TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Risto Alas 990746 IAPM

Using permissioned blockchains for security risk mitigation: an analysis framework and case studies

Master's Thesis

Supervisor Ahto Buldas
PhD
Co-supervisor Jaan Ginter
PhD

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Risto Alas 990746 IAPM

Loaliste plokiahelate kasutamine turvariskide vähendamiseks: analüüsiraamistik ning juhtumiuuringud

Magistritöö

Juhendaja Ahto Buldas
PhD
Kaasjuhendaja Jaan Ginter
PhD

Tallinn 2021

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: Risto Alas .....
(signature)

Date: January 11, 2021

Abstract

Blockchain and distributed ledger technology (DLT) have become increasingly popular due to their ability to perform the so-called "trustless" execution: they allow running software without trusting any single party for the correctness of execution. Instead, the correctness is established via mutual consensus protocols and many-sided client verification. In some cases, blockchains can have their own significant inefficiencies as well, depending on the use-case at hand. This brings up two questions: first, when can we trust a single party after all, and second, when we do need more parties, does blockchain technology have alternatives? To study the above questions we developed a model for assessing human behavior, based on consultations with a criminology expert. It estimates the probability of dishonest acts depending on the number of people involved, their likelihood of getting caught, how well they know each other, the attractiveness of the prize and several other factors. The resulting model is general enough to apply for many different situations outside our original problem space.

We then use the model to extract insights about the differences between blockchain technology and centralised multi-party security ceremonies. Based on the model, we suggest considering security ceremonies with remote participation as potential centralised alternatives for some permissioned blockchain deployments; we also discuss potential downsides and suggest areas for further study.

The thesis is in English language and contains 84 pages of text, 5 chapters, 1 figure, 8 tables.

Annotatsioon

Plokiahel (inglise keeles *blockchain*) ning üldisemalt hajusraamatu tehnoloogiad (*distributed ledger technology* ehk *DLT*) on muutunud viimastel aastatel järjest populaarsemaks. Nende peamine eriomadus on võimekus luua süsteeme mille toimimist ei ole vaja usaldada ühe konkreetse osapoole kätte, olgu siis põhjuseks kas usalduse puudumine ühe süsteemioperaatori vastu või arvamus, et osaopoolel puuduvad vahendid teenuse kvaliteetseks tööshoidmiseks. Selliste hajussüsteemide korral saavutatakse suurem usaldus tänu mitut osapoolt hõlmavatele konsesusprotokollidele ning kliendipoolsele verifitseerimisele. Kuna aga taolised mitme aktiivse osapoolega süsteemid võivad mõnedel juhtudel olla ka märgatavalt ebaefektiivsemad (sõltuvalt valitud kasutusjuhust), soovime nende uurimiseks esitada järgmised kaks küsimust: esiteks, millistel juhtudel saaksime hakkama ka ilma plokiahelata, ja teiseks, kas ülejäänud juhtudel eksisteerib plokiahelale ka alternatiive?

Konsulteerides krmininoloogia eksperdiga töötasime välja antud küsimuste uurimiseks üldisema mudeli mis püüab ette ennustada inimeste ebaausa käitumise tõenäosust erinevates olukordades, võttes avesse inimeste arvu otsuse tegemise juures, vahelejäämise tõenäosust ebaausa käitumise korral, aga ka seda kui hästi inimesed üksteist tunnevad ning ebaausa ründega saavutatava võidu suurust ja mitmeid muid faktoreid. Katsetasime mudelit paari kasutusjuhu peal ning mudel töötas subjektiivse hinnangu järgi ootuspäraselt. Samuti on mudel piisavalt üldine, et seda saaks kasutada ka paljudes muudes olukordades.

Uurime potentsiaalseid alternatiive loalistele (*permissioned*) hajusraamatutele tsenstraalsete turvatseremooniate näol, kus erinevalt plokiahela tehnoloogiast ei pea erinevad arvutid üle kontrollima üksteise arvutustulemusi, vaid usaldus tulemuste vastu saavutatakse protseduuriliselt: kõik turvalisuse mõttes olulised operatsioonid tehakse mitme inimese juuresolekul ning ühelgi inimesel pole üksinda juurdepääsu arvutitele, ei füüsiliselt ega üle võrguühenduse. Kasutasime loodud mudelit võrdlemaks plokiahela turvalisust antud tseremooniatega. Mudeli järgi peaks kaaluma plokiahelaga päris sarnaste turvaomadusete saamkseks vähemalt mõnede tseremoonia osaliste poolt virtuaalset osalemist, ilma teiste tseremoonia osalistega suhtlemata. Viimane aitab tagada nii osapoolte anonüümsust, raskendades võimallike vägivaldseid rünnakuid nende vastu, kui ka vähendab samal ajal tõenäosust, et osapooled saavad üksteisega tuttavamaks ning setõttu julgevad üksteisele

teha ka suurema tõenäosusega ettepanekut kuritegelikuks käitumiseks.

Sellisel tsentraalsel tseremoonial on ka fundamentaalsemaid puuduseid – näiteks kui puudub usaldus ühegi konkreetse arvutiriistvara tootja vastu siis võib plokiahelates kasutatav konsensus erinevate arvutite vahel muuta süsteemi töökindlamaks ja usaldusväärsemaks. Töö mainib põgusalt ka võimalust kasutada hübriidlahendusi mis kasutavad korraga nii ülalmainitud tseremooniaid kui ka plokiahelate poolt populariseeritud konsensusprotkolle. Antud lahendus võib olla atraktiivne näiteks juhul kui ühtegi konkreetset arvutit ei usaldata, aga samas ei saa ka majanduslikel või efektiivsuse põhjustel kasutada rohkem kui mõnda üksikut arvutit kuid lisaks ei soovita ka usaldada iga masinat haldama vaid ühte konkreetset operatorit.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 84 leheküljel, 5 peatükki, 1 joonist, 8 tabelit.

List of Abbreviations and Terms

0-day vulnerability	A vulnerability that is in software or hardware but is not known to the parties interested in mitigating it (including vendors)
BFT	Byzantime fault-tolerance
Byzantime fault-tolerance	A characteristic of consensus protocols that guarantees resilience even when some number of participants fail in arbitrary ways, including lying on purpose.
Blockchain	A decentralised architecture for achieving integrity and availability of computation and state (memory) for a service. In general, it is implemented using byzantine fault-tolerant (BFT) consensus coupled with state storage and shared validation rules.
Centralised architecture	An architecture where the security properties under study *can* be broken at execution time by compromising a single secretly chosen computer. This is the traditional way of building services; most systems today work this way.
CPU	Central Processing Unit

Decentralised architecture	An architecture where the security properties under study *cannot* be broken at execution time by arbitrarily compromising one (ideally more) secretly chosen computer(s). For integrity and availability, this is often achieved with Byzantine fault-tolerant (BFT) consensus. For confidentiality purposes, tools like secure multi-party computation (MPC) can be used. Such computers are typically owned and operated by different entities, resulting in the additional resilience property that no single operating entity can compromise the security properties of the service on their own.
Distributed ledger tech.	For the purposes of current work, synonymous with blockchain. Otherwise, the term is often used to refer to blockchain-like technologies that do not necessarily use a chain of blocks for their main data model.
DLT	Distributed ledger technology
Multi-party computation	A decentralised architecture for the purposes of achieving privacy of the data being computed on.
MPC	Multi-party computation
Permissioned blockchain	A blockchain where participants are permissioned and vetted using their identities.
Permissionless blockchain	A blockchain where participants are not permissioned and vetted using their identities and often are allowed to be anonymous to everybody but themselves. In practice, major public blockchains like Bitcoin and Ethereum often aim for the additional property of censorship resistance.
Public blockchain	A blockchain whose contents are public for everyone. Sometimes used as a synonym for permissionless blockchains.
Security ceremony	Expansion of the term "protocol" to include the actions of human actors, often used for the purposes of thinking about security.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 The Problem

Cryptocurrencies like Bitcoin, Ether and countless others have become increasingly popular in the recent years [1]. Likewise, the key underlying technology of cryptocurrencies, the distributed ledger technology (DLT) or simply *blockchain* technology, is also seeing wider use outside the strictly cryptocurrency space, now encompassing a wide array of applications, including health and energy sector applications, cryptocurrency exchanges, smart contracts, games and many others that are too numerous to mention here [2]. Additionally, the financial impact of this technology has received considerable praise; the global research and advisory firm Gartner has predicted that "the business value generated by blockchain will grow rapidly, reaching $176 billion by 2025 and $3.1 trillion by 2030" [3].

