_S} = \mathsf{c_C} = 1$, then set $\mathsf{c_C} \leftarrow 2$.
Trigger: If $\mathsf{c_C} \geq 2$ and b_{KG} then send $(\mathsf{corrupt\text{-}pin}, \mathsf{PIN})$ to $\mathcal{S}$.
Forwarding: If $\mathsf{c_C} = 3$ [resp. $\mathsf{c_S} = 1$], then inputs from client [resp. server] are forwarded to $\mathcal{S}$; and $\mathcal{S}$ determines the outputs that client [resp. server] receives.

Fig. 17. Ideal functionality for blind server-supported signing $\mathcal{F}_{\mathsf{tsign}}$ (1/3)

Signing by client and server: On input $(\mathsf{sign}, \mathsf{sid}, M, \mathsf{PIN}')$ from the client and $(\mathsf{sign}, \mathsf{sid})$ from the server, ignore unless $b_{\mathrm{KG}} \wedge b_{\mathrm{OK}}$. Otherwise:

- Call $\mathsf{clone\text{-}check}(1)$ and $\mathsf{process\text{-}pin}(\mathsf{PIN}')$. If successful, compute σ using $\mathsf{request\text{-}sig}(\mathsf{sid}, M)$.
- If $\sigma \neq \perp$, output $(\mathsf{sign\text{-}success}, \mathsf{sid}, M, \sigma)$ to client, and $(\mathsf{sign\text{-}success}, \mathsf{sid})$ to server. Otherwise, return $(\mathsf{sign\text{-}fail}, \mathsf{sid})$ to client and server.

Signing by client and adversary: On input $(\mathsf{sign}, \mathsf{sid}, M, \mathsf{PIN}')$ from the client and $(\mathsf{sign\text{-}server}, \mathsf{sid})$ from $\mathcal{S}$, ignore unless $b_{\mathrm{KG}} \wedge [\mathsf{c}_{\mathrm{S}} = 1]$. Otherwise:

- Call $\mathsf{process\text{-}pin}(\mathsf{PIN}')$ and then compute σ by querying $\mathsf{request\text{-}sig}(\mathsf{sid}, M)$.
- If $\sigma \neq \perp$, output $(\mathsf{sign\text{-}success}, \mathsf{sid}, M, \sigma)$ to client. Otherwise, return $(\mathsf{sign\text{-}fail}, \mathsf{sid})$ to both client and $\mathcal{S}$.

Signing by adversary and server: On input $(\mathsf{sign\text{-}client}, \mathsf{sid}, \mathsf{PIN}')$ from $\mathcal{S}$ and $(\mathsf{sign}, \mathsf{sid})$ from the server: ignore unless $b_{\mathrm{KG}} \wedge b_{\mathrm{OK}} \wedge [\mathsf{c}_{\mathrm{C}} > 0]$. Otherwise go to step $\mathsf{c}_{\mathrm{C}} \in \{1, 2, 3\}$:

1. Call $\mathsf{clone\text{-}check}(0)$ and $\mathsf{process\text{-}pin}(\mathsf{PIN}')$. If successful, set $\mathsf{c}_{\mathrm{C}} \leftarrow 2$; go to step 3.
2. Call $\mathsf{clone\text{-}check}(0)$. If successful, go to step 3.
3. Send $(\mathsf{sign\text{-}init}, \mathsf{sid})$ to $\mathcal{S}$ and upon receiving $(\mathsf{sign\text{-}success}, \mathsf{sid})$, output $(\mathsf{sign\text{-}success}, \mathsf{sid})$ to server. Set $\mathsf{ctr}_s \leftarrow \mathsf{ctr}_s + 1$. Otherwise, return $(\mathsf{sign\text{-}fail})$ to both $\mathcal{S}$ and server.

Verification: On input $(\mathsf{verify}, \mathsf{sid}, M, \sigma)$ from a party P, send $(\mathsf{verify}, \mathsf{sid}, M, \sigma)$ to $\mathcal{S}$. Upon receiving $(\mathsf{verified}, \mathsf{sid}, M, \sigma, \phi)$ from $\mathcal{S}$ with $\phi \in \{1, 0\}$:

- if (M, σ, b') is recorded, then set $b = b'$
- if $\phi = 1$, (M, σ, b) has not been recorded before, and $b_{\mathrm{sk}} = 1$, set $\mathsf{ctr}_s \leftarrow \mathsf{ctr}_s - 1$:
  - if $\mathsf{ctr}_s \geq 0$, record $(M, \sigma, b = \phi)$. // saving valid signature message pairs that have not been saved during "signing by adversary and server" queries
  - if $\mathsf{ctr}_s < 0$, set $b = 0$ and record (M, σ, b) // (one-more unforgeability) the adversary created $n + 1$ valid signatures with n signing queries
- else, set $b = \phi$ and record (M, σ, b).
- output $(\mathsf{verified}, \mathsf{sid}, M, \sigma, b)$.

Fig. 18. Ideal functionality for blind server-supported signing $\mathcal{F}_{\mathsf{tsign}}$ (2/3)

Supporting subroutine process-pin(PIN'):

- If $\mathsf{PIN} = \mathsf{PIN}'$, set $T \leftarrow 0$ and give control back to the invoker.
- Otherwise, increment T. If $T \geq T_0$ then set $b_{\mathrm{OK}} \leftarrow 0$. Return $(\mathsf{sign\text{-}fail}, \mathsf{sid})$ to the two parties (out of client, server, and $\mathcal{S}$) that initiated the signing.

Supporting subroutine clone-check(d):

- If $b_{\mathrm{lq}} = 1 - d$, set $b_{\mathrm{OK}} \leftarrow 0$ and return $(\mathsf{sign\text{-}fail}, \mathsf{sid})$ to the two parties that initiated the signing.
- Otherwise, set $b_{\mathrm{lq}} \leftarrow d$ and return to the invoker.

Supporting subroutine request-sig(sid, M):

- Send $(\mathsf{sign\text{-}init}, \mathsf{sid})$ to $\mathcal{S}$.
- Upon receiving $(\mathsf{signature}, \mathsf{sid}, \sigma)$ from $\mathcal{S}$, check whether $(M, \sigma, 0)$ is already stored. If true, restart request-sig(sid, M).
- Create record $(M, \sigma, 1)$ and return σ.

Fig. 19. Ideal functionality for blind server-supported signing $\mathcal{F}_{\mathsf{tsign}}$ (3/3)

tialisation for a corrupted user since, as we mentioned before, for simplicity we will consider further corruption after the key generation. Other than that, initialisation affects only the AMS and the Issuer.

Malicious AMS (storage):

1. On input $(\mathsf{keygen}, \mathsf{sid})$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}$, send $(\mathsf{keygen\text{-}init}, \mathsf{sid})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}$. Wait for $(\mathsf{key}, \mathsf{sid}, \mathsf{Ver})$ from $\mathcal{A}$. Store Ver as verification function for threshold signatures.
2. On input $(\mathsf{keygen}, \mathsf{sid})$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}$, send $(\mathsf{keygen\text{-}init}, \mathsf{sid})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}$. Wait for $(\mathsf{key}, \mathsf{sid}, \mathsf{pk})$ from $\mathcal{A}$. Store pk as encryption key for the threshold decryption scheme.
3. On input $(\mathsf{setup}, \mathcal{X}, \mathsf{sid})$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{rev}}$, check that $|\mathcal{X}| \geq N$ (here N is a parameter of $\mathcal{F}_{\mathsf{rev}}$). Store $\mathcal{X}$ and initialise empty sets of revoked tokens $X^- \leftarrow \emptyset$ and corrupted parties $\mathcal{C} \leftarrow \emptyset$. Initialise empty sets $\mathcal{U}_x$ and $\mathcal{R}_x$.
4. Send $(\mathsf{setup}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ideal}}$.
5. Initialise internal table $\mathsf{T}_{\mathsf{enc}}^{\mathsf{uID}}$ to store all the plaintext-ciphertext pairs, $\mathsf{T}_{\mathsf{sign}}^{\mathsf{uID}}$ to store all the generated threshold signatures, $\mathsf{T}_{\mathsf{Isign}}^{\mathsf{IID}}$ to store all the issuer signatures, $\mathsf{T}_{\mathsf{issue}}$ to store all the issued credentials. Instantiate set of possible attribute values $\mathcal{M}_{\mathsf{aID}}$ for an attribute identifier aID.
6. Define $\mathsf{Present}(\mathsf{cred}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}})$ as
   (a) For all $\mathsf{aID}_i \in \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}$, find corresponding atr_i in cred and add them to the $\overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}$ array.

(b) Set $\mathsf{pres} = (\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}, \mathsf{IID})$, where IID is extracted from cred, and return pres.

Define $\mathsf{Ver}(\mathsf{cred}, \mathsf{pres})$ as

(a) Parse $\mathsf{pres} = (\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}, \mathsf{IID})$.

(b) For all $\mathsf{aID}_i \in \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}$ and $\mathsf{atr}_i \in \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}$, verify that there is exist aID'_j and atr'_j in cred such that $\mathsf{aID}_i = \mathsf{aID}'_j$ and $\mathsf{atr}_i = \mathsf{atr}'_j$. Verify that cred contains IID. If these checks pass, set $b = 1$. Otherwise, set $b = 0$. Return b.

7. On message $(\mathsf{setup\text{-}ok})$ from $\mathcal{F}_{\mathsf{ideal}}$, send $(\mathsf{setup\text{-}ok}, \mathsf{sid}, \mathsf{Present}, \mathsf{Ver})$ to $\mathcal{F}_{\mathsf{ideal}}$.

Malicious AMS (storage) and Issuer:

1. On input $(\mathsf{keygen}, \mathsf{sid})$ from $\mathcal{A}$ to $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}$, send $(\mathsf{keygen\text{-}init}, \mathsf{sid})$ to $\mathcal{A}$ on behalf of $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}$. Wait for $(\mathsf{key}, \mathsf{sid}, \mathsf{Ver})$ from $\mathcal{A}$. Store Ver as verification function for threshold signatures.
2. On input $(\mathsf{keygen}, \mathsf{sid})$ from $\mathcal{A}$ to $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$, send $(\mathsf{keygen\text{-}init}, \mathsf{sid})$ to $\mathcal{A}$ on behalf of $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$. Wait for $(\mathsf{key}, \mathsf{sid}, \mathsf{pk})$ from $\mathcal{A}$. Store pk as encryption key for threshold decryption scheme.
3. On input $(\mathsf{setup}, \mathcal{X}, \mathsf{sid})$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{rev}}$, check that $|\mathcal{X}| \geq N$ (here N is a parameter of $\mathcal{F}_{\mathsf{rev}}$). Store $\mathcal{X}$ and initialise empty sets of revoked tokens $X^- \leftarrow \emptyset$ and corrupted parties $\mathcal{C} \leftarrow \emptyset$. Initialise empty sets $\mathcal{U}_x$ and $\mathcal{R}_x$.
4. Send $(\mathsf{setup}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ideal}}$.
5. Initialise internal table $\mathsf{T}^{\mathsf{uID}}_{\mathsf{enc}}$ to store all the plaintext-ciphertext pairs, $\mathsf{T}^{\mathsf{uID}}_{\mathsf{sign}}$ to store all the generated threshold signatures, $\mathsf{T}^{\mathsf{IID}}_{\mathsf{Isign}}$ to store all the issuer signatures, $\mathsf{T}_{\mathsf{issue}}$ to store all the issued credentials.
6. Define $\mathsf{Present}(\mathsf{cred}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}})$ as

(a) For all $\mathsf{aID}_i \in \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}$, find corresponding atr_i in cred and add them to the $\overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}$ array.

(b) Set $\mathsf{pres} = (\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}, \mathsf{IID})$, where IID is extracted from cred, and return pres.

Define $\mathsf{Ver}(\mathsf{cred}, \mathsf{pres})$ as

(a) Parse $\mathsf{pres} = (\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}, \mathsf{IID})$.

(b) For all $\mathsf{aID}_i \in \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}$ and $\mathsf{atr}_i \in \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}$, verify that there exist aID'_j and atr'_j in cred such that $\mathsf{aID}_i = \mathsf{aID}'_j$ and $\mathsf{atr}_i = \mathsf{atr}'_j$. Verify that cred contains IID. If these checks pass, set $b = 1$. Otherwise, set $b = 0$. Return b.

7. On message $(\mathsf{setup\text{-}ok})$ from $\mathcal{F}_{\mathsf{ideal}}$, send $(\mathsf{setup\text{-}ok}, \mathsf{sid}, \mathsf{Present}, \mathsf{Ver})$ to $\mathcal{F}_{\mathsf{ideal}}$.

Malicious User: if the adversary corrupts user before key generation, corruption reaches level $c_{\mathsf{U}} = 3$ and we cannot provide any security or privacy guarantees to the user in that case. In the analysis below, we do not assume full corruption of the device and analyse user corruption on level $c_{\mathsf{U}} = 1$ (that can increase to $c_{\mathsf{U}} = 2$ throughout the interaction with system).

C.2 Adversary-controlled subroutine calls

In the real protocol, we are using functionalities with which the adversary may interact at any time, without input of any honest parties. Let us describe which cases there are, and how they are simulated.

1. The adversary may **encrypt** any message using $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$.
   On message $(\mathsf{Encrypt}, \mathsf{sid}, m)$ to $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$ from corrupted party interface, send $(\mathsf{Encrypt}, \mathsf{sid}, m)$ to $\mathcal{A}$ and wait for $(\mathsf{Encrypt}, \mathsf{sid}, c)$. Store the pair (m, c) in table $\mathsf{T}_{\mathsf{enc}}$. Send $(\mathsf{Encrypted}, \mathsf{sid}, c)$ to $\mathcal{A}$.
2. The adversary can **decrypt** a message using $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$ only if both the user and the AMS are corrupted.
   On message $(\mathsf{Decrypt}, \mathsf{sid}, c)$ from corrupted user, and $(\mathsf{Decrypt}, \mathsf{sid})$ from AMS, if there exists a pair (m, c) in $\mathsf{T}_{\mathsf{enc}}$:
   - If m is not unique, set $m' = \perp$,
   - If m is unique, set $m' = \mathsf{rev}$.

   Output decrypted m' to the user.
3. The adversary may **verify** any signature using $\mathcal{F}^{\mathsf{lID}}_{\mathsf{sign}}$.
   Upon receiving $(\mathsf{verify}, \mathsf{sid}, msg, \sigma)$ from adversary, if $(m, \sigma, 1)$ is recorded then set $f = 1$. Else, if the signer is not corrupted, and no entry $(m, \sigma', 1)$ for any σ' is recorded, then set $f = 0$ and record the entry $(m, \sigma, 0)$. Else, if there is an entry (m, σ, f') recorded, then set $f = f'$. Else, set $f = \phi$, and record the entry (m, σ', ϕ). Output $(\mathsf{verified}, \mathsf{sid}, msg, f)$ to adversary.
4. Similarly, the adversary may **verify** any signature using $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}$.
5. If the issuer is corrupted, the adversary may **sign** any message on their behalf using $\mathcal{F}^{\mathsf{lID}}_{\mathsf{sign}}$.
   Upon receiving $(\mathsf{sign}, \mathsf{sid}, msg)$ to $\mathcal{F}^{\mathsf{lID}}_{\mathsf{sign}}$ from corrupted signer interface, set $\sigma = \mathsf{s}(msg)$ and verify that $\mathsf{v}(\sigma, msg) = 1$. If verification succeeds, output $(\mathsf{signature}, \mathsf{sid}, msg, \sigma)$ to $\mathcal{A}$ and record entry (msg, σ) to $\mathsf{T}_{\mathsf{lsign}}$. Otherwise, output error message to $\mathcal{A}$.
6. Similarly, the adversary may **sign** a message using $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}$. Differently from $\mathcal{F}^{\mathsf{lID}}_{\mathsf{sign}}$, the adversary needs to corrupt *both* the user and the AMS.
7. If the AMS is corrupted, the adversary can make **requests to** $\mathcal{F}_{\mathsf{rev}}$, which allow to issue tokens to corrupted parties, get all revoked tokens of corrupted users, and revoke tokens by a corrupted party. Only the last point may have an impact on an honest party and needs to be synchronised with $\mathcal{F}_{\mathsf{ideal}}$.
   Let rev be the token that $\mathcal{A}$ wants to silently revoke. $\mathcal{S}$ checks whether rev has already been linked to some cID in the table $\mathsf{T}_{\mathsf{issue}}$ of all issued credentials.
   - If there is such cID (or even several cID-s, which is possible for a corrupted user), then the adversary should have learned rev from a corrupted user, or a corrupted issuer. $\mathcal{S}$ uses silent request interface of $\mathcal{F}_{\mathsf{ideal}}$ to add cID to T_r of $\mathcal{F}_{\mathsf{ideal}}$.
   - If there is no such cID, then rev has not been issued yet (using $\mathcal{F}_{\mathsf{ideal}}$). In this case, $\mathcal{S}$ only adds rev to the set X^- of simulated $\mathcal{F}_{\mathsf{rev}}$. // As tokens are freshly sampled, this may happen only for a corrupted user, for whom addition to T_r (revocation of corrupted user's credential by a corrupted party) does not need to be simulated, and cID of a presented credential may even not exist if it does not concern any honest issuers.

C.3 Malicious User Authentication

A malicious user ($c_{\mathsf{U}} = 3$) is able to call routines of $\mathcal{F}_{\mathsf{ideal}}$ at any time. However, if the user has been honest in the beginning, the adversary does not have direct access to the user input interface of $\mathcal{F}_{\mathsf{ideal}}$, and needs to use the special silent request interface, which succeeds based on the corruption level of the user. In the real protocol, the adversary needs to get access to the secret key of $\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}$, which authenticates the user, which means guessing the PIN, leading to corruption $c_{\mathsf{U}} = 2$. We define a separate process of the simulator, which deals with user authentication.

1. Wait for a connection request of the form $(\mathsf{tag}_{routine}, \mathsf{sid}, \mathsf{uID})$ from $\mathcal{A}$.
2. Upon receiving input $(\mathsf{sign\text{-}phone}, \mathsf{sid}, msg, \mathsf{PIN}')$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}$, send $(\mathsf{invoke}, routine, \mathsf{sid}, \mathsf{uID})$ to $\mathcal{F}_{\mathsf{ideal}}$. Receive $(\mathsf{auth}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{ideal}}$.
3. Simulate threshold signature:
   (a) If $\mathsf{PIN}' = \mathsf{PIN}$, set $T \leftarrow 0$, $c_{\mathsf{U}} \leftarrow 2$, send $(\mathsf{corrupt\text{-}party}, \mathsf{uID}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ideal}}$ and proceed with the next step // $\mathcal{A}$ was able to guesses PIN and hence increased its corruption level. Otherwise, call $\mathsf{clone\text{-}check}(0)$ and return $(\mathsf{sign\text{-}fail})$ to $\mathcal{A}$ and $(\mathsf{auth\text{-}fail})$ to $\mathcal{F}_{\mathsf{ideal}}$ // $\mathcal{A}$ was not able to guesses PIN and authentication attempt fails.
   (b) Call $\mathsf{clone\text{-}check}(0)$, if successful proceed with the next step. // If adversary guesses the PIN and knows clone-detection bitstring, it can successfully create at least one signature
   (c) Send $(\mathsf{sign\text{-}init}, \mathsf{sid})$ to $\mathcal{A}$ through $\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}$. Upon receiving $(\mathsf{sign\text{-}success}, \mathsf{sid})$ from $\mathcal{A}$, set $\mathsf{ctr}_s \leftarrow \mathsf{ctr}_s + 1$. Otherwise, return $(\mathsf{sign\text{-}fail})$.
4. Verify signature: after receiving $(\sigma_{\mathsf{req}}, m)$ from $\mathcal{A}$, send $(\mathsf{verify}, \mathsf{sid}, m, \sigma_{\mathsf{req}})$ to $\mathcal{A}$. Upon receiving $(\mathsf{verified}, \mathsf{sid}, m, \sigma_{\mathsf{req}}, \phi)$ from $\mathcal{A}$ with $\phi \in \{1, 0\}$:
   – if $(m, \sigma_{\mathsf{req}}, b')$ is recorded, then set $b = b'$
   – if $\phi = 1$, $(m, \sigma_{\mathsf{req}}, b)$ has not been recorded before, and $b_{\mathsf{sk}} = 1$, set $\mathsf{ctr}_s \leftarrow \mathsf{ctr}_s - 1$:
     • if $\mathsf{ctr}_s \geq 0$, record $(m, \sigma_{\mathsf{req}}, b = \phi)$. // saving valid signature message pairs that have not been saved during "signing by adversary and server" queries
     • if $\mathsf{ctr}_s < 0$, set $b = 0$ and record $(m, \sigma_{\mathsf{req}}, b)$ // (one-more unforgeability) the adversary created $n + 1$ valid signatures with n signing queries
   – else, set $b = \phi$ and record $(m, \sigma_{\mathsf{req}}, b)$.
5. Send $(\mathsf{auth\text{-}ok}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ideal}}$. Assign $\mathsf{req}(routine, \mathsf{uID}, \mathsf{sid}) = (msg, \sigma)$.
6. As the result, whenever $\mathsf{req}(routine, \mathsf{uID}, \mathsf{sid})$ is defined, $\mathcal{F}_{\mathsf{ideal}}$ is waiting for a message $(\mathsf{input}, \mathsf{uID}, routine, \mathsf{sid}, x)$ from the simulator to proceed with the routine call.

C.4 Issuing

Since there are three parties involved, the simulator needs to simulate behaviour of several parties at once. For clarity, we label each simulation step, which party is being simulated.

We do not consider the cases where AMS is *honest*. This case would only be interesting in the case when we are *protecting* the AMS, as in the other cases corrupting AMS only makes the adversary stronger. However, AMS has no inputs and outputs to $\mathcal{Z}$, so there would be nothing to additionally simulate for an honest AMS.

Malicious AMS (storage):

1. *User*: On message $(\mathsf{issue\text{-}req}, \mathsf{sid}, \mathsf{uID}, \mathsf{IID}, \overrightarrow{\mathsf{aID}})$ from $\mathcal{F}_{\mathsf{ideal}}$, establish $(\mathsf{tag}_{issue}, \mathsf{sid}, \mathsf{uID})$ connection with the server. Generate $id_{\mathsf{cID}} \leftarrow \mathcal{X}_{\mathsf{id}}$, set $msg = (\mathsf{tag}_{issue}, id_{\mathsf{cID}}, \mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID})$, and wait for input $(\mathsf{sign\text{-}server}, \mathsf{sid})$ from $\mathcal{A}$. Send $(\mathsf{sign\text{-}init}, \mathsf{sid})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}$. Upon receiving $(\mathsf{signature}, \mathsf{sid}, \sigma)$ from $\mathcal{A}$, check whether $(msg, \sigma, 0)$ is already stored. In this case, restart signing. Otherwise, store $(msg, \sigma, 1)$ in table $\mathsf{T}_{\mathsf{sign}}^{\mathsf{uID}}$ and send $(\mathsf{sign\text{-}success}, \mathsf{sid})$ to the $\mathcal{A}$. Otherwise, stop execution.
2. *User*: Send $msg = (\mathsf{tag}_{issue}, id_{\mathsf{cID}}, \mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID})$ and σ to $\mathcal{A}$.
3. *Issuer*: Upon receiving $(U', id'_{\mathsf{cID}}, \mathsf{uID}', \overrightarrow{\mathsf{aID}}', \sigma')$ from $\mathcal{A}$ with n entries in $\overrightarrow{\mathsf{aID}}'$, verify the signature σ' using $\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}'}$. This ensures that the message originates from the user uID'.
4. *Issuer*: Let $\mathsf{cID} \leftarrow (id'_{\mathsf{cID}}, \mathsf{summary}(\mathsf{uID}', \overrightarrow{\mathsf{aID}}', \mathsf{IID}))$. Verify freshness of cID (and hence id'_{cID}). This ensures that the corrupted AMS has not made a repetition attack, so $\mathsf{uID} = \mathsf{uID}'$, $\overrightarrow{\mathsf{aID}}' = \overrightarrow{\mathsf{aID}}$ and $id'_{\mathsf{cID}} = id_{\mathsf{cID}}$.
5. *User*: Simulate U generating a fresh token rev_U for itself using $\mathcal{F}_{\mathsf{rev}}$:
   (a) Sample $\mathsf{rev}_U \leftarrow\!\!\$\ \mathcal{X}$. Set $L = \mathcal{L}_{issue}^{\{\mathsf{ST}\}}((U, U), \mathsf{rev}_U) = (U, U)$, and send $(\mathsf{issue\text{-}req}, \mathsf{sid}, L)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{rev}}$.
   (b) On input $(\mathsf{issue\text{-}req\text{-}ok}, \mathsf{sid})$ from $\mathcal{A}$, verify that there is no $U' \notin \mathcal{C}$ s.t. $U' \in \mathcal{U}_{\mathsf{rev}_U}$ (as rev_U is freshly sampled, it is unlikely for a sufficiently large N). Assign $\mathcal{U}_{\mathsf{rev}_U} \leftarrow \mathcal{U}_{\mathsf{rev}_U} \cup \{U\}$, $\mathcal{R}_{\mathsf{rev}_U} \leftarrow \mathcal{R}_{\mathsf{rev}_U} \cup \{U\}$.
   (c) Update $\mathcal{X} \leftarrow \mathcal{X} \setminus \{\mathsf{rev}_U\}$.
6. *User*: Let $\mathcal{A}$ send (I', IID) to U on behalf of AMS. Simulate U sending the revocation token rev_U to I'.
7. *Issuer*: Simulate I receiving the revocation token rev'_U from U'.
8. *User* and *Issuer*: If $U = U'$ and $I = I'$, jointly generate a fresh token rev_I using $\mathcal{F}_{\mathsf{rev}}$:
   (a) Sample $\mathsf{rev}_I \leftarrow\!\!\$\ \mathcal{X}$. Set $L = \mathcal{L}_{issue}^{\{\mathsf{ST}\}}(I, U, \mathsf{rev}_I) = (I, U)$, and send $(\mathsf{issue\text{-}req}, \mathsf{sid}, L)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{rev}}$.
   (b) On input $(\mathsf{issue\text{-}req\text{-}ok}, \mathsf{sid})$ from $\mathcal{A}$, verify that there is no $U' \notin \mathcal{C}$ s.t. $U' \in \mathcal{U}_{\mathsf{rev}_I}$ (as rev_I is freshly sampled, it is unlikely for a sufficiently large N). Assign $\mathcal{U}_{\mathsf{rev}_I} \leftarrow \mathcal{U}_{\mathsf{rev}_I} \cup \{U\}$, $\mathcal{R}_{\mathsf{rev}_I} \leftarrow \mathcal{R}_{\mathsf{rev}_I} \cup \{I\}$.
   (c) Update $\mathcal{X} \leftarrow \mathcal{X} \setminus \{\mathsf{rev}_I\}$.
   (d) Output $(\mathsf{issue\text{-}resp\text{-}u}, \mathsf{sid}, \mathsf{ok})$ and $(\mathsf{issue\text{-}resp\text{-}i}, \mathsf{sid}, \mathsf{ok})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{rev}}$. Now $\mathcal{A}$ may additionally choose that I or U receive $\perp$ from $\mathcal{F}_{\mathsf{rev}}$. Proceed with simulating each of these parties only if it has not received $\perp$.

   If $U' \neq U$ or $I \neq I'$, then the honest party interacts with the adversary impersonating uID or IID. As the result, $\mathcal{A}$ learns the issued rev_I. An honest issuer receives a token rev'_I. // We are *not* hiding the token rev that will be

assigned to the credential cID from a corrupted AMS. Hence, the adversary may know the link between cID and rev of issued credentials.

9. *Issuer*: Perform attribute retrieval and compose a credential with $\mathsf{rev} = (\mathsf{rev}'_I, \mathsf{rev}'_U)$:
   (a) Send $(\mathsf{issue\text{-}ok}, \mathsf{sid}, \mathsf{cID})$ to the $\mathcal{F}_{\mathsf{ideal}}$ (start interaction with $\mathcal{Z}$).
   (b) Wait for $(\mathsf{issue\text{-}atr}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{ideal}}$ (I received attribute values from $\mathcal{Z}$).
   (c) Send encryption request $(\mathsf{Encrypt}, \mathsf{sid})$ to $\mathcal{A}$ through $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$. Wait for response $(\mathsf{Encrypt}, \mathsf{sid}, c_{\mathsf{rev}})$ from $\mathcal{A}$ and store the pair $(\mathsf{rev}, c_{\mathsf{rev}})$ in table $\mathsf{T}^{\mathsf{uID}}_{\mathsf{enc}}$.
   (d) Set $\mathsf{atr}_0 \leftarrow \mathsf{pk}_{\mathsf{uID}}$ // with $\mathsf{pk}_{\mathsf{uID}}$ coming from verification Ver defined during the initialisation phase
   (e) Sample n arbitrary attribute values $\mathsf{atr}_i \leftarrow\!\!\$\ \mathcal{M}_{\mathsf{aID}_i}$. Send $n+1$ encryption requests $(\mathsf{Encrypt}, \mathsf{sid})$ to $\mathcal{A}$ on behalf of $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$. Wait for responses $(\mathsf{Encrypt}, \mathsf{sid}, c_i)$ from $\mathcal{A}$ and store the pairs (atr_i, c_i) in table $\mathsf{T}^{\mathsf{uID}}_{\mathsf{enc}}$.
   (f) Set $\mathsf{cred} = \langle(\mathsf{aID}_0, c_0), (\mathsf{aID}_1, c_1), \ldots, (\mathsf{aID}_n, c_n), (\mathsf{aID}_{\mathsf{rev}}, c_{\mathsf{rev}})\rangle$.
   (g) Generate signature on the credential cred as $\sigma_{\mathsf{cred}} = \mathsf{s}(\mathsf{cred})$ and verify that $\mathsf{v}(\sigma_{\mathsf{cred}}, \mathsf{cred}) = 1$ (s and v are parameters of $\mathcal{F}^{\mathsf{IID}}_{\mathsf{sign}}$). Save $(\sigma_{\mathsf{cred}}, \mathsf{cred})$ to table $\mathsf{T}^{\mathsf{IID}}_{\mathsf{Isign}}$.
   (h) Send $(\mathsf{cred}, \sigma_{\mathsf{cred}})$ to corrupt AMS server.
10. *User*: Verify the received credential:
   (a) Upon receiving $(\mathsf{cred}', \sigma'_{\mathsf{cred}})$ from corrupted AMS server, check that cred' indeed contains $\overrightarrow{\mathsf{aID}}$.
   (b) Verify issuer's signature σ'_{cred} on cred' using $\mathcal{F}^{\mathsf{IID}}_{\mathsf{sign}}$:
      – if $\mathsf{v}(\sigma'_{\mathsf{cred}}, \mathsf{cred}') = 1$ and no entry $(\sigma'_{\mathsf{cred}}, \mathsf{cred}')$ is recorded in $\mathsf{T}^{\mathsf{IID}}_{\mathsf{Isign}}$, output $(\mathsf{issue\text{-}atr}, \mathsf{sid}, \bot)$ to $\mathcal{F}_{\mathsf{ideal}}$,
      – otherwise, output is $\mathsf{v}(\sigma'_{\mathsf{cred}}, \mathsf{cred}')$.
   (c) Simulate decryption of c'_{rev} (contained in cred') by sending $(\mathsf{decrypt\text{-}init}, \mathsf{sid})$ to $\mathcal{A}$ on behalf of $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$.
   Upon receiving $(\mathsf{decrypt\text{-}complete}, \mathsf{sid}, \mathsf{Dec})$, if there exists a pair (m, c'_{rev}) in $\mathsf{T}^{\mathsf{uID}}_{\mathsf{enc}}$:
      – if m is not unique, set $m' = \bot$,
      – if m is unique, set $m' = m$.
   Otherwise, compute $m' := \mathsf{Dec}(c'_{\mathsf{rev}})$. Add (m', c'_{rev}) to $\mathsf{T}^{\mathsf{uID}}_{\mathsf{enc}}$.
   (d) Proceed if $m' = (\mathsf{rev}_I, \mathsf{rev}_U)$. This implies $\mathsf{rev}'_I = \mathsf{rev}_I$ and $\mathsf{rev}'_U = \mathsf{rev}_U$.
   (e) Simulate decryption of c'_0 (contained in cred') by sending $(\mathsf{decrypt\text{-}init}, \mathsf{sid})$ to $\mathcal{A}$ on behalf of $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$.
   Upon receiving $(\mathsf{decrypt\text{-}complete}, \mathsf{sid}, \mathsf{Dec})$, if there exists a pair (m, c'_0) in $\mathsf{T}^{\mathsf{uID}}_{\mathsf{enc}}$:
      – if m is not unique, set $m' = \bot$,
      – if m is unique, set $m' = m$.
   Otherwise, compute $m' := \mathsf{Dec}(c'_0)$. Add (m', c'_0) to $\mathsf{T}^{\mathsf{uID}}_{\mathsf{enc}}$.
   (f) Proceed if $m' = \mathsf{pk}_{\mathsf{uID}}$.
   (g) Simulate decryption of $n = |\overrightarrow{\mathsf{aID}}|$ ciphertexts inside cred' by sending $(\mathsf{decrypt\text{-}init}, \mathsf{sid})$ to $\mathcal{A}$ on behalf of $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$. Upon receiving $(\mathsf{decrypt\text{-}complete}, \mathsf{sid}, \mathsf{Dec})$, if there exists a pair (atr_i, c_i) in $\mathsf{T}^{\mathsf{uID}}_{\mathsf{enc}}$:

– If atr_i is not unique, set $m' = \perp$,
– If atr_i is unique, set $m' = \mathsf{atr}_i$.
Otherwise, compute $m' := \mathsf{Dec}(c_i)$. Add (m', c_i) to $\mathsf{T}^{\mathsf{uID}}_{\mathsf{enc}}$.
Let $\mathsf{atr}_i = m'$. Proceed if $\mathsf{atr}_i \in \mathcal{M}_{\overrightarrow{\mathsf{aID}}_i}$.

