TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Cybersecurity

Nikita Snetkov 184614IVCM

PRACTICAL IMPLEMENTABILITY OF TWO-PARTY ECDSA SIGNATURE SCHEMES

Master's thesis

Supervisor: Prof. Dr. Ahto Buldas

Tallinn 2020

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Kuberturvalisus

Nikita Snetkov 184614IVCM

KAHE OSAPOOLEGA ECDSA SIGNEERIMISPROTOKOLLI PRAKTILINE TEOSTATAVUS.

Magistritöö

Juhendaja: Prof. Dr. Ahto Buldas

Tallinn 2020

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Snetkov Nikita

03.08.2020

Abstract

Usage of digital signatures became an important part of our lives. The main security of digital signatures is a storage of private keys. If private key gets corrupted adversary will be able to impersonate legitimate owner of the stolen private key by forging their digital signature. To address the issue of secure private key storage it is possible to use threshold cryptography. Using this approach one part of the private key can be stored on a client side, other part – on a server, and valid digital signature can be created only with interaction between client and server. One of the most widely used digital signature schemes nowadays is ECDSA because it fast and efficient. Still, there is no optimal threshold ECDSA developed, however it is currently active area of cryptographic research.

The purpose of this thesis is to implement and analyse one of the proposed threshold ECDSA schemes created by Northeastern University and compare implementation with existing solutions. The scheme was implemented in Java, which is the first implementation of this scheme this programming language. Developed implementation shows quite competitive results for client-server communication and can be used in further research.

During the work, it was revealed that chosen scheme is the most efficient in terms of key generation and signature generation. However, it requires higher network bandwidth compared to other schemes.

This thesis is written in English and is 60 pages long, including 6 chapters, 10 figures and 5 tables.

Annotatsioon

Digitaalallkirja kasutamine on saanud tähtsaks osaks meie elus. Digitaalallkirjaga seonduv peamine turvaülesanne on privaatvõtmete haldus. Kui privaatvõti lekib, siis on võimalik võltsida võtme omaniku digitaalallkirja. Privaatvõtme haldusega seotud riskide maandamiseks võib kasutada lävikrüptograafiat, kus osa privaatvõtmest salvestatakse kasutaja seadmesse, teine aga serverisse, kusjuures korrektse digitaalallkirja saab moodustada üksnes kliendi seadme ja serveri koostöös. Üks tänapäeval laialdasemalt kasutatavaid digitaalallkirja skeeme on Elliptkõverail põhinev ECDSA, sest ta on kiire ja efektiivne. Seni teadaolevate lävikrüptograafia meetodite kasutamise efektiivsus ECDSA korral aga jätab aga soovida, ehkki selles vallas käib aktiivne teaduslik uurimistöö.

Selle magistritöö eesmärk on programselt teostada ja analüüsida üht efektiivseimat ECDSA lävikrüptograafial põhinevat meetodit, mille esitasid Northeastern University teadurid, ja võrrelda selle efektiivsust teiste teadaolevate meetodite ja nende teostustega. Skeem programmeeriti Javas, mis on esmakordne selles keeles teostus nimetatud skeemile. Teostuse analüüs näitab, et skeem on praktikas kasutatav klient-server tüüpi lahendustes ja seda võib kasutada nimetatud skeemi efektiivseid teostusi käsitlevates edasistes uurimistöödes.

Töö selgitas välja, et valitud skeem on väga efektiivne võtme genereerimise ja samuti digitaalallkirja moodustamise seisukohalt, kuid jääb oluliselt alla teistele skeemidele võrgusuhtluse mahu poolest.

Lõputöö on kirjutatud [mis keeles] keeles ning sisaldab teksti 60 leheküljel, 6 peatükki, 10 joonist, 5 tabelit.

List of abbreviations and terms

AES	Advanced Encryption Standard
API	Application Programming Inteface
CDH	Computational Diffie-Hellman
DCRA	Decisional Composite Residuosity Assumption
DES	Data Encryption Standard
DDH	Decisional Diffie–Hellman
DKG	Distributed Key Generation
DLP	Discrete Logarithm Problem
DSA	Digital Signature Algorithm
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
FIPS	Federal Information Processing Standard
GCD	Greatest Common Divisor
HPS	Hash Proof System
JCA	Java Cryptography Architecture
JCE	Java Cryptography Extension
MD5	Message Digest Algorithm
MPC	Multiparty Computation
NIST	National Institute of Standards and Technology
NP-Problem	Nondeterministic Polynomial Time
OT	Oblivious Transfer
SHA	Secure Hash Algorithm
SMP	Subgroup membership problem
SSH	Secure Shell
TLS	Transport Level Security
TSS	Threshold Signature Scheme
ZKP	Zero-knowledge proof

Table of contents

List of figures

List of tables

1 Introduction

1.1 Research Motivation

The Elliptic Curve Digital Signature Algorithm (ECDSA) is one of the most used cryptographic designs in modern IT infrastructure. ECDSA is the predominant algorithm used in protection of data exchange, bank transactions, secure communication, and national electronic identification documents such as the Estonian ID card. As ECDSA is based on asymmetric cryptography, there exists security issues due to the storing of ECDSA private keys. The private key should not be known to anyone besides the author of the digital signature. For many years there were mostly two ways to keep private keys secure: storage of private keys protected by the filesystem security model, where it is exposed to privilege escalation vulnerabilities, or, compiled inside software binary code, where it could be decompiled by malicious adversary.

Threshold signatures bring new solution for the previously stated issue. Instead of having one private key, which can be stolen, there can be numerous parts of private keys distributed to different parties that can group up to create one signature. For example, first part of the private key is given to a server, second part is given to a client. Only through combining both parts of the private key the digital signature can be created. Executing threshold signatures in a secure manner is challenging due to the mathematical design of ECDSA, unlike, other less used signature schemes.

Due to the prevailing interest of blockchain technology and cryptocurrency, in past four years, there were several academic papers proposed on the topic of Threshold ECDSA schemes. The main motivation for this research comes from Northeastern University "Secure Two-party Threshold ECDSA from ECDSA Assumptions" [1]. This paper proposed one of the first solutions that required only elliptic curve primitives and elliptic curve assumptions. Still, there was no research conducted on the topic of testing this scheme in terms of server-client scenario. The motivation of this paper is to provide new contribution in the area of threshold Two-Party signatures.

1.2 Scope

This paper concentrates on a client-server scenario and excludes Threshold Multiparty ECDSA scheme from the research's scope, due to the fact most of them were developed for blockchain application with multiple amounts of communicating parties.

1.3 Research Questions

The following research questions are to be answered:

1. What are two-party ECDSA solutions already existing beside [1]?
2. Which one has fastest key generation?
3. Which one has fastest signature creation process?
4. Which one does require the lowest network bandwidth?

1.4 Research Goals

Author's goal is to implement "Secure Two-party Threshold ECDSA from ECDSA Assumptions" [1] for client-server scenario using Java programming language. After that compare it with the original code of scheme written in Rust. Additionally, compare the code with other two-party implementations of ECDSA.

1.5 Student's contribution

The schemes were researched in this thesis were created by Northeastern University. I contribute implementation of scheme [1] and comparison of existing two-party schemes with my implementation.

1.6 Thesis Structure

This thesis is organized in six chapters. Chapter 1 is the introduction and motivation of this work. Chapter 2 details the background of cryptographic primitives and digital signatures. Chapter 3 consist of explaining of concepts of Threshold Signatures and related work. Chapter 4 explains principles of Secure Two-party Threshold ECDSA from ECDSA Assumptions scheme. Chapter 5 explains the implementation of Secure Two-party Threshold ECDSA from ECDSA Assumptions scheme. Chapter 6 is the conclusion and the future work.

2 Background

This chapter summarizes the background information, concepts and algorithms used during the analysis, development, and evaluation of the present research.

2.1 Field theory

Cryptography relies on numerous areas of mathematics. Without an understanding of the mathematics behind cryptography, it is impossible to understand cryptography itself. Field theory is one of the essential mathematical topics implied in modern cryptography. The following part of the thesis contains definitions and explanations of the basic field theory concepts that are needed for understanding Elliptic Curve Cryptography (ECC) and ECDSA.

2.1.1 Set

Definition 2.1.1. (Set): Defined collection, compilation, or assemblage of definite objects in mathematics is called **set.** *Examples of sets*: set of natural numbers, set of Latin symbols, set of TalTech students' names.

Definition 2.1.2. (Subset): Let *A* and *B* be some sets. Set *B* called **subset** of *A* if and only if all elements of *B* belong to the set A.

Definition 2.1.3. (Empty set): Set consisting of zero elements is called **empty set**.

Definition 2.1.4. (Binary operation): Computation implicating two elements of the set to produce another element of the set is **binary operation.** Examples: addition, multiplication, exponentiation in the set of natural numbers.

Definition 2.1.5. (Cardinality): Number of elements in a set is called **cardinality**.

Definition 2.1.6. (Finite and infinite sets): If set consist of finite number of elements, it is called **finite set.** Otherwise the set is **infinite.**

2.1.2 Group

Definition 2.1.7. (Group): Let *G* be a non-empty finite or infinite set with a defined binary operation (•), this set is called a **group** if and only if the following properties are satisfied:

- *Closure.* If two elements belong to a set *G*, the result of binary operation also belong to set *G:* $\forall a, b \in G: a \bullet b \in G$;
- *Identity element.* There is a unique element $e \in G$, called **identity element** such that for all other elements in a set we have: $\exists e \in G, \forall a \in G: a \bullet e = e \bullet a = a$; For additive operation identity element is usually denoted as 0; for multiplicative operation as 1.
- *Inverse element.* For every element of a set there is a unique element in a set called an **inverse element:** $\forall a \in G\ \exists b: a \bullet b = b \bullet a = e$;. For additive operation inverse element for an element $a \in G$ is denoted as *-a;* for multiplicative operation inverse of an element $a \in G$ is a^{-1}.
- *Associativity.* $\forall a, b, c \in G \Rightarrow (a \bullet b) \bullet c = a \bullet (b \bullet c)$

A group with defined addition operation is notated as *(G, +)* and called *additive group,* a group with multiplication operation – *(G, *), +)* and called *multiplicative group. Example of additive group:* group of integers. *Example of multiplicative group:*

Definition 2.1.8. (Abelian group): Let *G* be a group with an operation (•). The group is called **abelian** or **commutative,** if for every element of a group following is true: $\forall a, b \in G \Rightarrow a \bullet b = b \bullet a$

Definition 2.1.9. (Cyclic group): Let *G* be a group with an operation (•). The group is called **cyclic,** if every element of *G* can be calculated by repeated utilization of the group operation over one exact element, which is called a **generator**.

Definition 2.1.10. (Subgroup): Let *H* be a non-empty subset of group *G* with an operation (•). Set *H* is a **subgroup** of group *G*, if *H* forms a group with an operation (•).

Definition 2.1.11. (Order of a group): Let *G* be a group. The number of elements of a group *G* is called an **order of a group.** In other words, the order of a group is a cardinality of a group.

Definition 2.1.12. (Order of an element): Let *G* be a group. The **order of element** $a \in G$ is the smallest integer *n*, such that $a^n = e$ in multiplicative notation; $\left(a + \underset{n}{a +} \ldots + a\right) = e$ in additive notation, where *e* is the identity element of group *G*.

2.1.3 Field

Definition 2.1.13. (Field): Let *F* be a non-empty finite or infinite set with two defined binary operations (+ and *), the set is called a **field**, if and only if the following properties are met:

- *Additive group: F* forms an abelian group with operation + (*addition*). This group *(F, +)* is called an additive group of the field.
- *Multiplicative group: F* without 0 *(F \ {0})* forms an abelian group with operation * (*multiplication*). This group *(F, *)* is called a multiplicative group of the field.
- *Distributivity:* $\forall a, b, c \in F \Rightarrow (a + b) * c = a * c + b * c$

Definition 2.1.14. (Finite and infinite fields): Let *F* be a field. It is called a **finite** field if it consists of finite numbers of elements. Otherwise, ***F*** is an **infinite** field.

Examples of infinite fields: rational numbers ($\mathbb{Q}$), real numbers ($\mathbb{R}$), complex numbers ($\mathbb{C}$). Examples of infinite fields: residues of modulo *p (*0 mod p, 1 mod p, 2 mod p, …, p-1 mod p)*,* where *p* is a prime number.

Definition 2.1.15. (Field characteristic): Let *F* be a field. If there exists an integer number *p* such that for every element of the field F $(a \in F)$, the equation $a * p = 0$ holds, then the smallest *p* is called **field characteristic.** Field characteristic can be an only prime number. The field characteristic of *F* is denoted as *char(F)=p.*

Definition 2.1.16. (Order of a field): Let *F* be a field. The number of elements of finite field is called **order of a field** *F*. Finite field of order *q* is usually denoted as $\mathbb{F}_\mathrm{q}$. The field only exists if $\mathrm{q} = \mathrm{char(F)}^\mathrm{n}$, where $n \geq 1$.