Although there are no universally accepted definitions for the technology, one of the most concise definitions is provided by Wikipedia [4]:

> A distributed ledger (also called a shared ledger or distributed ledger technology or DLT) is a consensus of replicated, shared, and synchronized digital data geographically spread across multiple sites, countries, or institutions.[5] Unlike with a distributed database, there is no central administrator.[6]

In particular, it is the lack of a central administrator that is an important characteristic of DLT. It allows the building of services where no single party can exert control over the system, neither for profit nor malice. For more information on blockchains in general and how they differ from traditional database systems, refer to section 2.

Typically, the blockchain protocol will not allow any single entity to tamper with data, software or calculations without the approval of other parties. Thus, even when any (sufficiently small) subset of participants are completely compromised by attackers, the service continues to function normally: data remains tamper-proof, the system outputs correct results to client requests and remains available to its users. This is achieved by the fact that many parties would be running redundant copies of the system, always

synchronising in real time with each other over a consensus protocol, essentially taking frequent majority votes in real-time over the results of any computations in the service. The current work assumes this consensus protocol to be tolerant of arbitrary faults by its members, including lying on purpose – a Byzantine fault tolerant (BFT) consensus for blockchains [7].

This lack of a single central administrator is also the reason why blockchains are praised for their potential for reducing the number of trusted third parties in society, as Wüst and Gervais write in [8]:

> Blockchain is being praised as a technological innovation which allows to revolutionize how society trades and interacts. This reputation is in particular attributable to its properties of allowing mutually mistrusting entities to exchange financial value and interact without relying on a trusted third party. A blockchain moreover provides an integrity protected data storage and allows to provide process transparency.

Thus far we have mentioned the security properties of integrity and availability and how they can be enhanced using DLT. Although out of scope for the current work, using additional machinery, such *decentralised* systems can also give *privacy* guarantees of similar nature, resulting in systems where no single party can leak secret data from the system, albeit sometimes at a noticeable cost in performance [9], perhaps due to techniques like secure multi-party computation (MPC) or zero-knowledge proofs. Therefore, this overall space of decentralised technologies could potentially improve all aspects of security – confidentiality, integrity and availability – by means of reducing reliance on single parties.

Yet such decentralisation also comes with costs. The most obvious and perhaps the most fundamental cost is that of re-verification: multiple computers will have to verify the same computations again and again, as the participants cannot simply trust the computations performed by others. This results in excess computing power and electricity use; depending on a use-case this may or may not be a significant problem. Additionally, there is also the performance degradation of synchronising between multiple parties (and sometimes waiting for slower computers to catch up). Famously, some consensus protocols can also have huge costs of their own, such as Bitcoin with its Proof-of-Work consensus [10], although it should be mentioned that Proof-of-Work consensus has some unique advantages as well, and other, significantly more efficient forms of consensus exist also – particularly, Proof-of-Stake. As an example, Ouroboros Genesis is a Proof-of-Stake protocol [11] and Ethereum is also planning a move to a Proof-of-Stake consensus in the

near future [12]. Permissioned blockchains often use efficient protocols as well, such as the Tendermint BFT protocol [7], which we will use as our canonical example. Moving to the area of privacy, calculating with confidential data in a decentralised manner can also take additional computing power that otherwise would not be needed [9], and due to the replicated nature of blockchains, there can be a bigger need for such advanced privacy techniques than there would otherwise be (Bitcoin serves as a notable example [13]). Blockchains are also a relatively recent technology, thus it is natural to expect them to have early adoption costs: training for developers and users, immature tooling and so forth. Therefore, it makes sense to consider where blockchains truly add value – the areas where they are worth their costs.

It is also not immediately obvious where such decentralised architectures offer anything truly unique: regarding the often claimed strengths of blockchains – trust and security [8, 14] – the more traditional (centralised) architectures can also offer good levels of assurance. To pick a few intuitive examples, people often trust banks with their life savings (perhaps simultaneously relying on insurance companies and government bailouts), they trust companies like Microsoft and Apple to upload backdoor-free closed-source operating system updates straight to their computers (repeatedly), and they trust that their Intel or AMD CPUs (central processing units) are not backdoored with covert/steganographic channels set to receive secret commands from afar. Even the nuclear weapons of the world are not controlled by blockchains. Additionally, centralised cloud services seem to be getting more trustworthy over time: Amazon AWS has had a GovCloud region for some US government services since 2011 [15] and they have even introduced Top Secret workloads for the intelligence community as an air-gapped service [16].

Thinking about this further, in many ways the centralised and decentralised systems seem to be more similar than it would at first appear. For example, the traditional centralised systems can also be co-administered by multiple entities from different organisations. Specifically, centralised administration ceremonies can be used where multiple witnesses watch the entire process of administering computers, and the process could even be filmed or live-streamed online; various physical defences can also be employed, including tamper-detection, surveillance cameras and security guards. Public examples of such procedures include the DNSSEC key signing ceremony [17, 18], as well as banks and certificate authorities (CA) who often use similar methods for securing their centralised systems (sometimes called the "four-eyes principle")[1]. In other words, solving reliance on single

[1] In the simplest case, the four-eyes principle could mean that the server's root password is split between two administrators, thus ensuring that no single administrator has access to the server alone. Similarly for the physical defenses – the security code to disable the alarm in the server room can also be split. In practice, it is advisable to use more flexible and more secure schemes, for example using two-factor authentication and allowing for back-up administrators.

individuals and organisations does not necessarily require decentralised blockchains.

At the same time, decentralised blockchains are often *less* decentralised than it would at first seem. As security in general is often considered to be a "weakest link in the chain"-type of a problem where the least secure component can often determine the security level of the overall system [19], it can be important to keep all major components as secure as possible (except when the weaker components are shown to be well isolated from the rest of the system).

Of particular concern is the fact that the more privileged components of a computer system can have almost full control over the less privileged ones: a CPU can alter and observe the software it runs, an operating system can similarly alter and observe the software that it runs (excluding secure isolation mechanisms like Intel SGX and other secure hardware [20]), and the human operators can often alter and observe both hardware and software under their control. With that in mind, security bottlenecks can be created when a major set of users are sharing the same hardware and software combinations, thus making it easier for one manufacturer to compromise a large number of participants at once (either by mistake or due to malice).

Often the blockchain software itself has only one popular implementation available, and even if all users compile that code directly from the source using a trusted compiler, not many people double-check each and every line of the code involved, and then double-check it again after every update. In a sense this is worsened by the potential for split-view attacks, where even if only a small number of blockchain consensus participants are compromised, they could maintain parallel ledgers such that correct computers would not detect a consensus failure (only the users of the compromised computers would be affected, and may not notice the problem either for a while).