(h) Compute $h_{\mathsf{cred}} \leftarrow H(h_{\mathsf{cred}}, (\mathsf{cID}, \mathsf{cred}, \sigma_{\mathsf{cred}}))$.
(i) Send message $(\mathsf{atr\text{-}ok}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ideal}}$ and save $(\mathsf{cID}, \mathsf{cred}', \sigma'_{\mathsf{cred}}, \mathsf{rev})$ to $\mathsf{T}_{\mathsf{issue}}$.
// If both U and I finish successfully, then their views on the issued cID, $\overrightarrow{\mathsf{aID}}$, $\overrightarrow{\mathsf{atr}}$ match. This is what gets to $\mathsf{T}_{\mathsf{issue}}$ and the internal table T_a of $\mathcal{F}_{\mathsf{ideal}}$. The credential data cred' and σ'_{cred} correspond to the user's view, as this is what adversary expect to see during presentation.

Malicious AMS (storage) and Issuer: Since the AMS does not provide additional protection for the user, we do not consider separately the case where only the issuer is corrupted, as the simulation would be the essentially same.

1. On message $(\mathsf{issue\text{-}req}, \mathsf{sid}, \mathsf{uID}, \mathsf{IID}, \overrightarrow{\mathsf{aID}})$ from $\mathcal{F}_{\mathsf{ideal}}$, establish $(\mathsf{tag}_{issue}, \mathsf{sid}, \mathsf{uID})$ connection with the server. Generate $id_{\mathsf{cID}} \leftarrow \mathcal{X}_{\mathsf{id}}$, set $msg = (\mathsf{tag}_{issue}, id_{\mathsf{cID}}, \mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID})$, and wait for input $(\mathsf{sign\text{-}server}, \mathsf{sid})$ from $\mathcal{A}$. Send $(\mathsf{sign\text{-}init}, \mathsf{sid})$ to $\mathcal{A}$ on behalf of $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}$. Upon receiving $(\mathsf{signature}, \mathsf{sid}, \sigma)$ from $\mathcal{A}$, check whether $(msg, \sigma, 0)$ is already stored. In this case, restart signing. Otherwise, store $(msg, \sigma, 1)$ in table $\mathsf{T}^{\mathsf{uID}}_{\mathsf{sign}}$ and send $(\mathsf{sign\text{-}success}, \mathsf{sid})$ to the $\mathcal{A}$. Otherwise, stop execution.
2. *User*: Send $msg = (\mathsf{tag}_{issue}, id_{\mathsf{cID}}, \mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID})$ and σ to $\mathcal{A}$.
3. Simulate U generating a fresh token rev_U for itself using $\mathcal{F}_{\mathsf{rev}}$:
   (a) Sample $\mathsf{rev}_U \leftarrow\!\!\$\ \mathcal{X}$. Set $L = \mathcal{L}^{\{\mathsf{ST}\}}_{issue}((U,U), \mathsf{rev}_U) = (U,U)$, and send $(\mathsf{issue\text{-}req}, \mathsf{sid}, L)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{rev}}$.
   (b) On input $(\mathsf{issue\text{-}req\text{-}ok}, \mathsf{sid})$ from $\mathcal{A}$, verify that there is no $U' \notin \mathcal{C}$ s.t. $U' \in \mathcal{U}_{\mathsf{rev}_U}$ (as rev_U is freshly sampled, it is unlikely for a sufficiently large N). Assign $\mathcal{U}_{\mathsf{rev}_U} \leftarrow \mathcal{U}_{\mathsf{rev}_U} \cup \{U\}$, $\mathcal{R}_{\mathsf{rev}_U} \leftarrow \mathcal{R}_{\mathsf{rev}_U} \cup \{U\}$.
   (c) Update $\mathcal{X} \leftarrow \mathcal{X} \setminus \{\mathsf{rev}_U\}$.
4. Let $\mathcal{A}$ send (I', IID) to U on behalf of AMS. Simulate U sending the token rev_U to I'.
5. Generate a fresh token rev_I using $\mathcal{F}_{\mathsf{rev}}$:
   (a) Upon receiving $(\mathsf{issue}, \mathsf{sid})$ from $\mathcal{A}$ as the input of the issuer in $\mathcal{F}_{\mathsf{rev}}$, sample $\mathsf{rev}_I \leftarrow\!\!\$\ \mathcal{X}$. Set $L = \mathcal{L}^{\{I\}}_{issue}(I, U, \mathsf{rev}_I) = (I, U, \mathsf{rev}_I)$, and send $(\mathsf{issue\text{-}req}, \mathsf{sid}, L)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{rev}}$.
   (b) On input $(\mathsf{issue\text{-}req\text{-}ok}, \mathsf{sid})$ from $\mathcal{A}$, verify that there is no $U' \notin \mathcal{C}$ s.t. $U' \in \mathcal{U}_{\mathsf{rev}_I}$ (as rev_I is freshly sampled, it is unlikely for a sufficiently large N). Assign $\mathcal{U}_{\mathsf{rev}_I} \leftarrow \mathcal{U}_{\mathsf{rev}_I} \cup \{U\}$, $\mathcal{R}_{\mathsf{rev}_I} \leftarrow \mathcal{R}_{\mathsf{rev}_I} \cup \{I\}$.
   (c) Update $\mathcal{X} \leftarrow \mathcal{X} \setminus \{\mathsf{rev}_I\}$.
   (d) Output $(\mathsf{issue\text{-}resp\text{-}r}, \mathsf{sid}, \mathsf{ok})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{rev}}$.
6. Generate $\mathsf{cID} \leftarrow (id_{\mathsf{cID}}, \mathsf{summary}(\mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID}))$.
7. Send $(\mathsf{issue\text{-}ok}, \mathsf{sid}, \mathsf{cID})$ to the $\mathcal{F}_{\mathsf{ideal}}$.
8. Upon receiving $(\mathsf{input}, \mathsf{IID}, \mathsf{issue}, \mathsf{sid}, \overrightarrow{\mathsf{atr}'})$ from $\mathcal{F}_{\mathsf{ideal}}$ (input of a malicious issuer from $\mathcal{Z}$), notify $\mathcal{A}$ that IID received the input $\overrightarrow{\mathsf{atr}'}$.
9. Upon receiving $(\mathsf{cred}, \sigma_{\mathsf{cred}})$ from corrupted AMS, check that cred indeed contains $\overrightarrow{\mathsf{aID}}$.

10. Verify issuer's signature σ_{cred} on cred using $\mathcal{F}_{\mathsf{sign}}^{\mathsf{IID}}$ of IID as $b \leftarrow \mathsf{v}(\sigma_{\mathsf{cred}}, \mathsf{cred})$. Continue if $b = 1$.
11. Simulate decryption of c_{rev} (contained in cred) by sending $(\mathsf{decrypt\text{-}init}, \mathsf{sid})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}$. Upon receiving $(\mathsf{decrypt\text{-}complete}, \mathsf{sid}, \mathsf{Dec})$, if there exists a pair (m, c_{rev}) in $\mathsf{T}_{\mathsf{enc}}^{\mathsf{uID}}$:
    - If m is not unique, set $m' = \perp$,
    - If m is unique, set $m' = m$.

    Otherwise, compute $m' := \mathsf{Dec}(c_{\mathsf{rev}})$. Add (m', c_{rev}) to $\mathsf{T}_{\mathsf{enc}}^{\mathsf{uID}}$.
    All cases are possible since credential has been generated by the adversary and ciphertexts may not have been queried though the encryption of $\mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}$.
12. Proceed if $m' = \mathsf{rev}$ for $\mathsf{rev} = (\mathsf{rev}_I, \mathsf{rev}_U)$.
13. Simulate decryption of c_0 (contained in cred) by sending $(\mathsf{decrypt\text{-}init}, \mathsf{sid})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}$.
    Upon receiving $(\mathsf{decrypt\text{-}complete}, \mathsf{sid}, \mathsf{Dec})$, if there exists a pair (m, c_0) in $\mathsf{T}_{\mathsf{enc}}^{\mathsf{uID}}$:
    - if m is not unique, set $m' = \perp$,
    - if m is unique, set $m' = m$.

    Otherwise, compute $m' := \mathsf{Dec}(c_0)$. Add (m', c_0) to $\mathsf{T}_{\mathsf{enc}}^{\mathsf{uID}}$.
14. Proceed if $m' = \mathsf{pk}_{\mathsf{uID}}$.
15. Simulate decryption of $n = |\overrightarrow{\mathsf{aID}}|$ ciphertexts from cred by sending $(\mathsf{decrypt\text{-}init}, \mathsf{sid})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}$. Upon receiving $(\mathsf{decrypt\text{-}complete}, \mathsf{sid}, \mathsf{Dec})$, if there exists a pair (atr_i, c_i) in $\mathsf{T}_{\mathsf{enc}}^{\mathsf{uID}}$:
    - If atr_i is not unique, set $m' = \perp$,
    - If atr_i is unique, set $m' = \mathsf{atr}_i$.

    Otherwise, compute $m' := \mathsf{Dec}(c_i)$. Add (m', c_i) to $\mathsf{T}_{\mathsf{enc}}^{\mathsf{uID}}$.
    All cases are possible since credential has been generated by the adversary and ciphertexts may not have been queried though the encryption of $\mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}$.
    Let $\mathsf{atr}_i = m'$. Proceed if $\mathsf{atr}_i \in \mathcal{M}_{\mathsf{aID}_i}$.
16. Send $(\mathsf{input}, \mathsf{IID}, \mathsf{issue}, \mathsf{sid}, \overrightarrow{\mathsf{atr}})$ to $\mathcal{F}_{\mathsf{ideal}}$. This will be the input of malicious issuer in $\mathcal{F}_{\mathsf{ideal}}$.
17. Compute $h_{\mathsf{cred}} \leftarrow H(h_{\mathsf{cred}}, (\mathsf{cID}, \mathsf{cred}, \sigma_{\mathsf{cred}}))$.
18. Wait for $(\mathsf{issue\text{-}atr}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{ideal}}$.
    Send message $(\mathsf{atr\text{-}ok}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ideal}}$ and save $(\mathsf{cID}, \mathsf{cred}, \sigma_{\mathsf{cred}}, \mathsf{rev})$ to $\mathsf{T}_{\mathsf{issue}}$.
    // Here we add to $\mathsf{T}_{\mathsf{issue}}$ the credential as viewed by the honest user.

Malicious AMS(Storage) and User: While AMS does provide additional protection for the issuer by authenticating the user, it still cannot prevent the user from malicious behaviour. Hence, we do not consider separately the case where only the user is corrupted, as the simulation would be essentially the same.

1. Simulate I receiving msg and σ from $\mathcal{A}$. Extract uID, IID, $\overrightarrow{\mathsf{aID}}$ from msg. Verify σ and msg using $\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}$.
2. Send $(\mathsf{input}, \mathsf{uID}, \mathsf{issue}, \mathsf{sid}, (\overrightarrow{\mathsf{aID}}, \mathsf{IID}))$ to $\mathcal{F}_{\mathsf{ideal}}$.
3. Receive $(\mathsf{issue\text{-}req}, \mathsf{sid}, \mathsf{uID}, \mathsf{IID}, \overrightarrow{\mathsf{aID}})$ from $\mathcal{F}_{\mathsf{ideal}}$.
4. Let $\mathsf{cID} \leftarrow (id_{\mathsf{cID}}, \mathsf{summary}(\mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID}))$. Verify freshness of cID (and hence id_{cID}).

5. Simulate I receiving the revocation token rev_U from $\mathcal{A}$.
6. Generate a fresh token rev_I using $\mathcal{F}_{\mathsf{rev}}$:
   (a) Upon receiving $(\mathsf{issue}, \mathsf{sid})$ from $\mathcal{A}$ as the input of the issuer in $\mathcal{F}_{\mathsf{rev}}$, sample $\mathsf{rev}_I \leftarrow\!\!\$\ \mathcal{X}$. Set $L = \mathcal{L}_{issue}^{\{U\}}(I, U, \mathsf{rev}_I) = (I, U, \mathsf{rev}_I)$, and send $(\mathsf{issue\text{-}req}, \mathsf{sid}, L)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{rev}}$.
   (b) On input $(\mathsf{issue\text{-}req\text{-}ok}, \mathsf{sid}, \overline{\mathsf{rev}_I})$ from $\mathcal{A}$, take $\mathsf{rev}_I \leftarrow \overline{\mathsf{rev}_I}$. Verify that there is no $U' \notin \mathcal{C}$ s.t. $U' \in \mathcal{U}_{\mathsf{rev}_I}$ (as rev_I is freshly sampled, it is unlikely for a sufficiently large N). Assign $\mathcal{U}_{\mathsf{rev}_I} \leftarrow \mathcal{U}_{\mathsf{rev}_I} \cup \{U\}, \mathcal{R}_{\mathsf{rev}_I} \leftarrow \mathcal{R}_{\mathsf{rev}_I} \cup \{I\}$.
   (c) Update $\mathcal{X} \leftarrow \mathcal{X} \setminus \{\mathsf{rev}_I\}$.
   (d) Output $(\mathsf{issue\text{-}resp\text{-}r}, \mathsf{sid}, \mathsf{ok})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{rev}}$.
7. Send $(\mathsf{issue\text{-}ok}, \mathsf{sid}, \mathsf{cID})$ to the $\mathcal{F}_{\mathsf{ideal}}$ (start interaction with $\mathcal{Z}$).
8. Wait for $(\mathsf{issue\text{-}atr}, \mathsf{sid}, \overrightarrow{\mathsf{atr}})$ from $\mathcal{F}_{\mathsf{ideal}}$ (I received attribute values from $\mathcal{Z}$).
9. Send encryption request $(\mathsf{Encrypt}, \mathsf{sid})$ to $\mathcal{A}$ through $\mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}$. Wait for response $(\mathsf{Encrypt}, \mathsf{sid}, c_{\mathsf{rev}})$ from $\mathcal{A}$ and store the pair $(\mathsf{rev}, c_{\mathsf{rev}})$ in table $\mathsf{T}_{\mathsf{enc}}^{\mathsf{uID}}$.
10. Set $\mathsf{atr}_0 \leftarrow \mathsf{pk}_{\mathsf{uID}}$ // with $\mathsf{pk}_{\mathsf{uID}}$ coming from verification Ver defined during the initialisation phase
11. Simulate encryptions by honest issuer, sending $n = |\overrightarrow{\mathsf{aID}}| + 1$ encryption requests $(\mathsf{Encrypt}, \mathsf{sid})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}$. Wait for response $(\mathsf{Encrypt}, \mathsf{sid}, c_i)$ from $\mathcal{A}$ and store the pair (atr_i, c_i) in table $\mathsf{T}_{\mathsf{enc}}^{\mathsf{uID}}$.
12. Set $\mathsf{cred} = \langle(\mathsf{aID}_0, c_0), (\mathsf{aID}_1, c_1), \ldots, (\mathsf{aID}_n, c_n), (\mathsf{aID}_{\mathsf{rev}}, c_{\mathsf{rev}})\rangle$.
13. Generate signature on the credential cred as $\sigma_{\mathsf{cred}} = \mathsf{s}(\mathsf{cred})$ and verify that $\mathsf{v}(\sigma_{\mathsf{cred}}, \mathsf{cred}) = 1$ using $\mathcal{F}_{\mathsf{sign}}^{\mathsf{IID}}$. Save $(\sigma_{\mathsf{cred}}, \mathsf{cred})$ to table $\mathsf{T}_{\mathsf{lsign}}^{\mathsf{IID}}$.
14. Send $(\mathsf{cred}, \sigma_{\mathsf{cred}})$ to the user.
15. Send message $(\mathsf{atr\text{-}ok}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ideal}}$ and save $(\mathsf{cID}, \mathsf{cred}, \sigma_{\mathsf{cred}}, \mathsf{rev})$ to $\mathsf{T}_{\mathsf{issue}}$. // Here we add to $\mathsf{T}_{\mathsf{issue}}$ the credential as viewed by the honest issuer.

C.5 Get all credential identifiers

Malicious AMS (storage):

1. On message $(\mathsf{getcids\text{-}req}, \mathsf{sid}, \mathsf{uID})$ from $\mathcal{F}_{\mathsf{ideal}}$, establish $(\mathsf{tag}_{cids}, \mathsf{sid}, \mathsf{uID})$ connection with the server. Set $msg = (\mathsf{tag}_{cids}, \mathsf{uID})$ and wait for input $(\mathsf{sign\text{-}server}, \mathsf{sid})$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}$. Send $(\mathsf{sign\text{-}init}, \mathsf{sid})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}$. Upon receiving $(\mathsf{signature}, \mathsf{sid}, \sigma)$ from $\mathcal{A}$, check whether $(msg, \sigma, 0)$ is already stored. In this case, restart signing. Otherwise, store $(msg, \sigma, 1)$ in table $\mathsf{T}_{\mathsf{sign}}^{\mathsf{uID}}$ and send $(\mathsf{sign\text{-}success}, \mathsf{sid})$ to the $\mathcal{A}$. Otherwise, stop execution.
2. Send $msg = (\mathsf{tag}_{cids}, \mathsf{uID})$ and σ to $\mathcal{A}$.
3. Wait for input $\overrightarrow{\mathsf{cID}}, \overrightarrow{\mathsf{cred}}$ from $\mathcal{A}$ on behalf of corrupted server.
4. Let $\overrightarrow{\mathsf{cID}}'$ be all the credential identifiers stored in $\mathsf{T}_{\mathsf{issue}}$ so far, and $\overrightarrow{\mathsf{cred}}'$ the corresponding credentials. If $\overrightarrow{\mathsf{cID}} = \overrightarrow{\mathsf{cID}}$ and $\overrightarrow{\mathsf{cred}} = \overrightarrow{\mathsf{cred}}'$, send $(\mathsf{getcids\text{-}ok}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ideal}}$. Note that the digest check by the user ensures that the check passes in the real protocol iff success is reported by $\mathcal{F}_{\mathsf{ideal}}$ in the simulation.

Malicious User

1. Let $\mathsf{req}(\mathsf{getcids}, \mathsf{uID}, \mathsf{sid}) = (msg, \sigma)$ result from a successful malicious user authentication as in Sec. C.3. Verify that $msg = (\mathsf{tag}_{cids}, \mathsf{uID})$.
2. Send $(\mathsf{input}, \mathsf{uID}, \mathsf{getcids}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ideal}}$.
3. Wait for $(\mathsf{getcids\text{-}req}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{ideal}}$. Let $\overrightarrow{\mathsf{cID}}$ be all the credential identifiers stored in $\mathsf{T}_{\mathsf{issue}}$ so far, and $\overrightarrow{\mathsf{cred}}$ the corresponding credentials. Send $\overrightarrow{\mathsf{cID}}$ and $\overrightarrow{\mathsf{cred}}$ to $\mathcal{A}$ on behalf of AMS.
4. Send $(\mathsf{getcids\text{-}ok}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ideal}}$.

C.6 Revocation by issuer

To simulate revocation by an honest revoker, we observe that $\mathcal{F}_{\mathsf{ideal}}$ should leak cID if (1) it has been presented to some corrupted RP before, or (2) if it belongs to an honest user. Otherwise, we take a random token $\mathsf{rev} \leftarrow\!\!\$\ \mathcal{X}$, as in this case rev is not leaked to the adversary. The simulator still needs to simulate a successful revocation, even though the internal state of $\mathcal{F}_{\mathsf{rev}}$ will not change.

Later, upon full user corruption or presentation to a corrupted RP, it may turn out that a particular credential cID should have been revoked. The simulator will then silently add the corresponding token rev to the set X^-.

Malicious AMS server (storage): $\mathcal{F}_{\mathsf{ideal}}$ does not leak cID to $\mathcal{S}$ if both the user and the issuer are honest. In this case either $(\mathsf{cID}, \mathsf{rev})$ has already been presented to some corrupted RP (and the simulator can reuse it), or $\mathcal{A}$ has not seen $(\mathsf{cID}, \mathsf{rev})$ pair before and we can run revocation for the randomly chosen revocation token.

1. On message $(\mathsf{revoke\text{-}req}, \mathsf{sid}, P = \mathsf{IID}, \mathsf{uID})$ from $\mathcal{F}_{\mathsf{ideal}}$, establish $(\mathsf{tag}_{rev}, \mathsf{sid}, \mathsf{IID})$ connection with the server. Simulate encryption using $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$ with a ciphertext c_{cID}. Generate a consent signature on behalf of IID using $\mathcal{F}^{\mathsf{IID}}_{\mathsf{sign}}$.
2. Come up with token rev to revoke as follows:
   - If the actual cID has already been presented to some corrupted RP before, or it belongs to a corrupted user, then cID should have come from $\mathcal{F}_{\mathsf{ideal}}$. The token rev corresponds to entry from $\mathsf{T}_{\mathsf{issue}}$ with cID.
   - Otherwise, take random $\mathsf{rev} \leftarrow\!\!\$\ \mathcal{X}$. // As rev is freshly sampled, collision with any issued token (in the past or in the future) is unlikely for a sufficiently large total number of tokens N
3. On input $(\mathsf{revoke}, \mathsf{sid})$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{rev}}$ on behalf of AMS:
   - if rev has been presented to a corrupted RP before, set $L = \mathcal{L}^{\{\mathsf{ST}, RP\}}_{revoke}(I, \mathsf{rev}) = (I, b, \mathsf{rev})$;
   - if rev belongs to a corrupted user, set $L = \mathcal{L}^{\{\mathsf{ST}, \mathcal{U}_{\mathsf{rev}}\}}_{revoke}(I, \mathsf{rev}) = (I, b, \mathsf{rev})$;
   - otherwise, set $L = \mathcal{L}^{\{\mathsf{ST}\}}_{revoke}(I, \mathsf{rev}) = (I, b)$;

   where $b = (\mathsf{rev} \notin X^-) \wedge (I \in \mathcal{R}_{\mathsf{rev}}) = \mathsf{true}$, as in the real protocol an honest revoker only revokes a legitimately issued token that it has not revoked yet. Send $(\mathsf{revoke\text{-}req}, \mathsf{sid}, L)$ to $\mathcal{A}$.
   // At this point, $\mathcal{A}$ learns rev only if it corresponds to the true cID that is being revoked.
4. Wait for $(\mathsf{revoke\text{-}req\text{-}ok}, \mathsf{sid})$ from $\mathcal{A}$.

5. Simulate successful check $I \in \mathcal{R}_{\mathsf{rev}}$ without modifying the internal state of $\mathcal{F}_{\mathsf{rev}}$.
6. Output $(\mathsf{revoke\text{-}resp\text{-}s}, \mathsf{sid}, \mathsf{ok})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{rev}}$.
7. Output $(\mathsf{revoke\text{-}ok}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ideal}}$.
8. If cID belongs to a corrupted user, there may be several cID'-s that correspond to rev. Look for all records $(\mathsf{cID}', \_, \_, \mathsf{rev})$ in $\mathsf{T}_{\mathsf{issue}}$, and revoke all cID' on behalf uID (extracted from cID') using silent revocation interface.
   // Since this case is only possible for $c_{\mathsf{U}} \geq 2$, all authentications of uID by $\mathcal{F}_{\mathsf{ideal}}$ succeed.

Malicious Issuer: Since the malicious Issuer initiates revocation process, the adversary chooses the corresponding revocation token rev.

1. Wait for $\mathcal{A}$ to establish a connection $(\mathsf{tag}_{rev}, \mathsf{sid}, \mathsf{IID})$ with AMS on behalf of I.
2. Wait for a message $msg = (\mathsf{tag}_{rev}, \mathsf{IID}, c_{\mathsf{cID}})$ and signature σ_{req} from $\mathcal{A}$ on behalf of I. On behalf of AMS, verify the signature using $\mathcal{F}_{\mathsf{sign}}^{\mathsf{IID}}$. // To ensure that this step succeeds only for $\mathsf{IID} \in \mathcal{C}$ even in the case of replay attacks, we assume additional authentication of IID.
3. On input $(\mathsf{revoke}, \mathsf{sid}, x)$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{rev}}$ on behalf of I, set $L = \mathcal{L}_{revoke}^{\{I\}}(I, x) = (I, b)$, where $b = (x \notin X^-) \wedge (I \in \mathcal{R}_x)$ is easy to simulate, as $\mathcal{S}$ knows whether the corrupted IID is a revoker of x, and whether it has already revoked x. Send $(\mathsf{revoke\text{-}req}, \mathsf{sid}, L)$ to $\mathcal{A}$.
4. Wait for $(\mathsf{revoke\text{-}req\text{-}ok}, \mathsf{sid})$ from $\mathcal{A}$.
5. If $I \in \mathcal{R}_x$ or $\exists R' \in \mathcal{R}_x \cap \mathcal{C}$, add x to X^-.
6. Output $(\mathsf{revoke\text{-}resp\text{-}r}, \mathsf{sid}, \mathsf{ok})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{rev}}$.
7. Let cID be such that there is an entry $(\mathsf{cID}, \_, \_, (x, \_))$ or $(\mathsf{cID}, \_, \_, (\_, x))$ in $\mathsf{T}_{\mathsf{issue}}$. If there is none, set $\mathsf{cID} \leftarrow \perp$.
8. Send $(\mathsf{input}, \mathsf{IID}, \mathsf{revoke}, \mathsf{sid}, \mathsf{cID})$ to $\mathcal{F}_{\mathsf{ideal}}$.
9. If $\mathsf{cID} = \perp$, then $\mathcal{F}_{\mathsf{ideal}}$ outputs $(\mathsf{revoke\text{-}resp}, \mathsf{sid}, \perp)$ to IID.
   Otherwise, a message $(\mathsf{revoke\text{-}req}, \mathsf{sid}, \ldots)$ comes from $\mathcal{F}_{\mathsf{ideal}}$. Respond with $(\mathsf{revoke\text{-}ok}, \mathsf{sid})$ to ensure that a record $(\mathsf{uID}, \mathsf{cID}, \mathsf{IID})$ gets into T_r.
10. For a corrupted user, there may be several cID'-s that correspond to x. Look for all records $(\mathsf{cID}', \_, \_, (x, \_))$ and $(\mathsf{cID}', \_, \_, (\_, x))$ in $\mathsf{T}_{\mathsf{issue}}$, and revoke all $\mathsf{cID}' \neq \mathsf{cID}$ on behalf uID (extracted from cID') using silent revocation interface.
    // Since this case is only possible for $c_{\mathsf{U}} \geq 2$, all authentications of uID by $\mathcal{F}_{\mathsf{ideal}}$ succeed.

C.7 Revocation by user

While the general idea behind token handling is the same, the simulation is more complicated than issuer revocation, as it involves threshold decryption. However, it still does not leak decrypted c_{rev} (and hence cID) to the adversary due to its blinding property. Also, revocation requires a consent signature which needs to be generated using threshold signing $\mathcal{F}_{\mathsf{tsign}}$.

Malicious AMS server (storage):

1. On message $(\mathsf{revoke\text{-}req}, \mathsf{sid}, P = \mathsf{uID})$ from $\mathcal{F}_{\mathsf{ideal}}$, establish $(\mathsf{tag}_{rev}, \mathsf{sid}, \mathsf{uID})$ connection with the server. Simulate encryption using $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$ with a ciphertext c_{cID}.
2. Set $msg = (\mathsf{tag}_{rev}, \mathsf{uID}, c_{\mathsf{cID}})$ and wait for input $(\mathsf{sign\text{-}server}, \mathsf{sid})$ from $\mathcal{A}$. Send $(\mathsf{sign\text{-}init}, \mathsf{sid})$ to $\mathcal{A}$ on behalf of $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}$. Upon receiving $(\mathsf{signature}, \mathsf{sid}, \sigma)$ from $\mathcal{A}$, check whether $(msg, \sigma, 0)$ is already stored. In this case, restart signing. Otherwise, store $(msg, \sigma, 1)$ in table $\mathsf{T}^{\mathsf{uID}}_{\mathsf{sign}}$ and send $(\mathsf{sign\text{-}success}, \mathsf{sid})$ to the $\mathcal{A}$. Otherwise, stop execution.
3. Send $msg = (\mathsf{tag}_{rev}, \mathsf{uID}, c_{\mathsf{cID}})$ and σ to $\mathcal{A}$.
4. Come up with token rev to revoke as follows:
   - If the actual cID has already been presented to some corrupted RP before, then cID should have come from $\mathcal{F}_{\mathsf{ideal}}$. The token rev corresponds to entry from $\mathsf{T}_{\mathsf{issue}}$ with cID.
   - Otherwise, take random $\mathsf{rev} \leftarrow\!\!\$\ \mathcal{X}$. // As rev is freshly sampled, collision with any issued token (in the past or in the future) is unlikely for a sufficiently large total number of tokens N
5. Simulate decryption of c_{rev} by sending $(\mathsf{decrypt\text{-}init}, \mathsf{sid})$ to $\mathcal{A}$ on behalf of $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$.
   Upon receiving $(\mathsf{decrypt\text{-}complete}, \mathsf{sid}, \mathsf{Dec})$, if there exists a pair (m, c_{rev}) in $\mathsf{T}^{\mathsf{uID}}_{\mathsf{enc}}$:
   - if m is not unique, set $m' = \perp$,
   - if m is unique, set $m' = m$.

   Otherwise, compute $m' := \mathsf{Dec}(c_{\mathsf{rev}})$. Add (m', c_{rev}) to $\mathsf{T}^{\mathsf{uID}}_{\mathsf{enc}}$.
   // As simulator has just computed c_{rev} itself, we have $m = m' = \mathsf{rev}$.
6. On input $(\mathsf{revoke}, \mathsf{sid})$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{rev}}$ on behalf of AMS:
   - if rev has been presented to a corrupted RP before, set $L = \mathcal{L}^{\{\mathsf{ST}, RP\}}_{revoke}(U, \mathsf{rev}) = (U, b, \mathsf{rev})$;
   - otherwise, set $L = \mathcal{L}^{\{\mathsf{ST}\}}_{revoke}(U, \mathsf{rev}) = (U, b)$;

   where $b = (\mathsf{rev} \notin X^-) \wedge (U \in \mathcal{R}_{\mathsf{rev}}) = \mathsf{true}$, as an honest revoker only revokes a legitimately issued token that it has not revoked yet. Send $(\mathsf{revoke\text{-}req}, \mathsf{sid}, L)$ to $\mathcal{A}$.
   // At this point, $\mathcal{A}$ learns rev only if it corresponds to the true cID that is being revoked.
7. Wait for $(\mathsf{revoke\text{-}req\text{-}ok}, \mathsf{sid})$ from $\mathcal{A}$.
8. Simulate successful check $U \in \mathcal{R}_{\mathsf{rev}}$ without modifying the internal state of $\mathcal{F}_{\mathsf{rev}}$.
9. Output $(\mathsf{revoke\text{-}resp\text{-}s}, \mathsf{sid}, \mathsf{ok})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{rev}}$.
10. Output $(\mathsf{revoke\text{-}ok}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ideal}}$.
11. If cID belongs to a corrupted user, there may be several cID'-s that correspond to rev. Look for all records $(\mathsf{cID}', \_, \_, \mathsf{rev})$ in $\mathsf{T}_{\mathsf{issue}}$, and revoke all cID' on behalf uID (extracted from cID') using silent revocation interface.
    // Since this case is only possible for $c_{\mathsf{U}} \geq 2$, all authentications of uID by $\mathcal{F}_{\mathsf{ideal}}$ succeed.

Malicious User:

1. Let $\mathsf{req}(\mathsf{revoke}, \mathsf{uID}, \mathsf{sid}) = (msg, \sigma)$ result from a successful malicious user authentication as in Sec. C.3. Verify that $msg = (\mathsf{tag}_{rev}, \mathsf{uID}, \ldots)$.
2. Interact with the corrupted user in the threshold decryption protocol through $\mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}$ as follows:
   (a) Wait for input $(\mathsf{decrypt}, \mathsf{sid}, c)$ from $\mathcal{A}$ on behalf of corrupted U.
   (b) Send $(\mathsf{decrypt\text{-}init}, \mathsf{sid}, c)$ to $\mathcal{A}$.
   (c) Upon receiving $(\mathsf{decrypt\text{-}complete}, \mathsf{sid})$ from $\mathcal{A}$, set $y_2 \leftarrow \top$. Upon receiving $(\mathsf{decrypt\text{-}fail}, \mathsf{sid})$ from adversary, set $y_2 \leftarrow \bot$. Here y_2 is the server output (success of fail).
   (d) Wait for $(\mathsf{Output}, 1, y_1)$ from $\mathcal{A}$ through $\mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}$ and send $(\mathsf{Decrypted}, \mathsf{sid}, y_1)$ to $\mathcal{A}$. Here y_1 is the user output chosen by $\mathcal{A}$.
3. On input $(\mathsf{revoke}, \mathsf{sid}, x)$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{rev}}$ on behalf of U, set $L = \mathcal{L}_{revoke}^{\{U\}}(U, x) = (U, b)$, where $b = (x \notin X^-) \wedge (I \in \mathcal{R}_x)$ is easy to simulate, as $\mathcal{S}$ knows whether the corrupted user is a revoker of x, and whether it has already revoked x. Send $(\mathsf{revoke\text{-}req}, \mathsf{sid}, L)$ to $\mathcal{A}$.
4. Wait for $(\mathsf{revoke\text{-}req\text{-}ok}, \mathsf{sid})$ from $\mathcal{A}$.
5. If $U \in \mathcal{R}_x$ or $\exists R' \in \mathcal{R}_x \cap \mathcal{C}$, add x to X^-.
6. Output $(\mathsf{revoke\text{-}resp\text{-}r}, \mathsf{sid}, \mathsf{ok})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{rev}}$.
7. Let cID be such that there is an entry $(\mathsf{cID}, \_, \_, (x, \_))$ or $(\mathsf{cID}, \_, \_, (\_, x))$ in $\mathsf{T}_{\mathsf{issue}}$. If there is none, set $\mathsf{cID} \leftarrow \bot$.
8. Send $(\mathsf{input}, \mathsf{uID}, \mathsf{revoke}, \mathsf{sid}, \mathsf{cID})$ to $\mathcal{F}_{\mathsf{ideal}}$.
9. If $\mathsf{cID}' = \bot$, then $\mathcal{F}_{\mathsf{ideal}}$ outputs $(\mathsf{revoke\text{-}resp}, \mathsf{sid}, \bot)$ to uID.
   Otherwise, a message $(\mathsf{revoke\text{-}req}, \mathsf{sid}, \ldots)$ comes from $\mathcal{F}_{\mathsf{ideal}}$. Respond with $(\mathsf{revoke\text{-}ok}, \mathsf{sid})$ to ensure that a record $(\mathsf{uID}', \mathsf{cID}, \mathsf{uID})$ gets into T_r.
10. For a corrupted user, there may be several cID'-s that correspond to x. Look for all records $(\mathsf{cID}', \_, \_, (x, \_))$ and $(\mathsf{cID}', \_, \_, (\_, x))$ in $\mathsf{T}_{\mathsf{issue}}$, and revoke all $\mathsf{cID}' \neq \mathsf{cID}$ on behalf uID (extracted from cID') using silent revocation interface.
    // Since this case is only possible for $c_{\mathsf{U}} \geq 2$, all authentications of uID by $\mathcal{F}_{\mathsf{ideal}}$ succeed.