Definition 2.1.17. (Finite field of modulo *p*): Finite field of modulo *p* consists of integers from 0 to *p*-1, where *p* is a prime number. It is denoted as $\mathbb{F}_p$. Additive group is formed with modular addition ($a, b \in \mathbb{F}_p;\ a + b \bmod p$, where *mod p* denotes the remainder of division by *p)* and integers from 0 to *p*-1. Multiplicative group is formed with modular multiplication ($a, b \in \mathbb{F}_p;\ a * b \bmod p$, where *mod p* is remainder of division by *p)* and integers from 1 to *p*-1.

2.2 Elliptic curves

Since ECDSA uses elliptic curves, the following section will introduce the basic concepts of the elliptic curves: elliptic curves over real numbers, elliptic curves over finite fields and operation over elliptic curves.

2.2.1 Elliptic curves over real numbers

Definition 2.2.1. (Curve): curve is a set of points, which could be described by some mathematical equation in Euclidian plane. This mathematical object is similar to a line, however it does not need to be straight.

Definition 2.2.2. (Slope): a value that describes both the direction and steepness of the line is called **slope**.

Definition 2.2.3. (Tangent): line that touches curve in the one point is called **tangent** [Figure 1] line to that point.

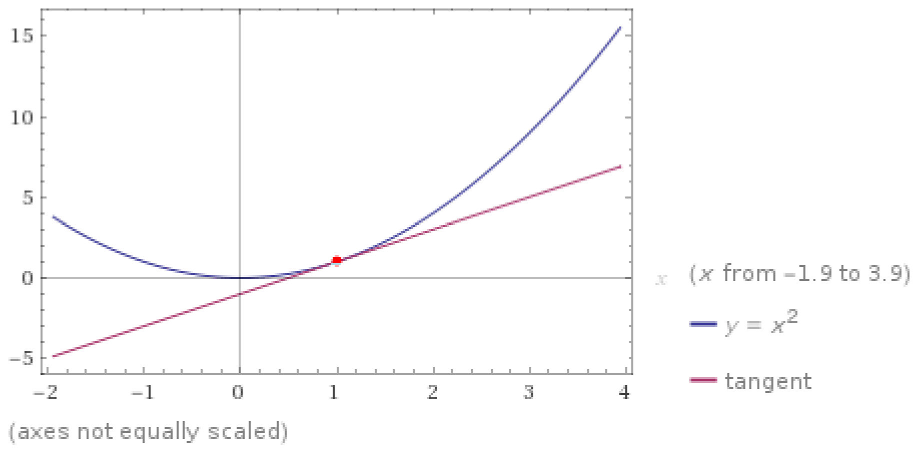


Figure 1: Tangent line to parabola

Definition 2.2.4. (Point at infinity): the point at the edges of all lines parallel to the *y*-axis.

Definition 2.2.5. (Elliptic curve over real numbers): elliptic curve [Figure 2] is a curve defined with equation $y^2 = x^3 + ax + b$, where $y, x, a, b \in \mathbb{R}$ and $4a^3 + 27b^2 \neq 0$ with defined point at infinity (noted as *O*). Definition also can be redefined:

$$\{(x, y) \in \mathbb{R}^2 \mid y^2 = x^3 + ax + b, 4a^3 + 27b^2 \neq 0\} \cup \{O\}$$

If $4a^3 + 27b^2 = 0$ it will be impossible to draw tangent line to every point of elliptic curve.

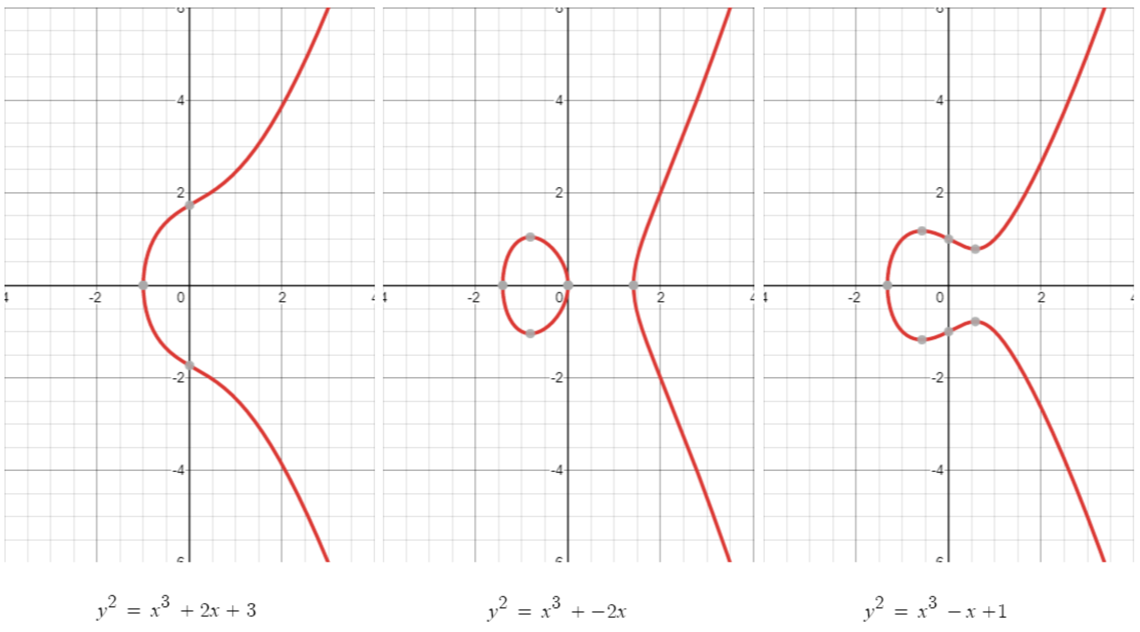


Figure 2: Graphs of elliptic curves

Group [Definition 2.1.7] can be defined for elliptic curve over real numbers:

1. Points of a curve and point at infinity *O* are elements of a group.

2. Binary operation is *geometric addition (+)*: for two non-zero points *P* and *Q* draw a line passing through *P* and *Q*, which will intersect a curve in third point *R*. Inverse of third point (*point symmetrical to X-axis*) is result of geometric addition: $P + Q = -R$ [Figure 3]

3. Point at infinity *O* is defined as *neutral element* of this group. For a point *P* on a curve -- $P + O = O + P = P$ [Figure 4]

4. For any point P exists point Q, which is *inverse element*, such as $Q = -P$ and $P + Q = O$, where O is point at infinity. [Figure 5]

5. To add point to itself tangent line is built in point P. Result of this geometric addition will be inverse point of intersection between the curve and the line tangent in point P. [Figure 6]

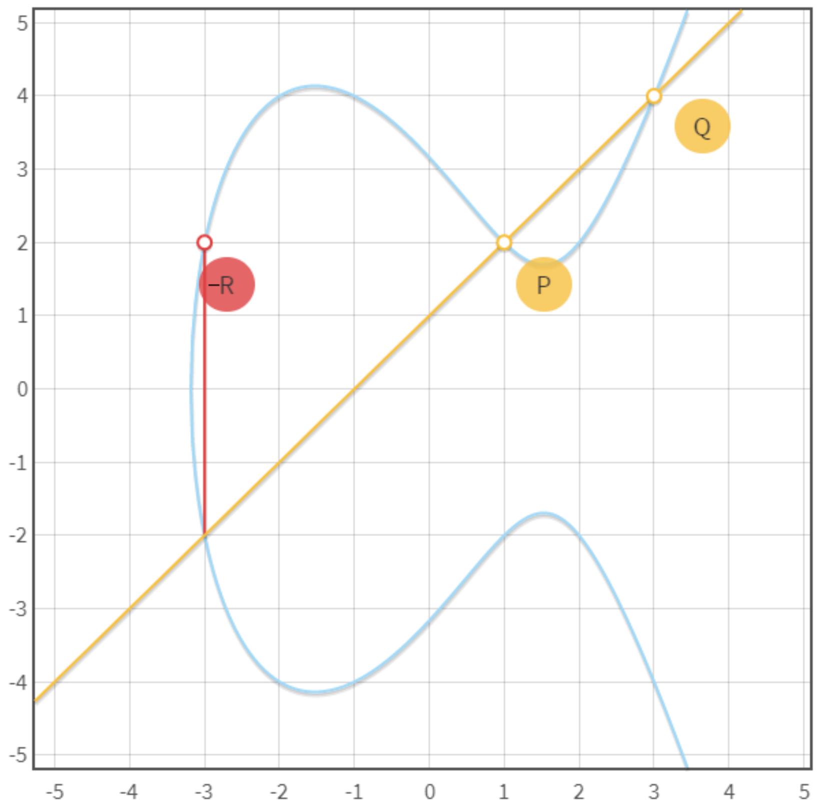


Figure 3: Geometric point addition $P + Q = -R$ over the elliptic curve $y^2 = x^3 - 7x + 10$

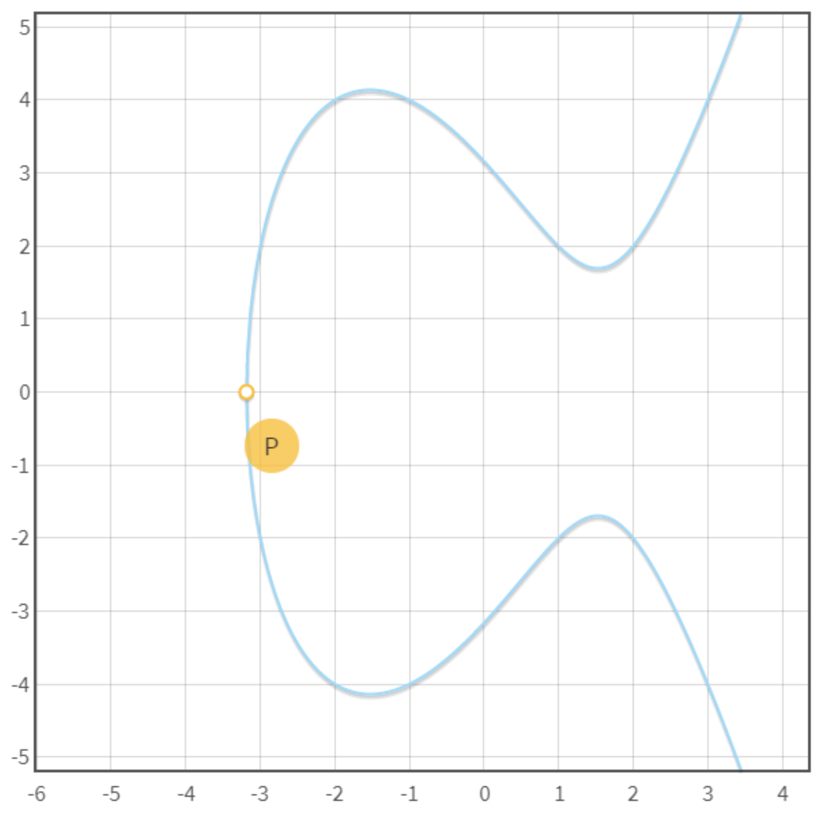


Figure 4: Geometric point addition $P + O = P$ over the elliptic curve $y^2 = x^3 - 7x + 10$

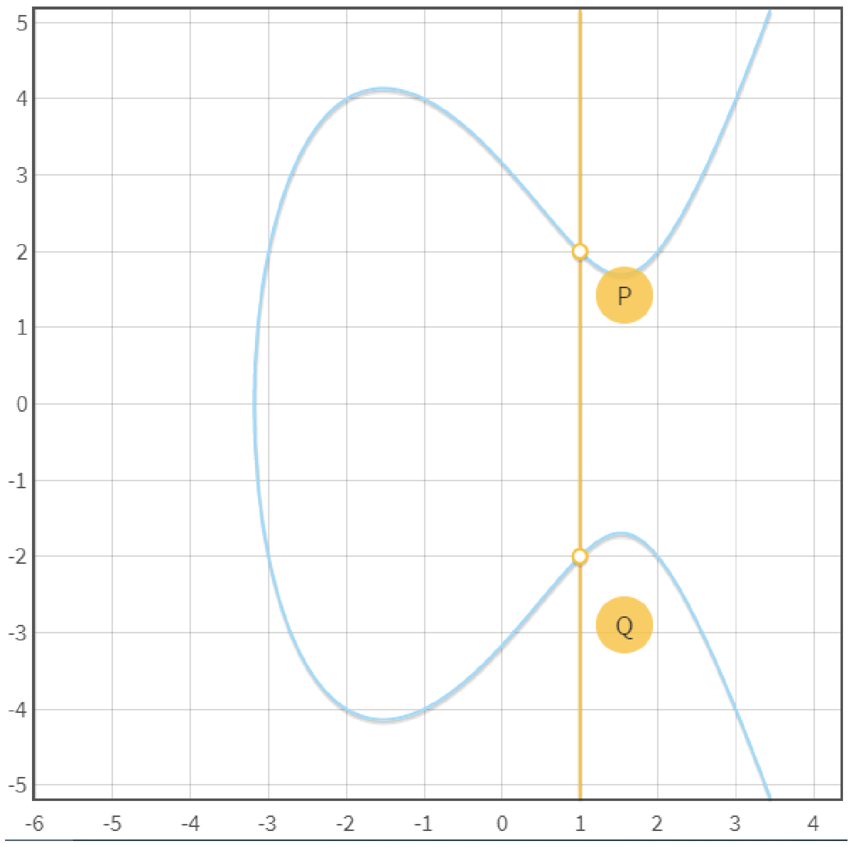


Figure 5: Geometric point addition $P + Q = P - P = O$ over the elliptic curve $y^2 = x^3 - 7x + 10$