Secondly, even the most secure blockchain in the world has to be observed through the lens of a user's computer, and that computer needs to be trusted by its user. Yet the machine probably receives various binary updates over the air and contains hardware that nobody has verified under a microscope. Even the typical best-case scenario has potential holes: a diligent user of open-source software could build her own binaries straight from the source that she downloads from GitHub; however, she typically still trusts GitHub to give her the exact same source code that auditors and reviewers have looked at, without any modifications; additionally, the first compiler that she downloaded was also probably a binary file that she did not inspect. Thus, given that there are limits to the trustworthiness of users' machines (which amount to the users' "eyes and ears"), it may not even be necessary to make the service itself much more trustworthy than the combined "eyes and ears" of

its users (otherwise, the attackers could simply move their attacks to the users' machines instead, as in some cases they already have [21]).

Compromised blockchain computers may lie and equivocate in arbitrarily clever ways, lying to other computers as needed and also lying to their own users: the computers could for example display data that does not match its own cryptographic hash values, making it appear to their users that the often-used sanity checking values blockchain block hashes are correct even though the actual block contents are different. To catch the problem, users would have to compare with non-compromised users the exact blockchain data contents (even though on a non-compromised computer, merely comparing the cryptographically strong hash values would suffice).

Thus, it can be useful to investigate in a more detailed way what the practical differences are between the security properties of blockchains and centralised architectures, especially given that decentralisation also has associated costs.

1.2 Restrictions on the Work

For readability purposes, the terms *distributed ledger technology* (DLT) and *blockchain* are considered synonymous in this work. Some sources use the terms in the same way (as an example, Wüst and Gervais appear to use the terms interchangeably in [8]). Other sources consider blockchain to be a special case of DLT (for example, [22] and [4]). For the current work, the difference between blockchain and DLT is considered to be an implementation detail: unless otherwise specified, anything that applies to blockchain also applies to DLT overall, and vice versa.

We say that a blockchain is *permissioned* when participants have to be explicitly allowed to use the system (typically using their known and verified identities). This is in contrast to *permissionless* blockchains where anyone is allowed to participate, typically including anonymous actors.

We focus our study on *permissioned blockchains*, which we define as (permissioned) decentralised services that aim for greater integrity and availability than to their single participants' machines. Typically they utilise a Byzantine fault-tolerant (BFT) consensus protocol that allows any small subset of participants to fail in arbitrary ways (including lying on purpose), without breaking the integrity or availability of the system (for background on blockchains and consensus, refer to section 2.1). Where a specific example is needed, we often refer to Tendermint as a canonical permissioned blockchain platform [7]. Tendermint uses a BFT consensus algorithm with strong similarities to the traditional Practical Byzantine fault-tolerant consensus (PBFT) algorithm [23], it includes full client-side validation and supports a gossip protocol [24] to ensure eventual delivery of important data across any available routes among participants.

For our purposes, unless otherwise noted, we only consider blockchains which use a Byzantine fault tolerant (BFT) consensus model, which is the strong model of consensus as it allows its participants to arbitrarily deviate from the protocol, even to maliciously lie on purpose. Our findings apply for various network models as well (synchronous, asynchronous, partially synchronous, among others).

We also analyse a type of architecture that is, in a sense, between the blockchains and centralised architecture (and can be combined with each if necessary): the on-site security ceremony. Such architectures are technically centralised (that is, the computations are *not* re-verified on multiple machines), but there are procedures in place to ensure that no single person or organisation (as needed) could access the system single-handedly (both in terms of physical and remote access). We describe such ceremonies in more detail in section 2.2

and analyse their key properties in section 3.4.

1.3 Research Problems

As we wish to compare the security risks of blockchains against their more centralised counterparts (including the on-site ceremonies), we need a way to assess these specific security risks which differ across the architectures.

In traditional, more centralised architectures, users have to *trust* a single party to run the system correctly (see section 3.1 for our definitions of trust), whereas in blockchain systems like Bitcoin, Ethereum and Tendermint, several parties re-verify the accuracy of the computation and state of the service. This acts as a diversification mechanism against the failures of any single (central) party. However, we must remember that the client computers need to be secure enough as well. Thus, we should consider the risks of both.

Mark Silver et al. write that an information system "comprises hardware, software, data, people, and procedures" [25]. Of these, we focus on the following three components: hardware, software and people.

Thus we end up with the list of sub-items that we wish to take into account as can be seen in figure 1.1.

```
1 Server side, including consensus if any:
    1.1 People (integrity, coercion, management, phishing)
    1.2 Software (vulnerabilities + backdoors)
    1.3 Hardware (vulnerabilities + backdoors)
2 Client side, consensus and/or "full node" verification
    2.1 People (same as above)
    2.2 Software (same as above)
    2.3 Hardware (same as above)
```

Figure 1.1. Aspects of security to be taken into account, both on the server side and on the client side, including people, hardware and software.

Our goals are:

- Taking the above needs into account, create a model that can be used in conjunction with a simple multi-parameter attack tree model based on Buldas et al. [26] to take into account the risks enumerated in figure 1.1 in the following way:
  - Using expert estimates, create a simple model of human behavior to estimate

the likelihood that an employee will act honestly, based on the seriousness of misbehavior, likelihood of getting caught, the social status of the employee, and various coercion methods (extortion, managers changing team contents, infiltrating an organisation through legitimate channels, bribing). Additionally, assess the likelihood that a person will give up another person to authorities for suggesting a conspiracy and how that affects the other person's willingness to suggest a conspiracy.

  - Using publicly available information on the history of 0-day vulnerabilities (roughly, 0-day vulnerabilities are "secret" vulnerabilities which exist in production software and are not known to the vendor and users at large), assess which ones have so far been critical for securing web services, including operating systems and web server software; estimate the rate of critical 0-day vulnerabilities to be found for operating systems and web servers to estimate under which condition blockchain architecture could be used to achieve resilience against them.

- Perform a simple validation of the model from the previous goal by comparing a blockchain and a centralised architecture for the same system.

1.4 Research Methods

The work analyses socio-technical aspects of security and was carried out using the design science methodology.

The main work product is a model designed to assess the socio-technical aspects of security for heterogeneous systems. A criminology expert was consulted with to assess the parameters suitable for the model.

Suitability of the model was validated by applying it to sample use-cases, after which subjective opinions were formed to assess the performance of the model.

To evaluate the model, the widely used attack tree approach was chosen due to its apparent fitness for purpose and for its game-theoretic semantics, which allow to quantitatively study the socio-technological aspects. The attack model was based on a model from Buldas et al. from 2006 [26]. Even though the models for computation of attack trees have since been improved [27, 28, 29, 30, 31, 32, 33, 34], the earlier model from [26] was sufficient for the current work.

2. Background

2.1 Blockchains and DLT

This section will introduce blockchain technology. To define what blockchains are, we come back to the definition proposed by Wikipedia [4]:

> A distributed ledger (also called a shared ledger or distributed ledger technology or DLT) is a consensus of replicated, shared, and synchronized digital data geographically spread across multiple sites, countries, or institutions.[5] Unlike with a distributed database, there is no central administrator.[6]

Thus we can think of DLT/blockchains as a kind of distributed database technology. Like many other databases, blockchains can store data, set permissions and execute code on the data. In the database world the code is often called *stored procedures*, whereas in the blockchain world it is often called *smart contracts* [35].

As the above definition from Wikipedia specifies, blockchains differ from distributed databases for their lack of central administrators. Thus, no single administrator can tamper with the database: not with its data, nor with its code and neither with its permissions. Unless otherwise noted, we are assuming a Byzantine fault tolerant (BFT) model, a strong model of consensus which allows participants to break the protocol in arbitrary ways, including lying maliciously on purpose.