C.8 Present

Malicious AMS (storage) (and Issuer):

Here, we are simulating communication that is expected by the adversary through the $\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}, \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}, \mathcal{F}_{\mathsf{rev}}$ functionalities with a randomly chosen credential, as this step does not leak any information about the cID to $\mathcal{A}$.

1. On message $(\mathsf{present\text{-}req}, \mathsf{sid}, l_{\mathsf{atr}} = |\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}|)$ from $\mathcal{F}_{\mathsf{ideal}}$, establish $(\mathsf{tag}_{pres}, \mathsf{sid}, \mathsf{uID})$ connection with the server. Simulate encryption using $\mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}$ with a ciphertext c_{cID}. Set $msg = (\mathsf{tag}_{pres}, \mathsf{uID}, n, c_{\mathsf{cID}})$ and wait for input $(\mathsf{sign\text{-}server}, \mathsf{sid})$ from $\mathcal{A}$. Send $(\mathsf{sign\text{-}init}, \mathsf{sid})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{tsign}}$. Upon receiving $(\mathsf{signature}, \mathsf{sid}, \sigma)$ from $\mathcal{A}$, check whether $(msg, \sigma, 0)$ is already stored. In this case, restart signing. Otherwise, store $(msg, \sigma, 1)$ in table $\mathsf{T}_{\mathsf{sign}}^{\mathsf{uID}}$ and send $(\mathsf{sign\text{-}success}, \mathsf{sid})$ to the $\mathcal{A}$. Otherwise, stop execution.
2. Send $msg = (\mathsf{tag}_{pres}, \mathsf{uID}, n, c_{\mathsf{cID}})$ and σ to $\mathcal{A}$.

3. Simulate retrieval of credential data.
   - Select any entry $(\mathsf{cID}, \mathsf{cred}, \sigma_{\mathsf{cred}}, \mathsf{rev})$ from $\mathsf{T}_{\mathsf{issue}}$, where cID contains uID. // Since presentation request has come from $\mathcal{F}_{\mathsf{ideal}}$, a suitable credential must exist.
   - Parse $\mathsf{cred} = \langle(\mathsf{aID}_0, c_0), (\mathsf{aID}_1, c_1), \ldots, (\mathsf{aID}_n, c_n), (\mathsf{aID}_{\mathsf{rev}}, c_{\mathsf{rev}})\rangle$.
4. Simulate decryption of c_{rev} by sending $(\mathsf{decrypt\text{-}init}, \mathsf{sid})$ to adversary on behalf of $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$. Upon receiving $(\mathsf{decrypt\text{-}complete}, \mathsf{sid}, \mathsf{Dec})$, look for a pair $(\mathsf{rev}, c_{\mathsf{rev}})$ in $\mathsf{T}^{\mathsf{uID}}_{\mathsf{enc}}$. If rev is not unique, set $\mathsf{rev} \leftarrow \perp$. // $\mathsf{rev} \neq \perp$ always exists, since c_{rev} was taken from $\mathsf{T}_{\mathsf{issue}}$, where it has been previously computed using $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$
5. Simulate revocation notification for $x \in \{\mathsf{rev}_I, \mathsf{rev}_U\}$ where $(\mathsf{rev}_I, \mathsf{rev}_U) \leftarrow \mathsf{rev}$.
   - Upon receiving $(\mathsf{unotify}, \mathsf{sid})$ from the $\mathcal{A}$ as input to $\mathcal{F}_{\mathsf{rev}}$. Set $L = \mathcal{L}^{\{S,I\}}_{unotify}(U, x) = (U, b')$, where $b' = (U \in \mathcal{U}_x) = \mathsf{true}$, as the honest user will only make this query for x issued to it. Send $(\mathsf{unotify\text{-}req}, \mathsf{sid}, L)$ to $\mathcal{A}$.
   - Wait for $(\mathsf{unotify\text{-}req\text{-}ok}, \mathsf{sid})$ from $\mathcal{A}$.
   - Let $b = \mathsf{true}$ iff $x \notin X^-$ and $U \in \mathcal{U}_x$.
   - Send $(\mathsf{unotify\text{-}resp\text{-}s}, \mathsf{sid}, \mathsf{ok})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{rev}}$.
6. Simulate decryption of l_{atr} attributes from cred by sending $(\mathsf{decrypt\text{-}init}, \mathsf{sid})$ to adversary on behalf of $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$. Upon receiving $(\mathsf{decrypt\text{-}complete}, \mathsf{sid}, \mathsf{Dec})$, look for a pair (atr_i, c_i) in $\mathsf{T}^{\mathsf{uID}}_{\mathsf{enc}}$ If atr_i is not unique, set $\mathsf{atr}_i = \perp$.
   // Here $\mathsf{atr}_i \neq \perp$ always exists since cred has been taken from $\mathsf{T}_{\mathsf{issue}}$, where it has been previously computed using $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$.
   // Since our decryption protocol is blind, adversary will not notice that we are decrypting some random attribute, but we do l_{atr} decryption queries as adversary expects to see this amount of queries.
7. Send $(\mathsf{present\text{-}req}, \mathsf{sid}, \mathsf{ok})$ to $\mathcal{F}_{\mathsf{ideal}}$.

Malicious User: In this case, we are simulating presentation of attributes requested by the malicious user. Malicious user is able to proceed with the presentation request only if the user is fully corrupted ($c_{\mathsf{U}} = 2$ or $c_{\mathsf{U}} = 3$).

1. Let $\mathsf{req}(\mathsf{present}, \mathsf{uID}, \mathsf{sid}) = (msg, \sigma)$ result from a successful malicious user authentication as in Sec. C.3. Verify that $msg = (\mathsf{tag}_{pres}, \mathsf{uID}, n, \ldots)$.
2. Let n be the reported length of $\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}$ which is a part of msg that the server received. Interact with the corrupted user in $n + 2$ decryptions using the threshold decryption protocol through $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$ as follows:
   (a) Wait for input $(\mathsf{decrypt}, \mathsf{sid}, c_i)$ from $\mathcal{A}$ on behalf of corrupted U.
   (b) Send $(\mathsf{decrypt\text{-}init}, \mathsf{sid}, c_i)$ to $\mathcal{A}$.
   (c) Upon receiving $(\mathsf{decrypt\text{-}complete}, \mathsf{sid})$ from $\mathcal{A}$, set $y_2 \leftarrow \top$. Upon receiving $(\mathsf{decrypt\text{-}fail}, \mathsf{sid})$ from adversary, set $y_2 \leftarrow \perp$. Here y_2 is the server output (success of fail).
   (d) Wait for $(\mathsf{Output}, 1, y_1)$ from $\mathcal{A}$ through $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$ and send $(\mathsf{Decrypted}, \mathsf{sid}, y_1)$ to $\mathcal{A}$. Here y_1 is the user output chosen by $\mathcal{A}$.
3. Interact with the corrupted user in two notifications as follows:

- Upon receiving $(\mathsf{unotify}, \mathsf{sid}, x)$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{rev}}$ on behalf of U, set $L = \mathcal{L}^{\{U\}}_{unotify}(U, x) = (U, b')$ where $b' = \mathsf{true}$ iff $U \in \mathcal{U}_x$. Send $(\mathsf{unotify\text{-}req}, \mathsf{sid}, L)$ to the $\mathcal{A}$.
- Wait for $(\mathsf{unotify\text{-}req\text{-}ok}, \mathsf{sid})$ from $\mathcal{A}$.
- Let $b = \mathsf{true}$ iff $x \notin X^-$ and $U \in \mathcal{U}_x$.
- Output $(\mathsf{unotify\text{-}resp\text{-}u}, \mathsf{sid}, b)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{rev}}$.

4. Let the adversary choose the output. Based on adversary data, we in general cannot come up with the "actual" inputs cID and $\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}$ for a valid presentation. Hence, send $(\mathsf{inputs}, \mathsf{uID}, \mathsf{present}, \mathsf{sid}, \bot)$ to $\mathcal{F}_{\mathsf{ideal}}$, causing it to fail, and choose the output pres for corrupted U. The outcome of $\mathcal{F}_{\mathsf{ideal}}$ does not matter, as it has no impact neither on the inner state of $\mathcal{F}_{\mathsf{ideal}}$ nor $\mathcal{Z}$.

C.9 Verification

Revocation functionality $\mathcal{F}_{\mathsf{rev}}$ assumes that the verifier party V has called it at least once using the subroutine $\mathsf{vnotify}$ of $\mathcal{F}_{\mathsf{rev}}$. It needs to remember the set X^-_V of the tokens revoked at that point. Similarly to $\mathcal{F}_{\mathsf{rev}}$, the simulator defines X^-_V as the current state of X^- that it maintains locally.

Malicious AMS: Since the RP is honest, the verification is fully simulated by the simulator. Therefore, we can simulate generating signature for a random challenge and random credential. Due to blindness property of the signature scheme, AMS server will not distinguish the executions.

1. Upon receiving $(\mathsf{verify\text{-}req}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{ideal}}$, sample $\mathsf{ch} \leftarrow \mathcal{CH}$ and send $(\mathsf{verify\text{-}req\text{-}ok}, \mathsf{sid}, \mathsf{ch})$ to $\mathcal{F}_{\mathsf{ideal}}$ to confirm proceeding.
2. Take any entry from $\mathsf{T}_{\mathsf{issue}}$ with cID for which a successful presentation has been constructed before. Take corresponding cred, σ_{cred}, rev, IID, and construct $\mathsf{pres} = (\overrightarrow{\mathsf{aID}}, \overrightarrow{\mathsf{atr}}, \mathsf{IID})$, which $\mathcal{A}$ will not see anyway (including the number of disclosed attributes).
   // If we have reached this point, such a credential does exist.
3. Establish $(\mathsf{tag}_{verify}, \mathsf{sid}, \mathsf{uID})$ connection with the server on behalf of user.
4. Set $msg = (\mathsf{pres}, \mathsf{ch}', \mathsf{cred}, \sigma_{\mathsf{cred}})$ and wait for input $(\mathsf{sign\text{-}server}, \mathsf{sid})$ from adversary. Send $(\mathsf{sign\text{-}init}, \mathsf{sid})$ to adversary on behalf of $\mathcal{F}_{\mathsf{tsign}}$. Upon receiving $(\mathsf{signature}, \mathsf{sid}, \sigma_{\mathsf{pres}})$ from adversary, check whether $(msg, \sigma_{\mathsf{pres}}, 0)$ is already stored. In this case, restart signing. Otherwise, store $(msg, \sigma_{\mathsf{pres}}, 1)$ in table $\mathsf{T}^{\mathsf{uID}}_{\mathsf{sign}}$.
5. Simulate sending $\langle \mathsf{uID}, \mathsf{cred}, \sigma_{\mathsf{cred}}, \mathsf{pres}, \mathsf{rev}_I, \mathsf{rev}_U, \sigma_{\mathsf{pres}} \rangle$ from the user to RP.
6. Simulate verifying presentation by RP.
7. Send $(\mathsf{verify\text{-}req\text{-}ok}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ideal}}$.

Malicious RP (and AMS):

1. On message $(\mathsf{verify\text{-}req}, \mathsf{sid}, \mathsf{uID}, \mathsf{cID}, \mathsf{pres}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}})$ from $\mathcal{F}_{\mathsf{ideal}}$, parse $(\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}, \mathsf{IID}) \leftarrow \mathsf{pres}$. Find entry from $\mathsf{T}_{\mathsf{issue}}$ with cID and take corresponding cred, σ_{cred}, rev.
2. Establish a connection $(\mathsf{tag}_{verify}, \mathsf{sid}, \mathsf{uID})$ with RP.
3. Receive ch that $\mathcal{A}$ sends on behalf of RP.

4. For all atr_i in $\overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}$, update the entry (atr_i, c_i) in $\mathsf{T}^{\mathsf{uID}}_{\mathsf{enc}}$, where c_i are taken from cred. // This is needed since c_i have been encryptions of potentially garbage values atr'_i during the issuing phase. It is safe to update atr'_i that have not been used in any simulations yet, and if atr'_i has already been simulated to $\mathcal{A}$, then $\mathsf{atr}_i = \mathsf{atr}'_i$, as the only place where atr_i are opened are the verification routine (here) or the adaptive user corruption (described in App. C.10), where atr_i are taken from T_a of $\mathcal{F}_{\mathsf{ideal}}$.
5. Establish $(\mathsf{tag}_{verify}, \mathsf{sid}, \mathsf{uID})$ connection with the server on behalf of user.
6. Set $msg = (\mathsf{pres}, \mathsf{ch}, \mathsf{cred}, \sigma_{\mathsf{cred}})$ and construct presentation signature:
   - if storage is corrupted, wait for input $(\mathsf{sign\text{-}server}, \mathsf{sid})$ from adversary and proceed.
   - if storage is not corrupted, proceed.

   Send $(\mathsf{sign\text{-}init}, \mathsf{sid})$ to adversary on behalf of $\mathcal{F}_{\mathsf{tsign}}$. Upon receiving $(\mathsf{signature}, \mathsf{sid}, \sigma_{\mathsf{pres}})$ from adversary, check whether $(msg, \sigma_{\mathsf{pres}}, 0)$ is already stored. In this case, restart signing. Otherwise, store $(msg, \sigma_{\mathsf{pres}}, 1)$ in table $\mathsf{T}^{\mathsf{uID}}_{\mathsf{sign}}$.
7. Simulate sending $\langle \mathsf{uID}, \mathsf{cred}, \sigma_{\mathsf{cred}}, \mathsf{pres}, \mathsf{rev}, \sigma_{\mathsf{pres}} \rangle$ to corrupted RP.
8. Parse $(\mathsf{rev}_I, \mathsf{rev}_U) \leftarrow \mathsf{rev}$.
9. On message $(\mathsf{verify}, \mathsf{sid})$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{rev}}$ on behalf of RP, define leakage L as follows:
   - if storage is not corrupted, take $L = \mathcal{L}^{\{RP\}}_{verify}(U, RP, x) = (x, b')$;
   - if storage is corrupted, take $L = \mathcal{L}^{\{\mathsf{ST}, RP\}}_{verify}(U, RP, x) = (x, b', \mathsf{rev\_sids}(x))$;

   where $x = \mathsf{rev}_I$ and $b' \in \mathsf{notif}(U)$ is true iff U has received a successful notification for x before RP. Since an honest U does not proceed to verification without a successful prior notification, $b' = \mathsf{true}$. The set $\mathsf{rev\_sids}(x)$ should contain all session identifiers for which x has been revoked so far, which corresponds to S_{ID} received from $\mathcal{F}_{\mathsf{ideal}}$ for a corrupted ST. For all $\mathsf{sid}' \in S_{ID}$ that ended up successfully, the simulator also adds rev_P to X^-, where P is the revoker party of the session sid'.

   Send $(\mathsf{verify\text{-}req}, \mathsf{sid}, L)$ on behalf of $\mathcal{F}_{\mathsf{rev}}$ to $\mathcal{A}$.
10. Wait for input $(\mathsf{verify\text{-}req\text{-}ok}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{rev}}$ from $\mathcal{A}$.
11. Assert $x \notin X^-_{RP}$.
12. Output $(\mathsf{verify\text{-}resp\text{-}v}, \mathsf{sid}, x)$ on behalf of $\mathcal{F}_{\mathsf{rev}}$ to $\mathcal{A}$.
13. Repeat steps (9)-(12) with $x = \mathsf{rev}_U$.
14. On message $(\mathsf{Verify}, \mathsf{sid}, (c_{\mathsf{rev}}, \mathsf{rev}))$ from $\mathcal{A}$ to $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$ on behalf of RP, if there exists a pair $(\mathsf{rev}, c_{\mathsf{rev}})$ in $\mathsf{T}^{\mathsf{uID}}_{\mathsf{enc}}$ and rev is unique, output $(\mathsf{Verify\text{-}success}, \mathsf{sid})$ to $\mathcal{A}$. Otherwise, output $(\mathsf{Verify\text{-}fail}, \mathsf{sid})$ to $\mathcal{A}$.
15. For $i = 0, \ldots, |\overrightarrow{\mathsf{atr}}|$, on message $(\mathsf{Verify}, \mathsf{sid}, (c_i, \mathsf{atr}_i))$ from $\mathcal{A}$ to $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$ on behalf of RP, if there exists a pair (atr_i, c_i) in $\mathsf{T}^{\mathsf{uID}}_{\mathsf{enc}}$ and atr_i is unique, output $(\mathsf{Verify\text{-}success}, \mathsf{sid})$ to $\mathcal{A}$. Otherwise, output $(\mathsf{Verify\text{-}fail}, \mathsf{sid})$ to $\mathcal{A}$.
16. Send $(\mathsf{verify\text{-}req\text{-}ok}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ideal}}$.

Malicious User: It is important that revocations of honest revokers are covered by $\mathcal{F}_{\mathsf{rev}}$. It is possible that, for some rev included into $\mathcal{F}_{\mathsf{rev}}$ (revocations by a corrupted revoker), the corresponding credential cID is *not* included into T_r of $\mathcal{F}_{\mathsf{ideal}}$. In this case, instead of letting $\mathcal{F}_{\mathsf{ideal}}$ rely on the check $(\_, \mathsf{cID}, \_) \notin \mathsf{T}_r$, the simulator will explicitly make $\mathcal{F}_{\mathsf{ideal}}$ fail using the adversary's interface.

1. Wait for $\mathcal{A}$ to establish a connection $(\mathsf{tag}_{verify}, \mathsf{sid}, \mathsf{uID})$ with RP on behalf of U.
2. On behalf of the RP, generate a random challenge $\mathsf{ch} \leftarrow\!\!\$\ \mathcal{CH}$. Send ch to $\mathcal{A}$.
3. Wait for $\mathcal{A}$ to establish $(\mathsf{tag}_{verify}, \mathsf{sid}, \mathsf{uID})$ connection with the AMS. On behalf of AMS, participate in generating a blind signature using $\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}$.
4. Let $\mathcal{A}$ choose the message pres' sent to RP on behalf of U.
   Parse $\langle \mathsf{uID}, \mathsf{cred}, \sigma_{\mathsf{cred}}, \mathsf{pres} = (\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}, \mathsf{IID}), \mathsf{rev}, \sigma_{\mathsf{pres}} \rangle \leftarrow \mathsf{pres}'$.
5. Look for an entry $(\mathsf{cID}, \mathsf{cred}, \sigma_{\mathsf{cred}}, \mathsf{rev})$ in $\mathsf{T}_{\mathsf{issue}}$. If there is no such entry:
   - If $\mathsf{IID} \in \mathcal{C}$, use silent issuing interface to generate an entry with a suitable data cred, provided with a fresh identifier cID. This creates a record $\mathsf{cred} \in \mathsf{T}_a$ in $\mathcal{F}_{\mathsf{ideal}}$, with $(\_, \mathsf{cID}, \_) \notin \mathsf{T}_r$. If $\mathsf{rev} \in X^-$, use silent revocation interface of $\mathcal{F}_{\mathsf{ideal}}$ to add $(\mathsf{U}, \mathsf{cID}, \mathsf{U})$ to T_r.
     // In this case, the adversary may still cause verification to fail due to presentation signature check or failures by $\mathcal{F}_{\mathsf{rev}}$, but such a failure will be handled separately.
   - Otherwise, take $\mathsf{cID} = \perp$.
     // In this case, then the verification will fail in both $\mathcal{F}_{\mathsf{ideal}}$ and the real protocol. This is because, if the issuer is honest, the adversary is only able to come up with valid issuer signatures that are stored in $\mathsf{T}_{\mathsf{issue}}$.
6. Send $(\mathsf{input}, \mathsf{uID}, \mathsf{verify\text{-}req}, \mathsf{sid}, \mathsf{pres}, \mathsf{RP})$ to $\mathcal{F}_{\mathsf{ideal}}$ to confirm proceeding.
7. Wait for $(\mathsf{verify\text{-}req}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{ideal}}$.
8. On behalf of RP, perform checks that $\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}$ are contained in cred, and $|\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}| = |\overrightarrow{\mathsf{atr}}|$. Verify the signatures σ_{cred} and σ_{pres} using $\mathcal{F}_{\mathsf{sign}}^{\mathsf{IID}}$ and $\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}$.
9. On message $(\mathsf{verify}, \mathsf{sid}, x)$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{rev}}$ on behalf of U, set $L = \mathcal{L}_{verify}^{\{U\}}(U, RP, x) = b'$, where $b' \in \mathsf{notif}(U)$ is true iff U has received a successful notification before RP (as U is corrupted, both events are known to the simulator). Send $(\mathsf{verify\text{-}req}, \mathsf{sid}, L)$ on behalf of $\mathcal{F}_{\mathsf{rev}}$ to $\mathcal{A}$.
10. Wait for input $(\mathsf{verify\text{-}req\text{-}ok}, \mathsf{sid})$ from $\mathcal{A}$.
11. Assert $x \notin X_{RP}^-$
12. Output $(\mathsf{verify\text{-}resp\text{-}u}, \mathsf{sid}, \mathsf{ok})$ on behalf of $\mathcal{F}_{\mathsf{rev}}$ to $\mathcal{A}$.
13. Parse $(\mathsf{rev}_I, \mathsf{rev}_U) = \mathsf{rev}$. On behalf of RP, check $x = \mathsf{rev}_I$. Repeat steps (9)-(12) and check $x = \mathsf{rev}_U$.
14. On message $(\mathsf{Verify}, \mathsf{sid}, (c_{\mathsf{rev}}, \mathsf{rev}))$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}$ on behalf of U, if there exists a pair $(\mathsf{rev}, c_{\mathsf{rev}})$ in $\mathsf{T}_{\mathsf{enc}}^{\mathsf{uID}}$ and rev is unique, output $(\mathsf{Verify\text{-}success}, \mathsf{sid})$ to RP. Otherwise, output $(\mathsf{Verify\text{-}fail}, \mathsf{sid})$ to RP.
15. For $i = 0, \ldots, |\overrightarrow{\mathsf{atr}}|$, on message $(\mathsf{Verify}, \mathsf{sid}, (c_i, \mathsf{atr}_i))$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}$ on behalf of U, if there exists a pair (atr_i, c_i) in $\mathsf{T}_{\mathsf{enc}}^{\mathsf{uID}}$ and atr_i is unique, output $(\mathsf{Verify\text{-}success}, \mathsf{sid})$ to RP. Otherwise, output $(\mathsf{Verify\text{-}fail}, \mathsf{sid})$ to RP.
16. If all checks by RP succeed, send $(\mathsf{verify\text{-}req\text{-}ok}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ideal}}$.

C.10 Corruption

In the real protocol, the adversary is corrupting *parties*. Whenever a party receives a special corruption message, it is considered corrupted in every subfunctionality ($\mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}$, $\mathcal{F}_{\mathsf{sign}}^{\mathsf{IID}}$, $\mathcal{F}_{\mathsf{rev}}$) used by the protocol. The simulator needs to

propagate this corruption to $\mathcal{F}_{\mathsf{ideal}}$.On input ($\mathsf{corrupt}$) from $\mathcal{A}$ to a party P, send ($\mathsf{corrupt\text{-}party}, P$) to $\mathcal{F}_{\mathsf{ideal}}$.

For the functionality $\mathcal{F}_{\mathsf{tsign}}$, the corruptions are handled differently, as the adversary may explicitly send messages ($\mathsf{corrupt\text{-}server}$) and ($\mathsf{corrupt\text{-}user}, \ell$) to $\mathcal{F}_{\mathsf{tsign}}$. For all uID, the simulator constantly observers the internal state of $\mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}$, acting as follows:

- Whenever $c_{\mathsf{S}} = 1$ is set in $\mathcal{F}_{\mathsf{tsign}}$, send ($\mathsf{corrupt\text{-}party}, \mathsf{ST}$) to $\mathcal{F}_{\mathsf{ideal}}$.
- If setup has not been performed, only $c_{\mathsf{U}} = 3$ can be set in $\mathcal{F}_{\mathsf{tsign}}$ (full corruption). This implies that adversary has full control over the user's device and there are no privacy guarantees for user any more. Send ($\mathsf{corrupt\text{-}party}, \mathsf{uID}$) to $\mathcal{F}_{\mathsf{ideal}}$, so that it will set $c_{\mathsf{U}} = 3$ as well.
- If the setup has been completed and $c_{\mathsf{U}} = 1$ is set in $\mathcal{F}_{\mathsf{tsign}}$, send ($\mathsf{corrupt\text{-}party}, \mathsf{uID}$) to $\mathcal{F}_{\mathsf{ideal}}$, so that it will increase $c_{\mathsf{U}} = 1$ as well. This happens when adversary has managed to learn phone's memory sometime between threshold signing sessions and has copied encrypted memory.
- If the setup has been completed, and $c_{\mathsf{U}} = 2$ is set in $\mathcal{F}_{\mathsf{tsign}}$, send ($\mathsf{corrupt\text{-}party}, \mathsf{uID}$) to $\mathcal{F}_{\mathsf{ideal}}$, so that it will increase $c_{\mathsf{U}} = 2$ as well. This happens when $\mathcal{A}$ guesses PIN during the authentication, having managed to gain access to the secret key storage. In practice, this means that adversary can be able to initiate multiple interactions with AMS server (by creating correct authentication signature) before the legitimate user tries to initiate signing transaction. At the legitimate user's signing attempt, clone detection mechanism will trigger and the user will be blocked, meaning that neither the legitimate user nor the adversary will be able to initiate any further request.

Whenever corruption level $c_{\mathsf{U}} = 2$ is reached, $\mathcal{F}_{\mathsf{ideal}}$ sends a special message ($\mathsf{full\text{-}leakage}, \mathsf{sid}, \mathsf{uID}, (\mathsf{T}_a^{\mathsf{uID}}, \mathsf{T}_r^{\mathsf{uID}})$), which contains projections of tables to the user uID. As discussed above, the practical leakage will be limited due to clone detection mechanism, but for safety we are assuming the worst-case here, as we do not know what exactly the adversary will manage to decrypt before being detected. Update the internal tables as follows:

- For all $(\mathsf{uID}, \mathsf{cID}, \overrightarrow{\mathsf{aID}}, \mathsf{atr}) \in \mathsf{T}_a$, find an entry in $\mathsf{T}_{\mathsf{issue}}$ with the corresponding cID and $\mathsf{cred} = \langle(\mathsf{aID}_0, c_0), (\mathsf{aID}_1, c_1), \ldots, (\mathsf{aID}_n, c_n), (\mathsf{aID}_{\mathsf{rev}}, c_{\mathsf{rev}})\rangle$. For every pair (aID_i, c_i), update the entry (atr_i, c_i) in $\mathsf{T}_{\mathsf{enc}}^{\mathsf{uID}}$ for the given c_i. If some (atr_i, c_i) has already been opened to $\mathcal{A}$ before (during presentation), then it is the same atr_i, taken from T_a.
- For all $(\mathsf{uID}, \mathsf{cID}, P) \in \mathsf{T}_r$, find an entry in $\mathsf{T}_{\mathsf{issue}}$ with the cID and the corresponding revocation token $\mathsf{rev}= (\mathsf{rev}_I, \mathsf{rev}_U)$. Add rev_P to the set X^- of the simulated $\mathcal{F}_{\mathsf{rev}}$ (it is possible that rev_P is already there). This ensures that cID will be treated as revoked during subsequent presentations.

C.11 Indistinguishability of the real and the simulated protocol.

Issuing In the case of honest issuer, simulator indistinguishably simulates issuance of credential

$$\mathsf{cred} = \langle(\mathsf{aID}_0, c_0), (\mathsf{aID}_1, c_1), \ldots, (\mathsf{aID}_n, c_n), (\mathsf{aID}_{\mathsf{rev}}, c_{\mathsf{rev}})\rangle$$

by including the public key $\mathsf{pk}_{\mathsf{uID}}$ of the user uID into c_0, taking attribute values atr_i from the pre-defined set $\mathcal{M}_{\mathsf{aID}_i}$ (that is the same as in the real protocol) for $i = 1..n$, sampling revocation token from $\mathcal{X}$, assembling information together and creating signature. Indistinguishability of the signing and encryption processes is inherited from the underlying used constructions. Number of required attributes n and the attribute identifiers $\overrightarrow{\mathsf{aID}}$ are received from the $\mathcal{F}_{\mathsf{ideal}}$. The values rev are allowed to repeat for a corrupted user.

In the case of malicious issuer, adversary itself generates credential for the user. Before saving issued credential to the internal tables, simulator verifies that credential contains correct revocation token $\mathsf{rev} \in \mathcal{X}$ and correct attribute values $\mathsf{atr}_i \in \mathcal{M}_{\mathsf{aID}_i}$. Since rev is freshly sampled, we have that $\mathsf{rev} \notin X^-$ and cID is unique with overwhelming probability.

In the case of malicious user, the issuer generates the credential for the user identity uID that is eligible to receive the credential, even if the request is sent on behalf of a malicious party. If uID is honest, the issuer will only generate a credential if uID has indeed requested it, which is proven by the fresh signature of uID. If AMS is corrupted, it is possible that the user uID will not receive the credential, but the adversary does not learn the attribute values, since they are encrypted by the public key of uID.

The simulator saves the credential that passes all checks to the internal table $\mathsf{T}_{\mathsf{issue}}$. A credential with cID containing uID and IID is only added to $\mathsf{T}_{\mathsf{issue}}$ if the following conditions are satisfied:

- The credential has indeed been requested by the user uID (if uID is honest).
- The credential has indeed been issued by the issuer IID (if IID is honest).

Here unfairness w.r.t. user is allowed, i.e. the user will not necessarily receive the credenital due to network delay.

Presentation If the user is honest, presentation process is all simulated and adversary does not get to see credential presentation pres. This allows simulator to simulate sub-protocols of the presentation phase with randomly selected credential. Usage of the blind threshold decryption ensures that the adversary does not learn which attribute is being decrypted. Thus, in this case simulated protocol is indistinguishable from the real protocol.

In the case of malicious user, adversary is able to initiate presentation process only if it has guessed PIN from the underlying threshold signature scheme. If the PIN check is successful, simulator simulates all the answers to the sub-protocol queries using its internal tables for the decryption and revocation using ciphertexts c_{atr_i} and c_{rev} supplied by adversary (note that adversary may decrypt *any* ciphertexts of uID at this point, and this is why there are no privacy guarantees for the user when achieving $c_{\mathsf{U}} = 2$). The adversary expects to see the correct notification from $\mathcal{F}_{\mathsf{rev}}$. This is indeed so, as the simulated X^- in this case is consistent with the actually revoked tokens (either by the user or some issuer),

both honest and corrupted. In the end of presentation, the output is produced by the simulator, and there is no effect on $\mathcal{F}_{\mathsf{ideal}}$.

Verification If the RP is malicious, it sees the presentation that is output of the ideal functionality, correctly constructed from the attributes requested by the user. We need to additionally ensure that the simulated revocation token is consistent with the previous revocations. During the previous protocol phases, whenever credential cID of an honest user has been revoked, the following has happened:

- In the cases when $\mathcal{F}_{\mathsf{ideal}}$ leaked cID to the simulator, the corresponding rev was added to X^-
- Otherwise, adding rev to X^- was postponed (a random rev has been revoked, which coincides with at least one real token with probability $\frac{n}{N}$ for n issued credentials). During the verification, the simulator has added all relevant tokens rev to X^- as soon as it received the corresponding information from $\mathcal{F}_{\mathsf{ideal}}$, before simulating verification using $\mathcal{F}_{\mathsf{rev}}$.

If user is malicious and RP is honest, the simulated challenge ch is unique with probability $m/|\mathcal{CH}|$ for m verification sessions. The simulator performs checks of credential presentation as an honest verifier. The simulated protocol accepts the credential only if

- credential signature σ_{cred} verifies,
- revocation token is not among the revoked tokens: $\mathsf{rev} \notin X^-$,
- disclosed attributes atr_i correspond to ciphertexts from credential cred.
- the user could respond to the fresh challenge, proving ownership of $\mathsf{pk}_{\mathsf{uID}}$

As malicious verifications are only allowed in the case $c_{\mathsf{U}} = 2$ and $c_{\mathsf{U}} = 3$, it holds that either (1) the user has been fully corrupted from the beginning or (2) the attacker guessed PIN of the user. In the case (1), the simulator has all the data of this user included into internal tables, so all previous checks can be done against them. In the case (2), some data of this user stored in the simulator tables (up to the first corruption) is garbage. However, the true data is provided by $\mathcal{F}_{\mathsf{ideal}}$ upon corruption, and the simulator fills its local tables accordingly, ensuring consistency of simulated tables with the tables of $\mathcal{F}_{\mathsf{ideal}}$.

Revocation by issuer In case of malicious issuer, adversary knows the connection between the credential identifier and the revocation token and expects correct credential to be revoked. Simulator uses its internal table $\mathsf{T}_{\mathsf{issue}}$ to ensure that correct credential is marked as being revoked.

If AMS is corrupted, and cID has already been presented to a corrupted RP, or it belongs to a corrupted user, then rev_I should be leaked to the adversary by $\mathcal{F}_{\mathsf{rev}}$. In both cases, cID is leaked by $\mathcal{F}_{\mathsf{ideal}}$, and the simulator takes the corresponding rev_I from $\mathsf{T}_{\mathsf{issue}}$.

If AMS is corrupted, but the credential has not been presented to a corrupted RP yet, and it belongs to an honest user, then the simulator does not receive

cID from $\mathcal{F}_{\mathsf{ideal}}$. However, as $\mathcal{F}_{\mathsf{rev}}$ does not leak rev_I neither, using random rev_I is indistinguishable from using the actual revocation token rev_I from $\mathsf{T}_{\mathsf{issue}}$ (unless rev_I coincides by chance with some legitimate token, which is $\frac{n}{N}$ for n issued credentials).