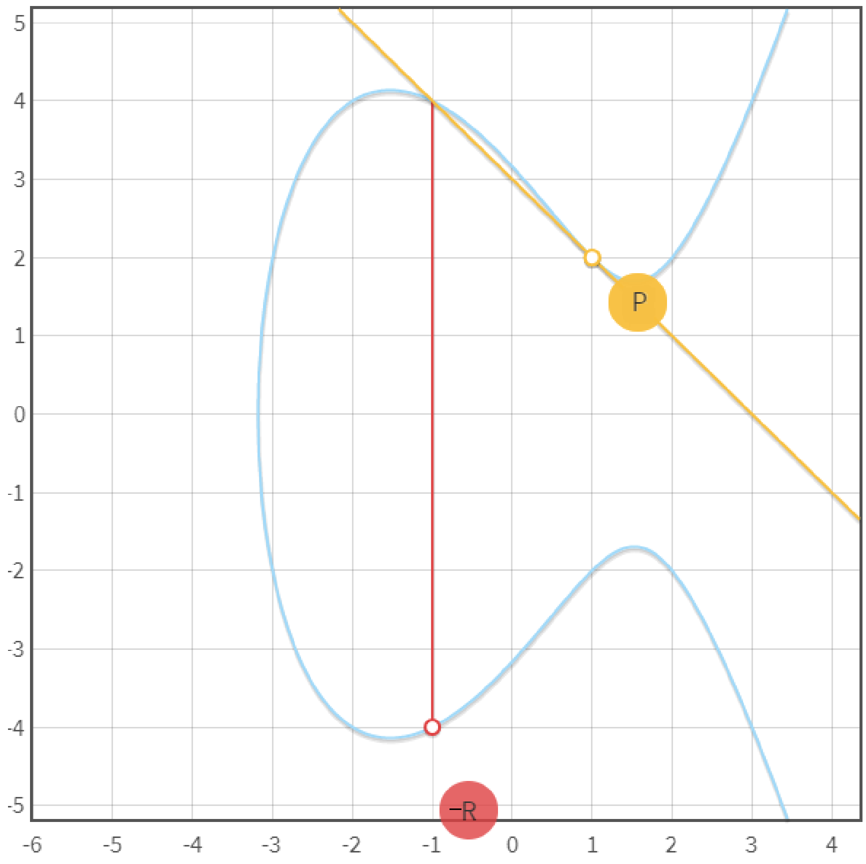


Figure 6: Geometric point addition $P + P = -R$ over the elliptic curve $y^2 = x^3 - 7x + 10$

Geometric addition can be represented with algebraic addition for points P and Q with Euclidian coordinates x_p, y_p and x_q, y_q accordingly:

- For a point *P* with coordinates (x_p, y_p) and point at infinity *O* the result of addition will be: $P + O = P$
- For points *P* and *-P* with coordinates (x_p, y_p) and $(-x_p, y_p)$: *P - P=O*
- For points *P* and *Q (P≠Q)* with coordinates (x_p, y_p) and (x_q, y_q):
    - Calculate slope [Definition 2.2.2] with formula: $s = \frac{y_p - y_q}{x_p - x_q}$
    - Coordinates of a point *R:*

$$x_r = s^2 - x_p - x_q$$
$$y_r = y_p + s(x_r - x_p)$$

    - Find inverse point *-R* by negating x_r. The result of addition *P+Q=-R* with coordinates $(-x_r, y_r)$
- For points *P* and *Q (P=Q)* with coordinates (x_p, y_p):
    - Calculate slope with formula: $s = \frac{3x_p^2 + a}{2y_p}$, where a *is* one of parameter of elliptic curve [Definition 2.2.5.]
    - Coordinates of a point *R:*

$$x_r = s^2 - x_p - x_q$$
$$y_r = y_p + s(x_r - x_p)$$

    - Find inverse point *-R* by negating x_r. The result of addition *P+Q=-R* with coordinates $(-x_r, y_r)$

Definition 2.2.6. (Scalar multiplication): Let *P* be a point on defined elliptic curve. *P* is an element of a group *G* with operation + (geometric/algebraic addition). **Scalar multiplication** is operation defined with formula: $k * P = kP = \left(P + P \underset{k}{+} \ldots + P\right) = Q$, where Q is point of elliptic curve and element of group *G.*

2.2.2 Elliptic curves over field of prime numbers.

There is a problem of storing real numbers in binary due to the rounding. Therefore, in cryptography, and ECDSA specifically, elliptic curves are specified over finite fields of modulo *p*, where *p* is prime number.

Definition 2.2.7. (Elliptic curves over $\mathbb{F}_\mathrm{p}$): elliptic curve over $\mathbb{F}_\mathrm{p}$ [Figures 7-9] is a curve defined with equation $y^2 = x^3 + ax + b(modp)$, where $y, x, a, b \in \mathbb{F}_{\mathrm{p},}$ $\mathrm{char}(\mathbb{F}_{\mathrm{p},}) \neq 2^\mathrm{n}, 3^\mathrm{n}$; where $\mathrm{n} \geq 1$ and $4a^3 + 27b^2 \not\equiv 0(modp)$with defined point at infinity (noted as *O*). Definition also can be redefined:

$$\{(x, y) \in (\mathbb{F}_\mathrm{p})^{\,2} \,|\, y^2 = x^3 + ax + b\ (mod\ p), 4a^3 + 27b^2 \not\equiv 0\ (mod\ p)\} \cup \{O\}$$

In case of $\mathrm{char}(\mathbb{F}_{\mathrm{p},}) = 2^\mathrm{n}$, where $\mathrm{n} \geq 1$, elliptic curve will be:

$$\{(x, y) \in (\mathbb{F}_\mathrm{p})^{\,2} \,|\, y^2 + y = x^3 + ax + b\ (mod\ p), 4a^3 + 27b^2 \not\equiv 0\ (mod\ p)\} \cup \{O\}$$

In case of $\mathrm{char}(\mathbb{F}_{\mathrm{p},}) = 3^\mathrm{n}$, where $\mathrm{n} \geq 1$, elliptic curve will be:

$$\{(x, y) \in (\mathbb{F}_\mathrm{p})^{\,2} \,|\, y^2 + y = x^3 + cx^2 + ax + b\ (mod\ p), 4a^3 + 27b^2 \not\equiv 0\ (mod\ p)\} \cup \{O\}$$

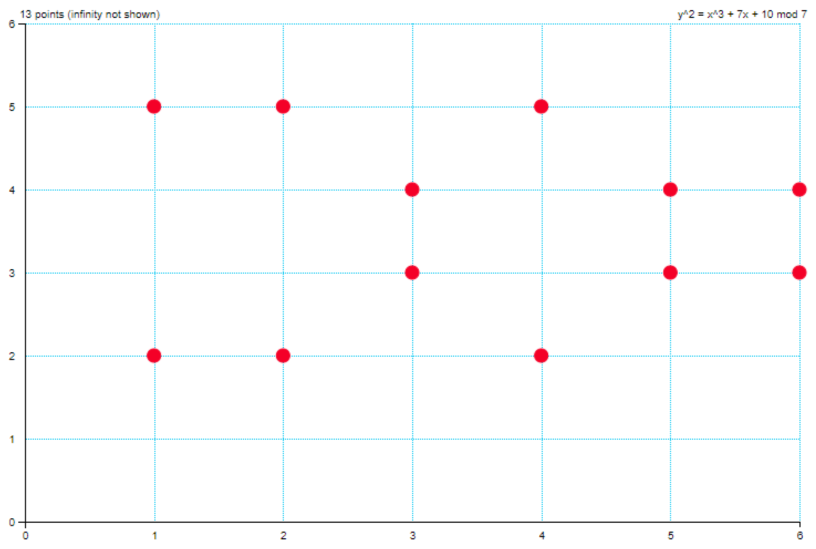


Figure 7: Graph of elliptic curve $y^2 = x^3 - 7x + 10$ over $\mathbb{F}_7$

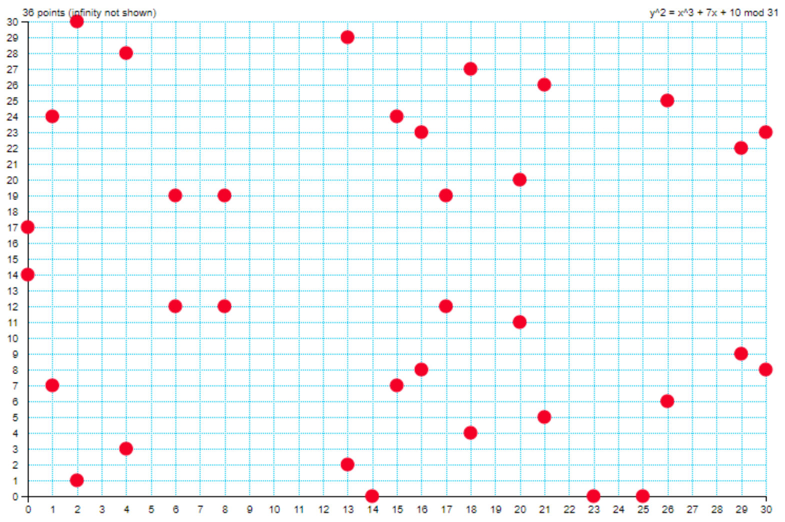


Figure 8: Graph of elliptic curve $y^2 = x^3 - 7x + 10$ over $\mathbb{F}_{31}$

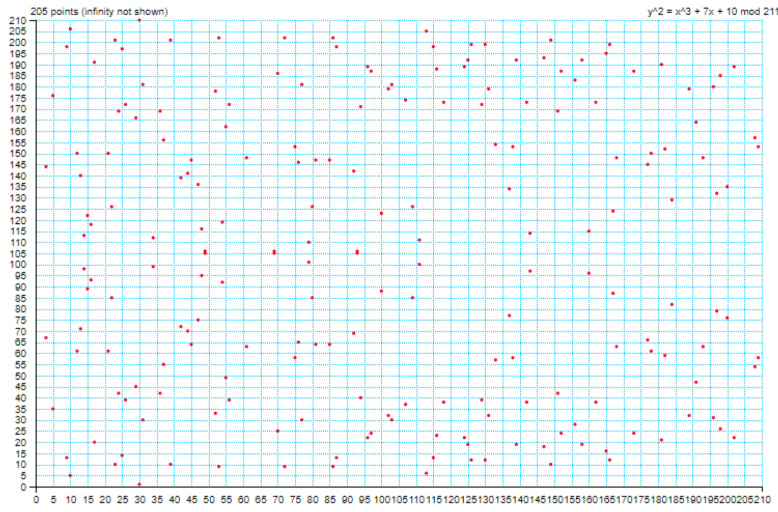


Figure 9: Graph of elliptic curve $y^2 = x^3 - 7x + 10$ over $\mathbb{F}_{211}$

Like with elliptic curves of real numbers, points of elliptic curve over $\mathbb{F}_\mathrm{p}$ form a group with binary operation of algebraic point addition:

- For a point *P* with coordinates (x_p, y_p) and point at infinity *O* the result of addition will be: *P* + *O* = *P*
- For points *P* and -*P* with coordinates (x_p, y_p) and $(-x_p, y_p)$: *P* - *P*=*O*
- For points *P* and *Q* (*P*≠*Q*) in $\mathbb{F}_\mathrm{p}$ with coordinates (x_p, y_p) and (x_q, y_q):
  - Calculate slope with formula: $s = (y_p - y_q)(x_p - x_q)^{-1} \, mod \, p$. ($q^{-1} mod \, p$ – is modulo inverse operation)
  - Coordinates of a point *R:*

$$x_r = (s^2 - x_p - x_q) \, mod \, p$$

$$y_r = y_p + s\left(x_r - x_p\right) mod \, p$$

    - Find inverse point -*R* by negating x_r. The result of addition *P*+*Q*=-*R* with coordinates $(-x_r \, mod \, p, y_r)$
- For points *P* and *Q (P=Q)* in $\mathbb{F}_p$ with coordinates (x_p, y_p):
    - Calculate slope with formula: $s = (3x_p^2 + a)(2y_p)^{-1} \, mod \, p$, where *a is* one of parameter of elliptic curve [Definition 2.2.7.]
    - Coordinates of a point *R:*

$$x_r = (s^2 - x_p - x_q) \, mod \, p$$
$$y_r = (y_p + s(x_r - x_p)) \, mod \, p$$

Find inverse point -*R* by negating x_r. The result of addition *P*+*Q*=-*R* with coordinates $(-x_r \, mod \, p, y_r)$

Definition 2.2.8. (Scalar multiplication in $\mathbb{F}_p$): For an elliptic curve over $\mathbb{F}_p$ point *P* and integer k ≥ 1, define **scalar multiplication in** $\mathbb{F}_p$: $kP = \left(P + P \underset{k}{+} \ldots + P\right) = Q$, where + is geometric/algebraic addition.