Any reading or writing to the database will have to be approved by a consensus between multiple participants. The exact details vary between systems, but as a simple mental model, we can imagine that the participating computers, owned by different entities, are coordinating and voting in real time between each other whether to approve any specific state updates for the system. In some cases, such real-time voting is literally a majority voting protocol, albeit the voting happens in multiple rounds and phases – as is the case with the PBFT/Tendermint consensus [7] – and in other cases like Bitcoin and Ethereum, their Proof-of-Work consensus achieves *eventual consistency* across multiple participants over time [36, 37].

Naturally, every bit of shared state is exactly the same across all parties (provided they have synchronised up to the same point). Often this means that any state-changing transactions that are executed in the system must be fully deterministic, as they must always lead to exactly the same result on all replicas (and the replicas cannot simply trust others to do the computations for them). A notable exception is Hyperledger Fabric which does allow non-deterministic transactions, but eventually, the consensus still decides on exactly the same value for everyone (that is, the transactions have to be deterministic on enough "endorser nodes" at the same time during the time of "endorsing", but they do not need to be fully deterministic at all time and across all machines) [38].

Typically, blockchain systems replicate data in discrete data packets called *blocks*:

$$b_0, b_1, ..., b_n$$

The first block b_0 is often referred to as a *genesis block*, and each following following block b_i (where $0 < i \leq n$) links to its previous block b_{i-1} via a cryptographically strong hash value (traditionally, this has often been the SHA-256 hash function taken over the contents of the entire previous block). Since the hash value of every block is calculated over the entire block contents, it therefore also hashes over the hash value of the previous block. Thus, the hash of the last block b_n uniquely authenticates the entire chain of blocks (and as hash values tend to be short, they can be conveniently compared between participants even using low-bandwidth communication mediums, even by humans speaking over the phone, if the need ever arises).

Over time, this forms an ever growing hash chain of blocks which allows auditing and of the entire system over time, bearing similarities to the event sourcing architectural pattern. Under normal operation, no blocks ever get removed or modified, only appended (except as part of an eventually consistent consensus algorithms like Bitcoin and Ethereum use, where the more recent blocks may take some time to settle).

In general (without sharding and privacy considerations), all participants receive the exact same set of blocks. Therefore, everyone shares the same history: there should not be a parallel history that differs (except again for short time periods in eventually consistent consensus algorithms). Such parallel histories are called "forks" of blockchains, and consensus algorithms attempt to prevent them as best as they can. When illegal forks happen, many blockchain security properties can no longer be guaranteed; perhaps the most famous example of this is the double-spending problem, where the users can spend the same amount of money multiple times (once on each fork), and each fork on its own

looks legitimate to observers.

For completeness, the term "fork" can also refer to perfectly welcome changes in the consensus protocol which may lead to parallel chains: one chain with the old protocol and another chain with the new protocol. However, as these changes are planned, measures can be taken to ensure that people are not negatively harmed by them (for instance, reminding everyone to upgrade their software can help ensure that they do not stay on the old parallel chain) [39].

In many designs (including Bitcoin and Ethereum) all fully validating clients download all blocks (even if they are not consensus participants themselves); in real time, as the blocks become available, such clients verify the entire contents of the block (according to shared validity conditions) and only accept such blocks whose state updates fully satisfy their validity conditions. For example, in the case of Bitcoin, double-spending transactions are not allowed to be included (these are transactions that spend money that the owner has already spent, without receiving it back from anyone) – thus even if the consensus participants create a block with an invalid double-spending transaction, the other fully validating clients will reject that block and any updates that it contains are ignored.

The blocks also often include any necessary metadata for verifying the correctness of the block contents – after all, the verifying participants cannot simply trust that the block contents are valid –, for example, currency transfer transactions may be signed by the respective account owners, and the resulting digital signatures are included inside the blocks. The blocks also come with authenticating information to verify that the consensus participants have indeed approved the block, as required by the consensus algorithm (in a simple case, this could be a list of digital signatures from consensus participants over the block).

An important classification of blockchains is around their openness to welcome new participants. *Permissionless blockchains* like Bitcoin and Ethereum are open for participation by anyone: merely having enough computation power is sufficient to allow anyone to participate in the consensus of these blockchains, and anyone is allowed to transact on the systems. Such platforms attempt to stay as neutral as possible, welcoming any participation by all people, often anonymously, generally without the need to perform Know-Your-Customer (KYC) verification.

Permissioned blockchains, on the other hand, allow only identified participants to perform in consensus and/or to transact on the blockchain. The permissioning entity could be a single entity or it can be a coalition of multiple parties. Examples of permissioned

blockchain technology include Hyperldger Fabric and R3 Corda.

Public blockchains are visible to everyone, *private* blockchains are visible to a smaller number of parties.

Public proof-of-work blockchains (for example, Bitcoin and Ethereum) typically call their consensus participants *miners*, whereas permissioned blockchains often refer to them as *validators* (note that when public Ethereum will be transitioning to proof-of-stake consensus, it too will start to call their consensus participants as validators).

Many blockchains have the property of making appearing more resilient, more secure and more trustworthy compared to some central parties alone. Some public permissionless blockchains (particularly, Bitcoin and Ethereum) are further claimed to have the property of *censorship resistance* from governments; although this property is not analysed in the current work, it is an often-claimed benefit of such blockchains.

For more information on blockchains in general, Wikipedia has an introduction article and links to resources for further study [40].

2.2 On-site Security Ceremonies

Carl Ellison introduced the idea of security ceremonies in 2007, with the following abstract in his paper [41]:

> The concept of ceremony is introduced as an extension of the concept of network protocol, with human nodes alongside computer nodes and with communication links that include UI, human-to-human communication and transfers of physical objects that carry data. What is out-of-band to a protocol is in-band to a ceremony, and therefore subject to design and analysis using variants of the same mature techniques used for the design and analysis of protocols. Ceremonies include all protocols, as well as all applications with a user interface, all workflow and all provisioning scenarios. A secure ceremony is secure against both normal attacks and social engineering. However, some secure protocols imply ceremonies that cannot be made secure.

In the current work, we focus specifically on *on-site* security ceremonies as an *alternative* to blockchain consensus and other forms of verification available to blockchain users. To be clear, blockchain consensus is also a kind of security ceremony in itself, involving

both people and protocols; we contrast blockchain consensus with on-site ceremonies that typically take place in a fixed location and lead to an agreement between humans that they a set of computers are trustworthy, without the need for a blockchain-type verification afterwards. Examples of such ceremonies include the DNSSEC key signing ceremony [18, 17], various private PKI key ceremonies on the digital certification market [42] and ceremonies for counting the votes in the national internet voting systems in Estonia, as well as electronic voting pilots in Norway and Australia [43], among others.

Although using a blockchain consensus can be more secure compared to meeting in person due to less physical risks and the fact that the operators of the blockchain consensus computers do not necessarily even have to know each other, there are also ways to make the on-site ceremonies more secure: not everyone needs to be physically present: cameras and other sensors can be used for remote participation, for which the participants can send trusted representation to set them up.

Although the economics of the solution are left for future work, a potentially simple solution for remote participation could be the use of smart phones as sensors on the ceremony site, potentially more than one phone per user (multiple phones allow the user both to get a better view of the ceremony environment and also to get a better view of whether the phones themselves are being maliciously tampered with). Users who trust the same phones could also share the feeds from the same phones. Phone cameras can be used for getting a live recording of the ceremony, various wireless connections on the phone might be usable for connecting to electronic tamper-evident seals and depth sensors on the phones can provide for additional data. On a very high level, this may be expected to replicate the canonical blockchain verification model – in both cases, multiple users are "watching", they all have to trust their own machines (in this case, also their sensors), and they all connect to each other "electronically" without being physically present.