User revocation If the user is corrupted, adversary is able to initiate revocation process only if it has guessed PIN from the underlying threshold signature scheme. If the PIN check is successful, then either the user has been corrupted from the beginning ($c_{\mathsf{U}} = 3$), or we have $c_{\mathsf{U}} = 2$, and in both cases the simulator has filled its internal tables according to T_a and T_r of $\mathcal{F}_{\mathsf{ideal}}$, so it has all necessary data for the simulation. The simulator simulates all the answers to the sub-protocol queries using its internal tables for the decryption and revocation.

If AMS is corrupted, and cID has already been presented to a corrupted RP, then rev_U should be leaked to the adversary by $\mathcal{F}_{\mathsf{rev}}$. In this case, cID is leaked by $\mathcal{F}_{\mathsf{ideal}}$, and the simulator takes the corresponding rev_U from $\mathsf{T}_{\mathsf{issue}}$.

If AMS is corrupted, but the credential has not been presented to a corrupted RP yet, then the simulator does not receive cID from $\mathcal{F}_{\mathsf{ideal}}$. However, as $\mathcal{F}_{\mathsf{rev}}$ does not leak rev_U neither, and decryption by means of $\mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}$ is blind, using a random rev_U is indistinguishable from using the actual revocation token rev_U from $\mathsf{T}_{\mathsf{issue}}$ (unless rev_U coincides by chance with some legitimate token, which is $\frac{n}{N}$ for n issued credentials).

Appendix 5

Publication IV

P. Laud and J. Vakarjuk. A comparison-based methodology for the security assurance of novel systems. In S. K. Katsikas, F. Cuppens, C. Kalloniatis, J. Mylopoulos, F. Pallas, J. Pohle, M. A. Sasse, H. Abie, S. Ranise, L. Verderame, E. Cambiaso, J. M. Vidal, M. A. S. Monge, M. Albanese, B. Katt, S. Pirbhulal, and A. Shukla, editors, *Computer Security. ESORICS 2022 International Workshops - CyberICPS 2022, SECPRE 2022, SPOSE 2022, CPS4CIP 2022, CDT&SECOMANE 2022, EIS 2022, and SecAssure 2022, Copenhagen, Denmark, September 26-30, 2022, Revised Selected Papers*, volume 13785 of *Lecture Notes in Computer Science*, pages 625–644. Springer, 2022

A Comparison-Based Methodology for the Security Assurance of Novel Systems

Peeter Laud and Jelizaveta Vakarjuk

Cybernetica AS
peeter.laud|jelizaveta.vakarjuk@cyber.ee

Abstract. In this paper, we advocate the position that the security certification of one system should make the certification of other similar systems easier, if one can present the evidence that the second system is at least as secure as the first system. We present a development of this idea, stating the components of such comparative evidence. We stretch the idea of propagating the certification to less similar systems, if one can present a sequence of systems from the certified one to the novel one, where each system is evidenced to be at least as secure as the previous one. We apply our methodology to authentication systems, where we show that a system based on threshold cryptography is at least as secure as widely used smartcard-based systems.

1 Introduction

Our critical systems are built on top of, and their security assurances depend on smaller subsystems, whose security is of utmost importance, and where the society has chosen security certification as the methodology to ensure that these subsystems satisfy the required properties. These small systems and devices include smartcards, hardware security modules, trusted execution environments, but also certain communication devices, operating systems and application software. A lot of trust is placed on the outcomes of certification, hence this process is expensive [9] and conservative [20]. When a certified system has been updated, then the new version has to pass the certification again. A system making use of novel security technologies may have hard time passing certification at all, because these technologies may not map cleanly to the categories specified in certification procedures.

In Common Criteria (CC) certification [8], we speak of *Targets of Evaluation* (ToE); this is the well-delimited system that is expected to pass certification. A *Security Target* (ST) document is compiled for certifying a particular ToE; it gives the boundaries of ToE, and lists the security requirements that ToE is expected to fulfill. These requirements are taken from standardised lists, enumerated in CC specifications. A security target may comply with, i.e contain one or more *Protection Profiles* (PP), which are themselves lists of security requirements. The security requirements are either *Security Functional Requirements* (SFR), referring to the security technologies that the ToE may use in a certain

manner in order to obtain certain security properties, or *Security Assurance Requirements* (SAR), which refer to the good practices used during the design, development, and operation of the ToE. In this paper, we use CC-related terms, but our proposed methodology should be applicable to other certification frameworks, too.

When a ToE evolves, then it may still fulfill the same SFRs listed in the ST of the old ToE, but it may fulfill them in novel manner. When a ToE makes use of novel technologies, it may be difficult to map the technical protections it offers to the fulfilment of SFRs in the ST. In both cases, it may be helpful to not match the new ToE directly against SFRs, but to compare the new ToE to the old one and make sure that the new one is *at least as secure*. This is the position we advocate in this paper: comparing two ToE-s — one with existing certification against a ST, and the other that we desire to get certified against a similar ST — can simplify the certification process, while preserving the assurances of the existing certification. The comparison establishes, that for any attack against the novel system, there exists an attack against the certified system that is no more difficult to perform, and with the same or worse consequences. We argue that such arguments should be accepted by certification bodies. Even if they are not directly accepted, the outcomes of the comparison should guide the vendor in changing the evidence documentation that supports the claims of ToE satisfying the ST.

Our proposal is similar to *game-based* security proofs of cryptographic primitives [1]. In cryptography, security definitions define an interactive attack game that is played between the adversary and the environment; the latter provides an interface to the adversary through which the primitive may be invoked in well-defined ways. The primitive is deemed secure if no adversary's probability of *winning* the game exceeds a certain value. A typical security proof presents a different game where the same adversary obviously has only a low probability of winning, and then argues that in the original game, the adversary's win is not much more likely.

In these security proofs, it is common to introduce *intermediate* games, such that any two neighbouring games differ only slightly from each other, simplifying the argument that the adversary's winning probability in i-th game is at most negligibly larger than in the $(i+1)$-st game. Here the zeroth game comes from the security definition, and the final game is the one where the adversary's win is obviously unlikely. We advocate that we can do something similar for certification: the zeroth ToE is the novel one that we want to certify, the final ToE has already been certified, and we may introduce "intermediate" ToE-s that are easier to compare against the previous and next ones.

Related work. The certification of evolving systems, the simplification of updating the evidence, and the reuse of evaluation results have been the topic of a statement from the CC Recognition Arrangement Management Committee [16]. Methods for using CC together with agile product development [18,19], and for continued certification of cyber security products [5] have been proposed. But we are aware of only a few attempts to show the security of one system by compar-

ing it to another one, where the latter one has already been deemed secure [3]. The other example is Trusted Computer System Evaluation Criteria (TCSEC), where the system evaluation process included the Rating Maintenance Phase (RAMP). The goal of RAMP was to provide an instrument to extend evaluation of a certified system to a new version of that system by analysing changes that were introduced [10].

In the rest of this paper, we describe our proposal for comparing two systems in more detail. We will then give an example of comparing two systems, where one of them is based on well-established hardware security technologies, while the other one makes use of threshold cryptography. In order to compare them against each other, we come up with a couple of intermediate systems, and argue that each of them is at least as secure as the next one.

2 Proposed methodology

Suppose we have a ToE T', for which we want to provide evidence that it matches a security target. We also have another system T, which we have already shown to match the same or a similar security target. We have the detailed specifications of both T and T', such that we can make a list of all potential weaknesses that an adversary may exploit in T or in T'. We may already have used these lists to create the security target [7]. For describing the processes of T and T', BPMN [11] may be a good choice, perhaps annotated further [14,17] to show the protection mechanisms in use. We create lists of weaknesses with high granularity, considering each data storage, movement, or processing step as something that the adversary may obtain information from, or tamper with the inputs and outputs of. The high granularity also means that an adversary may exploit one weakness either fully, or not exploit it at all, but there are no partial exploitations. Let W be the set of all weaknesses of T, and W' the set of all weaknesses of T'. Also, let $\mathcal{W}$ [resp. $\mathcal{W}'$] be the set of subsets of W [resp. W']. When an adversary performs an attack against T or T', then the set of weaknesses it exploits is an element of $\mathcal{W}$ or $\mathcal{W}'$.

The security target specifies the operational environment of the ToE, including the threats, organisational policies, and security assumptions. Hence not any set of weaknesses is considered exploitable against the system T. Rather, the security target specifies the set of *considered sets of weaknesses* $\mathcal{W}_{\mathrm{C}} \subseteq \mathcal{W}$ that contains only such sets of weaknesses that an adversary may *attempt* to exploit according to the security target. Here the attempts by the adversary do not correspond to his success; rather, the ToE is meant to employ measures that make the adversary unsuccessful. It is natural to assume that $\mathcal{W}_{\mathrm{C}}$ is downwards closed: if $\mathbf{w}_1, \mathbf{w}_2 \in \mathcal{W}$, $\mathbf{w}_1 \subseteq \mathbf{w}_2$, and $\mathbf{w}_2 \in \mathcal{W}_{\mathrm{C}}$, then also $\mathbf{w}_1 \in \mathcal{W}_{\mathrm{C}}$.

The security target for T' similarly specifies $\mathcal{W}'_{\mathrm{C}} \subseteq \mathcal{W}'$. As T' has not been certified yet, our methodology may bring adjustments to the precise definition of $\mathcal{W}'_{\mathrm{C}}$; hopefully not reducing it by too much.

To each set of weaknesses of T or T' we can attach two (abstract) quantities: the *difficulty* (for the adversary) of exploiting this set of weaknesses, and the

seriousness of the effects of successful exploitation by the adversary. We do not define the structure of these quantities, but we require that there is a (partial) order on them. Hence we get two preorders on $\mathcal{W} \cup \mathcal{W}'$: for two sets of weaknesses $\mathbf{w}_1$ and $\mathbf{w}_2$ we write $\mathbf{w}_1 \prec_\mathrm{D} \mathbf{w}_2$, if exploiting all weaknesses in $\mathbf{w}_1$ is no more difficult for the adversary than exploiting all weaknesses in $\mathbf{w}_2$. We also write $\mathbf{w}_1 \prec_\mathrm{S} \mathbf{w}_2$ if the effects of exploiting $\mathbf{w}_1$ are no worse than the effects of exploiting $\mathbf{w}_2$. A natural corollary of these definitions is, that if $\mathbf{w}_1 \subseteq \mathbf{w}_2$, then also $\mathbf{w}_1 \prec_\mathrm{D} \mathbf{w}_2$ and $\mathbf{w}_1 \prec_\mathrm{S} \mathbf{w}_2$.

It is possible that for some $\mathbf{w}_1 \neq \mathbf{w}_2$ we have both $\mathbf{w}_1 \prec_\mathrm{D} \mathbf{w}_2$ and $\mathbf{w}_2 \prec_\mathrm{D} \mathbf{w}_1$ (denote this $\mathbf{w}_1 \sim_\mathrm{D} \mathbf{w}_2$), hence $\prec_\mathrm{D}$ is not an order, but only a preorder. The same applies to $\prec_\mathrm{S}$. It may be difficult to describe all refinements that $\prec_\mathrm{D}$ and $\prec_\mathrm{S}$ have with respect to the subset inclusion. However, an imprecise description does not invalidate the outcomes of our proposed analysis.

We want to show that for each $\mathbf{w}' \in \mathcal{W}'_\mathrm{C}$, there exists some $\mathbf{w} \in \mathcal{W}_\mathrm{C}$, such that $\mathbf{w} \prec_\mathrm{D} \mathbf{w}'$ and $\mathbf{w}' \prec_\mathrm{S} \mathbf{w}$. We thus have to present a function $f : \mathcal{W}'_\mathrm{C} \to \mathcal{W}_\mathrm{C}$, such that $f(\mathbf{w}') \prec_\mathrm{D} \mathbf{w}'$ and $\mathbf{w}' \prec_\mathrm{S} f(\mathbf{w}')$ for all $\mathbf{w}' \in \mathcal{W}'_\mathrm{C}$. We have found it easier to present not a function f, but a relation $F \subseteq \mathcal{W}' \times \mathcal{W}$, requiring that each $\mathbf{w}' \in \mathcal{W}'_\mathrm{C}$ is related to at least one element of $\mathcal{W}_\mathrm{C}$.

Each element of $\mathcal{W}'$ [resp. $\mathcal{W}$] is a subset of W' [resp. W]; hence it has the natural representation as a characteristic vector of type $W' \to \{0,1\}$ [resp. $W \to \{0,1\}$]. The entire relation F is thus represented as a boolean function $\phi : ((W' \dot\cup W) \to \{0,1\}) \to \{0,1\}$, which in turn is conveniently represented as a boolean formula. The variables of this formula are exactly the weaknesses of T and T'.

We say that two systems T and T' are *similar* if there is enough relationships $\prec_\mathrm{D}$ and $\prec_\mathrm{S}$ between the corresponding attack sets. If the systems T and T' are sufficiently dissimilar, then the construction of ϕ may fail, because of the lack of relationships $\prec_\mathrm{D}$ and $\prec_\mathrm{S}$ between attack sets $\mathbf{w} \in \mathcal{W}$ and $\mathbf{w}' \in \mathcal{W}'$. Such lack of (obvious) relationships may inform us of the security mechanisms that we need to use in T' in order to follow our methodology — the security mechanisms used by T' have to be strong enough to justify the addition of $\prec_\mathrm{D}$- or $\prec_\mathrm{S}$-relationships between attack sets against T and T'. These mechanisms may also be necessary for T' to match the security target. If intermediate systems have been introduced, then the requirements to use stronger security mechanisms may propagate from T through these systems all the way back to the system T'.

3 Example

In this section, we show how to use the proposed methodology to compare two authentication systems, where the user's browser establishes a secure connection with a relying party's (RP) server, and the user is authenticated during this process. In both systems, the RP server, knowing the public key of the user, generates a challenge, which has to be signed with the corresponding private key. The systems differ in how the private key is stored, and how the signature is generated.

An authentication system has to provide processes for the full lifecycle of a user's keypair: how it is generated, how a certificate is issued for it, how the RP finds the public key, how the challenges are signed, how the key is revoked. A full comparison of the two systems has to consider all of them. For concreteness, in this paper we will only focus on the authentication process itself, which includes the signing, and also some details of the storage of the private key.

The first system T has a smartcard as its centerpiece, containing the user's private key, and issuing signatures with it when activated. There exist protection profiles for such devices [13,12], and a number of cards or chips on the card have been certified to satisfy them. A typical use of a smartcard is as part of the Client Certificate Authentication (CCA) in the Transport Layer Security (TLS) protocol [15].

The second system T' uses threshold cryptography [4], sharing the private key between the user's phone and a central server. It can be used to authenticate the client end in an established TLS session. When attempting to directly certify such a system, it may be difficult to argue that the user has sufficient control over the private key — neither the phone nor the remote server alone provides such control.

3.1 Authentication with a smartcard

When a relying party wants to make sure it is talking to a device of a user with a given public key, it will create a challenge bit-string and ask the device to sign it. If the corresponding private key is stored in a smartcard, used with a card reader, then the challenge goes from the client application to the card reader, then together with the PIN (entered by the user on the PINpad if the card reader has one, and on the keyboard otherwise) to the smartcard, which checks the PIN and creates the signature, which then moves back the same way. If there have been too many wrong entries of the PIN, then the card locks up. The whole process is displayed as a BPMN diagram in Figure 1, split into relatively atomic pieces.

The card stores the private key, and the PIN. The user also knows the PIN. Inside the card, the PIN entered by the user is compared against the PIN that the card is storing. The environment of the card is protected, meaning that the PIN and the private key (or any details about it) cannot be read out, and the logic that it executes cannot be thwarted. The card reader is also protected, in that the PIN entered on its pad will only be sent to the card and not anywhere else.

This process, and the protection offered by it is considered a good example of two-factor authentication, showing that the user knows the PIN (knowledge factor), and the user has the card (possession factor). It is considered a good example even if the PIN entry is less isolated from the outside world, i.e. when the user enters the PIN on the computer.

Weaknesses of a system with a smartcard. In this process, the following weaknesses can be identified. These are the elements of the set W — the pos-

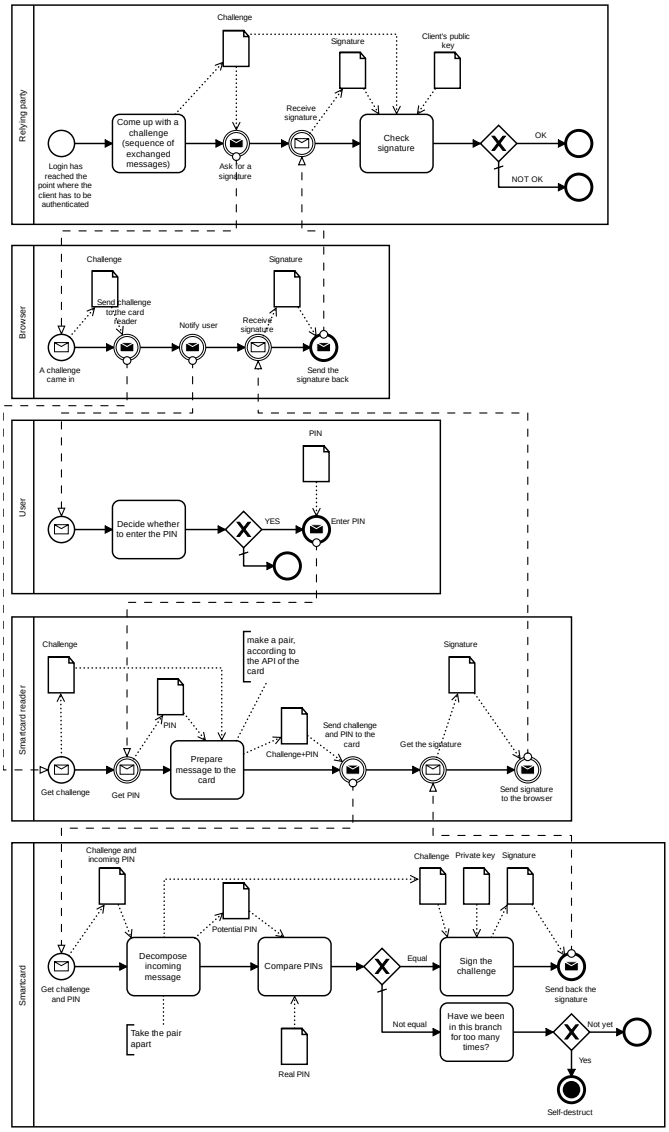


Fig. 1. The smartcard system

sible locations where an adversary could mount an attack. As the system T is considered secure, none of these locations, or their combinations are actually an exploitable weakness, i.e. $\mathcal{W}_C = \mathcal{W}$.

Possible attacks against the relying party

(RP1) Affect the computation of the challenge
(RP2) Learn the challenge
(RP3) Modify the challenge while it is sent to the browser
(RP4) Change the outcome of the signature check
(RP5) Accept the log-in, even if the signature does not check

Possible attacks against the browser

(B1) Learn the challenge
(B2) Modify the challenge while it is sent to the card reader
(B3) Modify the signature while it is sent to the relying party

Possible attacks against the user

(U1) Learn the PIN from the user
(U2) Change the PIN

Possible attacks against the smartcard reader

(SCR1) Learn the PIN that the user entered
(SCR2) Change the challenge that is sent to the smartcard
(SCR3) Change the PIN that is sent to the smartcard
(SCR4) Change the signature while it is sent to the browser

Possible attacks against the smartcard

(SC1) Learn the PIN
(SC2) Interfere with the PIN comparison procedure
(SC3) Make the decision of the PIN check take the other path
(SC4) Make the decision about counts of incorrect PINs take the other path
(SC5) Learn the private key
(SC6) Change the private key
(SC7) Change the challenge that enters the computation of the signature
(SC8) Learn the signature
(SC9) Change the signature sent back to the smartcard reader

3.2 Authentication with SplitKey

In systems relying on threshold cryptography, where the private key is stored in a secret-shared manner, several storage devices with the shares of the key have to be present in order to make use of that private key. If all the storage devices offer strong properties of unclonability, tamper-resistance and similar, then we can have even stronger security properties for the authentication process, compared to using just a single device.

However, we typically want to use threshold cryptography in order to reduce the requirements on the devices storing, and on the procedures for accessing individual shares of the private key. If we still want to argue that our authentication procedure has strong security properties similar to logging in with a smartcard, then we have to involve the properties of several devices in our arguments.

In particular, the SplitKey technology [2] for signature creation shares the private key among two devices, such that one of the devices – the phone – does not offer a strong protection for its key share. Hence, in an authentication system based on SplitKey, when arguing about the possession of a private key, we have to involve the second device in our arguments. This involvement may touch the physical properties of that device, as well as technical and organisational properties controlling the access to it.

The authentication procedure is depicted in Figure 2, in similar detail to the previous figure. Let us recall that in SplitKey, an RSA private key has been shared between the user's phone and the server, and the keyshare on the phone is encrypted with a low-entropy key (derived from the PIN that the user enters). If the adversary has obtained just the encrypted keyshare, then it cannot recognize, which PIN is the correct one. However, the server can recognize if it has received a signature share from the phone that has been created with a wrong keyshare.

When the phone has received the challenge, it asks the user to input the PIN. In order to help the user understand the context, in which the signing of the challenge is about to occur, the phone shows the user a short *control code*, derived from the challenge. The same control code is shown to the user by the relying party's website. The user is supposed to enter the PIN only if the control codes match.

Weaknesses of a SplitKey-based system. Considering the steps in Fig. 2, we can list the following possible weaknesses.

Attacks against the relying party:

(RP1) Affect the computation of the challenge
(RP2) Learn the challenge
(RP3) Affect the computation of the control code
(RP4) Learn the control code
(RP5) Modify the challenge while it is sent to the phone
(RP6) Change the outcome of the signature check
(RP7) Accept the log-in, even if the signature does not check

Attacks against the browser

(B1) Learn the control code
(B2) Change the control code, when it is shown to the user

Attacks affecting the user

(U1) Learn the PIN from the user
(U2) Change the PIN

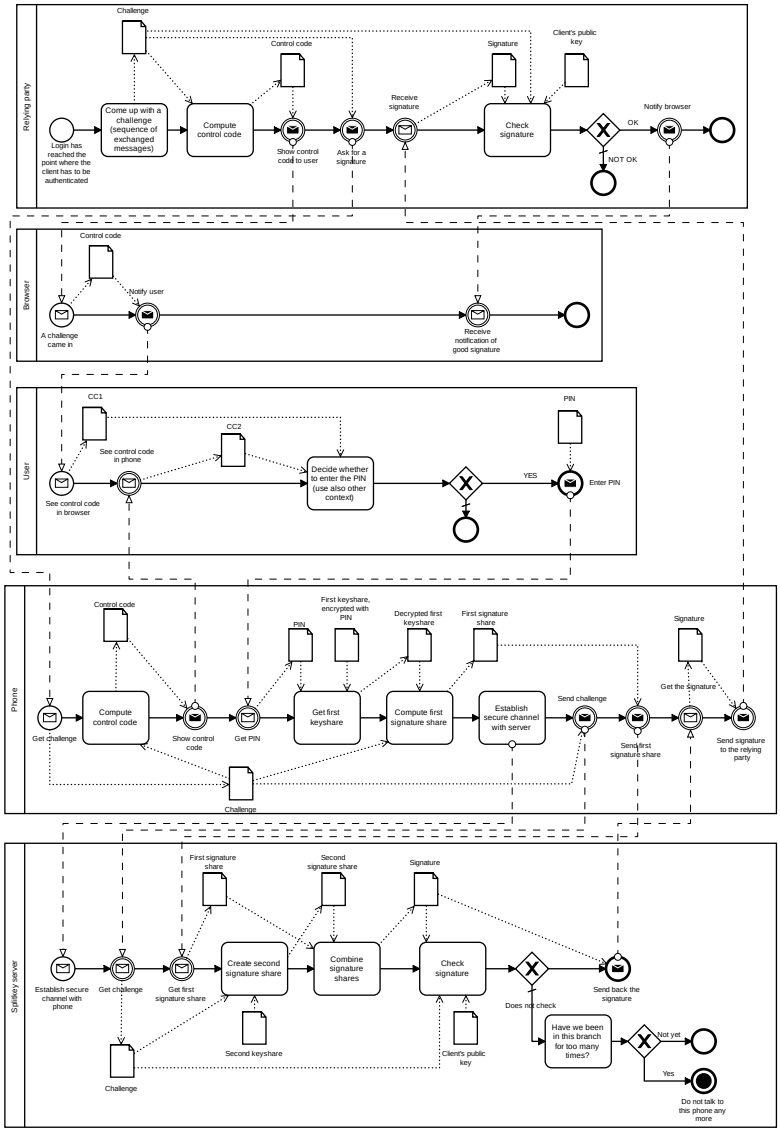


Fig. 2. SplitKey authentication system

(U3) Cause the user to incorrectly compare the control codes

Attacks against the phone

(P1) Affect the computation of the control code
(P2) Change the control code on the phone screen, where it is shown to the user
(P3) Learn the PIN that the user enters
(P4) Learn the encrypted first keyshare (encrypted with PIN)
(P5) Learn the plain first keyshare
(P6) Change the encrypted first keyshare, before it has been decrypted with the PIN
(P7) Change the plain first keyshare
(P8) Change the challenge that enters the computation of the first signature share
(P9) Learn the first signature share
(P10) Change the first signature share
(P11) Interfere with the establishment of the secure channel with the SplitKey server, thereby obtaining the capability to read and/or change messages sent over it
(P12) Change the challenge sent to the SplitKey server
(P13) Change the first signature share sent to the SplitKey server
(P14) Change the signature sent to the relying party

Attacks against the Splitkey server

(SS1) Interfere with the establishment of the secure channel with the phone, thereby obtaining the capability to read and/or change messages sent over it
(SS2) Interfere with the establishment of the secure channel with the phone, thereby confusing the server on the identity of the phone
(SS3) Learn the second keyshare
(SS4) Change the second keyshare
(SS5) Change the challenge that goes into the second signature share creation process
(SS6) Learn the second signature share
(SS7) Interfere with the signature combination process
(SS8) Learn the signature
(SS9) Change the client's public key, against which the signature is checked
(SS10) Interfere with the signature checking procedure
(SS11) Make the decision of the signature check go otherwise
(SS12) Change the signature sent back to the phone
(SS13) Make the decision about counts of unsuccessful signature creations take the other path

Among the attacks against the SplitKey system, we can identify the following relationships for $\prec_{\mathrm{D}}$ and $\prec_{\mathrm{S}}$. If the adversary learns both the PIN and the encrypted first keyshare, then he also has the plain first keyshare: $\{\mathrm{P5}\} \prec_{\mathrm{X}} \{\mathrm{P3}, \mathrm{P4}\}$ and $\{\mathrm{P5}\} \prec_{\mathrm{X}} \{\mathrm{U1}, \mathrm{P4}\}$, where X may be both D and S. Changing

a value has no greater effect than changing everything that is computed from this value. In particular, $\{SS4\} \prec_S \{SS7\} \prec_S \{SS10\} \prec_S \{SS11, SS12\}$ and $\{P6\} \prec_S \{P7\} \prec_S \{P12\} \prec_S \{SS7\}$. The same relationships generally continue to hold when we add the same additional attacks to both sides of $\prec_X$.

We are using threshold cryptography, where the adversary learning just a single share of a secret should not yet affect the seriousness of attacks that he can perform. In particular, we would like to state that $\{P5\} \cup \mathbf{w} \prec_S \mathbf{w}$ for a significant class of attack sets $\mathbf{w}$ that do not contain attacks against the second keyshare. We state this for all $\mathbf{w}$ that do not contain SS3 or SS6. Instead of the set $\{P5\}$, we can also consider other sets that imply the knowledge of the first keyshare or signature share: $\{P3, P4\}$ and $\{P9\}$.

3.3 First intermediate system

The systems T and T' are too different for the direct application of the comparison methodology we gave in Sec. 2. As we see below, we can cross this gap by proposing a couple of intermediate systems.

We change the system T' into the system T_1, where instead of computing the first signature share on the phone, we move all the computations into the Splitkey server. We let the Splitkey server store the user's private key share encrypted with the user's PIN and we send the PIN together with the challenge from the phone to the SplitKey server. The authentication process in the first intermediate system is depicted in Figure 3.

Weaknesses of the first intermediate system. Obviously the possible attacks against the relying party, browser, and user are the same as in the SplitKey-based system, as there have been no changes to this part of the system. Attacks against the phone and the server have changed; quite often, the change is simply in the target device.

Attacks against the phone

(P1) Affect the computation of the control code
(P2) Change the control code on the phone screen, where it is shown to the user
(P3) Learn the PIN that the user enters
(P4) Interfere with the establishment of the secure channel with the SplitKey server, thereby obtaining the capability to read and/or change messages sent over it
(P5) Change the challenge sent to the SplitKey server
(P6) Change the signature sent to the relying party
(P7) (**new**) Change the PIN sent to the SplitKey server

Attacks against the SplitKey server

(SS1) Interfere with the establishment of the secure channel with the phone, thereby obtaining the capability to read and/or change messages sent over it

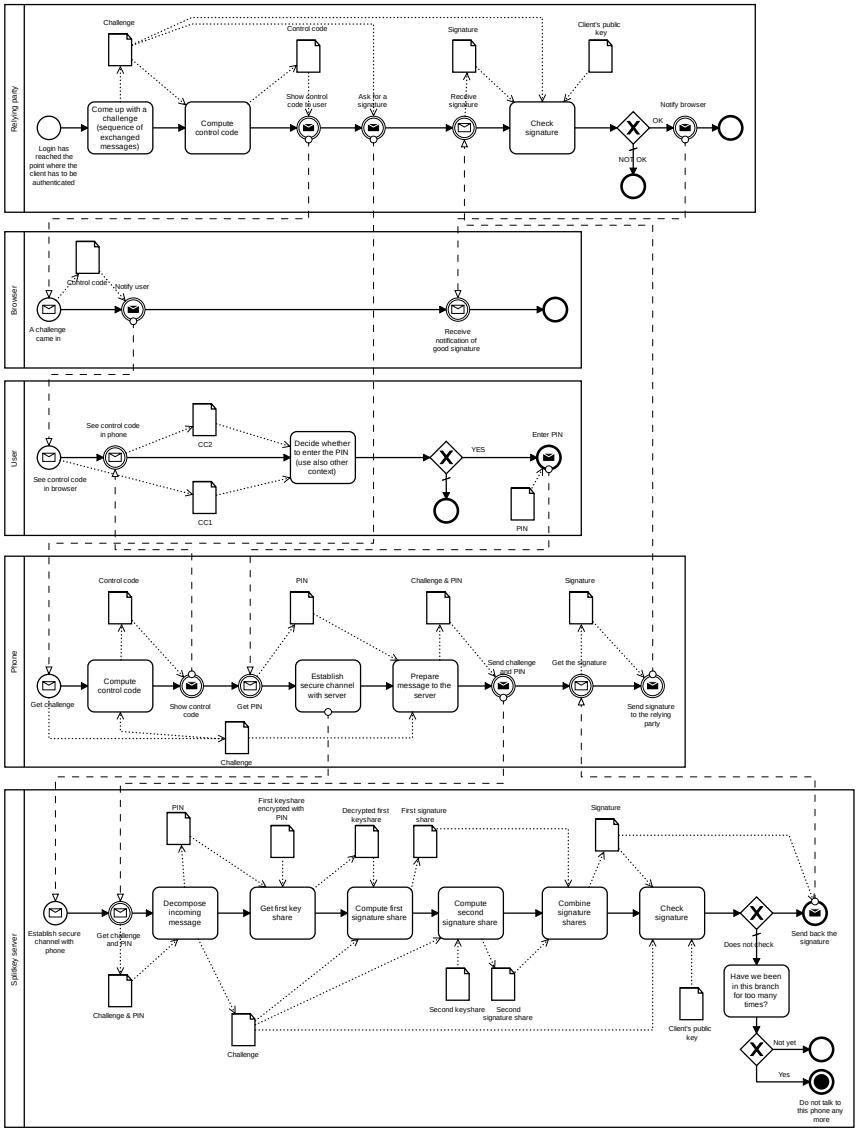


Fig. 3. The first intermediate system

(SS2) Interfere with the establishment of the secure channel with the phone, thereby confusing the server on the identity of the phone
(SS3) Learn the second keyshare
(SS4) Change the second keyshare
(SS5) Change the challenge that goes into the second signature share creation process
(SS6) Learn the second signature share
(SS7) Interfere with the signature combination process
(SS8) Learn the signature
(SS9) Change the client's public key, against which the signature is checked
(SS10) Interfere with the signature checking procedure
(SS11) Make the decision of signature check go otherwise
(SS12) Change the signature sent back to the phone
(SS13) Make the decision about counts of unsuccessful signature creations take the other path
(SS14) (**moved**) Learn the PIN received from the phone
(SS15) (**moved**) Learn the encrypted first keyshare (encrypted with PIN)
(SS16) (**moved**) Learn the plain first keyshare
(SS17) (**moved**) Change the encrypted first keyshare, before it has been decrypted with the PIN
(SS18) (**moved**) Change the plain first keyshare
(SS19) (**moved**) Change the challenge that enters the computation of the first signature share
(SS20) (**moved**) Learn the first signature share
(SS21) (**moved**) Change the first signature share

For the system T_1, we can again identify certain relationships for $\prec_{\mathrm{D}}$ and $\prec_{\mathrm{S}}$. All the relationships from T' are present, except that some of them have changed their numbers (and also moved their target from the phone to the server). We also assume that the different values the server computes during the signature creation process are equally difficult for an adversary to learn or change; thus $\{\mathrm{SS16}\} \prec_{\mathrm{D}} \{\mathrm{SS3}\}$, $\{\mathrm{SS6}\} \sim_{\mathrm{D}} \{\mathrm{SS20}\} \sim_{\mathrm{D}} \{\mathrm{SS8}\}$ and $\{\mathrm{SS5}\} \sim_{\mathrm{D}} \{\mathrm{SS19}\}$.