Scalar multiplication over points for elliptic curves in $\mathbb{F}_p$ allows to generate cyclic subgroups of elliptic points. [Definitions 2.1.9-10]

Definition 2.2.9. (Order of a point): Let *P* be a point on defined elliptic curve over $\mathbb{F}_p$. *P* is an element of a group *G* with operation + (geometric/algebraic addition). The smallest natural number *n* such as $nP = O$, where is a point at infinity, is called **order of a point.** *P* generates cyclic subgroup of order *n* [Definition 2.1.11]. Point *P* is called **based point (generator).**

Definition 2.2.10. (Cofactor of a subgroup): Let *N* be an order of $\mathbb{F}_p$ additive group. Let *n* be an order of cyclic subgroup generated by point *P* of elliptic curve over $\mathbb{F}_p$. The number $\mathrm{h} = \frac{\mathrm{N}}{\mathrm{n}}$ is cofactor of the subgroup.

2.3 Digital Signatures

In the case of transmitting documents through insecure channels, there is a risk of their interception. An attacker can get access to a digital document, modify it and resend it to the original addressee in order to gain benefit. To avoid such scenario, a concept of digital signatures was introduced. In 1976, Whitfield Diffie and Martin E. Hellman in their paper

"New Directions in Cryptography" proposed an idea of system that could replace written signatures called *"digital signatures"*. [2] The main property of their proposal was that it must be easy to verify the signature, but not legitimate signer should not be able to reproduce it.

The **eIDAS**, a European Union regulation on electronic identification, gives the following definitions on digital signatures [3]:

- **electronic (digital) signature** – *data in electronic form which attach to or logically associated with other data in electronic form and which is used by the signatory to sign" (Article 3).* This definition of signature is quite broad and could mean as sophisticated cryptographic scheme, as person's initials at the end of digital document, as checkbox click of user agreement.

- **advanced electronic (digital) signature** – *an electronic signature which is uniquely linked to signatory (Articles 3,24)*. The way to achieve uniqueness requires usage of cryptographic methods.

- **qualified electronic (digital) signatures** – *advanced electronic signature that is created by a qualified electronic signature creation device, and which is based on a qualified certificate for electronic signatures (Article 3)*. For creation of such signature is important to use hardware security modules and the signature must be certified by the trusted authority. For example, signatures created using Estonian ID card are qualified electronic signatures.

Most digital signatures based on cryptographic means are advanced electronic signatures, and most of them could be turned into qualified electronic signatures. Cryptographic digital signatures should satisfy the following properties [4]:

1. **Authentic**. Digital signature must convince verifier that the signer wilfully made exact digital signature.

2. **Non-reusable.** Digital signature must be a part of document, it cannot be duplicated on another document.

3. **Unforgeable**. No one else besides the original signer can produce valid digital signature on behalf of the original signer.

4. **Unchangeable**. After producing digital signature, the document to which it was applied, cannot be changed, or altered in any way.

5. **Non-Repudiated.** The signer of digital signature cannot deny that they are the author of the document and signature.

Even though there are many different digital signature algorithms, they all can be divided in two groups: based on *symmetric* cryptography or *asymmetric (public-key)* cryptography. The first group uses block ciphers, hash functions or non-interactive zero-knowledge proofs to achieve properties listed before, in the second group signature schemes are based on highly complex mathematical problems (NP-Problems). As the topic of this paper is directly related to ECDSA, further the focus will be only on asymmetric signatures, to which ECDSA belongs.

A digital signature scheme consists of the following algorithms:

1. **Key generation.** The signer creates two keys - *private* and *public*.

2. **Signature generation (signing).** Signer uses their private key to make digital signature connected to document or message.

3. **Signature verification.** Anyone who obtained public key can verify the authorship and authenticity of the signature.

2.4 RSA signature

RSA is asymmetric cryptosystem proposed in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman [5]. Even though RSA was not the first proposed asymmetric cryptosystem and not only one, nowadays it is the most popular algorithm being used. The main purpose of RSA system is encryption; however, it could be used as digital signature scheme.

Definition 2.4.1. (Greatest common divisor): Let *x* and *y* be natural numbers. **Greatest common divisor** is the largest natural number that divides both of *x* and *y*. It is denoted as *gcd(x, y).*

Definition 2.4.2. (Coprime number): Let *x* and *y* be natural numbers. The numbers are **coprime** if and only if *gcd(x,y)*=1.

Definition 2.4.3. (Euler's totient function): Euler's totient function for any natural number *p* calculates the amount of natural numbers that are coprime to *p* and less than *p*. It is denoted as *φ(n),* where *n* is natural number. The formula for the function:

$$\varphi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right).$$

Definition 2.4.4. (Modulo inverse): Let a, n be integers. The inverse of an integer a modulo n is an integer b such that: $ab \equiv 1 \, mod \, n$. Modulo inverse is denoted as $a^{-1} \, mod \, n$.

Definition 2.4.5. (Hash function): a function that maps input of arbitrary size to fixed-sized output. Input is called *message* and output – *message digest*. Examples of hash functions: SHA, MD5, Streebog. Taking hash value for message *M* is denoted as *H(M).*

RSA key generation

- Generate randomly two big random prime numbers *p* and *q*.
- Calculate a product of *p* and *q*: $N = p * q$.
- Choose value *e* known as RSA public exponent in range of [1 < *e* < *φ(n)*]. The *φ(N)* equal to *(p-1)(q-1)*. If *e* and *φ(N)* are not coprime, in other words *gcd(x,y)*≠1, new *e* must be generated until this statement is satisfied.
- Calculate value *d* known as RSA private exponent s.t. $d * e \equiv 1$ (mod *φ(N)).*
- Pair of values *(e, N)* is **RSA public key**, pair of values *(d, φ(N))* is **RSA private key.**

RSA signature generation (signing)

- Let *M* be message. Calculate hash value of the message *h=H(M)*
- Message is signed with private exponent *d* using formula: $s = h^d mod \, N$
- Value *s* is RSA digital signature on message *M*

RSA signature verification

- Let M be message. Calculate hash value of the message $h=H(M)$
- Calculate $h' = s^e mod\ N$
- Signature s is valid if and only if values h and h' are equal

Security of RSA signature

RSA signature scheme security and unforgeability are based on assumption that there is no efficient algorithm for big integer factorization. If adversary finds factorization of RSA parameter $N=p*q$, they can calculate Euler's totient function, $\varphi(N)$, and RSA private exponent d. At the moment, there is no polynomial-time method for factoring integers using classical computers. However, there is Shor's algorithm [6] that can find factors of any integer, but it works only on quantum computers, which are yet meant to be built.

It is important to note, that 'textbook' RSA, described above is not secure for practical use due to the several reasons, in practice modified versions are used. Additionally, RSA is not secure, if p and q are not big enough numbers: there are already works that show 240-digit number factorization [7].

2.5 The Digital Signature Algorithm (DSA)

DSA (Digital signature Algorithm) is a public key algorithm for creating digital signatures proposed by National Institute of Standards and Technology (NIST) in 1991 and published as a part of Digital Signature Standard (DSS) in December 1998. [8]

DSA key generation

The DSA makes use of the following parameters:

- p - a prime number (*modulus*) generated in range $2^{L-1} < p < 2^L$ for $512 \leq L \leq 1024$ and L is multiple of number 64.
- q - prime divisor of (p – 1) in range $2^{159} < q < 2^{160}$

Key generation algorithm works as follows:

- Calculate value g with formula: $g = h^{\frac{p-1}{q}} mod$ p, where h is generated integer in range 1 < h < (p – 1) such that [($h^{\frac{p-1}{q}}$) $mod\, p$] > 1
- Generate integer *x:* (pseudo)randomly in range $0 < x < q$
- Calculate value *y* using formula: $y = g^x mod\, p$
- The integers p, q, and g can be publicly known and can be common to a group of users. **DSA user's private key** is *x* and **DSA public key** is *y*. The keys are usually fixed for a period of time.

DSA signature generation (signing)

- Generate (pseudo)randomly integer parameter *k* in range $0 < k < q$. Parameter *k* must be regenerated for each signature and must be kept in secret.
- Let *M* be message. Calculate hash value of the message *h=H(M)*
- The signature on a message *M* is the pair of numbers (*r, s)* computed according to the formulas:
    - $r = (g^k\, mod\, p)\, mod\, q$
    - $s = \left(k^{-1}(H(M) + xr)\right) mod\, q$, where k^{-1} is inverse of parameter *k* modulo q.

  If either r, or s is equal to zero, signature was generated incorrectly, and it must be generated again with the new parameter *k*. The pair *(r, s)* is **DSA signature** on message *M*.

DSA signature verification

- Let *M* be a message, let *(r, s)* be DSA signature that needs to be verified. Check if $0 < r < q$ and $0 < s < q$. In case either condition is not satisfied the signature must be rejected.
- Calculate value *w* according to the formula: $w = s^{-1}\, mod\, q$

- Calculate $u_1 = (H(M)w)\ mod\ q$ and $u_2 = (rw)\ mod\ q$
- Calculate verification value $v = ((g^{u_1}y^{u_2})\ mod\ p)\ mod\ q$. If $v = r$, then the signature *(r, s)* is verified and the verifier can have enough confidence that the received message was sent by the party holding the DSA private key x corresponding to public key y.

Both signing and verification processes are presented in Figure 10.

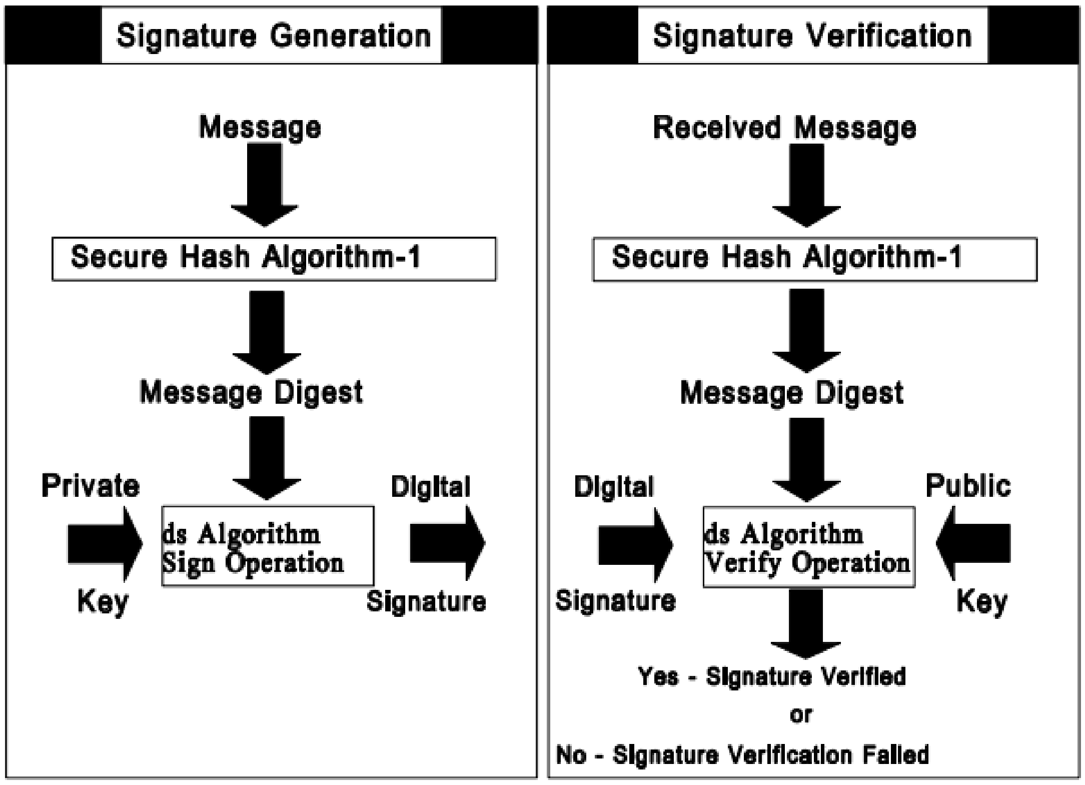


Figure 10: DSA signature generation and verification [8]

DSA Security (Discrete logarithm problem)

Security and unforgeability of DSA signature scheme are based on discrete logarithm problem (DLP). The computational hardness of this problem is not only essential for security of DSA, but for many other cryptographic protocols and primitives (for example ElGamal scheme [9] or Diffie-Helman key exchange [2]).