A remaining difference would be the machine itself that performs the computation of the service – thus, if one does not trust any particular operating system or hardware manufacturers, a BFT consensus would still be warranted. For completeness, it should be mentioned that the on-site ceremony also supports setting up a BFT consensus of machines on the site, both with or without a corresponding "regular full node" verification on the client side. Which model is the best for a given situation will have to be decided on a case-by-case basis.

An interesting use-case for using a hybrid BFT consensus together with a ceremony could be when the number of consensus nodes needs to be low (perhaps 2 or 3 computers) due to economical or performance reasons, yet the people using the system do not wish to trust

merely one operator for each of these computers. And at the same time they may not be willing to settle for just 1 computer, as they may not trust a given computer architecture enough. In that case, all such consensus computers could have a security ceremony of their own, with multiple participants (some participants potentially joining remotely).

Physical defenses can also be duplicated redundantly (for instance, if security hardware is not mutually trusted, each participant could send their own security hardware into the computer room). Section 3.4 goes into more details on the assumptions on how many people would have to collude in a ceremony to cause the system to misbehave.

We call the people who participate in the on-site ceremony *witnesses*. These people do not necessarily have to be security experts themselves; rather, the ceremony can often be designed in such a way that, with proper instruction, the participants can follow along and verify the honesty of the ceremony without much IT education. There are usually one or more leaders (perhaps called *ceremony administrators*) who perform the work of setting up and configuring computers, while witnesses are carefully watching to make sure the administrator follows his intended script and does not do anything suspicious.

It is also possible to extend the ceremony to the purchasing of computer equipment (in the simplest case, all participants could enter a computer store together; in more complex cases, the equipment could for example be sealed in tamper-evident packaging in factory and the on-site ceremony participants will observe the unboxing of the computer together, verifying that the tamper-evident packaging has not been tampered with).

In principle, the concepts of on-site security ceremonies can be extended to every aspect of IT: hardware manufacturing lines, software development, delivery services, and so forth – could all be implemented with remote and/or local witnesses present. Such approach might allow for manufacturing hardware that multiple non-friendly countries can trust simultaneously; however, the economics of such operations would have to be separately analysed, and lacking any such initiatives in practice, it might be make more sense for such countries to use a decentralised protocol between themselves, especially for an one-off case.

3. Body of Work

3.1 Blockchains as "Trust Machines"

Many factors can go into the decision process to use a blockchain, but almost universally, the most common advantages of blockchains are said to be trust, honesty, security and transparency. We have previously described how the properties of trust, honesty and security are interrelated in section 3.1. The remaining property of transparency is not strictly unique to blockchains: centralised architecture could just as easily provide more data for the world; the difference might be lack of trust in such data, which brings us back to the word "trust".

"Trustworthiness" is perhaps the most frequently discussed benefit of blockchains. In 2015, The Economist famously called blockchain the "trust machine" [14] and Wüst and Gervais write in [8] the following: “In general, using an open or permissioned blockchain only makes sense when multiple mutually mistrusting entities want to interact and change the state of a system, and are not willing to agree on an online trusted third party.” This suggests that a *trusted* online third party could generally serve as a replacement for blockchains, if such a party exists and is agreed to by the parties. Thus, blockchains are claimed to be useful in general only if one cannot find (and agree to) a trusted third party to run the (state-changing) functionality of the service.

The above conclusion also makes intuitive sense: you can take any blockchain system and transfer all its participating computers to a common owner, perhaps a well known cloud business. To the users, the blockchain would still function exactly the same, but what has changed is the ownership over participating computers: the blockchain would essentially be owned and operated by a single entity. The entity can then proceed to remove the blockchain technology without affecting functionality for their users: first the entity might reduce consensus participants to just one member (that is, one computer that they own), and then remove the consensus code altogether. The users would still be able to use everything like before, except eventually there will be little blockchain technology left in the system. If the central party is trusted by their users enough to do this, this appears to be a valid alternative to the blockchain, possibly with its own set of advantages (efficiency and cost would be likely candidates). Thus, the lack of trust in the central operator’s ability

to run the service single-handedly would be a clear and valid reason to use a blockchain instead (or to move to an alternative design altogether), yet if the trust was there, the functionality could, in principle, still be run centrally.

On the flip side, there can also be situations where consensus would be *less* trustworthy than a central actor: if the majority of consensus participants are maliciously colluding, it may be better to use a trusted central party instead. (And using multiple such trustworthy central parties together in BFT blockchain consensus would be better still.) Naturally, system designers are expected to build their systems according to the needs of a situation.

But what exactly is meant by the word "trust"? We have adopted the definitions from the 1995 paper due to Mayer et al. [44], where they propose three different aspects of organisational trust:

1. *Ability*. "Ability is that group of skills, competencies, and characteristics that enable a party to have influence within some specific domain." [44]
2. *Benevolence*. "Benevolence is the extent to which a trustee is believed to want to do good to the trustor, aside from an egocentric profit motive. Benevolence suggests that the trustee has some specific attachment to the trustor." They clarify the concept with an example involving a manager and an employee: "If the manager also were benevolent toward the employee, he or she may try to protect the employee from the possible ramifications of mistakes. A manager who is less benevolent to the focal employee may be more disposed to use the information in a way that helps the company most, even at the possible expense of the employee" [44]
3. *Integrity* means that "the trustee adheres to a set of principles that the trustor finds acceptable". It should be added that integrity includes personal morals and work ethic, people doing what they promise. "Such issues as the consistency of the party's past actions, credible communications about the trustee from other parties, belief that the trustee has a strong sense of justice, and the extent to which the party's actions are congruent with his or her words all affect the degree to which the party is judged to have integrity" Using again their example of a manager and an employee, they mention that "if the manager's integrity is questionable, can the employee help but wonder how long it will be until the manager betrays her or him as well?" [44]

We will be using the above definition of trust in the context of teams and employees, many of whom are working to provide useful services to their users. We suggest that the above *benevolence* property is seldom used in such business contexts – in a business setting, one does not generally rely on forming significant personal attachments to each other – thus we focus on the other two aspects of trust instead: *ability* and *integrity*. We tie both of

these aspects together under the following definition.

Definition 3.1.1 (Dishonesty)**.** We call a service *dishonest* or *untrustworthy* if it fails to comply with at least one of the following criteria for correctness at least once during a time period:

- All computers that make up the service (often called "servers") execute only their designated software, without alteration, and nothing else. (In practice, this definition has to be somewhat relaxed, so long as no significant harm will be done by this. For example, a legitimate system administrator asking the server to tell its local date and time does not necessarily constitute a trust violation.)
- All people working on the service follow their designated work protocols.
- All designated software and protocols of the service are created in such a way that most experts in the intended society would consider the system reasonably fair and secure.
- We include all aspects of security from the traditional security triad of confidentiality, integrity and availability, unless mentioned otherwise.

For the remainder of this work, it is assumed that customers would naturally want to trust only reputable companies and that such companies generally make a serious attempt to hire capable and reasonably trustworthy people. Thus we do not investigate the question of assessing capabilities of employees (for example, auditing and certification can be useful for such purposes). We also do not estimate reputations of companies; rather, we assume an existing good reputation and focus on how that company should build its processes: whether they should use a blockchain or not, whether some positions in the company would require multiple people approving certain actions.

3.2 Properties of Tendermint Consensus as a Typical Example

Byzantine fault-tolerant consensus algorithms can be parameterised in multiple ways – for example, to favor availability over integrity, or vice versa. A typical balance for permissioned blockchains is used by the PBFT/Tendermint consensus algorithm [7], a partially synchronous BFT consensus with strong similarities to the Practical Byzantine Fault Tolerance (PBFT) algorithm from [23]. The algorithm is also used in the Cosmos public blockchain network [45]. The properties of the Tendermint consensus are summarised as follows.