We also have relationships between the weaknesses of T' and T_1. We may assume that if $\mathbf{w}$ is a set of weaknesses for both of them (i.e. the names are the same), then $T'.\mathbf{w} \sim_{\mathrm{S}} T_1.\mathbf{w}$. If the targets of the weaknesses in $\mathbf{w}$ are also the same, then also $T'.\mathbf{w} \sim_{\mathrm{D}} T_1.\mathbf{w}$. For example: $\{T'.\mathrm{P4}\} \sim_{\mathrm{S}} \{T_1.\mathrm{SS15}\}$, but $\{T'.\mathrm{P4}\} \not\sim_{\mathrm{D}} \{T_1.\mathrm{SS15}\}$.

Comparison between T' and T_1. We can now write down the formula ϕ that matches each set of attacks against T' with a set of attacks against T_1. Denote $T'.\mathrm{K} \equiv (T'.\mathrm{P3} \wedge T'.\mathrm{P4}) \vee T'.\mathrm{P5}$ and $T_1.\mathrm{K} \equiv (T_1.\mathrm{SS14} \wedge T_1.\mathrm{SS15}) \vee T_1.\mathrm{SS16}$, i.e. "K" denotes the adversary's ability to learn the first keyshare. The formula ϕ is the conjunction of the following statements:

- $T'.\mathrm{RP}i \Leftrightarrow T_1.\mathrm{RP}i$ for $i \in \{1, \ldots, 7\}$
- $T'.\mathrm{B}i \Leftrightarrow T_1.\mathrm{B}i$ for $i \in \{1, 2\}$

– $T'.\mathrm{U}i \Leftrightarrow T_1.\mathrm{U}i$ for $i \in \{1, 2, 3\}$
– $T'.\mathrm{P}i \Leftrightarrow T_1.\mathrm{P}i$ for $i \in \{1, 2, 3\}$
– $T'.\mathrm{P}i \Leftrightarrow T_1.\mathrm{P}(i-7)$ for $i \in \{11, 12\}$
– $T'.\mathrm{P14} \Leftrightarrow T_1.\mathrm{P6}$
– $T'.\mathrm{SS}i \Leftrightarrow T_1.\mathrm{SS}i$ for $i \in \{1, \ldots, 13\}$
– (T'.K and T'.SS3) $\Leftrightarrow$ (T_1.K and T_1.SS3)
– (T'.K and T'.SS6) $\Leftrightarrow$ (T_1.K and T_1.SS6)
– (T'.P9 and T'.SS3) $\Leftrightarrow$ (T_1.SS20 and T_1.SS3)
– (T'.P9 and T'.SS6) $\Leftrightarrow$ (T_1.SS20 and T_1.SS6)

Sets of attacks against the relying party, browser and user have not changed, therefore the sets from the SplitKey system are equivalent to the corresponding sets in the first intermediate system. Also, certain attacks against the phone and the SplitKey server are the same in both systems as these are not connected to the changes we introduced in the first intermediate system.

Attacks involving the adversary learning one or both shares of the key and the signature are the most interesting ones. As the discussion of $\prec_{\mathrm{S}}$ above shows, we can disregard attacks where the adversary learns only one of the shares, as they are no more powerful than learning no shares at all. The last four equivalences above state that if an attack set $\mathbf{w}$ against T' allows the adversary to learn a first share (of the private key, or the signature) and a second share, then we match it with an attack set $\mathbf{w}'$ against T_1 where the adversary also learns the same first share and the same second share. The seriousness of these attacks is the same — $T'.\mathbf{w} \sim_{\mathrm{S}} T_1.\mathbf{w}'$, because the adversary obtains the same knowledge and interferes with the same operations. The difficulty is also the same, although the phone should be easier to attack than the Splitkey server — the set $\mathbf{w}$ contains either T'.SS3 or T'.SS6, hence $\mathbf{w}'$ contains either T_1.SS3 or T_1.SS6, but these are at least as difficult to perform as T_1.SS16 or T_1.SS20.

3.4 Second intermediate system

We change the system T_1 into the system T_2, where instead of computing a combined signature from two signature shares, the Splitkey server generates a single signature using a single private key. The private key is not encrypted with the user's PIN. Instead, the server verifies the correctness of the user's PIN received together with the challenge, by comparing the PIN received from the phone with one stored on the server side. The authentication process in the second intermediate system is depicted in Figure 4.

Weaknesses of the second intermediate system. Attacks against the relying party, browser, user, and phone are the same as for the first intermediate system.

Attacks against the Splitkey server

(SS1) Interfere with the establishment of the secure channel with the phone, thereby obtaining the capability to read and/or change messages sent over it

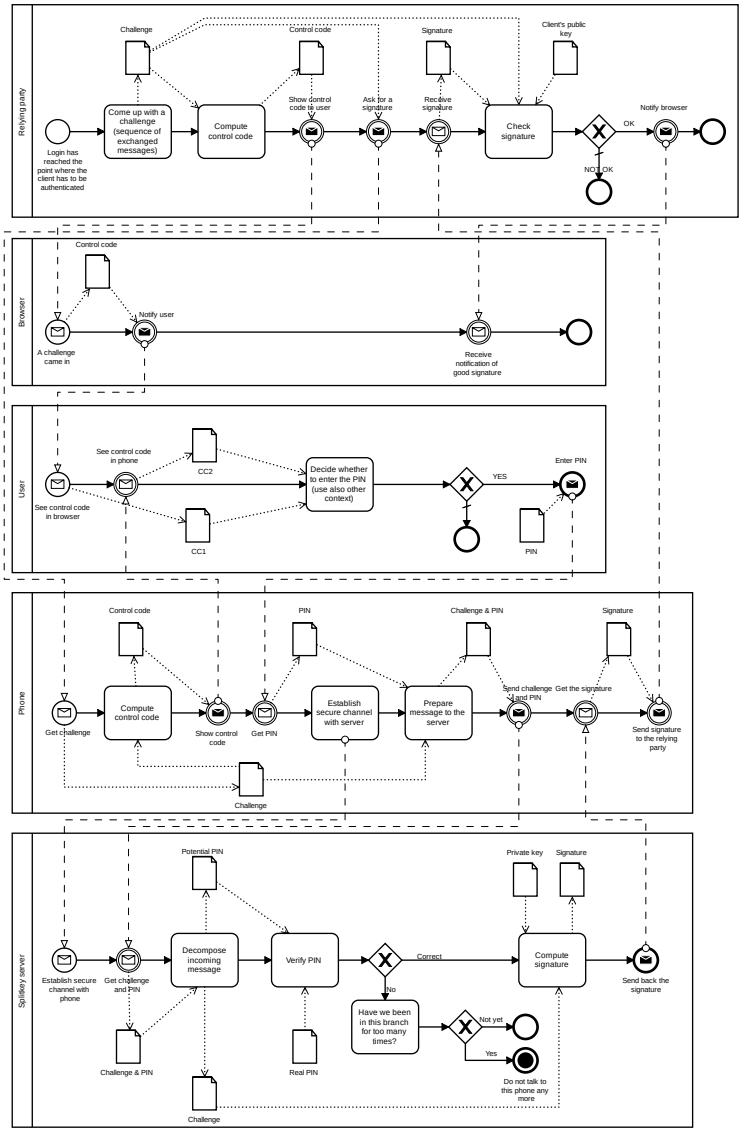


Fig. 4. The second intermediate system

(SS2) Interfere with the establishment of the secure channel with the phone, thereby confusing the server on the identity of the phone
(SS3) Learn the PIN received from the phone
(SS4) Interfere with the PIN verification process
(SS5) Make the decision of PIN check take the other path
(SS6) Make the decision about counts of incorrect PINs take the other path
(SS7) Learn private key
(SS8) Change private key
(SS9) Change the challenge that goes into the signature creation process
(SS10) Learn the signature
(SS11) Change the signature sent back to the phone

The second intermediate system is simpler than the first one because threshold cryptography is no longer used. We again have relationships between weaknesses of T_1 and T_2: we assume that learning the private key from the Splitkey server of T_2 is not harder than learning the second keyshare from the Splitkey server of T_1. We hence have $\{T_2.\text{SS7}\} \prec_\text{D} \{T_1.\text{SS3}\}$. We also have $\{T_1.\text{SS3}\} \prec_\text{S} \{T_2.\text{SS7}\}$: learning the whole private key definitely has at least as serious consequences as learning only one keyshare.

Comparison between T_1 and T_2. We can now write down the formula ϕ that matches each set of attacks against T_1 with a set of attacks against T_2. It is the conjunction of the following statements:

- $T_1.\text{RP}i \Leftrightarrow T_2.\text{RP}i$ for $i \in \{1, \ldots, 7\}$
- $T_1.\text{B}i \Leftrightarrow T_2.\text{B}i$ for $i \in \{1, 2\}$
- $T_1.\text{U}i \Leftrightarrow T_2.\text{U}i$ for $i \in \{1, 2, 3\}$
- $T_1.\text{P}i \Leftrightarrow T_2.\text{P}i$ for $i \in \{1, \ldots, 7\}$
- $T_1.\text{SS}i \Leftrightarrow T_2.\text{SS}j$ for $(i, j) \in \{(1, 1), (2, 2), (14, 3), (10, 4), (11, 5), (13, 6), (8, 10), (12, 11)\}$
- $(((T_1.\text{SS14}$ and $T_1.\text{SS15})$ or $T_1.\text{SS16}$ or $T_1.\text{SS20})$ and $(T_1.\text{SS3}$ or $T_1.\text{SS6})) \Leftrightarrow T_2.\text{SS7}$
- $(T_1.\text{SS4}$ or $T_1.\text{SS17}$ or $T_1.\text{SS18}) \Leftrightarrow T_2.\text{SS8}$
- $(T_1.\text{SS5}$ or $T_1.\text{SS19}$ or $T_1.\text{SS7}) \Leftrightarrow T_2.\text{SS9}$

Sets of attacks against the relying party, browser, user and phone have not changed as we have not introduced changes to these parts of the system. Additionally, certain attacks against the SplitKey server are the same in both systems as these are not related to the changes we introduced in T_2. In T_2, we have a single private key that is stored on the SplitKey server. It means that we can match a combination of attacks that retrieve keyshares from T_1 against a single attack in T_2, where the adversary extracts the private key from the SplitKey server. Similarly, attacks changing challenges, keyshares, or signature shares, or interfering with the combination of signatures in T_1 are no worse than the attack changing the challenge, key, or signature in T_2.

In T_1, the PIN is used to decrypt the first keyshare at the server, and only a single PIN can make the following signature check succeed. Hence it makes sense

to state that the attacks T_1.SS10 on interfering with the signature check, and T_1.SS11 on subsequently changing the control flow, are the same as the attacks T_2.SS4 and T_2.SS5 that interfere with the comparison of PINs and subsequent branching.

3.5 Comparing the second intermediate system and the smartcard system

The attacks against the smartcard system T have been described in Sec. 3.1. There is still a significant difference between T_2 and T, due to the use of different hardware components, but as we see below, we can overcome this by introducing and justifying $\prec_{\mathrm{D}}$-relationships among the attacks against T_2 and T.

Comparison between T_2 and T. The formula ϕ is the conjunction of the following statements:

- $T_2.\mathrm{RP}i \Leftrightarrow T.\mathrm{RP}i$ for $i \in \{1, 2\}$
- $T_2.\mathrm{RP}i \Leftrightarrow T.\mathrm{RP}(i-2)$ for $i \in \{5, 6, 7\}$
- $T_2.\mathrm{U}i \Leftrightarrow T.\mathrm{U}i$ for $i \in \{1, 2\}$
- $T_2.\mathrm{P}i \Leftrightarrow T.\mathrm{SCR}j$ for $(i, j) \in \{(3, 1), (5, 2), (7, 3), (6, 4)\}$
- $T_2.\mathrm{SS}i \Leftrightarrow T.\mathrm{SC}(i-2)$ for $i \in \{3, \ldots, 11\}$
- T_2.P4 or T_2.SS1 $\Leftrightarrow$ T.SCR1 and T.SCR2 and T.SCR3 and T.SC9

Most of the attacks against the relying party and user are the same in both systems. One difference comes from the fact that smartcard system does not have Control Codes, thus, the attacks against them in T_2 are vacuously successful in T. We matched a set of attacks against the phone in T_2 against similar attacks targeting smartcard reader in the smartcard system. We have not matched T_2.RP3, T_2.RP4, T_2.B1, T_2.B2, T_2.P1, and T_2.P2 with anything in T as all of these attacks are related to the Control Codes and the smartcard system does not have this protection mechanism employed.

The set of attacks against the SplitKey server in T_2 is almost identical to the set of attacks against the smartcard in the smartcard system, except attacks related to the communication channel. However, the SplitKey server and smartcard have different hardware, hence we need to introduce the relationships $\{T.\mathrm{SC}i\} \prec_{\mathrm{D}} \{T_2.\mathrm{SS}(i+2)\}$ for the analysis to go through. Also, the attacks related to the communication channel in T_2 are matched with the set of attacks where the adversary learns and changes the values sent over that channel (e.g. PIN, challenge), and this again requires extra $\prec_{\mathrm{D}}$-relationships. These requirements on $\prec_{\mathrm{D}}$ are discussed below.

3.6 Propagation of the security requirements

We described which attacks are equivalent to each other in the neighbouring systems. Now, we analyse these to identify security requirements that should be

in place in the SplitKey system for the above listed attacks to be equivalent to each other.

The communication channel between the phone and SplitKey server must offer the same level of protection as smartcard system offers against SCR1, SCR2, SCR3, SC9 attacks. This requirement arises when we move from the second intermediate system to the smartcard system, as attacks against the communication channel are removed at this point. However, we identified that attacks against the communication channel are equivalent to the attacks against the smartcard system that involve learning and modifying information sent to the smartcard. The requirement of protecting the communication channel propagates through T_2 and T_1 back to T'.

We introduced the relationship $\{T.\text{SC5}\} \prec_{\text{D}} \{T_2.\text{SS7}\}$, requiring that the Splitkey server protects the private key at least as strongly as the smartcard. The same requirement holds for T_1, except that there is no private key in T_1. We can verify that it will be sufficient to strongly protect only the second keyshare: $\{T_2.\text{SS7}\} \prec_{\text{D}} \{T_1.\text{SS3}\}$. The same requirement propagates to T'.

It should be as hard to interfere with the signature verification process in the SplitKey server as it is to interfere with the PIN verification in the smartcard. There are attacks targeting the PIN verification process in the smartcard system and in T_2. However, T_1 and T' have attacks against signature verification. Signature verification in the SplitKey system serves to verify whether the user correctly decrypted their part of the private key to create their signature share, meaning that the user entered the correct PIN code. In the smartcard system, the PIN is verified straightforwardly by comparing the code stored in the card with the one received from the user.

It should be as hard to interfere with the signature creation/combination process in the SplitKey server as it is in the smartcard system. In the smartcard system and in T_2, the adversary can modify the challenge that enters the computation of the signature. In T_1 and T', the adversary can modify challenges for two signature shares — on the phone side and on the server side. If the SplitKey server receives an incorrect signature share from the phone and decides not to discard it but use it further, it means that it can successfully execute attacks against the signature verification process, that were covered in the previous requirement. Therefore, in this requirement, we focus on the server side attacks, where the adversary can not only change the challenge for the second signature share but also interfere with the signature combination process.

It should be as hard to change the server's private key share that is used to create the signature in the Splitkey server as it is in the smartcard system. In the smartcard system and T_2, adversary can attempt to change the single private key that will be used to create signature. In T_1 and T', the adversary may target two shares of the private key. With the modified user's share of the private key, adversary will create a signature share that will not pass the verification procedure. If the server decides to use incorrect signature share, it means that one can execute attacks against signature verification that were covered before. Therefore, this requirement again focuses on the modifications on the server side.

It should be as hard to capture the PIN entered by the user from the phone as it is to capture the PIN from the smartcard reader. Starting from the smartcard system, where the PIN is sent to the smartcard through the smartcard reader this requirement propagates all the way back to the SplitKey system, where the user enters the PIN in the phone. It may be difficult to argue that this requirement is satisfied, if a smartcard reader with PINpad is used in T. It will be easier to argue for it, if computer's keyboard is used in T.

4 Conclusion

We have presented the case that comparison-based arguments can be useful for providing the evidence that a system satisfies certain security properties, and, in particular, that a ToE satisfies a Security Target. It remains to be determined, how scalable the proposed methodology is – how much effort this methodology will require in different practical applications for evolving and novel ToEs, and what kind of tool support will be useful. The proposed methodology also needs evaluation by certification laboratories.

In our example, we have compared authentication systems. We hope that it is possible to similarly show that a threshold cryptography based system may be a qualified electronic signature creation device (QSCD) [6], by again comparing it to systems making use of smartcards.

References

1. Bellare, M., Rogaway, P.: The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In: Vaudenay, S. (ed.) Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4004, pp. 409–426. Springer (2006)
2. Buldas, A., Kalu, A., Laud, P., Oruaas, M.: Server-Supported RSA Signatures for Mobile Devices. In: Foley, S.N., Gollmann, D., Snekkenes, E. (eds.) Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10492, pp. 315–333. Springer (2017)
3. Buldas, A., Saarepera, M.: Electronic Signature System with Small Number of Private Keys. In: Ellison, C.M., Polk, W.T., Hastings, N.E., Smith, S.W. (eds.) NISTIR 7085: 2nd Annual PKI Research Workshop Proceedings, pp. 110–122. National Institute of Standards and Technology (NIST) (2004)
4. De Santis, A., Desmedt, Y., Frankel, Y., Yung, M.: How to share a function securely. In: Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing (STOC '94). p. 522–533. Association for Computing Machinery, New York, NY, USA (1994)
5. Dupont, S., Ginis, G., Malacario, M., Porretti, C., Maunero, N., Ponsard, C., Massonet, P.: Incremental Common Criteria Certification Processes using DevSecOps Practices. In: IEEE European Symposium on Security and Privacy Workshops, EuroS&P 2021, Vienna, Austria, September 6-10, 2021. pp. 12–23. IEEE (2021)

6. European Parliament and Council of European Union: Regulation (EU) no 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC. OJ L 257, 28.8.2014, p. 73-114 (2014)
7. Hernandez-Ardieta, J.L., Blanco, P., Vara, D.: A methodology to construct Common Criteria security targets through formal risk analysis. In: Proceedings of XII Spanish Meeting on Cryptology and Information Security (RECSI 2012) (2012)
8. ISO/IEC 15408-1/2/3:2005 - Information technology — Security techniques — Evaluation criteria for IT security
9. Keblawi, F., Sullivan, D.: Applying the Common Criteria in Systems Engineering. IEEE Security and Privacy **4**(2), 50–55 (2006)
10. National Computer Security Center: Rating Maintenance Phase Program Document Version 2. Rainbow Series, NCSC-TG-013 V2 (1995), https://web.archive.org/web/20110720184904/http://iaarchive.fi/Rainbow/NCSC-TG-013%20PINK%20version%202.pdf
11. OMG: Business Process Model and Notation (BPMN), http://www.omg.org/spec/BPMN/2.0/
12. PP-Module for User Authentication Devices, Version 1.0. National Information Assurance Partnership (Jul 2019)
13. prEN 14169-1:2009: Protection profiles for Secure signature creation device — Part 2: Device with key generation. Technical Committee CEN/TC 224 (Dec 2009)
14. Pullonen, P., Matulevičius, R., Bogdanov, D.: PE-BPMN: privacy-enhanced business process model and notation. In: Carmona, J., Engels, G., Kumar, A. (eds.) Business Process Management - 15th International Conference, BPM 2017, Barcelona, Spain, September 10-15, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10445, pp. 40–56. Springer (2017)
15. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Aug 2018). https://doi.org/10.17487/RFC8446, https://www.rfc-editor.org/info/rfc8446
16. Reuse of Evaluation Results and Evidence (Oct 26th 2002), information Statement on behalf of the Common Criteria Recognition Arrangement Management Committee, Document no. 2002-08-009-002
17. Salnitri, M., Dalpiaz, F., Giorgini, P.: Designing secure business processes with SecBPMN. Softw. Syst. Model. **16**(3), 737–757 (2017)
18. Sinnhofer, A.D., Raschke, W., Steger, C., Kreiner, C.: Evaluation paradigm selection according to Common Criteria for an incremental product development. In: Tverdyshev, S. (ed.) International Workshop on MILS: Architecture and Assurance for Secure Systems, MILS@HiPEAC 2015, Amsterdam, The Netherlands, January 20, 2015. Zenodo (2015)
19. Sinnhofer, A.D., Raschke, W., Steger, C., Kreiner, C.: Patterns for Common Criteria Certification. In: Link, C., Eloranta, V. (eds.) Proceedings of the 20th European Conference on Pattern Languages of Programs, EuroPLoP 2015, Kaufbeuren, Germany, July 8-12, 2015. pp. 33:1–33:15. ACM (2015)
20. Sun, N., Li, C., Chan, H., Le, B.D., Islam, M.Z., Zhang, L.Y., Islam, M.R., Armstrong, W.: Defining security requirements with the common criteria: Applications, adoptions, and challenges. IEEE Access **10**, 44756–44777 (2022)

Appendix 6

Publication V

J. Vakarjuk, A. Pankova, and A. Kalu. Comparison of private key protection techniques for digital identity wallets. In *2026 IEEE European Symposium on Security and Privacy Workshops*, pages 396–412, 2026

Comparison of Private Key Protection Techniques for Digital Identity Wallets

Jelizaveta Vakarjuk
Cybernetica AS
Tallinn University of Technology
Tallinn, Estonia
jelizaveta.vakarjuk@cyber.ee

Alisa Pankova
Cybernetica AS
Tartu, Estonia
alisa.pankova@cyber.ee

Aivo Kalu
Cybernetica AS
Tallinn, Estonia
aivo.kalu@cyber.ee

***Abstract*—Considering the increased attention to the digital identity management systems, mandated by the eIDAS regulations, there are many open questions to consider, including protection of the user's private key. This work examines the ways of private key protection mechanisms for EU digital identity wallets. As currently developed solutions often rely on mobile-based wallet applications, this study analyses different approaches of protecting private key on mobile devices: systems with hardware-protected keys (local and remote), threshold cryptography, and all-or-nothing non-transferability. The goal is to evaluate whether these methods achieve equivalent security when fulfilling system requirements. Using Laud-Vakarjuk (SecAssure 2022) methodology grounded in cryptographic security proofs, the research compares newer systems (potentially employing non-standard cryptographic algorithms) against established ones that rely on hardware protection, ensuring they meet baseline security requirements. By identifying security guarantees and formalizing threat models, the analysis demonstrates that all four approaches can provide comparable security when implemented with appropriate protection mechanisms.**

***Index Terms*—identity management, key protection, security comparison**

I. Introduction

With the updated eIDAS regulation [1], EU member states are required to provide its members with the digital identity wallet solution (EUDI wallet). Digital identity wallet is aimed at providing users with fully mobile solutions enabling presentation of verifiable credentials to various relying parties. A verifiable credential contains claims about the user's attributes, signed by the issuer's private key. A verifiable presentation of the credential contains the verifiable credential signed by the user's private key. The EUDI wallet framework has several requirements for its architecture, including that the credential issued to the wallet must be bound to the hardware-protected device key and must be protected according to Level of Assurance (LoA) "high" requirements [1] (§29). This is often solved with the help of secure elements (SEs) or Remote Hardware Security Modules (HSM). On the other hand, there are other approaches explored in the literature that aim at protecting user's private key without relying purely on usage of SE. One example is usage of threshold cryptography, which allows to split a private key between mobile device and supporting server [2]. Both hardware based and threshold cryptography based approaches ensure that the private key is bound to the device, and possession of the corresponding private key (bound to the device) is checked by a verifier.

Another approach of binding the user's private key is discouraging user from sharing that key; this mechanism is called all-or-nothing non-transferability [3]. This approach focuses on the user of the system, preventing them from partly sharing their credentials with another user, as sharing one credential will result in sharing all the user's credentials. An alternative approach to non-transferability is adding biometric access control to the system [4], which can be integrated with the threshold cryptography approach [1]. The importance of non-transferability is best illustrated by its role in preventing unauthorized sharing or misuse of verifiable credentials. For example, when users present proofs of age or driver's licenses, non-transferability ensures that these credentials cannot be forwarded or reused by third parties. Verifiers, such as law enforcement agencies, rely on this property to securely request and validate attributes for access control and verification purposes [4].

In this work, we explore and analyse different ways of protecting user's private keys that are required to create verifiable presentation of the user's credential. The systems under analysis are the systems with hardware protected key (locally in a SE of the mobile device, or remotely in a Remote HSM), the system relying on threshold cryptography, and the system relying on all-or-nothing non-transferability (anonymous credentials). We reason that Remote HSM, threshold cryptography, and anonymous credential based systems are at least as secure as the system that locally stores the private key in the SE of the mobile device, under assumption that certain system requirements are satisfied. To analyse the systems, we use Laud-Vakarjuk methodology [5]. This methodology was created based on the idea of cryptographic security proofs to show that some system (possibly relying on non-standard cryptographic algorithms) is at least as secure as an already standardised system. We note that, while several analytical frameworks are available, a comprehensive comparison of methodological choices falls outside the scope of this work.

This paper is organised as follows. We start with defining the comparison methodology in Section II-A. We proceed with an

[1] https://cyber.ee/products/splitkeyplus/ (last visited 22nd April, 2026)

overview of the compared systems in Section II-B. The actual modelling of the systems, their comparison, and analysis of the results are presented in Section III. Finally, we conclude our paper in Section IV.

II. Background

A. Comparison Methodology

One of the possible approaches that allow to compare security guarantees of systems relying on cryptography is the methodology by Laud et al. [5]. It has been used to provide comparison between threshold cryptography based authentication and the smartcard based authentication [5]. Additionally, it has been applied to compare security guarantees of the i-voting system to the paper-based postal voting [6]. Therefore, we find it suitable for the purposes of analysing private key protection techniques for digital identity wallets.

The first step of the comparison methodology of [5] is to establish two systems S and S' and the processes that we want to compare. In this framework, S denotes the *reference* system, whose security properties have already been formally analysed or are supported by established certifications and standards. Contrarily, S' represents the *novel* system under evaluation. The objective is to demonstrate that the system S' achieves security guarantees at least equivalent to those of the system S. To facilitate a structured comparison and enable analysis of the chosen processes, the methodology recommends modelling both S and S' using BPMN diagrams.

Building on the BPMN models, we systematically identify potential weaknesses of both S and S', by analysing each data object, computation step (task), and decision point (gateway) of the BPMN diagrams. Table I classifies the main types of weaknesses mapped to these BPMN components. Within our threat model, exploitation of weakness is treated as a binary condition: an adversary either fully exploits a weakness or does not exploit it at all; partial or graded exploitation is excluded. This analytical approach helps to construct a list of potential attack vectors, capturing adversary's interaction with the system, including extreme or highly improbable scenarios such as direct private key compromise. Although certain weaknesses may be practically infeasible to exploit (e.g., extracting a key from a Hardware Security Module), we deliberately retain them in the analysis to identify corresponding security requirements.

This approach mirrors the structure of cryptographic security proofs. In such proofs, we typically assume that the adversary operates within a well-defined model, and aim to reduce the adversary's ability to break the system to solving some hard computational problem – such as the Decisional Diffie-Hellman (DDH) assumption, which is believed to be infeasible. Similarly, following this methodology, the goal is to trace potential attacks back to the fundamental security assumptions about the system components. This, for example, can help us to learn that, in order to provide the same security guarantees as the system S, the system S' should store some component inside HSM. With this methodology, we do *not* analyse technical cryptographic details of the protocols and cryptographic assumptions (like difference between RSA-based or pairing-based constructions), as the goal is to analyse weaknesses on the higher level of the system.

After all the potential weaknesses have been identified, the next step requires a systematic comparison of the listed weaknesses of S and S'. It is achieved by finding relations between the weaknesses. There are two types of relations that are used in this work – the *difficulty of exploiting* the weakness, and the *seriousness of consequences* if the weakness is exploited. For every weakness present in S', we aim to identify an equivalent weakness in S. If there is a weakness in S that has no equivalent in S', this indicates that S' offers stronger security guarantees with respect to that specific threat vector.

The Laud–Vakarjuk methodology [5] also provides an approach for comparing systems that lack sufficient structural or functional alignment to enable direct weakness mapping. However, the systems analysed in this work are similar enough to be comparable without introducing intermediate systems as defined in [5].

B. Credential systems

There is an Architecture and Reference Framework (ARF) for EUDI wallet, which consists of common standards, technical specifications, common guidelines and best practices for implementing European Digital Identity Framework [7]. ARF lists the following possible security architectures for Wallet Secure Cryptographic Device (WSCD):

1) a remote device, such as a Hardware Security Module (HSM), accessed over a network;
2) an external device, such as a smart card issued to the user specifically for this purpose;
3) a component within the user device, such as a SIM, e-SIM, or embedded Secure Element;
4) a component embedded in the user device and accessed via an API provided by the operating system.

Considering the regulation requirement of wallet to be *fully mobile* [1] (§29) and the proposed use cases for the EUDI wallet, options 1, 3, and 4 are the most fitting. Requiring user having access to the smart-card constrains the way user can interact with the wallet ecosystem.

Next, we describe the four systems compared in this paper, starting with the approach where WSCD can be implemented using options 1, 3, and 4 – the hardware-based credentials. Futher, we present detailed BPMN models defining presentation process of verifiable credential for each system.

a) Hardware-based credentials: The user's private key that is used to create verifiable credential presentation is stored in SE (Secure Element). The simplified presentation process is presented on Figure 1, summarised in the following steps:

0) User initiates presentation process through their mobile application and selects a credential for presentation.
1) Mobile device requests the user's knowledge factor (e.g. PIN/biometrics).
2) Mobile device passes the knowledge factor to SE for verification.

Component of diagram	Weakness types	Example
Data object	**Learn** – get unauthorised access to data object while in store or transit **Modify** – tamper with data object while in store or transit	Learn the private key. Meaning that adversary can use it for their purposes. Modify the private key. Meaning that the private key that is used further in the protocol is modified. This does not mean that adversary gets to learn the private key.
Task	**Interfere with** – change the way the task should be executed (including input and output data objects)	Interfere with the presentation decomposition process. Meaning that structure consists of several data objects – credential, signature, etc. and adversary may modify any of them.
Gateway	**Make decision go other way** – influence the outcome of decision point	Make the decision on issuer's signature verification go other way. Meaning that adversary can make the verifier accept signature even if it is not correct or reject correct signature.

TABLE I
MAIN WEAKNESS TYPES

3) Upon successful verification, mobile device asks SE to create a signature on the presentation of the selected credential using the user's private key.
4) Mobile device sends the signed credential presentation to the Relying Party (RP).
5) RP verifies the signature on the presentation using the user's public key, and verifies the credential using the issuer's public key.

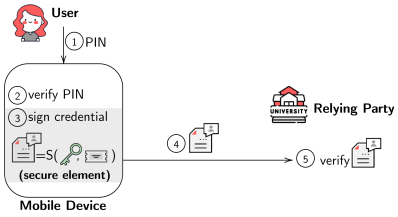


Fig. 1. Hardware-based credential

Protection of the user's private key is achieved through storing it inside SE from which the key cannot be extracted. The user cannot share its private key with other parties without giving full access to the device and the knowledge factor.

b) Remote HSM: The description of the Remote HSM approach is based on architecture document for German National EUDI Wallet [8]. The simplified presentation process is presented on Figure 2, summarised in the following steps:

0) User initiates presentation process through their mobile application and selects a credential for presentation.
1) Mobile device requests the user's knowledge factor (e.g. PIN/biometrics) that will be used to authenticate the user *towards Remote HSM*.
2) Mobile device sends an authentication request to Remote HSM.
3) Remote HSM verifies the authentication request.
4) Mobile device constructs a credential presentation and sends a signing request to Remote HSM.
5) Remote HSM signs the credential presentation and sends it to the mobile device.
6) Mobile device sends the credential presentation to the Relying Party (RP).
7) RP verifies the signature on the presentation using the user's public key, and verifies the credential using the issuer's public key.

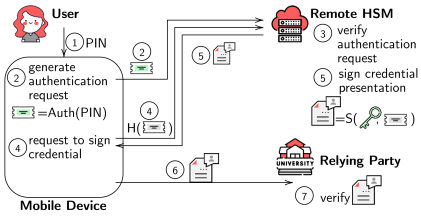


Fig. 2. Remote HSM-based credential

With this approach, the user's private key that is used to sign the credential presentation is stored in Remote HSM, and there is a 2-factor authentication mechanism in use to authenticate the user towards Remote HSM by creating signature on the challenge received from Remote HSM:

- *possession factor*: authentication private key that is stored on the device (stored private key);
- *knowledge factor*: authentication private key derived from the user's PIN (recomputed private key). It is not stored on the device, but is freshly generated from PIN for each authentication.

c) Anonymous credentials: Carsten Baum et al. [9] have given a feedback to the European Digital Identity ARF, pointing out that the currently proposed solution does not meet important properties that the system should achieve according to the regulation. As a solution, [9] proposes to use anonymous credentials as they were designed specifically to provide authentication and identification services that are both secure and privacy-preserving. Anonymous credential systems enable the user to anonymously prove a specific claim about their attributes to the verifier while staying anonymous. In the anonymous credential system proposed by Camenisch and Lysyanskaya [3], there is a notion and mechanism of *all-or-nothing non-transferability*. The aim of this mechanism is to discourage the user from sharing their private key with other parties. It is achieved through connecting all the credentials with a single *master secret key* and publishing partial credentials. If the user shares their master secret key with some party

(including the adversary), this party can look at the published records of all the issuers and obtain information about all the credentials of the user. This knowledge (that transferring the right to present a single credential leads to leaking all the credentials) prevents the user from willingly transferring this right to the other users. The issuing process of [3] with all-or-nothing non-transferability is outlined as follows:

1) User interacts with the issuer. They establish a pseudonym N_i (a value by which the user is known to the issuer) and a validating tag P_i, computed using the user's master secret key and a randomness to which both user and the issuer contribute.
2) For the non-transferability, the user additionally creates a ciphertext c_i, which is verifiable encryption of the validating tag randomness under the user's public key.
3) Issuer publishes the pseudonym N_i and the ciphertext c_i.
4) If N_i is eligible to get a credential on P_i, the issuer produces a credential by signing a statement containing P_i. The issuer sends the credential to the user and publishes the signature.
5) *Non-transferability*: if the user shares its master secret key with some party, this party can look at the published records of all the issuers and obtain information about all user's credentials through decrypting ciphertexts c_i.