Definition 2.5.1 (Discrete logarithm problem): Let *G* be a cyclic multiplicative group, g – generator of group *G* of order *p*, and h – element of the cyclic group *G* ($h \in G$). The **discrete logarithm problem** is to find such integer *x* that $g^x \equiv h\ mod\ p$.

Provided level of security by Definition 2.5.1 of DSA problem is not sufficient, therefore, additional assumptions related to the DLP are used to prove security of DSA and similar cryptographic schemes.

Definition 2.5.2. (Computational Diffie-Hellman Assumption): Let *G* be a cyclic multiplicative group of order *p*, g – randomly chosen generator of group *G,* and *a,b* – randomly generated elements of the cyclic group *G* ($a, b \in G$). **Computational Diffie-Hellman Assumption (CDH)** assumption states that given to adversary values (g, g^a, g^b), they cannot compute g^{ab} in polynomial time.

Definition 2.5.3. (Decisional Diffie-Hellman): Let *G* be a cyclic multiplicative group of order *p*, g – randomly chosen generator of group *G,* and *a, b, c* – randomly generated elements of the cyclic group *G* ($a, b, c \in G$). **Decisional Diffie-Hellman (DDH)** assumption states given values g^c and g^{ab} to adversary are computationally indistinguishable.

DLP, CDH assumption and DDH assumption are providing security DSA with correct chosen security parameters.

2.6 ECDSA

Neal Koblitz [10] and Victor Saul Miller [11] independently proposed an idea Elliptic Curve Cryptography (ECC), in other words cryptosystems based on using elliptic curves. The Elliptic Curve Digital Signature Algorithm (ECDSA) is elliptic curve version of DSA [12]. It was approved in 1999 as ANSI standard, and was accepted in year 2000 as IEEE and NIST standards.

ECDSA Domain Parameters

ECDSA requires that the private/public key pairs used for digital signature generation and verification be generated with respect to a distinct set of domain parameters. These domain parameters may be common to a group of users and may be public. Domain parameters may remain fixed for an extended time. [12] ECDSA parameters are:

- q – order of the elliptic curve field $\mathbb{F}_\mathrm{p}$
- a, b – parameters of elliptic curve equation $y^2 = x^3 + ax + b$

- G – base point generating cyclic subgroup
- n – an order of cyclic subgroup generated by G
- h – a cofactor of cyclic subgroup generated by G

ECDSA Domain Parameters generation

Chosen domain parameters directly affects security of the scheme. Certain types of elliptic curves are rather weak and allow the use of special algorithms for efficiently solving the discrete logarithm problem [13]. In computer systems and cryptographic protocols using ECDSA usually recommended elliptic curve domain parameters are implemented. The most widely used ECDSA Domain Parameters are NIST P-256 (secp256r1), NIST P-384 (secp384r1), NIST P-526 (secp521r1), Curve25519. [14]

ECDSA Key Generation

1. **ECDSA private key** d – is randomly chosen integer from a range $1 < d < (n - 1)$, where n an order of cyclic subgroup generated by G
2. **ECDSA public key** Q – is a point on elliptic curve over $\mathbb{F}_\mathrm{p}$ calculated with formula: $Q = dG$, where G is base point generating cyclic subgroup

ECDSA signature generation (signing)

- Generate (pseudo)randomly integer parameter k in range $1 < d < (n - 1)$. Parameter k must be regenerated for each signature and must be kept in secret.
- Calculate point using scalar multiplication $P = kG$
- Let M be message. Calculate hash value of the message $h=H(M)$
- The signature on a message M is the pair of numbers (r, s) computed according to the formulas:
    - $r = x_p \, mod \, n$, where x_p – x-coordinate of point P.
    - $s = \left(k^{-1}(h + rd)\right) mod \, n$, where k^{-1} is inverse of parameter k modulo n.

If either r, or s is equal to zero, signature was generated incorrectly, and it must be generated again with the new parameter *k*. The pair *(r, s)* is **ECDSA signature** on message *M*.

ECDSA signature verification

- Let *M* be a message, let *(r, s)* be ECDSA signature that needs to be verified. Check if $0 < r < n$ and $0 < s < n$. In case either condition is not satisfied the signature must be rejected.

- Calculate values $u_1 = (s^{-1}h)\ mod\ n$ and $u_2 = (s^{-1}r)\ mod\ n$

- Calculate point using scalar multiplication $P' = u_1G + uQ$

- If $r = x_{P'}\ mod\ n$, then the signature *(r, s)* is verified and the verifier can have enough confidence that the received message was sent by the party holding the ECDSA private key *d* corresponding to public key *Q*.

ECDSA security.

Security of ECDSA relies on the ECDLP problem.

Definition 2.6.1. (Elliptic Curve Discrete Logarithm Problem): Let *E* be elliptic curve over finite field $\mathbb{F}_\mathrm{p}$ with defined operation of algebraic addition, *P* and *Q* are point of *E* and elements of $\mathbb{F}_\mathrm{p}$. The **elliptic curve discrete logarithm problem** is to find such integer *n* that $nP = Q$.

At the moment of conducting this research, there is no non-quantum algorithm capable of computing discrete logarithms in polynomial time.

2.7 Security level

Definition 2.7.1. (Security level): If a symmetric cryptographic system with *n*-bit keys has no general attack faster than exhaustive key search (so called bruteforce), then it is traditionally said to have **security level** *n*. In general, a cryptographic system has security level *n* if a successful general attack of retrieving key can be expected to require effort approximately to operations 2^n. [15]

For cryptographic systems like Advanced Encryption Standard (AES) and Data Encryption Standard (DES) the length of a key equals level of security. [Table 1]

Name	**Key length (bits)**	**Security level**
DES	56	56
AES-128	128	128
AES-192	192	192
AES-256	256	256

Table 1: Security level table for AES and DES [15]

Digital signatures algorithms described in Sections 2.4-2.6 to achieve security level of *n* require larger key length than *n*. The reasoning for that these digital signatures rely on mathematical problems. Even though currently there is no efficient polynomial-time algorithms for breaking factorization problem, DLP and ECDLP, there are algorithms to solve these problems faster than exhaustive key search. [16] [17] [18].

Following table comparing RSA, DSA and ECDSA key sizes required to obtain exact security level [Table 2]. RSA and DSA key sizes are bigger than ECDSA keys with same level of security. ECDSA keys not only uses less memory space, but key and signature generation are faster than RSA and DSA.

Date	Security Strength	Symmetric algorithms	Factoring modulus (RSA)	Discrete Logarithm (DSA) (Key/Group)	Elliptic Curve (ECDSA)
2019-2030	112	AES-128 (3TDEA)	2048	224/2048	224
2019-2030 & beyond	128	AES-128	3072	256/3072	256
2019-2030 & beyond	192	AES-192	7680	384/7680	384
2019-2030 & beyond	256	AES-256	15360	512/15360	512

Table 2: NIST key size recommendations [19]

2.8 Multiparty computation

Multiparty computation (MPC) is a joint computation of some function that is performed by several parties. [20] There are different techniques to perform multiparty computation with **secret sharing** being one of these techniques. Intuitively, it can be seen as dividing input value between the several parties and allowing them to collaborate to obtain the evaluations of some function. These results can be combined to reconstruct the output value (final evaluation) of the function.

Oblivious transfer (OT) is a cryptographic primitive that is widely used in multiparty computation. The simplest version is 1-out-of-2 oblivious transfer that is an interactive protocol between sender and receiver. Sender has two different messages x_1 and x_2 and receiver wants to get one of these messages without revealing to the sender which message exactly they want to get. [21]

Gilboa OT-Multiplication : this function takes two inputs from each party, and returns these inputs to each party as additive secret shares of the product of those two inputs. (G-

$\text{OT}(a, b) = k_1, k_2$, where a and b are inputs from parties, $k_1, k_2 -$ additive secret shares such as $k_1 + k_2$ [22]

2.9 Zero Knowledge Proof

Zero-knowledge proof (ZKP) is an interactive protocol between prover and verifier that enables to prove knowledge of secret data without revealing the data. The *prover* and *verifier* both have common input x, prover has additional private input (witness) w that satisfies the relation $(x, w) \in R$. The example of such relation could be hash function $x = H(w)$, in this case witness is input to the hash function. During the execution of protocol prover attempts to prove the knowledge of witness to the verifier without telling the verifier any additional information about data needed to be proven [23]. ZKP should satisfy the following properties [24]:

- *Completeness*: the proof succeeds, if both parties are honest and run the protocol according to the definition.
- *Soundness*: prover cannot prove something wrong, only with some negligible probability.
- *Zero-knowledge*: verifier learns nothing except that the statement being proven is true or false.

For example, assume having secure public key encryption scheme, where public key is used to encrypt data and secret key is used to decrypt ciphertext. Prover wants to show verifier that they have access to the secret key that corresponds to the public key that verifier has, the protocol works as follows:

- Verifier encrypts some message of their choice with prover's public key and sends the ciphertext to the prover.
- Prover uses their secret key to decrypt ciphertext and sends the result of decryption back to the verifier.
- Verifier compares the message received from the prover and the message that was originally encrypted. Verifier accepts the proof if and only if both messages are equal. [23]

Commitments

The concept that is closely related to ZKP is a **commitment scheme**, interactive protocol between prover and verifier. Commitment enables prover to commit to some value, without immediately revealing that value and without being able to change the value inside the commitment after it was outputted. Later, prover can reveal the value that was inside the commitment and verifier can check if revealed value corresponds to the initial commitment [23]. Two properties that should be satisfied in commitment scheme are:

- **Binding property**: after outputting the commitment, prover should not be able to change the value inside the commitment.
- **Hiding property**: after receiving the commitment from prover, verifier cannot tell what is inside the commitment until prover opens the commitment.

Example of a commitment scheme could be illustrated with hash functions. Prover has some message *m* what they want to commit to, prover hashes the message *H(m)* and sends it to the verifier. Verifier, seeing only hash *H(m)*, cannot tell what message is inside the commitment. Later, prover can open the commitment by sending the message *m* to the verifier. Verifier hashes the message received from the prover and compares that hash with the initial commitment, if these hashes are equal, verifier accepts commitment.

2.10 Paillier Cryptosystem

Definition 2.12.1. (Additive homomorphism): property of a ciphertext when encrypting two messages *a* and *b* and then adding ciphertexts together gives the same result as adding messages together and only then encrypting them, $Enc(a) + Enc(b) = Enc(a+b)$

Definition 2.12.2. (Multiplicative homomorphism): property of a ciphertext when encrypting two messages *a* and *b* and then multiplying ciphertexts together gives the same result as multiplying messages together and only then encrypting them, $Enc(a) * Enc(b) = Enc(a*b)$

Paillier encryption scheme is not as widely used as for example [25] encryption scheme, but it is very useful in some applications. More specifically, Paillier encryption scheme has additive homomorphic property (**Definition 2.12.1)**. Due to the additive homomorphic property, Paillier encryption scheme is used in many multi-party computation applications, including threshold signatures. Paillier encryption scheme uses multiplicative group $Z^*_{N^2}$ and it is defined as follows:

Key Generation:

1. generate two distinct odd prime numbers p and q of equal length
2. compute $N = pq$ and $\phi(\mathrm{N}) = (\mathrm{p} - 1)(\mathrm{q} - 1)$
3. **Paillier public key** is *N*, **Paillier private key** is $(N, \phi(N))$.

Encryption:

1. Plaintext should be $m < N$
2. Choose random $r \leftarrow Z^*_N$, s.t. *gcd(r, N) = 1*
3. Compute ciphertext as $c \coloneqq [(1 + N)^m \cdot r^N \; mod \; N^2]$

Decryption:

1. Ciphertext should be $c < N^2$
2. Compute plaintext as $m \coloneqq \left[\frac{\left[[c^{\phi(N)} mod N^2] - 1\right]}{N} \cdot \phi(N)^{-1} \; mod \; N\right]$

Security

Definition 2.12.3. (Decisional Composite Residuosity Assumption): Let n be a composite number, z be an integer. **Decisional Composite Residuosity Assumption (DCRA)** states that it is hard to find integer y such as $y \equiv z^n \; (mod \; n^2)$ (it is hard to decide whether z is an n-residue modulo n^2. Pallier cryptosystem security relies on DCRA.