The mechanism uses a "round robin" leader rotation scheme where each consecutive block is proposed by a different consensus participant. This ensures that a small number of participants cannot censor transactions or keep the system from progressing for long.

In the simplest cases, all consensus participants have an equal number of voting power, but the following also generalises for more complex voter-weight assignments.

Here and in the rest of the document, we use the following notation to represent the floor and ceiling functions:

$$\lfloor x \rfloor = \max \{m \in \mathbb{Z} | m \leq x\}$$
$$\lceil x \rceil = \min \{n \in \mathbb{Z} | n \geq x\}$$

We represent the total number of consensus participants as n, where $n \geq 1$. In practice it often makes sense to use $n \geq 4$ consensus participants in order to allow for at least one redundant party, due to the fact that the protocol requires at least $\lfloor \frac{2}{3} n \rfloor + 1$ consensus parties to function correctly for it to finalise new blocks.

Depending on the number of consensus participants, failure scenarios can occur under the following conditions.

- An illegal *fork* in blockchain (that is, an illegal parallel blockchain) can occur if at least one of the following is true:
  - At least $n - 2(\lceil \frac{1}{3} n \rceil - 1)$ consensus participants are acting "maliciously" and network packets are being delayed to prevent the remaining ”honest” participants from seeing the other parallel chain before finalising on their own one. (See appendix on page 78 for an explanation of the formula.) For example, with $n = 10$ total consensus participants the attacker would need to co-opt at least 4 of the participants (assuming again that specific network packets can be

delayed sufficiently).
  - Over $^2/_3$ of consensus participants (that is, $\lfloor \frac{2}{3}n \rfloor + 1$ participants) are "malicious" and all network packets are arriving on time. For example, for $n = 10$, that would be 7 participants.
- An invalid state transition in blockchain (for instance, a payment of more money than an account holder actually has) can occur if over $^2/_3$ of consensus participants (that is, $\lfloor \frac{2}{3}n \rfloor + 1$ participants) are "malicious".
- A failure to update the blockchain state can occur if at least one of the following is true:
  - at least $\lceil \frac{1}{3}n \rceil$ the consensus participants are unavailable to participate. For example, for $n = 10$, that would be 4 participants.
  - the one consensus participant whose turn it is to propose a new block is unable to participate, and the same for the next proposer in line and so forth (one proposer at a time).
- A transaction can be censored if at least $\lceil \frac{1}{3}n \rceil$ the consensus participants are willing to reject blocks that include that transaction (and they can create blocks without that transaction when it is their time to propose a block).

There is often a general assumption that adding more participants to the consensus makes the system more reliable. However, this only holds true if the new participants are reliable enough. To illustrate this, we can calculate the likelihood that at least c among n parties are reliable using the binomial distribution, under the assumption that the all parties have the same probability p of being reliable (X is a random variable representing the number of honest parties):

$$Pr(X \geq c) = \sum_{i=c}^{n} \binom{n}{i} p^i (1-p)^{n-i}.$$

As an example, table 2 lists the availability probabilities of a consensus group with $n = 10$ participants within the Tendermint protocol (that is, the probabilities that over $^2/_3$ or 7 of the participants are available), depending on different reliability probabilities of each individual member (p).

We can see from the above table that availability of consensus with $n = 10$ participants can be dramatically better than availability of a single member, but only if the availability of a single member (p) is reasonably high. Accordingly, with smaller values of p, the consensus availability can be significantly less than that of a single member. In practice, it is generally assumed that the configurations are chosen favorably by the designers of the

Table 2. Probabilities that, in a group of $n = 10$ participants, we have at least 7 participants who are reliable, depending on the probability of a single participant being reliable (p), assuming the probabilities for all participants are independent.

p	$Pr(X \geq 7)$
0.0	0.0000000000
0.1	0.0000091216
0.2	0.0008643584
0.3	0.0105920784
0.4	0.0547618816
0.5	0.1718750000
0.6	0.3822806016
0.7	0.6496107184
0.8	0.8791261184
0.9	0.9872048016
0.99	0.9999979987
0.999	0.9999999998
1.000	1.0000000000

system.

3.3 Blockchain Validation Beyond Consensus

In many popular blockchain designs there is a distinction between blockchain clients who participate in consensus and other clients who do not. Those who do not participate in consensus can still verify blockchain contents for themselves, and depending on a design, even sporadically alert other users automatically of any wrongdoings they find, for example by broadcasting computer-verifiable fraud proofs [46], or even participating in protocols to punish the misbehavior of consensus participants [47]. Such blockchain clients who download and verify all blockchain contents are often called "full nodes" or "fully verifying nodes", as is the case for Bitcoin [39] and Ethereum [48], among many others.

Here we will distinguish between two kinds of full nodes: these who participate in consensus are called *validators* and these who do not are called *regular full nodes*. Often, *lightweight nodes* are also mentioned which do not verify all transactions [49]; such lightweight nodes are out of scope for the current work.

This, even if the consensus is compromised (either consensus participants agree to create "illegal" parallel blockchains – *forks* – or they agree to produce blocks that are considered invalid by the blockchain protocol, perhaps including transactions that spend money which does not exist), other regular full nodes can still verify a lot of their behavior and discover their wrongdoings. Let us classify the different types of such misbehavior that consensus participants can engage in and how well each can be detected by regular, non-validating full nodes. We will cover all aspects of security – confidentiality, integrity, availability. (For reference, chapter 3.2 lists the conditions under which integrity and availability failures can occur under a specific consensus algorithm.)

- Confidentiality of blockchain contents is typically not a task for consensus, unless it is a private chain or the system is using privacy techniques like secure multi-party computation (MPC). There are no good ways for regular full nodes to directly detect whether a computer has privately leaked information to outside parties who continue to keep it a secret; only probabilistic predictions can be made about the likelihood of such behavior.
- Integrity failures. Integrity failures can be created by the consensus in the following two ways:
  - Illegal blockchain forks (illegal parallel chains). These cannot always be detected with certainty by full nodes, unless they receive some help from other blockchain clients, due to the fact that lever consensus participants would not send multiple forks to the same client, but each fork would go to a different client. To learn about the duplicity of the validators, in the general case of a

typical PBFT consensus algorithm the targeted clients would need to hear from other clients that received a different parallel blockchain. Such synchronisation is often implemented using a peer-to-peer gossip algorithm – for an example, see [24] – but in principle, it could even be implemented manually (perhaps by the users occasionally exchanging block hash values with each other). It is also important to note that for this detection to work, the clients on the other side of the fork must be available, able and willing to send their information to the participants on other forks; attackers may attempt to disrupt or delay such synchronisation, for example by flooding the synchronisation protocol with their own messages.

For completeness, it should be mentioned that there exist also situations where the lack of parallel blockchains can be reasonably detected without communicating with other parties, such as when enough consensus participants are known to be extremely trustworthy, or when a consensus algorithm makes forking "physically impossible" (as an example, with Proof-of-Work, if the world's entire computational power is known and accounted for and most of it has clearly participated for long enough time in the creating of a single chain – as can be seen from the Proof-of-Work artefacts on the blockchain itself –, it would be extremely unlikely for another parallel chain to exist).

  - Invalid blocks (according to the validity conditions of a given blockchain). Assuming no bugs in software, this attack should be perfectly detected every time by every full client that receives the invalid blocks. Such attacks may fool more lightweight nodes that do not validate all block contents, however.

- Availability
  - Censoring updates to the chain (either all updates or just specific transactions, perhaps from certain users). This may be naturally detectable by the user(s) whose transactions fail to go through; in some cases, other observers on the network will be able to detect censorship as well, and even specific protocols have been proposed for this purpose (for one informal suggestion, see for example [50]).
  - Hiding block contents from parties. Technically, once the consensus participants have created a new block, they do not have to show the completed block to anyone, or on a smaller scale, they could show the block only to select participants. Obviously, anyone who does not receive a block will notice it, especially if the blocks have to arrive at a known rate (say, once per second on average).