The simplified presentation process is presented on Figure 3. The presentation process can be summarised as follows:

0) User initiates presentation process through their mobile application and selects a credential for presentation.
1) Mobile device proceeds without additional user authentication.
2) Mobile device generates a presentation containing zero-knowledge proofs (ZKP), convincing RP:
   - of possessing a signature generated by the issuer on a statement containing P_i (Proof of Possession, PoP),
   - of knowing the master secret key related to P_i.
3) Mobile device sends the presentation (containing ZKPs) to RP. It does *not* send the actual credential to RP.
4) RP verifies the proofs in the presentation.

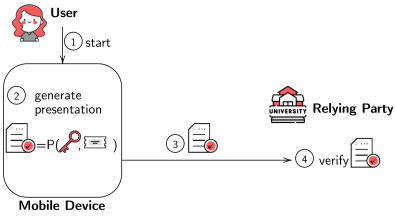


Fig. 3. Anonymous credentials

For the analysis and comparison below, we use the description as it is presented in [3], without assuming usage of SE or any other measures of protecting private key. Since the system description assumes that the user is not prevented from sharing their master secret key, it is plausible assumption. However, we note that it is possible to implement this system with additional protection measures as is done with one of the versions of the system in Yivi application [2].

d) Threshold cryptography: Another approach of protecting the user's private key, explored in the literature and used in practice, is relying on threshold cryptography. One of the examples is authentication and signing solution widely adopted in Baltic states[3], which relies on the server-supported RSA signature scheme by Buldas et al. [2]. Another scheme that is proposed to be directly applied to the digital identity wallet solutions is scheme by Dijkhuis [10], which combines usage of threshold signatures and SE. The scheme uses two signature key pairs – ECDSA key, which is stored in SE and FROST [11] key, which is distributed. Finally, in the Yivi [4] wallet application, there is also combined approach implemented. One secret key is stored in SE, and the second one is stored in distributed way.

In the description below, we do not rely on approaches of [10] or Yivi, since they use more nuanced schemes providing features that we do not aim to include in our comparison. We note that adding those properties to the systems will make them stronger than the simpler system analysed in this paper. The presentation process that we use is similar to the hardware-based credential, with the difference in the way how the user's signature is generated. Instead of generating signature inside SE, we use threshold signature scheme of Buldas et. al [2] to generate functionally interchangeable signature, i.e. which can be verified as an ordinary non-threshold signature. The simplified presentation process is presented on Figure 4 and can be summarised in the following steps:

0) User initiates presentation process through their mobile application and selects a credential for presentation.
1) Mobile device requests user's PIN.
2) Mobile device uses PIN to decrypt the first key share.
3) Mobile device proceeds with creating the first signature share using the decrypted first key share, and sends the obtained signature share to the supporting server.
4) Supporting server creates second signature share using second key share, combines two signature shares, and verifies the resulting signature using user's public key. Upon successful verification, supporting server sends combined signature to the mobile device.
5) Mobile device constructs credential presentation and sends it for the verification to the RP.
6) RP verifies signature on presentation using the user's public key, and verifies the credential using the issuer's public key.

Protection of the user's private key is ensured through generating private key in a threshold manner, storing the first share of the key on the user's device protected by the user's PIN, and storing the second share on the supporting server (possibly inside HSM). Thus, the adversary corrupting user's

[2]https://docs.yivi.app/keyshare-protocol (last visited 22nd April, 2026)
[3]https://www.smart-id.com (last visited 22nd April, 2026)
[4]https://docs.yivi.app (last visited 22nd April, 2026)

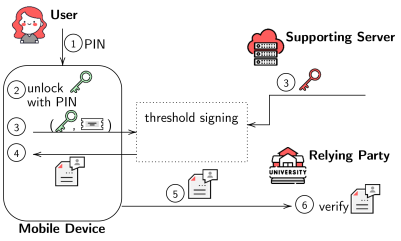


Fig. 4. Server-supported credentials

device is not able to create signatures without participation of the supporting server. Additionally, scheme [2] provides protection against offline guessing attacks on PIN and clone detection. Clone detection offers protection against "silent" attacks, where adversary copies the mobile device memory (without getting full access to the device).

One of the advantages of using this system, is that scheme [2] includes user authentication towards server (through PIN) as a part of the signing process. So there is no need for a separate authentication protocol that should be executed before signing.

III. Comparison

In this section, we will be using the following notation:

- HW system with hardware-based credentials;
- RH system with remote HSM;
- SS system with server-supported threshold signing;
- AN anonymous credential system based on all-or-nothing non-transferability.

The system with hardware-based credentials HW can be viewed as the simplest system. Intuitively, the other three systems should provide even stronger guarantees. However, additional protection mechanisms may introduce additional attack vectors that have not existed in HW. In this section, we present comparison of RH, SS and AN against the baseline system HW, coming up with sets of requirements that need to be satisfied in order to keep the systems RH, SS and AN at least as secure as HW. For this, we follow methodology of Section II-A. The analysis involves the following steps:

1) Construct BPMN diagrams for each system HW, RH, SS, AN, based on their descriptions from Sec. II-B.
2) Extract all potential weaknesses from every data object, task, and gateway of the BPMN diagrams, as in Table I.
3) Filter out repetitive and "irrelevant" weaknesses, taking into account that changing a value has no greater effect than changing all values computed from it. E.g. we can exclude the weakness *modify the selected credential identifier* if there is already the weakness *interfere with the selection of credential identifier* present.
4) For each relevant weakness (or their combination) of RH, SS, AN, look for an *equivalent* weaknesses in HW, i.e. exploiting which has the same impact on the system.
5) Compare (in terms of difficulty of exploitation) the combinations of weaknesses that can be exploited in different systems to run particular attacks against the wallet system.

A. BPMN diagrams and the lists of weaknesses

In Figures 9, 10, 11, 12 (given in the appendix), we present the BPMN models on which the analysis is based (analysis step 1). The name of each task, each gateway, and each data object contain (in parentheses) the label of the corresponding weakness. For a data object, we provide a pair of weaknesses, where the first component is the **learning**, and the second component is the **modification**, as described in Table I. As was mentioned before, we do not focus on weaknesses coming from underlying cryptographic primitives, so e.g. the proof forgery attack is covered by the weakness *interfere with the proof verification*.

In Figures 5, 6, 7, 8, we present the lists of identified weaknesses (analysis step 2). Weaknesses are classified by the party on whose side the weakness is exploited. In these figures, some weaknesses of the systems RH, SS, AN already contain a reference (in square brackets) to the corresponding "obviously" equivalent weakness of the system HW (e.g. $\mathsf{D1}[\mathsf{HW.D1}]$). Additionally, whenever exploiting the weakness W_0 is no more serious than exploiting the weakness W_1 (e.g. W_1 modifies the output and W_0 the input), we add a reference from W_0 to W_1 (e.g. $\mathsf{D}_5[\mathsf{D}_6]$), so that only W_1 will need to be considered further (analysis step 3).

B. Relations between weaknesses

When comparing the weaknesses (analysis step 4), we follow notation of [5]. We write $\mathsf{W}_1 \prec_{\mathrm{S}} \mathsf{W}_2$ to denote that consequences of exploiting weakness W_1 have no greater effect than exploiting weakness W_2 (**seriousness**). We write $\mathsf{W}_1 \prec_{\mathrm{D}} \mathsf{W}_2$ to denote that exploiting weakness W_1 is no more difficult than exploiting weakness W_2 (**difficulty**) and $\prec_{\mathrm{X}}$ to denote both seriousness and difficulty. We write $\mathsf{W}_1 \sim_{\mathrm{X}} \mathsf{W}_2$ to denote that weaknesses are equivalent in consequences of exploiting and difficulty of exploiting. We write $\mathsf{W}_1 \sim_{\mathrm{S}} \mathsf{W}_2$ to denote that weaknesses are equivalent in consequences of exploiting and $\mathsf{W}_1 \sim_{\mathrm{D}} \mathsf{W}_2$ in difficulty of exploiting. If for $\mathsf{W}_1 \neq \mathsf{W}_2$, it holds that $\mathsf{W}_1 \prec_{\mathrm{X}} \mathsf{W}_2$ and $\mathsf{W}_2 \prec_{\mathrm{X}} \mathsf{W}_1$, we write $\mathsf{W}_1 \sim_{\mathrm{X}} \mathsf{W}_2$. Notation $\{\mathsf{W}_1, \mathsf{W}_2\}$ means exploiting both W_1 and W_2.

a) Weaknesses of hardware-based system: Figure 5 presents identified weaknesses of the hardware-based system. There are the following relations between weaknesses within the system HW:

- getting access to the user's mobile device, results in learning everything that is stored there (except private key and PIN inside SE): $\{\mathsf{D6}, \mathsf{D10}, \mathsf{D12}, \mathsf{D17}\} \prec_{\mathrm{S}} \mathsf{U4}$;
- getting access to the user's mobile device is no more difficult than getting access to everything that is stored there (except private key and PIN inside SE): $\mathsf{U4} \prec_{\mathrm{D}} \{\mathsf{D6}, \mathsf{D10}, \mathsf{D12}, \mathsf{D17}\}$.

Weaknesses of the user side:

U1 Interfere with selection of credential to present
U2 Learn the knowledge factor (PIN) from the user
U3 Modify the knowledge factor (PIN) on the user side
U4 Influence user in giving out mobile device
U5 Modify the list of credential identifiers (displayed by the mobile device)
U6 Learn the list of credential identifiers
U7 Learn the selected credential identifier
U8 Modify the selected credential identifier

Weaknesses of the mobile device:

D1 Display incorrect credential identifiers
D2 Learn the knowledge factor (PIN) entered by the user
D3 (SE) Learn the knowledge factor (PIN) stored for verification
D4 (SE) Interfere with the knowledge factor (PIN) verification process
D5 Proceed with signing even if the knowledge factor (PIN) does not check
D6 (SE) Get access to signing by the user's private key (w/o extracting the key)
D7 (SE) Learn user's private key
D8 (SE) Modify user's private key
D9 Modify the message that enters SE for signing process
D10 Learn the signature that was created by SE
D11 Modify the signature that was created by SE
D12 Learn the credential that corresponds to the selected identifier
D13 Modify the credential identifier selected by the user
D14 Interfere with search of the selected credential
D15 Interfere with construction of the credential presentation
D16 Modify credential presentation (to be sent to the RP)
D17 Learn user's public key (used as a part of credential presentation)
D18 Modify user's public key (used as a part of credential presentation)
D19 Learn credential presentation (to be sent to the RP)
D20 Change the decision about too many failed PIN checks
D21 Modify presentation challenge (received from RP)
D22 Modify knowledge factor (PIN) entered by the user
D23 (SE) Modify knowledge factor (PIN) stored in the device for verification
D24 (SE) Interfere with signing by the user's private key
D25 Learn presentation challenge (received from RP)
D26 Learn the list of credential identifiers (stored in the device)
D27 Learn the credential identifier selected by the user
D28 Modify the credential chosen according to the selected identifier

Weaknesses of the RP:

RP1 Modify presentation challenge (to be sent to the mobile device)
RP2 Interfere with the presentation decomposition process
RP3 Interfere with the user's signature verification process
RP4 Make the decision on user's signature verification go other way
RP5 Interfere with the issuer's signature verification process
RP6 Make the decision on issuer's signature verification go other way
RP7 Modify issuer's public key (stored on the RP side)
RP8 Modify credential presentation (received from the user)
RP9 Learn user's public key (extracted from the decomposed presentation)
RP10 Learn issuer's signature (extracted from the presentation)
RP11 Learn user's signature (extracted from the presentation)
RP12 Learn credential presentation (received from the user)
RP13 Learn credential (extracted from the presentation)
RP14 Learn presentation challenge (generated by the RP)
RP15 Modify presentation challenge (generated by the RP)
RP16 Modify issuer's signature (extracted from the presentation)
RP17 Learn issuer's public key (stored on the RP side)
RP18 Modify user's public key (extracted from the presentation)
RP19 Modify user's signature (extracted from the presentation)
RP20 Modify credential (extracted from the presentation)

Fig. 5. Weaknesses of hardware-based credentials. We denote by (SE) weaknesses targeting operation in Secure Element.

b) Weaknesses of remote HSM-based system: Figure 6 presents identified weaknesses of the system based on Remote HSM. There are the following relations between weaknesses within the system RH:

- learning both the PIN and the seed stored on the device results in learning reconstructed private key: $\mathsf{D}4 \prec_{\mathrm{X}} \{\mathsf{D}2, \mathsf{D}3\}$ and $\mathsf{D}4 \prec_{\mathrm{X}} \{\mathsf{U}2, \mathsf{D}3\}$;
- learning both *recomputed* private key and *stored* private key results in gaining access to the HSM signing functionality: $\mathsf{HSM}4 \prec_{\mathrm{S}} \{\mathsf{D}4, \mathsf{D}7\}$; // assuming that it is as hard to get the key from SE of mobile device as from HSM, the relation can be $\prec_{\mathrm{X}}$
- tampering with creation of authentication request signature (to be sent to HSM) has no greater effect than interfering with computation of the subsequent signing request and modifying the resulting signed authentication request (sent to HSM): $\mathsf{D}16 \prec_{\mathrm{S}} \{\mathsf{D}12, \mathsf{D}15\}$;
- interfering with the authentication request verification process has no greater effect than making the authentication request verification decision go other way and modifying message that enters HSM for signing process: $\mathsf{HSM}2 \prec_{\mathrm{S}} \{\mathsf{HSM}3, \mathsf{HSM}7\}$;
- tampering with the request on the device (sender) side has no greater effect than interfering with processing the request on the server (receiver) side: $\mathsf{D}13 \prec_{\mathrm{S}} \mathsf{HSM}2$;
- modifying the sent signature on the server (sender) side has no greater effect than modifying it on the device (receiver) side: $\mathsf{HSM}9 \prec_{\mathrm{S}} \mathsf{D}14$;
- the stored seed contains no more information than the recomputed key: $\mathsf{D}3 \prec_{\mathrm{S}} \mathsf{D}4$;
- authentication request contains no more information than the recomputed authentication private key and the authentication challenge $\mathsf{D}33 \prec_{\mathrm{S}} \{\mathsf{D}4, \mathsf{D}31\}$;
- presentation signing request contains no more information than the credential and the presentation challenge $\mathsf{D}34 \prec_{\mathrm{S}} \{\mathsf{D}19, \mathsf{D}28\}$;
- message to remote HSM contains no more information than the authentication and presentation signing requests: $\mathsf{D}36 \prec_{\mathrm{S}} \{\mathsf{D}33, \mathsf{D}34\}$ and $\mathsf{HSM}14 \prec_{\mathrm{S}} \{\mathsf{D}33, \mathsf{D}34\}$;
- learning authentication challenge is not considered a leakage by itself: $\mathsf{HSM}12 \vee \mathsf{D}31 \sim_{\mathrm{S}} \{\}$.

This allows to remove D3, D13, D16, D31, D33, D34, D36, HSM2, HSM9, HSM12, HSM14 from the set of new relevant weaknesses.

For the new relevant weaknesses of system RH, Table II presents equivalent weaknesses in system HW. Relation connected to the private key (learning and modifying) is equivalent to the corresponding relation in HW in both seriousness and difficulty, as in both systems the private key is stored in protected environments (HSM or SE). Relations connected to gaining access to the SE (or HSM) signing functionality are also equivalent in both seriousness and difficulty. The reason is that in system RH, one of the user's authentication private keys (that is used to get access to Remote HSM signing) is stored in SE. There are more attacks in RH that are related to the interfering with knowledge factor verification procedure. They are equivalent in seriousness, but not equivalent in difficulty, since PIN verification in HW happens inside SE.

c) Weaknesses of threshold cryptography-based system: Figure 7 presents identified weaknesses of the system based on threshold cryptography. There are the following relations between weaknesses within the system SS:

Weaknesses of the user side: Same as for HW.
Weaknesses of the mobile device:

D1 [HW.D1] Display incorrect credential identifiers
D2 [HW.D2] Learn knowledge factor (PIN) entered by the user
D3 **[NEW] Learn the seed used to generate recomputed private key**
D4 **[NEW] Learn the recomputed private key**
D5 [D6] Modify the (locally stored) seed before generating recomputed key
D6 **[NEW] Interfere with generation of recomputed key (from seed and PIN)**
D7 **[NEW] (SE) Learn the stored authentication private key**
D8 **[NEW] (SE) Get access to signing by the stored authentication private key (w/o extracting the key)**
D9 **[NEW] Modify authentication challenge received from Remote HSM**
D10 [D16] (SE) Modify the stored authentication private key
D11 [D16] Interfere with computation of auth. request (to be signed by local SE)
D12 [D13] Interfere with computation of signing request (to be sent to HSM)
D13 **[NEW] Interfere with creation of the message sent to remote HSM**
D14 **[NEW] Modify the signature received from the remote HSM**
D15 [D13] Modify signed authentication request (to be sent to HSM)
D16 **[NEW] (SE) Interfere with auth. request signature creation**
D17 [HW.D15] Interfere with construction of the credential presentation
D18 [HW.D16] Modify credential presentation (to be sent to the RP)
D19 [HW.D12] Learn credential presentation (to be sent to the RP)
D20 [HW.D13] Modify the credential identifier selected by the user
D21 [HW.D14] Interfere with search of the selected credential
D22 [HW.D17] Learn user's public key (used as a part of credential presentation)
D23 [HW.D18] Modify user's public key (used as a part of credential presentation)
D24 [HW.D19] Learn credential presentation (to be sent to the RP)
D25 [HW.D21] Modify presentation challenge (received from RP)
D26 [HW.D22] Modify knowledge factor (PIN) entered by the user
D27 [HW.D10] Learn the signature received from remote HSM
D28 [HW.D25] Learn presentation challenge (received from RP)
D29 [HW.D26] Learn the list of credential identifiers (stored in the device)
D30 [HW.D27] Learn the credential identifier selected by the user
D31 **[NEW] Learn authentication challenge (received from HSM)**
D32 [D6] Modify the recomputed private key
D33 **[NEW] Learn signed authentication request (to be sent to HSM)**
D34 **[NEW] Learn presentation signing request (to be sent to HSM)**
D35 [D13] Modify presentation signing request (to be sent to HSM)
D36 **[NEW] Learn the message for remote HSM**
D37 [D13] Modify the message for remote HSM
D38 [HW.D28] Modify the credential chosen according to the selected identifier

Weaknesses of the Remote HSM:

HSM1 **[NEW] Interfere with authentication challenge generation**
HSM2 **[NEW] Interfere with the authentication request verification process**
HSM3 **[NEW] Make the auth. request verification decision go other way**
HSM4 **[NEW] (HSM) Get access signing by the user's private key (w/o extracting the key)**
HSM5 **[NEW] (HSM) Learn the user's private key**
HSM6 **[NEW] (HSM) Modify the user's private key**
HSM7 **[NEW] Modify the message that enters HSM for signing process**
HSM8 **[NEW] Learn the signature that was created by HSM**
HSM9 **[NEW] Modify the signature (to be sent to the mobile device)**
HSM10 **[NEW] Change the decision about too many failed auth. requests**
HSM11 **[NEW] (HSM) Interfere with signing by the user's private key**
HSM12 **[NEW] Learn authentication challenge (to be sent to the device)**
HSM13 [HSM1] Modify authentication challenge (to be sent to the device)
HSM14 **[NEW] Learn the message to be signed (sent by the device)**

Weaknesses of the RP: Same as for HW.

Fig. 6. Weaknesses of Remote HSM-based credentials (RH). The **highlighted** weaknesses are relevant weaknesses that are not present in system HW.

Weaknesses of the user side: Same as for HW.
Weaknesses of the mobile device:

D1 [HW.D1] Display incorrect credential identifiers
D2 [HW.D2] Learn knowledge factor (PIN) entered by the user
D3 **[NEW] Learn encrypted first keyshare**
D4 **[NEW] Learn the plain first keyshare**
D5 [D6] Modify encrypted first keyshare (before it gets decrypted)
D6 **[NEW] Modify plain first keyshare**
D7 **[NEW] Learn the combined signature (received from supporting server)**
D8 **[NEW] Learn first signature share**
D9 [HW.D12] Learn the credential that corresponds to the selected identifier
D10 [HW.D28] Modify the credential chosen according to the selected identifier
D11 [D12] Interfere with generation of the first signature share
D12 [D13] Interfere with the signature request combining process
D13 **[NEW] Modify signature request sent to the supporting server**
D14 **[NEW] Modify the signature (received from supporting server)**
D15 [HW.D15] Interfere with construction of the credential presentation
D16 [HW.D16] Modify credential presentation (to be sent to the RP)
D17 [HW.D13] Modify the credential identifier selected by the user
D18 [HW.D17] Learn user's public key (used as a part of credential presentation)
D19 [HW.D18] Modify user's public key (used as a part of credential presentation)
D20 [HW.D19] Learn credential presentation (to be sent to the RP)
D21 [HW.D21] Modify presentation challenge (received from RP)
D22 [HW.D22] Modify knowledge factor (PIN) entered by the user
D23 [HW.D14] Interfere with search of the selected credential
D24 **[NEW] Learn signature request**
D25 [HW.D25] Learn presentation challenge (received from RP)
D26 [HW.D26] Learn the list of credential identifiers (stored in the device)
D27 [HW.D27] Learn the credential identifier selected by the user
D28 [D11] Modify the first signature share
D29 [D6] Interfere with decryption of encrypted first keyshare

Weaknesses of the supporting server:

S1 **[NEW] (HSM) Learn the second keyshare**
S2 **[NEW] (HSM) Get access to signing by the second keyshare (w/o extracting the keyshare)**
S3 **[NEW] (HSM) Modify the second keyshare**
S4 **[NEW] Modify the message that enters HSM for signing process**
S5 **[NEW] Learn the second signature share**
S6 **[NEW] Interfere with the signature combination process**
S7 **[NEW] Learn the signature**
S8 [S9] Modify the client's public key, against which the signature is checked
S9 **[NEW] Interfere with the signature checking procedure**
S10 **[NEW] Make the decision of the signature check go otherwise**
S11 **[NEW] Modify the signature sent back to the phone**
S12 **[NEW] Change the decision about too many failed signature verifications**
S13 **[NEW] Learn the user's public key (stored locally)**
S14 **[NEW] Learn the first signature share**
S15 [S6] (SE) Interfere with signing by the second keyshare
S16 [S6] Modify the first signature share
S17 [S6] Modify the second signature share

Weaknesses of the RP: Same as for HW.

Fig. 7. Weaknesses of threshold cryptography-based credentials (SS). The **highlighted** weaknesses are relevant weaknesses not present in system HW.

- learning both the PIN and the encrypted first keyshare, results in learning the plain first keyshare: $\mathsf{D4} \prec_{\mathrm{X}} \{\mathsf{D2}, \mathsf{D3}\}$ and $\mathsf{D4} \prec_{\mathrm{X}} \{\mathsf{U2}, \mathsf{D3}\}$;
- tampering with the request sent to the supporting server has no greater effect than modifying the components of this request on the server side (the message and the first signature share): $\mathsf{D13} \prec_{\mathrm{S}} \{\mathsf{S4}, \mathsf{S16}\}$;
- the ciphertext leaks at most the plaintext: $\mathsf{D3} \prec_{\mathrm{S}} \mathsf{D4}$
- the signature request contains the message and the first signature share: $\mathsf{D24} \prec_{\mathrm{S}} \{\mathsf{D8}, \mathsf{D9}\}$.
- interfering with combination of signature shares has no greater effect than tampering with the signature and its verification procedure: $\mathsf{S6} \prec_{\mathrm{S}} \{\mathsf{S11}, \mathsf{S9}\}$

This allows to remove D3, D13, D24, S6 from the set of new relevant weaknesses.

For the new relevant weaknesses of system SS, Table III presents equivalent weaknesses in system HW. Weaknesses connected to learning the private key are equivalent both in seriousness and difficulty under the assumption that supporting server's share is stored in HSM. Weaknesses connected to modification of private key are equivalent in seriousness. Modification of the second key share in SS is equivalent in

Weakness(es) of RH	Rel. type	Weakness of HW	Description
HSM5	$\sim_X$	D7	Learning private key from HSM is equally serious and at least as hard as learning it from the device.
HSM8∨D27	$\sim_X$	D10	Learning the signature on presentation (from HSM or device) is equivalent to learning it from the device in HW.
D14	$\sim_X$	D11	Modifying signature received from HSM is equivalent to modifying signature generated by the device.
HSM6	$\sim_X$	D8	Modifying private key inside HSM is equally serious and at least as hard as modifying it inside device.
(D4 ∧ (D7∨D8)) ∨ HSM4	$\sim_X$	D6	Signing functionality can be accessed either directly on HSM side, or after getting both the authentication private key and access to device's local signing functionality.
HSM7	$\sim_X$	D9	Modifying message that enters signing process on the HSM side is equally serious, and at least as hard as doing it inside the device
D6 ∨ D9 ∨ HSM1	$\sim_S$	D4	Interfering with authentication challenge is no more serious than interfering with the knowledge factor verification process in HW.
HSM3	$\sim_X$	D5	Proceeding with the signing even if the knowledge factor fails in HSM is equally serious, and at least as hard as doing it inside the device.
HSM10	$\sim_X$	D20	Tampering with counting failures inside HSM is equally serious, and at least as hard as inside the device.
HSM11	$\sim_X$	D24	Tampering with signing inside HSM is equally serious, and at least as hard as inside the device.

TABLE II
NEW WEAKNESSES OF RH AND EQUIVALENT WEAKNESSES IN HW.

Weakness(es) of SS	Rel. type	Weakness of HW	Description
D4 ∧ S1	$\sim_X$	D7	Learning both key shares in SS is equivalent to learning user private key in HW.
(D8 ∧ S5) ∨ (S14∧S5)∨ S7 ∨ D7	$\sim_X$	D10	Learning both signature shares (or learning the combined signature) in SS is equivalent to learning the signature in HW.
D6 ∨ S3	$\sim_S$	D8	Modifying both key shares in SS is as serious as modifying the user's private key in HW.
D4 ∧ S2	$\sim_X$	D6	Getting the first keyshare and access to signing by the second keyshare is equivalent to getting access to signing by the user's private key in HW.
S4	$\sim_X$	D9	Modifying the message that enters signing process in SS is equivalent to modifying the corresponding message in HW.
S9	$\sim_S$	D4	Interfering with signature checking procedure by the server in SS is as serious as interfering with knowledge factor verification in HW.
S10	$\sim_S$	D5	Proceeding with signature generation if the check of first signature share fails in SS is as serious as proceeding with the signing if the knowledge factor fails in HW.
S11 ∨ D14	$\sim_X$	D11	Modifying the signature created by the server in SS is equivalent to modifying it in the device in HW.
S12	$\sim_S$	D20	Tampering with counting failures on the server side in SS as serious as tampering with failures on the device side in HW.
S13 ∨ D18	$\sim_X$	D17	Learning user's public key (from the server or the device) in SS is equivalent to learning it from the device in HW.

TABLE III
NEW WEAKNESSES OF SS AND EQUIVALENT WEAKNESSES IN HW.

both seriousness and difficulty to modification of private key in HW under the assumption that supporting server's share is stored in HSM. Interfering with signature verification in SS is equivalent in seriousness to interfering with knowledge factor verification as it influences the signature verification outcome. Signature verification in SS serves as a way to verify correctness of the user's PIN: if the PIN is correct and the signature shares are computed according to the protocol, the composed signature should pass verification. This weakness is *not* equivalent in difficulty, as PIN verification in HW is performed inside SE.

d) Weaknesses of anonymous credentials: Figure 8 presents identified weaknesses of the anonymous credential system with all-or-nothing non-transferability. There are the following relations within system AN:

- leaking the public key of RP is not treated as sensitive information: $\mathsf{D22} \prec_S \{\}$.

This allows to remove D22 from the set of new relevant weaknesses.

For the new relevant weaknesses of system AN, Table IV presents equivalent weaknesses in system HW. Relations connected to the private key are equivalent in seriousness to the corresponding ones from HW. In order to be equivalent in difficulty of exploiting, it is important to apply additional protection on the private key. If the key is not stored in SE, it is easier for external adversary to copy or modify it. However, if the user decides to willingly share their private key with other party, it would mean giving access to the device, assuming that an average user will not be able to directly access and share their private key from the device. Giving out mobile device in AN is equivalent in both difficulty and seriousness to giving out device in HW, as it would give another party access to all the user's credentials and ability to present them. Additionally, since sharing master secret key in AN means giving access to all the credentials, the equivalent weakness in HW is learning both the *private key* and the *credential*, which is superseded in seriousness (but not necessarily in difficulty) by *sharing user device*. Learning and modifying pseudonym for RP is equivalent in seriousness to learning and modifying user's public key. It can be seen as a less serious weakness of system AN as it does not leak user's identity, while learning public key in system HW can fully identify the user.

Weaknesses of the user side:

U1 [HW.U1] Interfere with selection of credential to present
U2 [HW.U5] Modify the list of credential identifiers (displayed by the mobile device)
U3 **[NEW] Influence user in giving out master secret key**
U4 [HW.U6] Learn the list of credential identifiers
U5 [HW.U7] Learn the selected credential identifier
U6 [HW.U8] Modify the selected credential identifier

Weaknesses of the mobile device:

D1 [HW.D1] Display incorrect credential identifiers
D2 [HW.D21] Modify presentation challenge (received from RP)
D3 **[NEW] Learn the master secret key**
D4 **[NEW] Modify the master secret key**
D5 [HW.D12] Learn the credential that corresponds to the selected identifier
D6 [HW.D28] Modify the credential chosen according to the selected identifier
D7 [D16] Modify the RP public key (needed for the zero-knowledge proof)
D8 **[NEW] Learn the pseudonym that the user has generated for RP**
D9 **[NEW] Modify the pseudonym that the user has generated for RP**
D10 [D11] Interfere with construction of the proof of credential possession
D11 [D14] Interfere with construction of the proof connecting pseudonym to the master secret key
D12 **[NEW] Learn the proof of credential possession**
D13 **[NEW] Learn the proof of pseudonym correspondence**
D14 [HW.D15] Interfere with construction of the credential presentation
D15 [HW.D16] Modify credential presentation (to be sent to the RP)
D16 [HW.D19] Learn credential presentation (to be sent to the RP)
D17 [HW.D13] Modify the credential identifier selected by the user
D18 [HW.D14] Interfere with search of the selected credential.
D19 [HW.D25] Learn presentation challenge (received from RP)
D20 [HW.D26] Learn the list of credential identifiers (stored in the device)
D21 [HW.D27] Learn the credential identifier selected by the user
D22 **[NEW] Learn the RP public key**
D23 [D10] Modify PoP of credenital
D24 [D11] Modify pseudonym-related ZKP

Weaknesses of the RP:

RP1 [HW.RP1] Modify presentation challenge (to be sent to the mobile device)
RP2 [HW.RP2] Interfere with the presentation decomposition process
RP3 **[NEW] Interfere with the proof of credential possession (PoP) verification**
RP4 **[NEW] Make the decision on PoP go other way**
RP5 **[NEW] Interfere with the zero knowledge proof (ZKP) of pseudonym connection to master secret key verification process**
RP6 **[NEW] Make the decision on ZKP verification go other way**
RP7 **[NEW] Learn user's pseudonym**
RP8 **[NEW] Learn the proof of credential possession**
RP9 **[NEW] Learn the proof of pseudonym correspondence**
RP10 [HW.RP12] Learn credential presentation (received from the user)
RP11 [HW.RP8] Modify credential presentation (received from the user)
RP12 [HW.RP14] Learn presentation challenge (generated by the RP)
RP13 [HW.RP15] Modify presentation challenge (generated by the RP)
RP14 [RP2] Modify user's pseudonym in the presentation
RP15 [RP2] Modify PoP of credential in the presentation
RP16 [RP2] Modify pseudonym-related ZKP in the presentation

Fig. 8. Weaknesses of the anonymous credentials (AN). The **highlighted** weaknesses are relevant weaknesses that are not present in system HW.

Weakness(es) of AN	Rel. type	Weakness of HW	Description
U3 ∨ D3	$\sim_S$	U4	Getting master secret key in AN is no more serious than getting full access to the user's device in HW.
D4	$\sim_S$	D8	Modifying master secret key in AN is no more serious than modifying the user private key in HW.
RP3 ∨ RP5	$\sim_X$	RP2	Interfering with proof verification in AN is equivalent to interfering with the user's signature verification process in HW.
RP4 ∨ RP6	$\sim_X$	RP3	Interfering with decisions on proof verification in AN is equivalent to interfering with decision on user's signature verification in HW.
D8 ∨ RP7	$\sim_S$	D17	Leaking user's pseudonym in AN is no more serious than leaking user's public key in HW.
D9	$\sim_S$	D18	Modifying user's pseudonym in AN is no more serious than modifying user's public key in HW.
D12 ∨ D13 ∨ RP8 ∨ RP9	$\sim_S$	D19	Proofs in AN contain no more information than the credential presentation in HW.

TABLE IV
NEW WEAKNESSES OF AN AND EQUIVALENT WEAKNESSES IN HW.

C. Analysis

We have shown that, for all weaknesses of systems RH, SS, AN, we can find equivalent weaknesses in the system HW in terms of *seriousness* (for some weaknesses, also in difficulty). We now want to compare the *difficulty* of exploiting these weaknesses in context of particular attacks (analysis step 5).

From the identified weaknesses, it can be seen that all the attacks that are connected to the credential itself are similar in all the systems, since we are only changing the way how the presentation of credential is being generated. Therefore, the comparison below focuses on the main and more serious weaknesses connected with the credential presentation and leaking private key. It is important to analyse the system as a whole (not just the key storage), as we want to identify what information might be leaked, and thus the consequences of a particular attack.