Definition 2.12.4. (Subgroup membership problem): Let G be a group, let H be its subgroup. **Subgroup membership problem** (**SMP**) states that it is hard to decide whether a given element $g \in G$ belongs to H. [26]

3 Two-party ECDSA

3.1 Threshold signatures schemes (TSS)

Threshold signature schemes is a special case of MPC. In threshold signature scheme (TSS) several parties share a private key between them. Only when an authorized subset of parties is met, they can produce a digital signature. This topic was initially studied in 1990, but due to the lack of interest, it was abandoned until recent years of cryptocurrency popularization. For scenario *t* parties of *m* needed, TSS is divided in three steps:

- **Key Generation:** there must be a public key generated, which will be used by anyone to verify signatures. Additionally, an individual secret for each party must be created, which will be a secret share, must be created. For generated public key available to all parties and secret shares for each party the following properties must be met:
    - **Privacy**: no secret shares data is leaked between the parties.
    - **Correctness**: the public key should always correspond to generated secret shares.
- **Signature generation (signing):** Using either *t* or more party secret shares, a digital signature is created for some abstract message with the property of privacy ensures that no leakage of secret shares occurred during the computation. The signing guarantees that a quorum *t* parties is needed to sign message.
- **Verification:** to be compatible with non-threshold signatures, everyone with knowledge of the public key should be able to verify and validate the digital threshold signature.

Due to the scope of this research, further I will use term TSS for ***two party scenarios (Two-Party Signature)***. In this scenario two parties (the client and the server) jointly:

- Generate secret shares with corresponding public key

- Produce digital signature. This signature can be produced only by both parties' cooperation.
- Periodically refresh parties' sharing of the secret key.

The ***Two-Party Signature*** generation remains secure if both parties are not compromised between successive refreshes.

3.2 Problem of Threshold ECDSA

The implementation of ECDSA in TSS in a secure manner is challenging. For ECDSA threshold scheme, the goal is to create secret shares for client and server. Recall the formula for ECDSA signature:

$$s = \left(k^{-1}(H(M) + rd)\right) \, mod \, n$$

where *M* is a message, H is a hash function, *d* is the ECDSA private key, *n* is order of cyclic subgroup, k is the instance key, k^{-1} is inverse modulo *n*, *r* is the x-coordinate of the elliptic curve point $R = nG$ (*G* being the generator for the curve).

In ECDSA signing, the equation for computing ECDSA signature includes k^{-1}. Assume we made shares of *k*, instance key, to client (share k_1) and server (share k_2). This brings two scenarios:

1. $k = k_1 + k_2 \, mod \, n$. (Additive sharing) In this case it is believed difficult to compute secret shares k_1', k_2' such that met condition: $k_1' + k_2' \, = k^{-1} \, mod \, n$
2. $k = k_1 * k_2 \, mod \, n$. (Multiplicative sharing) In this case it is possible to compute by each party locally $k_i' = k_i' \, mod \, n$. But it brings up the problem of combing k_1', k_2' to *k* securely over the channel of communication between server and client.

3.3 Related work

3.3.1 Two-Party Generation of DSA Signatures (P. MacKenzie, M. K. Reiter)

This paper was proposed in 2001 by Philip MacKenzie and Michael K. Reiter [27]. Until 2017 it was the most efficient DSA/ECDSA two-party threshold signature scheme. It was based on multiplicative secret shares of ECDSA private key d and instance key k.

Signature generation

Each party (P_1 and P_2) has secret shares d_1, d_2 and k_1, k_2 such as $k = k_1 * k_2$ and $d = d_1 * d_2$. Authors of the scheme proposed using Paillier encryption, due to its additive homomorphic property, to produce d and k securely:

1. Party P_1 computes $c_1 = Enc_{pk_1}(k_1^{-1} * H(m))$, pk_1 – party P_1 public key
2. Party P_1 computes $c_2 = Enc_{pk_1}(k_1^{-1} * d_1 * r)$
3. Party P_1 sends c_1 and c_2 to party P_2
4. Party P_2 computes $c = (k_2^{-1} \otimes c_1) \oplus \left((k_2^{-1} * d_2) \otimes c_2\right) = Enc(k_2^{-1}\left(k_1^{-1} * H(m)\right) + (k_2^{-1} * d_1) * (k_1^{-1} * d_1 * r) = k^{-1} * (H(m) + r * d))$ (encryption of ECDSA signature), where $\otimes$ - scalar multiplication, $\oplus$ - homomorphic addition

Security

However, using Paillier encryption requires ZKP to protect from malicious adversaries. One of the issue is party P_1 Paillier encryptions c_1 and c_2 include k_1^{-1} when party P_2 only knows $R_1 = k_1 G$, but not $R_2 = k_1^{-1} G$. Another issue is party P_1 must prove that the Paillier encryptions c_1 and c_2 exist in the exact range. Such ZKP causes this scheme to work inefficiently: digital signature generation takes several seconds, which is not appropriate for modern communication.

Due to such low-efficient characteristic, Philip MacKenzie and Michael K. Reiter threshold ECDSA scheme, only relevance was to future TSS academic research. It has not been used in any industry protocols, programmes, or cryptocurrencies. These issues are the reason it is excluded from Chapter 5 comparison.

3.3.2 Fast Secure Two Party ECDSA Signing (Yehuda Lindell)

In 2017 Yehuda Lindell proposed a Two-Party ECDSA protocol [28] which had better performance compared to [27], but was partly based on it. The main Lindell's approach was to remove as much ZKP as possible improving signature generation speed with similar security of P. MacKenzie, M. K. Reiter approach.

Key Generation

1. P_1 generates random d_1 and calculates commitment $Q_1 = d_1 G$
2. P_1 sends commitment Q_1 to party P_2 along with ZKP of its discrete log.
3. P_2 generates random d_2 and calculates $Q_2 = d_2 G$
4. P_2 sends commitment Q_2 to party P_1 along with ZKP of its discrete log
5. P_1 decommits, generates a Paillier key pair (pk, sk) and computes $c_{key} = Enc_{pk}(d_1)$.
6. P_1 sends encrypted key c_{key} to party P_2
7. P_1 proves to P_2 that a value encrypted in a given Paillier ciphertext is the discrete log of a given Elliptic curve point ($(c_{key},\ \mathrm{p}k,\ Q_1) \in L_{PDL}$)
8. P_2 verifies the proof. If proof was not verified - P_2 aborts key generation process.
9. Both parties locally compute the output, elliptic curve point $Q = d_1 * Q_1 = d_2 * Q_1 = d_1 * d_2 * G = dG$

Signature generation (signing)

Parties P_1 and P_2 hold multiplicative secret shares such as $d = d_1 * d_2$. Party P_2 holds $c_{key} -$ a Paillier encryption of party's P_1 secret share d_1.

1. P_1 and P_2 locally generate random k_1 and k_2
2. P_1 sends commitment $R_1 = k_1 G$ to party P_2 along with ZKP of its discrete log.

3. P_2 sends commitment $R_2 = k_2 G$ to party P_1 along with ZKP of its discrete log

4. P_2 verifies the proof. If proof was not verified - P_2 aborts key generation process.

5. Both parties locally compute $R = k_1 * k_2 * G$ and $r = r_x \; mod \; n$

6. Party P_2 locally computes following:

   a. $c_1 = Enc_{pk}((k_2^{-1} * H(m) \; mod \; n)$, where $pk -$ party's P_1 Paillier public key

   b. $v = (k_2^{-1} * r * d_2) \; mod \; n$

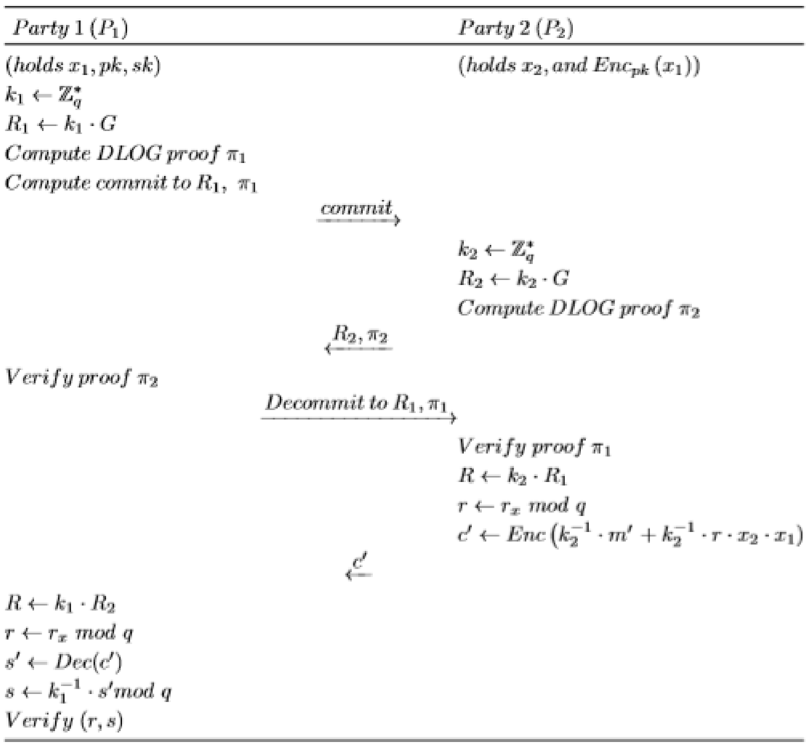
   c. $c_2 = v \otimes c_{key}$

   d. $c' = c_1 \oplus c_2$

7. Party P_2 sends c' to party P_1.

8. Party P_1 decrypts $Dec_{sk}(c') = s'$, $sk -$ party's P_1 Paillier private key.

9. Party P_1 calculates $s = k_1^{-1} * s'$ and outputs ECDSA signature (r, s)



$Party\ 1\ (P_1)$		$Party\ 2\ (P_2)$
$(holds\ x_1, pk, sk)$		$(holds\ x_2, and\ Enc_{pk}(x_1))$
$k_1 \leftarrow \mathbb{Z}_q^*$		
$R_1 \leftarrow k_1 \cdot G$		
$Compute\ DLOG\ proof\ \pi_1$		
$Compute\ commit\ to\ R_1,\ \pi_1$		
	$\xrightarrow{commit}$	
		$k_2 \leftarrow \mathbb{Z}_q^*$
		$R_2 \leftarrow k_2 \cdot G$
		$Compute\ DLOG\ proof\ \pi_2$
	$\xleftarrow{R_2, \pi_2}$	
$Verify\ proof\ \pi_2$		
	$\xrightarrow{Decommit\ to\ R_1, \pi_1}$	
		$Verify\ proof\ \pi_1$
		$R \leftarrow k_2 \cdot R_1$
		$r \leftarrow r_x\ mod\ q$
		$c' \leftarrow Enc\left(k_2^{-1} \cdot m' + k_2^{-1} \cdot r \cdot x_2 \cdot x_1\right)$
	$\xleftarrow{c'}$	
$R \leftarrow k_1 \cdot R_2$		
$r \leftarrow r_x\ mod\ q$		
$s' \leftarrow Dec(c')$		
$s \leftarrow k_1^{-1} \cdot s' mod\ q$		
$Verify\ (r, s)$		

Figure 11 : Fast Secure Two Party ECDSA Signing [28]

Security

The main difference between Lindell's scheme and [27]: security of the protocol relies that party P_2 and only party P_2 obtain during key generation $c_{key} = Enc_{pk}(d_1)$: ECDSA private key secret share of party P_1 encrypted with P_1 Paillier public key. That brings another issue – using ZKP prove to party P_2 that c_{key} contains the discrete log of a given Elliptic curve point *G*. If such proof is not performed, information may be revealed about P_2 secret share to a malicious P. That is the reason Lindell proposed new highly efficient zero-knowledge proof for the language LPDL in this paper [28].

In case of malicious P_1: party P_1 only participates in the generation of ECDSA parameter *R*. If we assume ZKP used is efficient (need to break ECDLP), corrupted party cannot do anything malicious and at the same time no additional zero-knowledge proofs are needed.

In case of malicious P_2: the main risk is party P_2 can send invalid (corrupted) value c' to party P_1. If it happens, then P_1 gets the ciphertext c', decrypts it's Paillier private key and multiplies it with k_1^{-1}. Produced value is ECDSA signature and P_1 can verify whether the obtained signature is correct. Thus, no additional ZKP is required.

Lindell's proposed scheme could be used for client-server scenario; however it is patented [29] which limits its usage in open-source projects.