To summarize, all integrity and availability failures can be detected by regular full nodes, with the caveats that detecting invalid forking requires exchanging information between

parties on different forks, and availability failures can only directly be detected by the parties to whom the system is not available (without additional observer protocols).

This ability of regular full nodes to detect consensus failures can be very important: consensus participants may be less likely to misbehave if they know they are being observed, and it allows other network users to take action against the misbehaving parties (including taking away their rights as consensus participants and punishing them with for example monetary fees; this has been automated in the so-called "slasher" protocols for example, where consensus participants have to put down security deposits that can be taken from them automatically if the network detects that they have signed off on multiple forks of the blockchain, have built an invalid block, or appear to be ”offline” for too long [47]).

3.4 Properties of On-Site Ceremonies

As specified in chapter 2.2, on-site security ceremonies can be seen as a natural alternative for blockchain consensus (at least for permissioned implementations), and the ceremonies can, in a sense, also function as an alternative for client-side full node verification (assuming there is enough security). This idea of using on-site security ceremonies is not new: variants of such security ceremonies have been in use for some time in different contexts (for example, see [18, 42, 43]). The current chapter answers the following problem: if we already know the integrity of the people involved, which we express as a probability of them acting dishonesty, how do we calculate a corresponding probability of dishonesty for an entire on-site security ceremony? Later chapters will investigate how to assign more concrete probabilities for different people.

In the case of BFT consensus protocols, a typical configuration is to allow the consensus to proceed so long as over $2/3$ of consensus participants have signed off on the block, thus producing an invalid block would require comprising an equal number of nodes: over $2/3$ of consensus participants. In principle, one could achieve even higher integrity guarantees by requiring a larger share of participants to sign off on block, all the way up to 100%, but there would be a price to pay in availability. For instance, in the extreme case of requiring 100% of participants to sign off on every block, the system would stop producing any blocks every time there was even one participant who was unwilling (or unable) to sign off on a block: every single participant would have veto power over any and all updates.

On the other hand, for on-site security ceremonies, it might not matter if someone vetoes a ceremony if we can simply reschedule it again at a later date, perhaps with different participants next time. Eventually we should have a ceremony that succeeds and we can start to use the computer we have just set up; at that point, nobody can veto it as it is already running and no single party would have access to it alone. There could still be situations where we need to act fast – perhaps when a data center has gone offline unexpectedly and one needs to move the application over to another data center quickly; at that point, it could be disadvantageous to have the entire decision to move data centers be vetoed easily by a single person.

For simplicity and for ease of comparison, we will assume that the on-site ceremony will also have a threshold of over $2/3$ participants needing to confirm any action of system administration that could result in a security violation. Various different models are possible as well (in particular, appendix on page 82 goes into a bit more details about the risks of allowing a single member of a team to veto the result of the ceremony and suggests some solutions).

Let us specify conditions analogous to chapters 3.2 and 3.3, which follow naturally from our established premises:

- Confidentiality. An on-site ceremony can be used to configure a computer that operates on secret data, analogous to secure multi-party computation but on a single computer and secured by the ceremony instead of decentralised architecture. Secure multi-party computation can be configured to accept various thresholds for honest members, as can the on-site ceremonies. Here, however, as stated above, we have chosen the option that over $^2/_3$ of the witnesses have to approve every ceremony.
- Integrity, which has 2 sub-parts.
  - Invalid forks. It is here that the witness ceremony has an advantage for the same number of people on a BFT consensus algorithm like Tendermint, due to the fact that invalid forking is not possible on a properly set up centralised architecture: it is not possible for a centralised computer to split into two computers, each on a separate "fork" of state updates. Thus, the number of people to achieve a "fork" in state would be again over $^2/_3$ of the witnesses, as that is the number of people who can successfully complete a ceremony to configure the computers however they see fit.
  - Invalid state updates (blocks), similarly, require over $^2/_3$ of the witnesses to be complicit, a number similar to Tendermint consensus.
- Availability.
  - Censoring some or all of the transactions, once the system is operational, would again require over $^2/_3$ of the witnesses to be complicit, as less people cannot re-configure the system in any way. This number is larger than with Tendermint consensus.
  - However, stopping the ceremony from functioning altogether can be achieved with merely $\lceil \frac{1}{3}n \rceil$ of witnesses (a number that is again comparable to the typical BFT consensus parameters).

3.5 Comparing Blockchain Consensus and On-site Security Ceremonies

As established in chapter 3.3, we are considering two modes (layers) of verification in a typical blockchain (including in Bitcoin and Ethereum, as well as the Tendermint consensus we use as an example): the two layers are:

1. The consensus protocol (e.g. the Tendermint consensus protocol).
2. Verification by regular full nodes.

Our cited source [49] additionally refers to lightweight nodes, which are out of scope for the current work. As stated in chapter 3.3, the second layer (verification by regular full nodes) is virtually perfect when detecting invalid blocks (to the extent that the verifying full node itself is trustworthy), whereas detecting forks in the blockchain with Tendermint consensus (and many others) requires successfully hearing from other parties who are willing, able, and are aware of the other parallel chain. Such fork detection is often implemented as an automated gossip protocol (for an example, see [24]; Bitcoin, Ethereum and many other protocols also utilise gossiping protocols).

Let us compare the properties of Tendermint consensus from chapters 3.2 and 3.3 to an on-site ceremony from chapter 3.4. Let us consider the different verification layers separately.

1. The Tendermint consensus protocol vs on-site ceremony. To summarise our findings above regarding the on-site ceremonies, we assume that on-site ceremonies for any system administration action (that can result in a significant security risk) can be stopped from occurring by $\lceil \frac{1}{3}n \rceil$ of witnesses, whereas it would take over $2/3$ of witnesses to make the ceremony succeed at breaking a security property. This is in some contrast to Tendermint consensus as elaborated in chapter 3.2, where some actions have similar numbers of minimum parties acting dishonestly as the on-site ceremony, whereas two types of attacks differ – a) forking using packet delays, and b) censorship/availability attacks on an already-working system. Such attacks can be achieved with less participants than with the ceremony, respectively: a) $n - 2(\lceil \frac{1}{3}n \rceil - 1)$ and b) $\lceil \frac{1}{3}n \rceil$.
2. The on-site ceremony, by default, lacks any verification comparable to that of regular full nodes. Although full node verification rests on the trustworthiness of the particular full node, and additionally the fork detection requires successfully communicating with other full nodes – many of which can be frequently offline, a

fact which can be exaggerated by attackers without much suspicion in the short run (by for example delaying network packets or flooding the gossip protocol with too many messages) – the fork detection may be a slow process, which thus may end up being a background process which may merely alert the user of a wrongdoing a long time after user has already committed to a course of action based on wrong facts (for example, a user selling a TV set may have given away their TV thinking that they have received a payment for it, only to have the payment later be declared suspect when their full node finally hears from the other fork). Even so, at least the crime will be eventually detected (depending on the protocol, perhaps in a matter of seconds, hours or days).

Thus, the best way to model the second layer of verification is the same: we can model it as something that will not stop an invalid transaction, but will find out about it in the near future. Assuming enough participants are trustworthy, the invalid fork may be found with relative certainty in reasonable time (this is particularly important if the number of regular nodes far outweighs the number of consensus participants). If we wish for the on-site ceremony to have a similar layer of verification, then either we would have to introduce a second service that verifies the first one in some way (perhaps using blockchain technology and/or computational integrity proofs), or alternatively, we can make the single ceremony more secure, perhaps by including more people (particularly, we could involve the people would would otherwise be running full nodes; in practice, however, we can probably achieve a good likelihood of reliability with just a few additional people). More details on how to make the ceremonies more secure are provided in section 3.7.1.