We differentiate attacks on *learning the private key* and on *getting access to SE* where the key is stored. It has been shown that it is possible to extract the private key from hardware modules through side-channel attacks or exploiting incorrect implementation [12]–[14]. The attacks on getting access to SE have also been present in practice, with StartSSL (2011)[5] and DigiNotar (2011) [6] incidents being one of the examples. It should be noted that in case of this attack, it is possible that the adversary gains access to signing functionality only temporarily and later loses it again. Thus, this attack is less serious in consequences as it grants access to the operations with the private key for limited time. Whereas, if the adversary learns the private key, it can create unlimited signatures.

We divide explanation below by the final consequence of exploiting weaknesses (e.g. present any user's credential, modify credential presentation). For each system, we list weaknesses (or combinations of weaknesses) that can be exploited to lead to the stated result.

To **initiate presentation process** in the systems HW, RH and SS, adversary may exploit the weaknesses listed below, assuming adversary has access to the user mobile device. Since the AN does not have any measures preventing unauthorised presentation, we do not include it. Successful initiation of

[5]https://www.darkreading.com/cyberattacks-data-breaches/how-startcom-foiled-comodohacker-4-lessons (last visited 22nd April, 2026)

[6]https://web.archive.org/web/20150919015849/https://www.rijksoverheid.nl/ministeries/ministerie-van-binnenlandse-zaken-en-koninkrijksrelaties/documenten/rapporten/2011/09/05/diginotar-public-report-version-1 (last visited 22nd April, 2026)

presentation means successful authentication to the system, which has different conditions in all three systems. We note that, successful initiation of presentation does not guarantee successful presentation of credential, it can be used by adversary to learn more information from the system.

HW Learn knowledge factor (U2) **OR** Learn knowledge factor that user enters (D2) **OR** Learn value stored for knowledge factor verification (D3) **OR** Interfere with knowledge factor verification process (D4) **OR** Proceed with signing even if knowledge factor check fails (D5)

RH (Learn recomputed private key (D4) **AND** Get access to signing by the stored private key (D7$\vee$D8)) **OR** Make the decision of the authentication request check go otherwise (HSM3)

SS Learn knowledge factor from user (U2) **OR** Learn knowledge factor from device (D2) **OR** Learn first keyshare (D4) **OR** Make the decision of the signature check go otherwise (S10)

For the systems to be considered equivalent, they must provide comparable protection against offline knowledge-factor (e.g., PIN) guessing and memory copying attacks. Currently, HW has an additional weakness of storing the knowledge-factor verification value on-device (within the Secure Element). This enables offline guessing attacks, representing a more severe weakness than attacks that require real-time device access during active authentication.

In the RH, there is no value stored on the mobile device that can be used to verify the PIN guessing attempts with. Instead, PIN is used to recompute authentication private key, which in turn is used to create authentication signature. The authentication is performed by the Remote HSM, which limits the number of incorrect guesses.

Similarly, in SS there is no value stored on the mobile device that can be used to verify the guessing attempts with [2]. PIN verification is performed by the supporting server while verifying the composed signature. Therefore, it is not possible for an adversary to start guessing PIN without contacting the server or extracting PIN from the mobile device. It is an important concern to consider as an adversary may try to get access to the user's device without user sharing corresponding knowledge factor (e.g. when the mobile device is lost or stolen). When implementing user authentication in AN, protection against offline guessing attacks and memory copying attacks should be implemented.

The **modification of presentation** in the systems may happen through exploiting the following weaknesses:

HW Modify private key (D8) **OR** Modify credential (D28) **OR** Modify message to be signed (D9) **OR** Interfere with presentation construction process (D15) **OR** Modify credential presentation sent to the RP (D16)

RH Modify private key (HSM6) **OR** Modify message that enters HSM for signing process (HSM7) **OR** Modify signature sent to the mobile device (HSM9) **OR** Modify the signature received from the remote HSM (D14) **OR** Interfere with the presentation construction process (D17) **OR** Modify credential presentation sent to the RP (D18)

SS Modify knowledge factor on the user side (U3) **OR** Modify encrypted first keyshare (D5) **OR** Modify plain first keyshare (D6) **OR** Modify second keyshare (S3) **OR** Modify credential (D10) **OR** Modify message to be signed (D12 $\vee$ S4) **OR** Modify signature request (D13) **OR** Modify signature from supporting server (D14) **OR** Interfere with presentation construction process (D15) **OR** Modify credential presentation sent to the RP (D16) **OR** Interfere with the signature combination process (S6) **OR** Modify the signature sent to the phone (S11)

AN Modify master secret key (D4) **OR** Modify pseudonym for RP (D9) **OR** Interfere with proof-of-possession construction process (D10) **OR** Interfere with zero-knowledge proof of pseudonym connection to master secret key construction process (D11) **OR** Interfere with the presentation construction process (D14) **OR** Modify credential presentation sent to the RP (D15)

There are more attacks that lead to modified presentation in RH than in HW. This can be explained by the fact that the presentation signature is created by the Remote HSM on the input provided by the mobile device. Thus, adversary has more points for the attack. Attack related to the modification of private key in RH is equivalent to the key modification attack in HW as in both cases keys are stored in secure environment. Therefore, the attacks against RH and HW are equivalent in seriousness and difficulty. Similarly, there are more attacks that result in modified presentation in SS because the process of generating signature for presentation is distributed between the mobile device and the supporting server. However, if the second key share is stored in HSM, the modification of the second key share on the supporting server side is equivalent in difficulty to modifying private key in HW. Modifying the master secret key in AN, however, is less difficult compared to modifying private key in HW, if it is not stored in any secure environment. Other attacks against supporting server, attacks against the mobile device in SS and AN, are equivalent in difficulty and seriousness. Therefore, it can be concluded that the attacks against HW and SS systems are equivalent, given that the second key share is stored in HSM. Additionally, if only one key share is modified in SS and other system component is not compromised, the modified presentation in SS will not be outputted to RP. The AN needs additional layer of protection for the private key to be equivalent to system HW. It can be achieved through introducing threshold cryptography or storing the private in SE.

To **change outcome of presentation verification**, adversary may either modify the presentation (discussed above), or exploit the following weaknesses:

HW Interfere with the user's signature verification (RP3) **OR** Interfere with the issuer's signature verification (RP5)

RH same as HW.

SS same as HW.

AN Interfere with the proof-of-possession verification (RP3)

OR Interfere with the verification of ZKP of pseudonym connection to the master secret key (RP5)

It can be seen that, in all four systems, weaknesses that result in changing outcome of presentation verification are equivalent. Difference comes only from the cryptographic verification algorithm, so the systems are equivalent assuming that the underlying cryptographic primitives are equally strong.

To **present (any) user's credential**, adversary may exploit the following weaknesses:

HW [Influence user to give out mobile device (U4) **AND** learn knowledge factor (U2∨D2∨D3)] **OR** [(Learn private key (D7) **OR** get access to SE signing functionality (D6)) **AND** learn credential (D12)]

RH [Influence user in giving out mobile device (U4) **AND** learn knowledge factor (U2∨D2)] **OR** [Learn recomputed authentication private key (D4) **AND** get access to SE signing functionality (D8) **AND** learn credential (D19)] **OR** [Get access to HSM signing functionality (HSM4) **AND** learn credential (D19)]
// the attacker may either (1) get full access to the device, or (2) get access to authentication keys, or (3) get access directly to HSM signing functionality

SS (Influence user in giving out mobile device (U4) **AND** Learn knowledge factor (U2 ∨ D2)) **OR** [Learn first key share (D4) **AND** (learn second key share (S5) **OR** get access to signing functionality (S2)) **AND** learn credential (D9)]

AN Learn master secret key (D3 ∨ U3)

To present credentials in RH, adversary needs to learn both authentication private keys that can be used to access the Remote HSM signing functionality. Another option is to get the signing key directly from Remote HSM. But in both cases, adversary needs to learn credentials. Another option is to get user's device and PIN code. This will give adversary ability to create valid authentication signatures and signing requests.

To be able to present credentials in SS, adversary needs to learn both key shares or learn key share from the mobile device and get access to the supporting server's signing functionality. Moreover, it is not enough to just learn private key, adversary also needs to learn the credential. Alternatively, adversary needs to get user's device and PIN code. In this way, they will be able to construct valid signature shares and requests to the supporting server.

Learning the master secret key in AN reveals information about all credentials, since the adversary using this key can decrypt partial credential information published by each issuer. Furthermore, adversary will be able to present corresponding credentials. Since this property was introduced in the system to mitigate transferring of the private key to the other parties, it also creates attack surface for the external adversary. All-or-nothing non-transferability is very dependent on the user's understanding of the system. If the user does not understand the full consequences of giving out private key, the system will be protecting user in less efficient way. In other systems, user is not required to know those technical details about the system to still stay protected against certain attacks.

In HW, assuming it is not possible to extract the private key from the SE, attacker needs to have full access to the device, where key is stored, in order to create signatures on credential presentations. Gaining full access to the device would also give adversary full access to all the credentials stored on that device. In the RH, the user can share authentication private keys with other party, but it is not enough to be able to create verifiable presentations. In the SS, the user can share the first private key share OR PIN code AND encrypted first key share with other party. However, the first share of the key is not enough to be able to create verifiable presentations. This party would also need to get access to the credentials and communicate with the supporting server for each presentation.

Finally, let us separately review the *consequences* of **leaking private key** in each of the systems.

HW Learn private key (D7): create signature for credential presentation

RH Learn recomputed and stored authentication private keys (D4 ∧ D7): authenticate towards remote HSM **AND** get access to remote HSM signing functionality // one of the keys alone will not lead to the attacks discussed in this work
Learn HSM private key (HSM5): create signatures for credential presentation

SS Learn first key share (D4): create first signature share **AND** issue signing requests to the supporting server
Learn second key share (S1): create second signature share
Learn both key shares (D4 ∧ S1): create signature for credential presentation

AN Learn master secret key (U3 ∨ D3) get ability to construct credential presentations **AND** learn all the user's credentials

When adversary learns *only* recomputed authentication key or stored authentication key in RH, it does not give them ability to successfully authenticate to the system. However, adversary may target Remote HSM directly to either extract signing key from there or get access to the signing functionality. In SS, when the adversary learns first private key share, it does not necessarily get ability to make valid requests to the supporting server. When instantiating system with [2], one gets additional property of clone detection that prevents adversary who only obtained first key share to issue unlimited signing requests to the supporting server. We note that learning second key share in SS is equivalent to learning private key in HW, if the second key share is stored in HSM. Moreover, having access to the second key share, does not provide ability to create signatures on credential presentation. To obtain this functionality, adversary would need to corrupt both mobile device and the supporting server, which is more difficult.

D. Requirements to the systems and additional guarantees

In this section, we list the identified requirements for the systems RH, SS and AN to achieve equivalent security

guarantees with the HW. We also recall which additional levels of protection are present in RH, SS and AN.

RH has protection of authentication values equivalent to the one in HW. Moreover, it provides stronger protection against guessing attacks as authentication of user happens on the Remote HSM side. However, in RH it is enough to get access to Remote HSM to perform signing operations. and the user has no control over this. Additionally, compromise of Remote HSM will lead to the compromise of signing keys of many system users. Therefore, the protection of Remote HSM from unauthorised access is crucial for the security of RH.

SS should make use of HSM for storing the second key share on the supporting server side. Moreover, it should make use of clone detection mechanism to detect if adversary has copied memory of the mobile device and is trying to initiate signing. In this way, SS provides stronger key protection mechanism compared to HW, as adversary would need to corrupt both mobile device and the supporting server to have access to the credential presentation functionality. Additionally, if one wants to introduce non-transferability guarantees, it can be done by extending the system to support biometric access control [4]. However, SS relies on specific cryptographic protocol that should be supported by the wallet infrastructure. The protocol of [2] has all the required supporting features (protection against offline guessing attacks, clone detection) but it is only capable of producing RSA signatures. Most of the attribute formats referenced in ARF, however, support ECDSA signatures [15], [16]. The work by Snetkov et al. [17] has demonstrated how to achieve ECDSA-based protocol with similar properties. Furthermore, in SS supporting server learns that the user initiated presentation of their credential. With some external information available, supporting server may learn which credential is being presented by the user. To mitigate this leakage, it is possible to extend the protocol to support blind signatures that hide content of message being signed from the supporting server [18].

Similarly, AN should also employ some mechanisms for the private key protection against external adversary. Learning the private key in AN not only gives adversary ability to present all the credentials, but also learn the information about the credentials themselves. This was a desirable property by the design of the system, to prevent the user from sharing their private key with other parties. However, it introduces more attack vectors by the external adversary. Giving out private key in AN is equivalent in consequences to giving out mobile device and the knowledge factor in HW, RH and SS. However, if there are no additional key protection mechanisms in AN, this attack is easier to execute than in HW, RH and SS. Similarly to SS, this system relies on nuanced cryptographic protocols that allow implementing efficient zero-knowledge proofs of signature possession. In this work, we do not focus on the cryptographic details of the protocols and additional nuance properties they can provide.

Table V presents possible improvements of the main weaknesses of the systems identified above.

System	The main weakness	Possible improvements
HW	All trust is placed on the user's device	Distribute storage of private key
RH	All trust is placed on the Remote HSM	Distribute storage of private key between several remote servers
SS	Protection of the supporting server's key share	Store server's share of the private key in HSM
	Leaking the credential in-formation to the supporting server	Introduce blinding property towards supporting server
AN	Protection for the private key from an external adversary	Distribute storage of private key
	User authentication	Introduce authentication (e.g. using knowledge factor)

TABLE V
POSSIBLE IMPROVEMENTS OF THE MAIN WEAKNESSES

IV. CONCLUSION

In this work, we explored different methods of private key protection in the context of digital identity wallets. While secure elements are considered to be the conventional approach for protecting cryptographic keys, this method centralises trust entirely within the mobile device and its manufacturer, creating a single point of failure if the hardware or its underlying infrastructure is compromised. Additionally, the analysis highlights that the system requires implementing protection against offline guessing attacks. Without such protections, having access to the device would enable an adversary to generate valid verifiable presentations.

The threshold cryptography-based approach provides equivalent security guarantees compared to hardware-based approach, provided that the private key share on the supporting server is stored in HSM. This method distributes trust between the mobile device and the supporting server, thereby mitigating risks associated with single points of failure, and ensuring that unauthorized access to the mobile device alone is insufficient for compromising the system. The approach of all-or-nothing non-transferability aims at preventing the user from sharing their credentials with the other users. Although this approach achieves its goal of discouraging user to share their credentials, the analysis shows that this approach should be integrated with the other measures of private key protection.

When combining threshold cryptography with non-transferability properties (either through all-or-nothing or biometric approach), the system achieves dual-layer protection: it is not only preventing from the credential sharing but also significantly complicates an adversary's ability to access the private key. This combination ensures a more resilient system, addressing both user-driven risks (e.g., intentional or accidental sharing) and external threats (e.g., device compromise or brute-force attacks).

ACKNOWLEDGMENT

This research has been supported by Estonian Research Council, grant No. PRG1780.

REFERENCES

[1] "Regulation (EU) 2024/1183 of the European Parliament and of the Council of 11 April 2024 amending Regulation (EU) No 910/2014 as regards establishing the European Digital Identity Framework," 2024.

[2] A. Buldas, A. Kalu, P. Laud, and M. Oruaas, "Server-Supported RSA Signatures for Mobile Devices," in *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I* (S. N. Foley, D. Gollmann, and E. Snekkenes, eds.), vol. 10492 of *Lecture Notes in Computer Science*, pp. 315–333, Springer, 2017.

[3] J. Camenisch and A. Lysyanskaya, "An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation," in *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding* (B. Pfitzmann, ed.), vol. 2045 of *Lecture Notes in Computer Science*, pp. 93–118, Springer, 2001.

[4] S. Pape, "A Survey on Non-transferable Anonymous Credentials," in *The Future of Identity in the Information Society - 4th IFIP WG 9.2, 9.6/11.6, 11.7/FIDIS International Summer School, Brno, Czech Republic, September 1-7, 2008, Revised Selected Papers* (V. Matyás, S. Fischer-Hübner, D. Cvrcek, and P. Svenda, eds.), vol. 298 of *IFIP Advances in Information and Communication Technology*, pp. 107–118, Springer, 2008.

[5] P. Laud and J. Vakarjuk, "A comparison-based methodology for the security assurance of novel systems," in *Computer Security. ESORICS 2022 International Workshops* (S. Katsikas, F. Cuppens, C. Kalloniatis, J. Mylopoulos, F. Pallas, J. Pohle, M. A. Sasse, H. Abie, S. Ranise, L. Verderame, E. Cambiaso, J. Maestre Vidal, M. A. Sotelo Monge, M. Albanese, B. Katt, S. Pirbhulal, and A. Shukla, eds.), (Cham), pp. 625–644, Springer International Publishing, 2023.

[6] J. Vakarjuk, N. Snetkov, and J. Willemson, "Comparing security levels of postal and internet voting," *Inf. Secur. J. A Glob. Perspect.*, vol. 34, no. 4, pp. 265–285, 2025.

[7] "The European Digital Identity Wallet Architecture and Reference Framework V2.4.0 ," 2025. Accessed: 26.02.2026.

[8] "German National EUDI Wallet: Architecture Documentation v0.4.0," 2025. Accessed: 26.02.2026.

[9] C. Baum, O. Blazy, J. Camenisch, J.-H. Hoepman, E. Lee, A. Lehmann, A. Lysyanskaya, R. Mayrhofer, H. Montgomery, N. K. Nguyen, *et al.*, "Cryptographers' Feedback on the EU Digital Identity's ARF," *Tech. Rep.*, 2024.

[10] S. Q. Dijkhuis, "Authentication and sole control at a high level of assurance on widespread smartphones with threshold signatures," *IACR Cryptol. ePrint Arch.*, p. 267, 2025.

[11] C. Komlo and I. Goldberg, "FROST: flexible round-optimized schnorr threshold signatures," in *Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers* (O. Dunkelman, M. J. J. Jr., and C. O'Flynn, eds.), vol. 12804 of *Lecture Notes in Computer Science*, pp. 34–65, Springer, 2020.

[12] A. Shakevsky, E. Ronen, and A. Wool, "Trust dies in darkness: Shedding light on samsung's trustzone keymaster design," in *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022* (K. R. B. Butler and K. Thomas, eds.), pp. 251–268, USENIX Association, 2022.

[13] F. Khalid and A. Masood, "Vulnerability analysis of qualcomm secure execution environment (QSEE)," *Comput. Secur.*, vol. 116, p. 102628, 2022.

[14] J. Mishra and S. K. Sahay, "Modern hardware security: A review of attacks and countermeasures," *CoRR*, vol. abs/2501.04394, 2025.

[15] M. Sporny, D. Longley, D. Chadwick, and I. Herman, "Verifiable Credentials Data Model v2.0," 2025.

[16] ISO Central Secretary, "Personal identification — ISO-compliant driving licence — Part 5: Mobile driving licence (mDL) application," Standard ISO/IEC 18013-5:2021, International Organization for Standardization, Geneva, CH, 2021.

[17] N. Snetkov, J. Vakarjuk, and P. Laud, "Universally Composable Server-Supported Signatures for Smartphones." Cryptology ePrint Archive, Paper 2024/1941, 2024.

[18] N. Snetkov, J. Vakarjuk, and A. Pankova, "Coppercloud: Blind server-supported RSA signatures." Cryptology ePrint Archive, Paper 2025/1824, 2025.

APPENDIX

In Figures 9, 10, 11, 12, we present the BPMN models on which the analysis is based. The BPMN models themselves can be found in the repository [7].

[7] https://github.com/jvakarjuk/comparison/tree/main (last visited 22nd April, 2026)

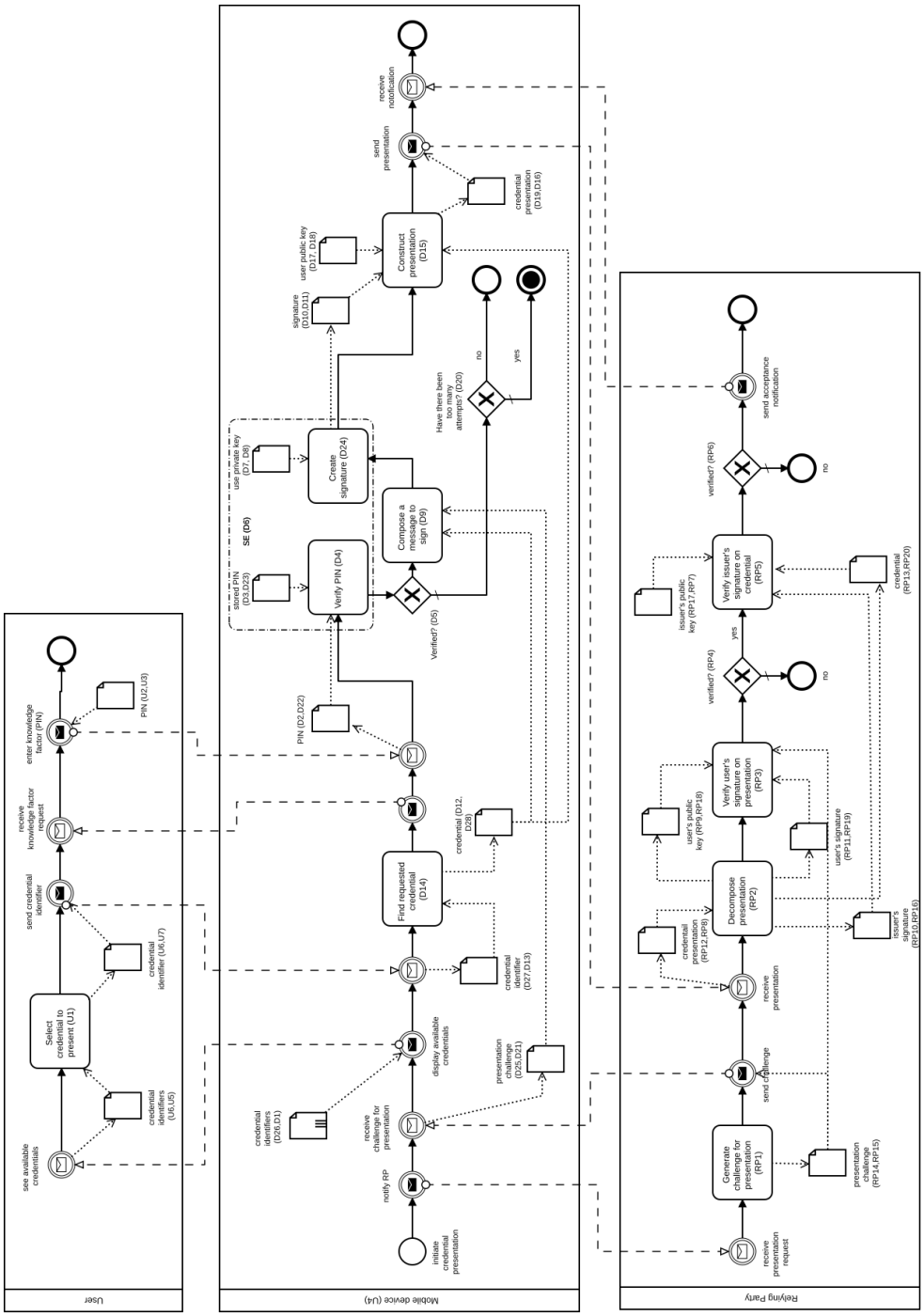


Fig. 9. BPMN model of hardware-based credentials.

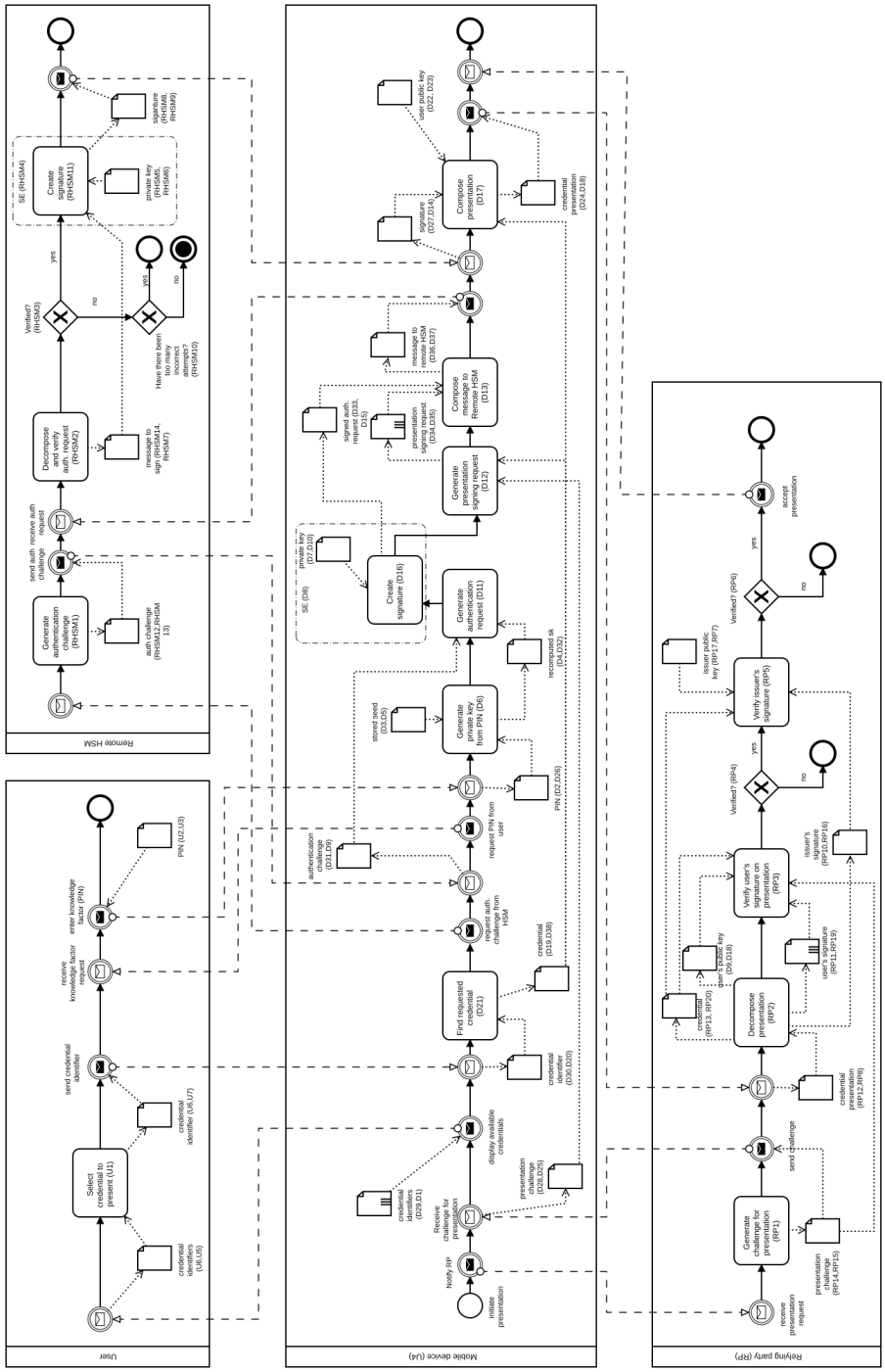


Fig. 10. BPMN model of Remote HSM-based credentials.

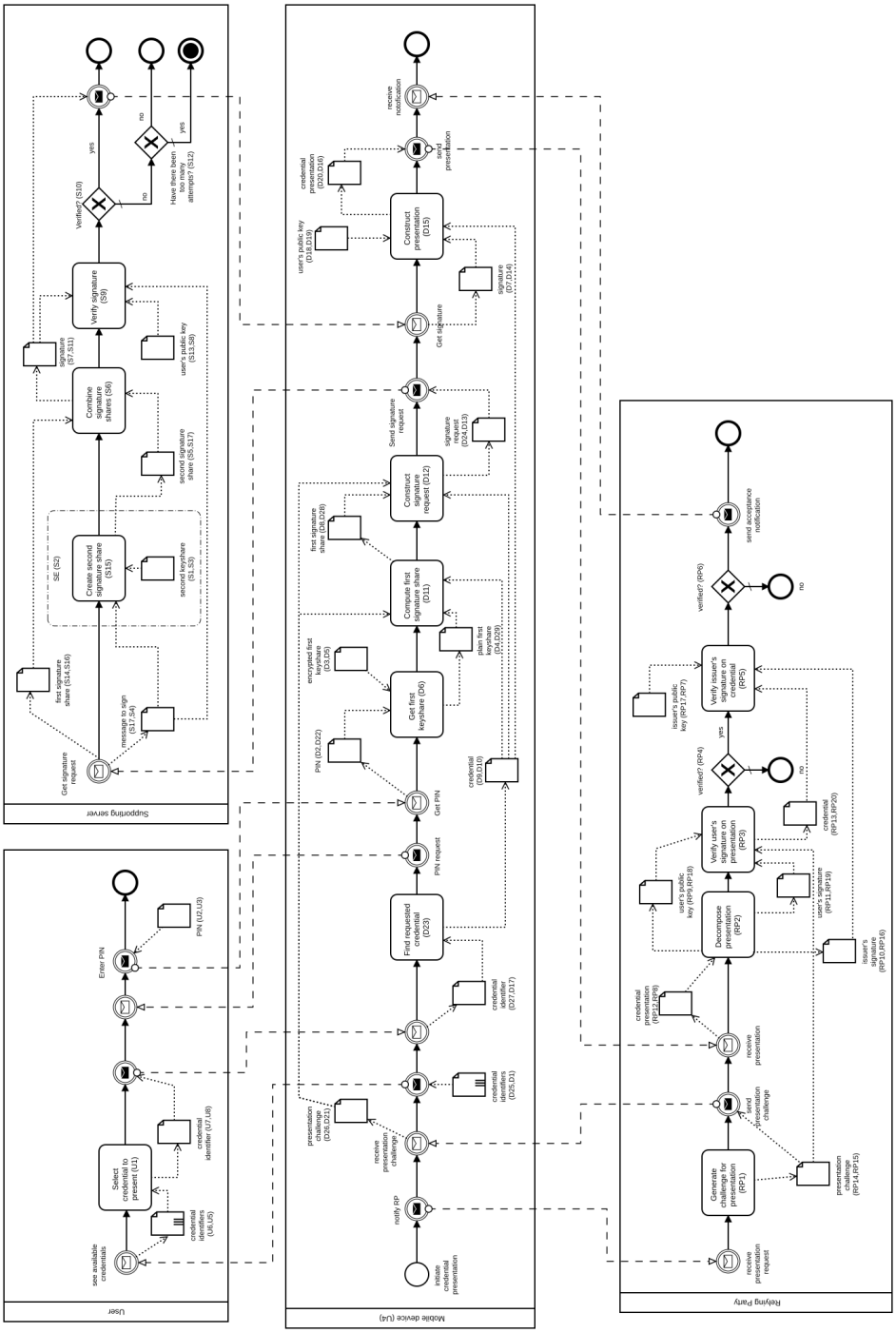


Fig. 11. BPMN model of threshold cryptography based credentials.

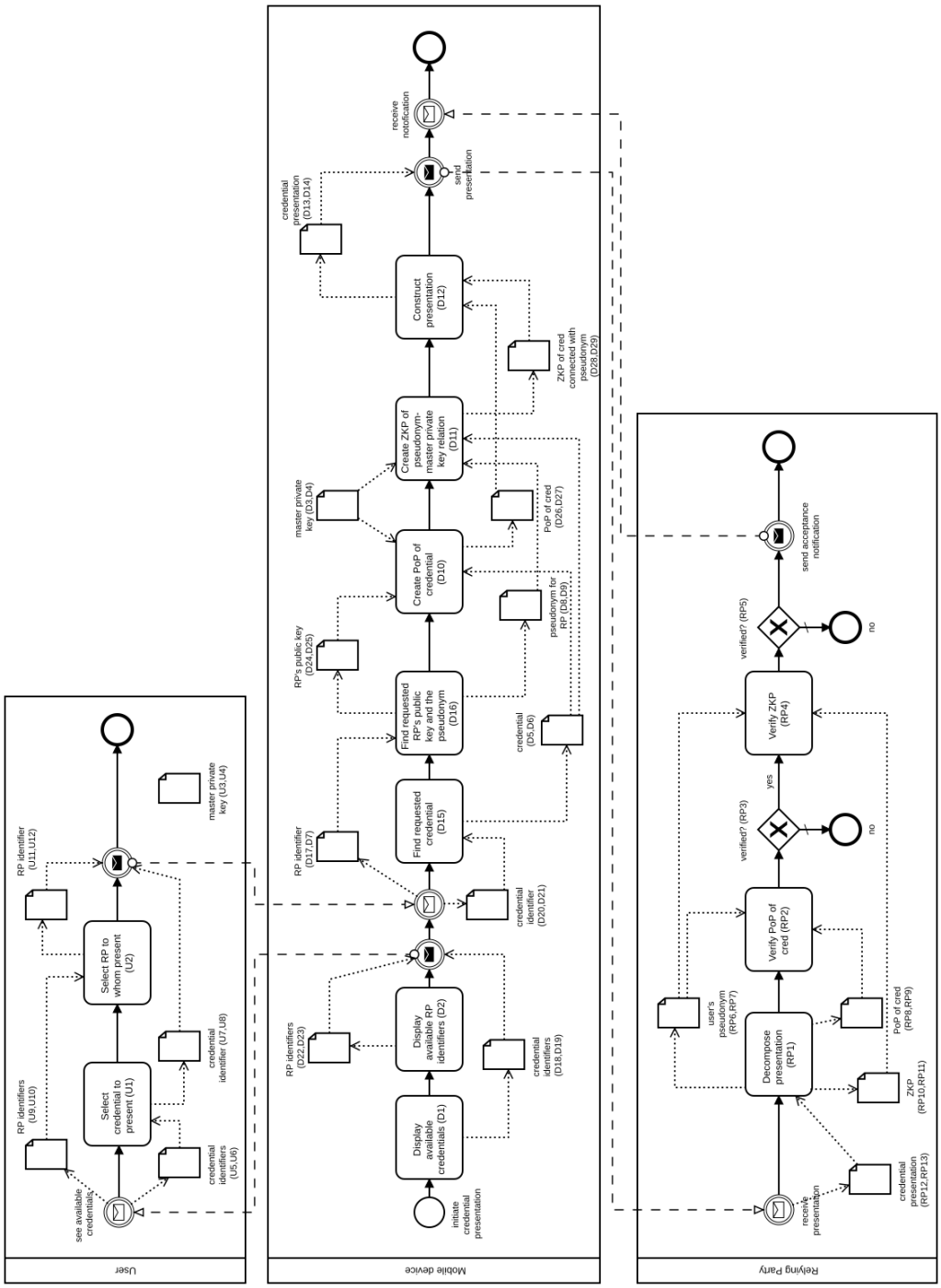


Fig. 12. BPMN model of anonymous credentials.