3.3.3 Two-Party ECDSA from Hash Proof Systems and Efficient Instantiations

In 2019 Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker proposed their scheme for Two-Party ECDSA [30]. Their scheme is built on Lindell's paper [28]. The authors were trying to fix following shortcomings in Lindell's scheme:

1. Lindell used artificial and interactive assumption on Paillier scheme proving security of protocol under malicious party P_2. In the other terms, uses guessing aborting a protocol under malicious P_2

2. Sending encrypted secret share d_1 with Paillier requires additional highly cost range checks: d_1 must be both in message space of Paillier cryptosystem and less than *n* (an order of cyclic subgroup generated by point *G).*

Main goals of authors' proposed scheme were:

- Secure scheme not relying on artificial and interactive assumptions.
- Tight security proof of the scheme
- No range proofs for message space

To achieve these goals authors decided to use variation of Paillier cryptosystem proposed in [31] also known as Hash Proof System (HPS). This system relies on Subgroup Membership Problem (SMP, Definition 2.12.4.). Using this cryptosystem does not compromise the system the way Paillier does in Lindell's approach [28].

Key generation.

1. P_1 and P_2 randomly generate d_1 and d_2 accordingly.
2. $P_1$and Bob computes $Q_1 = d_1G$ and $Q_2 = d_2G$ accordingly.
3. P_1sends commitment Q_1 to party P_2 along with ZKP of its discrete log.
4. P_2 sends commitment Q_2 to party P_1along with ZKP of its discrete log.
5. If none of the previous commitments were sound – protocol aborts.
6. $P_1$and P_2 compute ECDSA public key $Q = d_1 * Q_2 = d_2 * Q_1$
7. P_1 generates HPS key pair (sk, pk) and send $Enc_{pk}(x_1)$ to P_1

Signing

1. P_1 and P_2 randomly generate k_1 and k accordingly.
2. $P_1$and Bob computes $R_1 = k_1G$ and $R_2 = k_2G$ accordingly.
3. P_1sends commitment R_1 to party P_2 along with ZKP of its discrete log.

4. P_2 sends commitment R_2 to party P_1 along with ZKP of its discrete log

5. If none of the previous commitments were sound – protocol aborts.

6. $P_1$and P_2 compute ECDSA public key $R = k_1 * R_2 = k_2 * R_1$

7. Party P_2 homomorphically calculate $Enc_{pk}(s') = Enc_{pk}\left(k_2^{-1}(H(m) + r_x x_1)\right)$

8. Party P_1 decrypts $Dec_{sk}\left(Enc_{pk}(s')\right) = s'$ and calculates signature $s = k_1^{-1} s' mod\, n.$

9. Party P_1 verifies signature output. *(r,s)*

4 Algorithm Specification (Secure Two-party Threshold ECDSA from ECDSA Assumptions)

This paper was proposed by Jack Doerner and Yashvanth Kondi and Eysa Lee and abhi shelat in 2018 [1]. Authors criticize Lindell’s scheme dependence upon the Paillier encryption, since security of it relies upon DCRA, which is alien to ECDSA.,The Paillier cryptosystem contains computationally expensive operations compared to Elliptic Curve operations. They address this issue by developing the first Two-Party ECDSA scheme that is based solely upon Elliptic Curves and the assumptions that the ECDSA signature scheme itself already makes.

Once again recall the formula for ECDSA signature:

$$s = \left(k^{-1}(H(M) + rd)\right) mod\, n$$

where *M* is a message, *H* is a hash function, *d* is the ECDSA private key, *n* is order of cyclic subgroup, *k* is the instance key, k^{-1} is inverse modulo *n, r* is the x-coordinate of the elliptic curve point $R = nG$ (*G* being the generator for the curve). *Q=dG* is the ECDSA public key.

Main difference of this scheme from the previous – authors decided to use Gilboa OT multiplication [22] to compute secret shares.

4.1.1 Key generation

Let party P_1 be called Alice, let party P_2 be called Bob.

10. Alice and Bob randomly generate d_A and d_B accordingly.
11. Alice and Bob computes $Q_A = d_A G$ and $Q_B = d_B G$ accordingly.
12. Alice sends commitment Q_A to Bob along with ZKP of its discrete log.
13. Bob sends commitment Q_B to Bob along with ZKP of its discrete log.
14. If none of the previous commitments were sound – protocol aborts.
15. Alice and Bob compute $Q = d_A * Q_B = d_B * Q_A$

4.1.2 Signature generation

Signing can be split in 4 logical (not proceeding) steps:

- **Multiplication:** Alice and Bob using Gilboa OT-based multiplication protocol [32] calculate their multiplicative shares in additive shares.
- **Instance Key Exchange:** Alice and Bob calculate point *R* for ECDSA signature
- **Consistency check:** Alice and Bob verify that the first Gilboa multiplication uses inputs consistent with the value *R* from Instance Key Exchange step. In protocol it is concatenating a random padding φ to Alice's input, and then combining the padding with the Gilboa multiplication output and the known value *R* in such a way that Bob can retrieve the padding only if he is not malicious adversary. Second check is need for multiplications being consistent with each other and with the generated public key Q.
- **Signature and Verification:** The parties reconstruct the signature, which is given to Bob. If the signature is verified with ECDSA public key, then Bob outputs it.

Two-party signing algorithm is:

1. Bob randomly generates his secret share k_B of instance key k and computes point $D_B = k_B * G$. After that Bob sends D_B to Alice.

2. Alice randomly generates instance key seed k'_A and calculates her secret share $k_A = H(R') + k'_A$, where $R' = k'_A * D_B$. After that Alice computes point $R = k_A * D_B$

3. Alice generates random padding φ (1< φ <n) and calculates with Bob using Gilboa OT-Multiplication (F_{Mul}) two additive shares $F_{Mul}(\varphi + \mathrm{k}_A^{-1}, k_B^{-1}) = \varphi * \mathrm{k}_B^{-1} + \mathrm{k}_A^{-1} * k_B^{-1} = t_A^1 + t_B^1$ receiving their padded joint inverse instance key.

4. Alice and Bob calculates using Gilboa OT-Multiplication (F_{Mul}) two additive shares $F_{Mul}(d_A * \mathrm{k}_A^{-1},\ d_B * k_B^{-1}) = \varphi * \mathrm{k}_B^{-1} + \mathrm{k}_A^{-1} * k_B^{-1} = t_A^2 + t_B^2$ receiving their joint private key over their joint instance key.

5. Alice send R' to Bob

6. Bob calculates $R = H(R') * D_B + R'$. Now both Alice and Bob share point R with coordinates (r_x, r_y).

7. Alice using ZKP proves discrete log of R and k_A. Bob receives a bit indicating whether the proof was sound. If it was not, he aborts

8. Alice and Bob compute $m' = H(m)$

9. Alice computes check value using padding from step 3: $\Gamma^1 = G + \varphi * \mathrm{k_A} * G - t_A^1 * R$

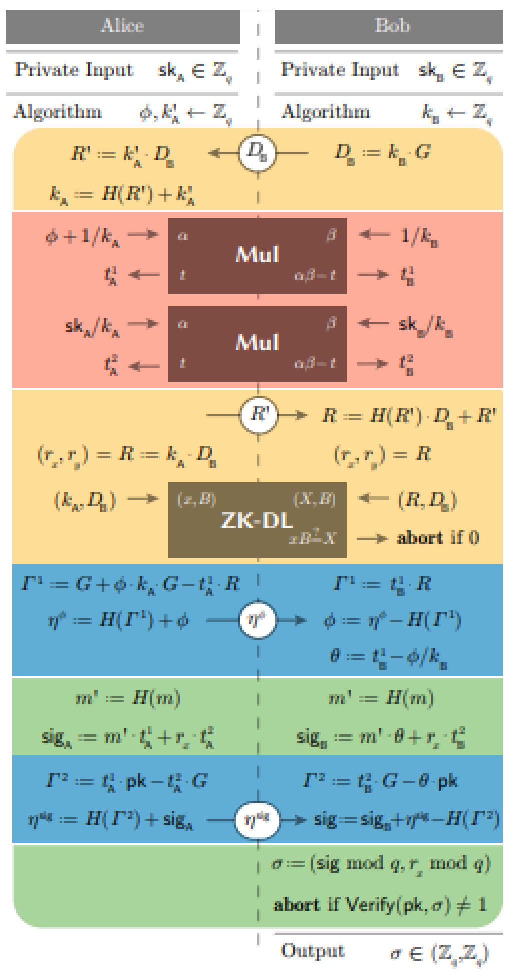
10. Alice sends calculated encryption $\eta^{\varphi} = H(\Gamma^1 + \varphi) +$ to Bob

11. Alice computes her signature share $s_A = (m' * t_A^1) + (r_x * t_A^2)$ and the second check value $\Gamma^2 = (t_A^1 * Q) - (t_A^2 * G)$. Alice encrypts s_A with Γ^2 and then sends the encrypted value to Bob: $\eta^{\mathrm{s}} = H(\Gamma^2) + s_A$

12. Bob computes check values Γ^1 and Γ^2 and reconstructs the signature:

- $\Gamma^1 = t^1 * R$
- $\varphi = \eta^{\varphi} - H(\Gamma^1)$
- $\theta = t_B^1 - \varphi * k_B^{-1}$
- $s_B = (m' * \theta) + (r_x * t_B^2)$
- $\Gamma^2 = (t_B^2 * G) - (\theta * Q)$
- $s = s_B + \eta^s - H(\Gamma^2)$

13. Bob verifies value (s, r_x) with Q if it is valid ECDSA signature on message m. If the verification fails, Bob aborts protocol. If verification succeeds, he outputs $(s, r_x)=\sigma$



Alice
Bob
Private Input $\mathsf{sk}_A \in \mathbb{Z}_q$
Private Input $\mathsf{sk}_B \in \mathbb{Z}_q$
Algorithm $\phi, k'_A \leftarrow \mathbb{Z}_q$
Algorithm $k_B \leftarrow \mathbb{Z}_q$
$R' := k'_A \cdot D_B$
D_B
$D_B := k_B \cdot G$
$k_A := H(R') + k'_A$
$\phi + 1/k_A$
$1/k_B$
Mul
α
β
t
$\alpha\beta - t$
t_A^1
t_B^1
sk_A/k_A
sk_B/k_B
Mul
t_A^2
t_B^2
R'
$R := H(R') \cdot D_B + R'$
$(r_x, r_y) = R := k_A \cdot D_B$
$(r_x, r_y) = R$
(k_A, D_B)
(x, B)
(X, B)
ZK-DL
$xB \stackrel{?}{=} X$
(R, D_B)
abort if 0
$\Gamma^1 := G + \phi \cdot k_A \cdot G - t_A^1 \cdot R$
$\Gamma^1 := t_B^1 \cdot R$
$\eta^\phi := H(\Gamma^1) + \phi$
η^ϕ
$\phi := \eta^\phi - H(\Gamma^1)$
$\theta := t_B^1 - \phi / k_B$
$m' := H(m)$
$m' := H(m)$
$\mathsf{sig}_A := m' \cdot t_A^1 + r_x \cdot t_A^2$
$\mathsf{sig}_B := m' \cdot \theta + r_x \cdot t_B^2$
$\Gamma^2 := t_A^1 \cdot \mathsf{pk} - t_A^2 \cdot G$
$\Gamma^2 := t_B^2 \cdot G - \theta \cdot \mathsf{pk}$
$\eta^{\mathsf{sig}} := H(\Gamma^2) + \mathsf{sig}_A$
η^{sig}
$\mathsf{sig} := \mathsf{sig}_B + \eta^{\mathsf{sig}} - H(\Gamma^2)$
$\sigma := (\mathsf{sig} \bmod q, r_x \bmod q)$
abort if $\mathsf{Verify}(\mathsf{pk}, \sigma) \neq 1$
Output $\sigma \in (\mathbb{Z}_q, \mathbb{Z}_q)$

Figure 12: Illustrated Two-party Signing Scheme. Operations are color-coded according to the logical component with which they are associated Gilboa Multiplication , Instance Key Exchange Consistency Check , and Verification/Signing. [1]

4.1.3 Security

In the papers author proves this scheme is secure under CDH (**Definition 2.5.2.)** and the assumption that the resulting signature is itself secure. These two assumptions do not require no additional assumptions relative to ECDSA itself.

5 Implementation

This chapter introduces the details of this research implementation as well as the benchmark results of the proposed implementation.

5.1 Proposed implementation

5.1.1 Programming language and libraries

The author created proof-of-concept implementation of 2-of-2 protocol proposed in “Secure Two-party Threshold ECDSA from ECDSA Assumptions” [1] in Java programming language, version 1.8. [33]. The code to implementation can be accesses using this link (https://github.com/Animehater/thesis). To implement client-communication author used standard libraries of Java Core and Java EE. For implementing cryptographic and mathematical operations used in protocol author used:

1. **Java Cryptography Extension (JCE) and Java Cryptography Architecture (JCA):** standard Java API is created to provide an abstraction layer for application developers. Their architecture is called *provider-based architecture*. JCE and JCA provide a set of classes and interfaces that a program/application developer writes to, together with factories that allows the creation of the objects that conform to the interfaces and classes. The collections of classes that provide these implementation objects are called *cryptographic providers.* [34] In implementation JCE and JCA are only used as level of abstraction.

2. **BouncyCastle**: cryptographic library developed by a group of volunteers and cryptology enthusiasts. Includes a lightweight cryptographic API, a provider for the JCE and JCA, a library for reading and writing encoded ASN.1 objects and

different generators/processors for X.509 certificates, S/MIME and CMS, OpenPGP, etc. [35].

In implementation author used elliptic curve primitives. finite fields, ECDSA key pairs, secure random generation, hash functions.

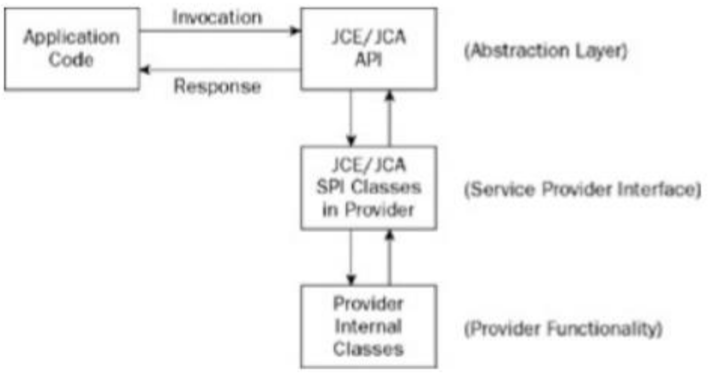


Figure 13: JCE and JCA communication with provider [34]

Self-implemented cryptographic primitives: unfortunately, BouncyCastle does not support primitives like Paillier encryption, Oblivious Transfer, ZKP, Gilboa OT-Multiplication, hash functions that can take elliptic curve point's coordinates as input. Eventually, all important primitives were implemented during this research [36]

5.1.2 Technical Environment

For conducting research, the author used following laptops:

- Lenovo Yoga 300 with 2 core Intel(R) Celeron(R) CPU N3060 @ 1.60GHz CPU as a client (Ubuntu 18.04).
- Lenovo X1 Carbon with 2 core Intel Core i5-3337 CPU N3060 @ 1.80GHz CPU as a server (Windows 10).

All communication was conducted in LAN using WiFi Pineapple with following characteristics:

• CPU: 533 MHz MIPS 74K Atheros AR9344 SoC

• Memory: 64 MB DDR2 RAM

• Disk: 2 GB NAND Flash

• Wireless: Atheros AR9344 + Atheros AR9580, both IEEE 802.11 a/b/g/n with quad integrated skybridge amplifiers and included 5 dBi antenna for a high 29 dBm gain EIRP

• Ports: (4) SMA Antenna, RJ45 Fast Ethernet, Ethernet over USB, Serial over USB, USB 2.0 Host, 12V/2A DC Power

5.1.3 Implementation protocol parameters

As ECDSA parameters for implementation testing author chose - NIST P-256 (secp256r1), NIST P-384 (secp384r1), NIST P-526 (secp521r1) curves. Alice (first party) – was chosen as client, Bob (second party) – was chosen as server. Such decision is motivated that in [1] scheme, Bob outputs signature or aborts protocol (4.1.2 Step 13). Even though both client and server can be compromised – author made assumption that server has less chances to be compromised giving him ability to abort protocol more often than client.

For each curve, the author ran 2000 execution of signing generation and 250 executions key generation.

5.2 Northeastern University implementation

Authors of [1] created a proof-of-concept implementation of their 2-of-2 ECDSA signing protocol in the Rust language. Implementation was benchmarked by Amazon instances from Amazon’s Virginia datacentre, both running Ubuntu 16.04 with Linux kernel 4.4.0. The code was compiled in Rust 1.27. The bandwidth between their instances was measured to be 5GBits/Second, and the round-trip latency to be 0.2 ms. They used P-256 curve standardised by NIST [12].

Since their implementation is proof-of-concept of protocol, this Rust implementation does not support P-384 and P-521 curves and does not compile on Windows or Mac without additional software installations. [37]

5.3 Key Generation

Due to the fact [37] does not support P-384 and P-521, there is no data on result for these curves from Northeastern paper [1]. Lindell [28] and Castagnos [30] papers provided data for all three standard curves.

Curve / Mean time	Lindell [28]	Castagnos [30]	Northeastern [1]	This paper
P-256	2456 ms	9 350 ms	43.41 ms	50 ms
P-384	2440 ms	35 491 ms	-	713 ms
P-521	3535 ms	103 095 ms	-	1322 ms

Table 3:Running times for key generation

Due to the fact [37] does not support P-384 and P-521, there is no data on result for these curves from Northeastern paper [1]. Lindell [28] and Castagnos [30] papers provided data for all three standard curves.

Curve / Mean Time	Lindell [28]	Castagnos [30]	Northeastern [1]	This paper [36]
P-256	36.8 ms	170 ms	3.77 ms	8.32 ms
P-384	47.11 ms	649 ms	-	26.21 ms
P-521	78.19 ms	1888 ms	-	68.91 ms

Table 4:Running times for signature generation

5.4 Bandwidth

All three papers [1] [28] [30] provided data for signing communication costs.

Table 5: Signing communication costs

Curve / Mean Time	Lindell [28]	Castagnos [30]	Northeastern [1]	This paper [36]
P-256	769 B	178 KiB	169.8 KiB	172.3 KiB
P-384	897 B	35 491ms	350.7 KiB	373.5 KiB
P-521	1043 B	103 095ms	615.3 KiB	665.3 KiB

6 Conclusion and Future work

In this work, the author implemented Secure Two-party Threshold ECDSA from ECDSA Assumptions scheme. The implementation of the code demonstrated support for, not only, P-256 curve, but additionally P-384 and P-521. Support of P-384 and P-521 gives higher level of security to author's implementation. Moreover, my implementation was tested on both Ubuntu/Windows, Northeastern university implementation [37] only managed to work on Ubuntu 16.04 operation system.

Compared to results of [28] and [30] the author's implementation showed best performance in signature generation and key generation, which makes [1] perfect for client-server scenario. However, author's implementation requires high network bandwidth, making it impossible to run in low bandwidth scenarios.

Additionally, during the research the author implemented various cryptographic primitives which can be transformed in full working API/Library for a further cryptographic research.

Moreover, it is first Java implementation of Secure Two-party Threshold ECDSA from ECDSA Assumptions. [36]

For future research author am planning:

- Analyze TSS ECDSA signatures for Java Card
- Make comparison of Multiparty ECDSA schemes
- Improve bandwidth latency in scheme [1]

References

[1] J. Doerner, Y. Kondi, E. Lee and A. Shelat, “Secure Two-party Threshold ECDSA from ECDSA Assumptions,” in *IEEE Symposium on Security and Privacy (SP),* San Francisco, CA, 2018.

[2] W. Diffie and M. Hellman, “New directions in cryptography,” *IEEE Transactions on Information Theory,* Vols. IT-22, no. 6, 1976.

[3] “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,* p. 73–114, 28.8.2014.

[4] B. Schneier, “A Primer on Authentication and Digital Signatures,” *Computer Security Journal,* vol. 10, no. 2, pp. 38-40, 1994.

[5] S. A. a. A. L. Rivest R. L., “A method for obtaining digital signatures and public-key cryptosystems.,” *Commun. ACM,* vol. 21, no. 2, pp. 120-126, Feb. 1978.

[6] P. W. Shor, “Algorithms for quantum computation: discrete logarithms and factoring,” in *Proceedings 35th Annual Symposium on Foundations of Computer Science,* Santa Fe, NM, USA, 1994.

[7] F. B. a. P. G. a. A. G. a. N. H. a. E. T. a. P. Zimmermann, “Comparing the difficulty of factorization and discrete logarithm: a 240-digit experiment,” *Cryptology ePrint Archive, Report 2020/697,* 2020.

[8] F. P. 186-1, “Digital Signature Standart,” U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology, 15 December 1998.

[9] T. ElGamal, “A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms,” *IEEE TRANSACTIONS ON INFORMATION THEORY,* Vols. IT-31, no. 4, July 1985.

[10] N. Koblitz, “Elliptic curve cryptosystems,” *Mathematics of Computation,* vol. number 48, pp. 203-209, 1987.

[11] V. S. Miller, “Use of elliptic curves in cryptography,” *Advances in Cryptology-Proceedings of CRYPTO’85,* pp. 417-426, 1986.

[12] F. I. P. STANDARDS, “FIPS PUB 186-4,” Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD , July 2013.

[13] T. I. a. F. M. a. J. Chao, “Classification of Elliptic/hyperelliptic Curves with Weak Coverings against the GHS attack under an Isogeny Condition,” Cryptology ePrint Archive, Report 2013/487, 2013.

[14] C. Research, SEC 2: Recommended Elliptic Curve Domain Parameters, Certicom Corp, 2000.

[15] A. K. Lenstra, Key Lengths Contribution to The Handbook of Information Security, 2010.

[16] T. Kleinjung, “On polynomial selection for the general number field sieve,” *Mathematics of Computation.,* vol. 75(256), p. 2037–2047, October 2006.

[17] O. T. V. S. A. Menezes A.J, “Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field,” *IEEE Transactions on Information Theory,* vol. V39, pp. 1639-1646, 1993.

[18] D. Gordon, “Discrete logarithms in GF(p) using the number field sieve,” *SIAM Journal on Discrete,* vol. 6, pp. 124-138, 1993.

[19] NIST, “Recommendation for Key Management,” *Special Publication 800-57,* vol. Part 1, no. Rev. 5, 05/2020.

[20] Y. Lindell, “Secure Multiparty Computation (MPC),” *Cryptology ePrint Archive, Report* , vol. 2020/300, 2020.

[21] T. C. a. C. Orlandi, “The Simplest Protocol for Oblivious Transfer,” *Cryptology ePrint Archive,* vol. Report 2015/267, 2015.

[22] N. Gilboa, “Two Party RSA Key Generation,” CRYPTO, 1999.

[23] I. D. a. J. B. Nielsen, “Commitment Schemes and Zero-Knowledge,” *Aarhus University, BRICS,* 2011.

[24] D. Unruh, “Cryptology I (Short notes),” spring 2020. [Online]. Available: http://kodu.ut.ee/~unruh/courses/crypto1/2020/notes.pdf.

[25] P. Paillier, “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes,” *EUROCRYPT. Springer,* p. pp. 223–238.

[26] T. S. Akihiro Yamamura, “SUBGROUP MEMBERSHIP PROBLEM AND ITS APPLICATIONS TO INFORMATION SECURITY,” *Scientiae Mathematicae Japonicae Online,* vol. 7, pp. 23-29, 2002.

[27] P. M. a. M. K. Reiter, “Two-Party Generation of DSA Signatures,” *International Journal of Information Security,* Vols. 2(3-4), p. 218–239, 2004.

[28] Y. Lindell, “Fast Secure Two-Party ECDSA Signing,” *Cryptology ePrint Archive, Report 2017/552,* 2017.

[29] Y. Lindell, “Digital signing by utilizing multiple distinct signing keys, distributed between two parties”. US Patent US20180359097A1, 13 Dec 2018.

[30] D. C. F. L. F. S. I. T. Guilhem Castagnos, “Two-Party ECDSA from Hash Proof Systems and Efficient Instantiations,” Cryptology ePrint Archive, Report 2019/503, 2019.

[31] V. S. Ronald Cramer, “Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption,” Cryptology ePrint Archive, Report 2001/085, 2001.

[32] R. I. a. M. Naor, “Efficient cryptographic schemes provably as secure as subset sum,” *J. Cryptol,* vol. 9, no. 4, Sep 1996.

[33] O. Coorparation, “Java™ Platform Standard Ed. 8,” [Online]. Available: https://docs.oracle.com/javase/8/docs/api/.

[34] D. Hook, Beginning Cryptography with Java, Wrox Press, 2005.

[35] V. S. Á. C. G. J. H. E. L. Gayoso Martínez, “Elliptic curve cryptography: Java implementation issues,” 2005.

[36] N. Snetkov, “Master Thesis Two Party ECDSA,” 2020. [Online]. Available: https://github.com/Animehater/thesis.

[37] J. Doerner, “Northeastern University Crypto,” [Online]. Available: https://gitlab.com/neucrypt/curves/-/blob/461899226de2812c859058014949d8e74e64168b/src/secp256k1.rs.