9 Formal definition of Prettiness system

The initialisation protocol is presented in Figure 36. The issuing protocol is presented in Figures 37 and 38. The credential retrieval protocol is presented in Figure 39. Revocation protocol initiated by user is presented in Figure 40) and protocol initiated by issuer is presented in Figure 41. The presentation protocol is presented in Figure 42. The verification protocol is presented in Figures 43 and 44.

$\mathsf{Initialise}_{\mathsf{U}}$	$\mathsf{Initialise}_{\mathsf{ST}}$
// get input from environment	// generate signing key pair
$\mathcal{Z} \longrightarrow: (\mathsf{initialise\text{-}req}, \mathsf{sid})$;	$\mathsf{pk} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}(\mathsf{keygen}, \mathsf{sid}, T_0)$
// generate signing key pair	// generate decryption key pair
$\mathsf{pk} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}(\mathsf{keygen}, \mathsf{sid}, L, T_0, \mathsf{PIN})$	$\mathsf{pk}_{\mathsf{s}} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{KeyGen}, \mathsf{sid})$
// generate decryption key pair	
$\mathsf{pk}_{\mathsf{c}} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{KeyGen}, \mathsf{sid})$	

Figure 36: Initialisation protocol (User, ST*)*

$\mathsf{Issue}_\mathsf{U}$	$\mathsf{Issue}_\mathsf{ST}$
// get input from environment	
$\mathcal{Z} \longrightarrow: (\mathsf{issue\text{-}req}, \mathsf{sid}, \overrightarrow{\mathsf{aID}}, \mathsf{IID})$;	
$id_{\mathsf{cID}} \leftarrow\!\!\$\ \mathcal{X}_{\mathsf{id}}$;	
// establish connection with the server	// wait for user connection
$\longrightarrow \mathsf{ST} : \mathsf{tag}_{issue}, \mathsf{sid}, \mathsf{uID}$;	$U \longrightarrow: \mathsf{tag}_{issue}, \mathsf{sid}, \mathsf{uID}$;
// generate request message	
$m = (\mathsf{tag}_{issue}, id_{\mathsf{cID}}, \mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID})$;	
// generate threshold signature	// generate and verify threshold signature
$\sigma_{\mathsf{req}} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}(\mathsf{sign}, \mathsf{sid}, m, \mathsf{PIN})$;	$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}(\mathsf{sign}, \mathsf{sid})$;
$\longrightarrow \mathsf{ST} : \sigma_{\mathsf{req}}, m$;	$\mathsf{U} \longrightarrow: \sigma_{\mathsf{req}}, m = (\mathsf{tag}_{issue}, id_{\mathsf{cID}}, \mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID})$;
	$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}(\mathsf{verify}, \mathsf{sid}, m, \sigma_{\mathsf{req}})$;
// Get connected to the issuer I	// authenticate I as IID to U
$\mathsf{ST} \longrightarrow: I, \mathsf{IID}$;	$\longrightarrow \mathsf{U} : I, \mathsf{IID}$;
// locally generate a revocation token for the user	// authenticate U as uID to I
$\mathsf{rev}_U \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{issue}, U, (\mathsf{sid}, 0))$;	$\longrightarrow \mathsf{I} : U, id_{\mathsf{cID}}, \mathsf{uID}, \overrightarrow{\mathsf{aID}}, \sigma_{\mathsf{req}}$;
$\longrightarrow I : \mathsf{rev}_U$;	// get encrypted credential from I
// jointly generate revocation token for the issuer	$\mathsf{I} \longrightarrow: (\mathsf{cred}, \sigma_{\mathsf{cred}})$;
$\mathsf{rev}_I \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{issue}, I, (\mathsf{sid}, 1))$;	$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{IID}}_{\mathsf{sign}}(\mathsf{verify}, \mathsf{sid}, \mathsf{cred}, \sigma_{\mathsf{cred}})$;
// get encrypted credential from AMS	// send encrypted credential to U
$\mathsf{ST} \longrightarrow: (\mathsf{cred}, \sigma_{\mathsf{cred}})$;	$\longrightarrow \mathsf{U} : (\mathsf{cred}, \sigma_{\mathsf{cred}})$;
$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{IID}}_{\mathsf{sign}}(\mathsf{verify}, \mathsf{sid}, \mathsf{cred}, \sigma_{\mathsf{cred}})$;	
// decrypt and verify content	// participate in threshold decryption
$(\mathsf{rev}'_I, \mathsf{rev}'_U) \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Decrypt}, \mathsf{sid}, c_{\mathsf{rev}})$;	$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Decrypt}, \mathsf{sid})$;
assert $(\mathsf{rev}_I, \mathsf{rev}_U) = (\mathsf{rev}'_I, \mathsf{rev}'_U)$;	
for $\mathsf{aID}_i \in \overrightarrow{\mathsf{aID}}$ **do** :	**for** $\mathsf{aID}_i \in \overrightarrow{\mathsf{aID}}$ **do** :
$\mathsf{atr}_i \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Decrypt}, (\mathsf{sid}, i), c_{\mathsf{atr}_i})$;	$ok \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Decrypt}, (\mathsf{sid}, i))$;
assert $\mathsf{atr}_i \in \mathcal{V}_{\mathsf{aID}_i}$;	
assert $\mathsf{atr}_0 = \mathsf{pk}_{\mathsf{uID}}$;	
$\mathsf{cID} \leftarrow (\mathsf{id}_{\mathsf{cID}}, \mathsf{summary}(\mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID}))$;	$\mathsf{cID} \leftarrow (\mathsf{id}_{\mathsf{cID}}, \mathsf{summary}(\mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID}))$;
$h_{\mathsf{cred}} \leftarrow H(h_{\mathsf{cred}}, (\mathsf{cID}, \mathsf{cred}, \sigma_{\mathsf{cred}}))$;	$\mathsf{DB}\mathcal{S}[\mathsf{uID}, \mathsf{cID}] \leftarrow (\mathsf{cred}, \sigma_{\mathsf{cred}})$;
$\longrightarrow \mathcal{Z} : (\mathsf{issue\text{-}resp}, \mathsf{sid}, \mathsf{cID})$;	

Figure 37: Issuing protocol (User, ST*)*

Issue_I

// receive an issue request from the server

$\mathsf{ST} \longrightarrow: \mathsf{sid}, U, id_{\mathsf{cID}}, \mathsf{uID}, \overrightarrow{\mathsf{aID}}, \sigma_{\mathsf{req}};$

// verify the request

$ok \leftarrow \mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}(\mathsf{verify}, \mathsf{sid}, m = (\mathsf{tag}_{issue}, id_{\mathsf{cID}}, \mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID}), \sigma_{\mathsf{req}})$

// verify freshness of the credential identifier

$\mathsf{cID} \leftarrow (id_{\mathsf{cID}}, \mathsf{summary}(\mathsf{uID}, \overrightarrow{\mathsf{aID}}, \mathsf{IID}))$; **assert** $\mathsf{cID} \notin \overrightarrow{\mathsf{cID}}$;

$\overrightarrow{\mathsf{cID}} \leftarrow \overrightarrow{\mathsf{cID}} \cup \mathsf{cID}$

// generate revocation tokens for the user and the issuer

$U \longrightarrow: \mathsf{rev}_U;$

$\mathsf{rev}_I \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{issue}, U, (\mathsf{sid}, 1));$

// request attribute values from the environment

$\longrightarrow \mathcal{Z} : (\mathsf{issue\text{-}req}, \mathsf{sid}, \mathsf{uID}, \mathsf{cID}, \overrightarrow{\mathsf{aID}});$

$\mathcal{Z} \longrightarrow: (\mathsf{issue\text{-}atr}, \mathsf{sid}, \overrightarrow{\mathsf{atr}});$

// verify attribute values obtained from the environment

assert $(\mathsf{atr}_1 \in \mathcal{V}_{\mathsf{aID}_1}, \ldots, \mathsf{atr}_n \in \mathcal{V}_{\mathsf{aID}_n})$;

// encrypt revocation tokens and the user public key

$c_{\mathsf{rev}} \leftarrow \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Encrypt}, \mathsf{sid}, (\mathsf{rev}_I, \mathsf{rev}_U));$

$\mathsf{atr}_0 \leftarrow \mathsf{pk}_{\mathsf{uID}};$

$c_{\mathsf{atr}_0} \leftarrow \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Encrypt}, \mathsf{sid}, \mathsf{atr}_0);$

// encrypt attribute values

for $\mathsf{aID}_i \in \overrightarrow{\mathsf{aID}}$ **do**

  $c_{\mathsf{atr}_i} \leftarrow \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Encrypt}, (\mathsf{sid}, i), \mathsf{atr}_i)$

// compose the credential

$\mathsf{cred} = \langle (\mathsf{aID}_0, c_{\mathsf{atr}_0}), (\mathsf{aID}_1, c_{\mathsf{atr}_1}), \ldots (\mathsf{aID}_n, c_{\mathsf{atr}_n}), (\mathsf{aID}_{\mathsf{rev}}, c_{\mathsf{rev}}) \rangle$

$\sigma_{\mathsf{cred}} \leftarrow \mathcal{F}_{\mathsf{sign}}^{\mathsf{IID}}(\mathsf{sign}, \mathsf{sid}, \mathsf{cred});$

$\longrightarrow \mathsf{ST} : (\mathsf{cred}, \sigma_{\mathsf{cred}});$

// remember revocation token

$\mathsf{DB}I[\mathsf{cID}] = \mathsf{rev}_I;$

$\longrightarrow \mathcal{Z} : (\mathsf{issue\text{-}resp}, \mathsf{sid}, \mathsf{cID});$

Figure 38: Issuing protocol (Issuer)

GetCIDs_U	GetCIDs_ST
// get input from environment	
$\mathcal{Z} \longrightarrow$: (getsids-req, sid);	
// establish connection with the server	// wait for user connection
$\longrightarrow \text{ST}$: tag_{cids}, sid, uID;	$U \longrightarrow$: tag_{cids}, sid, uID;
// generate request message	
$m = (\text{tag}_{cids}, \text{uID})$;	
// generate threshold signature	// generate threshold signature
$\sigma_\text{req} \leftarrow \mathcal{F}_\text{tsign}^\text{uID}(\text{sign}, \text{sid}, m, \text{PIN})$	ok $\leftarrow \mathcal{F}_\text{tsign}^\text{uID}(\text{sign}, \text{sid})$
$\longrightarrow S : \sigma_\text{req}, m$;	$U \longrightarrow$: $\sigma_\text{req}, m = (\text{tag}_{cids}, \text{uID})$;
	ok $\leftarrow \mathcal{F}_\text{tsign}^\text{uID}(\text{verify}, \text{sid}, m, \sigma_\text{req})$;
// get credential data from AMS	// send credential data to the user
$S \longrightarrow$: $\overrightarrow{\text{cID}}, \overrightarrow{\text{cred}}$	$(\overrightarrow{\text{cID}}, \overrightarrow{\text{cred}}) \leftarrow \text{DB}S[\text{uID}]$;
// check integrity of credential data	$\longrightarrow U : \overrightarrow{\text{cID}}, \overrightarrow{\text{cred}}$;
assert $h_\text{cred} = H(\ldots H(0, (\text{cID}_1, \text{cred}_1))\ldots, (\text{cID}_n, \text{cred}_n))$;	
$\text{DB}U[\overrightarrow{\text{cID}}] \leftarrow \overrightarrow{\text{cred}}$;	
$\longrightarrow \mathcal{Z}$: (getcids-resp, sid, $\overrightarrow{\text{cID}}$);	

Figure 39: Protocol for downloading credential backup from the server

Revoke_U	Revoke_ST
// get input from environment	
$\mathcal{Z} \longrightarrow$: (revoke-req, sid, cID);	
// select credential to revoke	
$(\text{cred}, \sigma_\text{cred}) \leftarrow \text{DB}U[\text{cID}]$;	
// establish connection with the server	// wait for user connection
$\longrightarrow \text{ST}$: tag_{rev}, sid, uID;	$U \longrightarrow$: tag_{rev}, sid, uID;
// generate request message	
$c_\text{cID} \leftarrow \mathcal{F}_\text{tdecrypt}^\text{uID}(\text{Encrypt}, sid, \text{cID})$;	
$m = (\text{tag}_{rev}, \text{uID}, c_\text{cID})$;	
// generate threshold signature	// generate and verify threshold signature
$\sigma_\text{req} \leftarrow \mathcal{F}_\text{tsign}^\text{uID}(\text{sign}, \text{sid}, m, \text{PIN})$	ok $\leftarrow \mathcal{F}_\text{tsign}^\text{uID}(\text{Sign}, \text{sid})$
$\longrightarrow S : \sigma_\text{req}, m$;	$U \longrightarrow$: $\sigma_\text{req}, m = (\text{tag}_{rev}, \text{uID}, c_\text{cID})$;
	ok $\leftarrow \mathcal{F}_\text{tsign}^\text{uID}(\text{verify}, \text{sid}, m, \sigma_\text{req})$;
// decrypt revocation token data	// participate in threshold decryption
$(\text{rev}_I, \text{rev}_U) \leftarrow \mathcal{F}_\text{tdecrypt}^\text{uID}(\text{Decrypt}, \text{sid}, c_\text{rev})$;	ok $\leftarrow \mathcal{F}_\text{tdecrypt}^\text{uID}(\text{Decrypt}, \text{sid})$;
// revoke the selected token	// confirm revocation
ok $\leftarrow \mathcal{F}_\text{rev}(\text{revoke}, \text{sid}, \text{rev}_U)$;	ok $\leftarrow \mathcal{F}_\text{rev}(\text{revoke}, \text{sid})$;
$\longrightarrow \mathcal{Z}$: (revoke-resp, sid, ok);	

Figure 40: Revocation protocol initiated by user

$\mathsf{Revoke}_\mathsf{I}$	$\mathsf{Revoke}_\mathsf{ST}$
// get input from environment	
$\mathcal{Z} \longrightarrow: (\mathsf{revoke\text{-}req}, \mathsf{sid}, \mathsf{cID})$;	
// select credential to revoke	
$\mathsf{rev}_I \leftarrow \mathsf{DB}I[\mathsf{cID}]$;	
// establish connection with the server	// wait for issuer connection
$\longrightarrow \mathsf{ST} : \mathsf{tag}_{rev}, \mathsf{sid}, \mathsf{IID}$;	$I \longrightarrow: \mathsf{tag}_{rev}, \mathsf{sid}, \mathsf{IID}$;
// extract uID from cID	
$\mathsf{uID} \leftarrow \mathsf{cID}$;	
// generate request message	
$c_{\mathsf{cID}} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Encrypt}, sid, \mathsf{cID})$;	
$m = (\mathsf{tag}_{rev}, \mathsf{IID}, c_{\mathsf{cID}})$;	
$\sigma_{\mathsf{req}} \leftarrow \mathcal{F}^{\mathsf{IID}}_{\mathsf{sign}}(\mathsf{sign}, \mathsf{sid}, m)$;	
$\longrightarrow \mathsf{ST} : (m, \sigma_{\mathsf{req}})$;	$\mathsf{I} \longrightarrow: (m = (\mathsf{tag}_{rev}, \mathsf{IID}, c_{\mathsf{cID}}), \sigma_{\mathsf{req}})$
	$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{IID}}_{\mathsf{sign}}(\mathsf{verify}, \mathsf{sid}, m, \sigma_{\mathsf{req}})$
// revoke the selected token	// confirm revocation
$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{revoke}, \mathsf{sid}, \mathsf{rev}_I)$;	$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{revoke}, \mathsf{sid})$;
$\longrightarrow \mathcal{Z} : (\mathsf{revoke\text{-}resp}, \mathsf{sid}, \mathsf{ok})$;	

Figure 41: Revocation protocol initiated by issuer

Present_U	Present_ST
// get input from environment	
$\mathcal{Z} \longrightarrow: (\text{present-req}, \text{sid}, \text{cID}, \overrightarrow{\text{aID}}_{\text{dscl}})$;	
// select credential to present	
$(\text{cred}, \sigma_{\text{cred}}) \leftarrow \text{DB}U[\text{cID}]$;	
assert $\overrightarrow{\text{aID}}_{\text{dscl}} \subseteq \overrightarrow{\text{aID}}$;	
// establish connection with the server	// wait for user connection
$\longrightarrow \text{ST} : \text{tag}_{pres}, \text{sid}, \text{uID}$;	$U \longrightarrow: \text{tag}_{pres}, \text{sid}, \text{uID}$;
// generate request message	
$c_{\text{cID}} \leftarrow \mathcal{F}^{\text{uID}}_{\text{tdecrypt}}(\text{Encrypt}, sid, (\text{cID}, \overrightarrow{\text{aID}}_{\text{dscl}}))$;	
$m = (\text{tag}_{pres}, \text{uID}, \lvert\overrightarrow{\text{aID}}_{\text{dscl}}\rvert, c_{\text{cID}})$;	
// generate threshold signature	// generate and verify threshold signature
$\sigma_{\text{req}} \leftarrow \mathcal{F}^{\text{uID}}_{\text{tsign}}(\text{sign}, \text{sid}, m, \text{PIN})$	$\text{ok} \leftarrow \mathcal{F}^{\text{uID}}_{\text{tsign}}(\text{sign}, \text{sid})$
$\longrightarrow \text{ST} : \sigma_{\text{req}}, m$;	$\text{U} \longrightarrow: \sigma_{\text{req}}, m = (\text{tag}_{pres}, \text{uID}, n, c_{\text{cID}})$;
	$\text{ok} \leftarrow \mathcal{F}^{\text{uID}}_{\text{tsign}}(\text{verify}, \text{sid}, m, \sigma_{\text{req}})$;
// check revocation status	// participate in subroutine calls
$(\text{rev}_I, \text{rev}_U) \leftarrow \mathcal{F}^{\text{uID}}_{\text{tdecrypt}}(\text{Decrypt}, \text{sid}, c_{\text{rev}})$;	$\text{ok} \leftarrow \mathcal{F}^{\text{uID}}_{\text{tdecrypt}}(\text{Decrypt}, \text{sid})$;
$b_I \leftarrow \mathcal{F}_{\text{rev}}(\text{unotify}, \text{sid}, \text{rev}_I)$;	$\text{ok} \leftarrow \mathcal{F}_{\text{rev}}(\text{unotify}, \text{sid})$;
$b_U \leftarrow \mathcal{F}_{\text{rev}}(\text{unotify}, \text{sid}, \text{rev}_U)$;	$\text{ok} \leftarrow \mathcal{F}_{\text{rev}}(\text{unotify}, \text{sid})$;
// decrypt disclosed attributes	// participate in threshold decryption
for $\text{aID}_i \in \overrightarrow{\text{aID}}_{\text{dscl}}$:	**for** $i \in 1..n$ **do**
$\overrightarrow{\text{atr}}_{\text{dscl}} \leftarrow \overrightarrow{\text{atr}}_{\text{dscl}} \cup \mathcal{F}^{\text{uID}}_{\text{tdecrypt}}(\text{Decrypt}, (\text{sid}, i), c_{\text{aID}_i})$;	$\text{ok} \leftarrow \mathcal{F}^{\text{uID}}_{\text{tdecrypt}}(\text{Decrypt}, (\text{sid}, i))$;
assert $\neg(b_I \vee b_U)$;	
$\longrightarrow \mathcal{Z} : (\text{present-resp}, \text{sid}, \text{present} = \overrightarrow{\text{atr}}_{\text{dscl}})$;	

Figure 42: Presentation with selective disclosure

$\mathsf{Verify}_{\mathsf{RP}}$

// get input from the environment

$\mathcal{Z} \longrightarrow: (\mathsf{verify\text{-}req}, \mathsf{sid});$

// get last revocation data (can be done in advance)

$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{vnotify}, \mathsf{sid});$

// wait for user connection

$U \longrightarrow: \mathsf{tag}_{verify}, \mathsf{sid}, \mathsf{uID};$

// generate a challenge for the user

$\mathsf{ch} \leftarrow\!\!\$\ \mathcal{CH};$

$\longrightarrow U : \langle \mathsf{ch} \rangle;$

// get response from the user

$U \longrightarrow: \langle \mathsf{uID}, \mathsf{cred}, \sigma_{\mathsf{cred}}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}, \mathsf{IID}, \mathsf{rev}_I, \mathsf{rev}_U, \sigma_{\mathsf{present}} \rangle$

// verify user and issuer signatures

$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{IID}}_{\mathsf{sign}}(\mathsf{verify}, \mathsf{sid}, \mathsf{cred}, \sigma_{\mathsf{cred}})$

$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tsign}}(\mathsf{verify}, \mathsf{sid}, (\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}, \mathsf{cred}, \sigma_{\mathsf{cred}}, \mathsf{ch}), \sigma_{\mathsf{present}})$

// verify revocation status

$\textbf{assert}\ \mathcal{F}_{\mathsf{rev}}(\mathsf{verify}, \mathsf{sid}) = (\mathsf{rev}_I, \mathsf{true});$

$\textbf{assert}\ \mathcal{F}_{\mathsf{rev}}(\mathsf{verify}, \mathsf{sid}) = (\mathsf{rev}_U, \mathsf{true});$

// verify credential data

$\langle (\mathsf{aID}_0, c_0), \ldots (\mathsf{aID}_n, c_n), (\mathsf{aID}_{\mathsf{rev}}, c_{\mathsf{rev}}) \rangle \leftarrow \mathsf{cred};$

$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Verify}, \mathsf{sid}, (c_{\mathsf{rev}}, \mathsf{rev}));$

$\mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Verify}, \mathsf{sid}, (c_0, \mathsf{pk}_{\mathsf{uID}}));$

$\textbf{for}\ (\mathsf{aID}_i, \mathsf{atr}_i) \in (\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}) :$

$\quad \mathsf{ok} \leftarrow \mathcal{F}^{\mathsf{uID}}_{\mathsf{tdecrypt}}(\mathsf{Verify}, \mathsf{sid}, (c_{\mathsf{aID}_i}, \mathsf{atr}_i));$

$\longrightarrow \mathcal{Z} : (\mathsf{verify\text{-}resp}, (\overrightarrow{\mathsf{aID}}, \overrightarrow{\mathsf{atr}}, \mathsf{IID}, \mathsf{ch}), 1);$

Figure 43: Verification of credential presentation – RP

$\mathsf{Verify}_{\mathsf{U}}$

// get input from the environment

$\mathcal{Z} \longrightarrow$: $(\mathsf{verify\text{-}req}, \mathsf{sid}, \mathsf{cID}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \mathsf{present} = \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}, \mathsf{RP})$;

// cred, σ_{cred}, rev_I, rev_U, IID come from presentation of cID

// Establish connection with RP

$\longrightarrow \mathsf{RP} : \mathsf{tag}_{verify}, \mathsf{sid}, \mathsf{uID}$;

// Get the challenge of RP

$\mathsf{RP} \longrightarrow$: $\langle \mathsf{ch} \rangle$

$m \leftarrow (\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}, \mathsf{IID}, \mathsf{cred}, \sigma_{\mathsf{cred}}, \mathsf{ch})$

// Establish connection with AMS

$\longrightarrow \mathsf{ST} : \mathsf{tag}_{verify}, \mathsf{sid}, \mathsf{uID}$;

$\sigma_{\mathsf{present}} \leftarrow \mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}(\mathsf{sign}, \mathsf{sid}, m, \mathsf{PIN})$

$\longrightarrow \mathsf{RP} : \langle \mathsf{uID}, \mathsf{cred}, \sigma_{\mathsf{cred}}, \overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}}, \mathsf{IID}, \mathsf{rev}_I, \mathsf{rev}_U, \sigma_{\mathsf{present}} \rangle$;

// prove that credential is not revoked

$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{verify}, \mathsf{sid}, \mathsf{rev}_I)$;

$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{rev}}(\mathsf{verify}, \mathsf{sid}, \mathsf{rev}_U)$;

// participate in proofs of encryption

$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Verify}, \mathsf{sid}, (c_{\mathsf{rev}}, \mathsf{rev}))$;

$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Verify}, \mathsf{sid}, (c_0, \mathsf{pk}_{\mathsf{uID}}))$;

for $(\mathsf{aID}_i, \mathsf{atr}_i) \in (\overrightarrow{\mathsf{aID}}_{\mathsf{dscl}}, \overrightarrow{\mathsf{atr}}_{\mathsf{dscl}})$:

  $\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{tdecrypt}}^{\mathsf{uID}}(\mathsf{Verify}, \mathsf{sid}, (c_{\mathsf{aID}_i}, \mathsf{atr}_i))$;

$\longrightarrow \mathcal{Z} : (\mathsf{verify\text{-}resp}, (\overrightarrow{\mathsf{aID}}, \overrightarrow{\mathsf{atr}}, \mathsf{IID}, \mathsf{ch}), 1)$;

$\mathsf{Verify}_{\mathsf{ST}}$

// wait for user connection

$U \longrightarrow$: $\mathsf{tag}_{verify}, \mathsf{sid}, \mathsf{uID}$;

// participate in generating threshold signature

$\mathsf{ok} \leftarrow \mathcal{F}_{\mathsf{tsign}}^{\mathsf{uID}}(\mathsf{sign}, \mathsf{sid})$

Figure 44: Verification of credential presentation – User and AMS

Curriculum Vitae

1. Personal data

Name	Jelizaveta Vakarjuk
Date and place of birth	17 September 1997 Tallinn, Estonia
Nationality	Estonian

2. Contact information

Address	Cybernetica AS, Mäealuse 2/1, 12618 Tallinn, Estonia
E-mail	jelizaveta.vakarjuk@cyber.ee

3. Education

2022–...	Tallinn University of Technology, School of Information Technologies, Department of Software Science, PhD studies
2019–2021	Tallinn University of Technology, School of Information Technologies, Cybersecurity, MSc *cum laude*
2016–2019	Tallinn University of Technology, School of Information Technologies, Business Information Technology, BSc *cum laude*

4. Language competence

Russian	native
Estonian	fluent
English	fluent

5. Professional employment

2021– ...	Cybernetica AS, Junior Researcher
2020–2022	STACC, Junior Researcher

9. Defended theses

- 2021, Converting a Post-Quantum Signature Scheme to a Two-Party Signature Scheme, MSc, supervisor Prof. Ahto Buldas, PhD Jan Villemson, Tallinn University of Technology, School of Information Technologies
- 2019, Theoretical and Analytical Overview of Lattice-Based Key Establishment Protocols, BSc, supervisor Prof. Ahto Buldas, Tallinn University of Technology, School of Information Technologies

10. Field of research

- privacy-preserving cryptography
- threshold cryptography
- post-quantum cryptography

11. Scientific work
Papers

1. A. Kalu, B. C. Kus, P. Laud, K. L. Leung, N. Snetkov, and J. Vakarjuk. CACS: A cloud privacy-preserving attribute management system. In *Proceedings of the 18th International Conference on Availability, Reliability and Security, ARES 2023, Benevento, Italy, 29 August 2023- 1 September 2023*, pages 122:1–122:9. ACM, 2023

2. P. Laud, A. Pankova, and J. Vakarjuk. Privacy-preserving server-supported decryption . In *2025 IEEE 38th Computer Security Foundations Symposium (CSF)*, pages 48–63, Los Alamitos, CA, USA, June 2025. IEEE Computer Society

3. J. Vakarjuk and A. Pankova. Prettiness – privacy preserving attribute management system, 2026

4. P. Laud and J. Vakarjuk. A comparison-based methodology for the security assurance of novel systems. In S. K. Katsikas, F. Cuppens, C. Kalloniatis, J. Mylopoulos, F. Pallas, J. Pohle, M. A. Sasse, H. Abie, S. Ranise, L. Verderame, E. Cambiaso, J. M. Vidal, M. A. S. Monge, M. Albanese, B. Katt, S. Pirbhulal, and A. Shukla, editors, *Computer Security. ESORICS 2022 International Workshops - CyberICPS 2022, SECPRE 2022, SPOSE 2022, CPS4CIP 2022, CDT&SECOMANE 2022, EIS 2022, and SecAssure 2022, Copenhagen, Denmark, September 26-30, 2022, Revised Selected Papers*, volume 13785 of *Lecture Notes in Computer Science*, pages 625–644. Springer, 2022

5. J. Vakarjuk, A. Pankova, and A. Kalu. Comparison of private key protection techniques for digital identity wallets. In *2026 IEEE European Symposium on Security and Privacy Workshops*, pages 396–412, 2026

6. J. Vakarjuk, N. Snetkov, and J. Willemson. Dilizium: A two-party lattice-based signature scheme. *Entropy*, 23(8):989, 2021

7. J. M. Kirss, P. Laud, N. Snetkov, and J. Vakarjuk. Server-supported decryption for mobile devices. In G. Lenzini and W. Meng, editors, *Security and Trust Management - 18th International Workshop, STM 2022, Copenhagen, Denmark, September 29, 2022, Proceedings*, volume 13867 of *Lecture Notes in Computer Science*, pages 71–81. Springer, 2022

8. J. Vakarjuk, N. Snetkov, and J. Willemson. Russian federal remote e-voting scheme of 2021 - protocol description and analysis. In D. Megías, R. D. Pietro, and J. García-Alfaro, editors, *EICC 2022: European Interdisciplinary Cybersecurity Conference, Barcelona, Spain, June 15 - 16, 2022*, pages 29–35. ACM, 2022

9. K. Krips, N. Snetkov, J. Vakarjuk, and J. Willemson. Trust assumptions in voting systems. In S. K. Katsikas, H. Abie, S. Ranise, L. Verderame, E. Cambiaso, R. M. Ugarelli, I. Praça, W. Li, W. Meng, S. Furnell, B. Katt, S. Pirbhulal, A. Shukla, M. Ianni, M. D. Preda, K. R. Choo, M. P. Correia, A. Abhishta, G. Sileno, M. Alishahi, H. K. Kalutarage, and N. Yanai, editors, *Computer Security. ESORICS 2023 International Workshops - CPS4CIP, ADIoT, SecAssure, WASP, TAURIN, PriST-AI, and SECAI, The Hague, The Netherlands, September 25-29, 2023, Revised Selected Papers, Part II*, volume 14399 of *Lecture Notes in Computer Science*, pages 309–329. Springer, 2023

10. J. Vakarjuk, N. Snetkov, and P. Laud. Identifying obstacles of PQC migration in e-estonia. In C. Kwan, L. Lindström, D. Giovannelli, K. Podins, and D. Strucl, editors, *16th International Conference on Cyber Conflict: Over the Horizon, CyCon 2024, Tallinn, Estonia, May 28-31, 2024*, pages 63–81. IEEE, 2024

11. N. Snetkov, J. Vakarjuk, and P. Laud. TOPCOAT: towards practical two-party crystals-dilithium. *Discov. Comput.*, 27(1):18, 2024

12. J. Vakarjuk, N. Snetkov, and J. Willemson. Comparing security levels of postal and internet voting. *Information Security Journal: A Global Perspective*, 0(0):1–21, 2024

13. N. Snetkov, J. Vakarjuk, and P. Laud. Electrum: UC fail-stop server-supported signatures. In G. Yang, S. Liu, C. Su, A. Otsuka, and Z. Lian, editors, *Provable and Practical Security - 19th International Conference, ProvSec 2025, Yokohama, Japan, October 10-12, 2025, Proceedings*, volume 16172 of *Lecture Notes in Computer Science*, pages 88–107. Springer, 2025

14. N. Snetkov, J. Vakarjuk, and P. Laud. Universally composable server-supported signatures for smartphones. *IACR Commun. Cryptol.*, 3(1):13, 2026

15. N. Snetkov, J. Vakarjuk, and A. Pankova. Coppercloud: Blind server-supported RSA signatures. In S. D. C. di Vimercati and P. Samarati, editors, *Proceedings of the 23rd International Conference on Security and Cryptography, SECRYPT 2026, Porto, Portugal, July 16-18, 2026, Volume 1*, pages 419–428. SCITEPRESS, 2026

Elulookirjeldus

1. Isikuandmed

Nimi	Jelizaveta Vakarjuk
Sünniaeg ja -koht	17.09.1997, Tallinn, Eesti
Kodakondsus	Eesti

2. Kontaktandmed

Aadress	Cybernetica AS, Mäealuse 2/1, 12618 Tallinn, Estonia
E-mail	jelizaveta.vakarjuk@cyber.ee

3. Haridus

2022–…	Tallinna Tehnikaülikool, Infotehnoloogia teaduskond, Tarkvarateaduse instituut, doktoriõpe
2019–2021	Tallinna Tehnikaülikool, Infotehnoloogia teaduskond, Küberkaitse, MSc *cum laude*
2016–2019	Tallinna Tehnikaülikool, Infotehnoloogia teaduskond, Äriinfotehnoloogia, BSc *cum laude*

4. Keelteoskus

vene keel	emakeel
eesti keel	kõrgtase
inglise keel	kõrgtase

5. Teenistuskäik

2021– …	Cybernetica AS, Nooremteadur
2020–2022	STACC, Nooremteadur

9. Kaitstud lõputööd

- 2021, Kahe osapoolega signeerimisprotokoll postkvant signatuuriskeemi baasil, MSc, juhendaja Prof. Ahto Buldas, PhD Jan Villemson, Tallinna Tehnikaülikool, Infotehnoloogia teaduskond
- 2019, Võrepõhiste võtmekehtestusprotokollide ülevaade ja analüüs, BSc, juhendaja Prof. Ahto Buldas, Tallinna Tehnikaülikool, Infotehnoloogia teaduskond

10. Teadustöö põhisuunad

- privaatsust säilitav krüptograafia
- lävikrüptosüsteemid
- postkvant-krüptograafia

11. Teadustegevus

Teadusartiklite, konverentsiteeside ja konverentsiettekannete loetelu on toodud ingliskeelse elulookirjelduse juures.

ISSN 2585-6901
ISBN 978-9916-80-568-8