3.6 A Model for the Integrity of Humans

The following model estimates the integrity of humans. The results from this model can be used to analyse various security risks associated with a service and can be plugged into for instance the attack tree model [51, 26].

Specifically, the model from this section (and subsections below) can be used in the following different ways:

- To estimate a probability of a dishonest act originating from a particular person (inside or outside an organisation), we can use section 3.6.1 to estimate the probability that a given employee will choose to act dishonestly, depending on various parameters.
- When analysing the act of enlisting additional people in a conspiracy (for example using bribes or violence), section 3.6.2 can provide estimates of success and the likelihood that the approached person will turn to the authorities instead.

Other sections in the document offer clarification on the integrity of hardware and software. Specifically, when providing rough estimates for capabilities of attackers and their exploit tool kits, sections 3.8 and 3.7.4 may help provide additional data points, whereas section 3.7.3 provides additional justification for our additional assumptions around backdoors being unlikely to be detected.

Specific sub-sections below will contain additional descriptions on how and for what purpose the information in the given sections can be applied.

The model for estimating human behavior has been built with consultations with a criminology expert, though (obviously) it cannot take into account the specifics of all situations. Thus, the model should be taken as a starting point and a possible approximation of general human trustworthiness.

Where the difference matters, when we talk about probability of dishonest acts (definition 3.1.1), we typically talk about the likelihood of (at least one) dishonest act in a year.

As usual for security analysis, one may have to consider many different kinds of attackers: state actors, criminal organisations, lone criminals, competitors, employees, employees of tool builders, delivery personnel, etc. Various taxonomies have been proposed for the task.

Any monetary values below are represented in Euros (€) and the examples assume an

average European Union (EU) context, but the general findings should translate to many other countries naturally.

3.6.1 Probability of Dishonest Acts Of Individuals, Depending on the Odds of Getting Caught.

Probability of dishonest acts from an individual matters in two cases: first, when an attacker is trying to enlist an employee into a conspiracy, only dishonestly acting people will join the attacker willingly. Second, when we are analysing a situation where the individual is the one who is *starting* the conspiracy, we can use the probabilities in this section to calculate the likelihood that the person who is legitimately employed in a position will start that conspiracy. (This differs from the situations where we would already assume the person to be an attacker beforehand.)

In our model, the probability of an individual acting dishonestly (p) depends on the following parameters:

1. person's probability of performing the dishonest act if they assume they won't get caught (w);
2. factor f that modifies the probability of a dishonest act depending on the probability of getting caught and the attractiveness of the motivating goal;
3. person's risk tolerance modifier r which we assume to depend on their social status; this is based on the assumption that people who feel like they have more to lose tend to take less risks of getting caught.

As an aside, we note the fact that since the person's probability of acting dishonestly depends on their likelihood of getting caught, this can make it somewhat harder to work with attack trees: the overall likelihood of getting caught may not be known until the entire attack tree is fully computed; only then can the probability of a dishonest act be computed for said person (unless we assume that the person is not well informed about the situation and assumes that a different attack tree is involved).

We calculate the probability of a person acting dishonestly (p) in the following way. We first start by estimating the following values, which we then use in a formula.

First, we estimate the person's "base" probability of acting dishonestly *without* factoring in the fear of getting caught. It is assumed that the probability depends on the seriousness of the crime. The value is represented with the letter w and its values can be read from table 3.

Secondly, we proceed to look up the factor f, which takes into account the person's

Table 3. Our assumed person's "base" probability for performing a dishonest act if they do not fear getting caught. We assume the probabilities differ depending on the seriousness of a crime.

Crime	Probability without getting caught (w)
White-collar crime	0.8
Threatening with violence to hide a crime	0.1
Actually carrying out violence to hide a crime	0.01

likelihood of getting caught and the attractiveness of the goal.

We begin by calculating someone's overall odds of getting caught during the attack, as they understand their own odds. The attackers are assumed to be rational, but some attackers may not be given enough information by others, misleading them for example into thinking that the odds of getting caught are less than they actually are. When calculating a person's willingness to attack, it should be calculated based on what the person thinks is true.

Thus, the odds of getting caught may include the odds that an auditing procedure will catch the crime, or perhaps the tax office will start a successful inquiry, or potentially another person being proposed to will report the whole conspiracy to authorities. This overall chance of getting caught depends on the use-case at hand and will have to be estimated on a case-by-case basis, for example using attack trees (which may mean that the attack trees will have to be computed partially out-of-order: the likelihoods of getting caught specified on the upper nodes of an attack tree have to be finished before the probability of participation can be calculated for lower nodes that specify the joining of the participants).

Once we know the person's likelihood of getting caught as the person estimates it (q), we can estimate their factor f for probability of a dishonest act by reading the value from table 4. The factor f depends on three factors: probability of getting caught, seriousness of the punishment and the attractiveness of the prize for a successful attack, represented as a monetary value in Euros in the year 2020.

Our assumptions represent a typical European Union citizen with a monthly net income of € 2 000 in 2020. For people with different income levels, the attractiveness of the monetary values will have to be adjusted accordingly; a linear adjustment is suggested as a first approximation: person A whose income is thousand times larger than that of person B would find a billion Euros as motivating as person B would find million Euros. In other words, the prize amounts listed in the table header should be divided by a person's income divided by 2000.

Additional adjustments may be necessary for people whose income level is not representative of their overall wealth level. Precise mechanics of such adjustments are out of scope for the current work, but one suggestion for consideration could be to compare a person's overall wealth to that of a typical EU citizen which the table represents. (For instance, we might assume that the net wealth of a typical adult in the EU is around 125 000 Euros and calculate analogously to the income calculation.)

The table lists values for factor f for specific probabilities and prize amounts in Euros. To calculate f for probabilities or prize amounts that are between the values listed in the table, one could use interpolation – for instance, one can interpolate between the monetary amounts and the corresponding value of f. It is out of scope for the current work to determine the most accurate ways of interpolation, but as a first approximation, standard linear interpolation techniques can be used, as shown for example in the following equation (f_0 and f_1 represent the values of f from the table, m_0 and m_1 represent corresponding money prize amounts from the table):

$$f = \frac{f_0(m_1 - m) + f_1(m - m_0)}{m_1 - m_0}.$$

We do not specify an analogous method for extrapolating to values outside the bounds listed in the table.

This table (like other tables about human behavior in this work) was created using consultations with a criminology expert and its estimations take into account the fact that people fear criminal charges but at the same time a huge sum of money may overwhelm their otherwise rational thinking. It also takes into account that there are a certain amount of people who do not necessarily act according to realistic likelihoods of getting caught.

Crimes of passion are excluded from the table, as they tend to generally last for a shorter amount of time and typically do not last long enough to create a conspiracy or a sophisticated attack. However, there exist also attacks that do not require sophisticated planning, and studying them could be part of a future study. For now, we may propose that a probability for passion crimes per year per attacking person might be on order of $p = 0.001$.

Thirdly, we must further take into account that people with higher social status have more to lose, and are thus less willing to take the risk of getting caught; for some of them, a chance of getting convicted at all might feel almost as bad as going to jail. We will use the

Table 4. Our assumed values for factor f, to be used in equation 3.1. Factor f depends on three variables: the likelihood of getting caught (q, on the vertical axis), seriousness of the punishment (also on the vertical axis) and the attractiveness of the prize for a successful attack (on the horizontal axis). The table represents a typical European Union citizen with a monthly net income of € 2 000. For people with different income levels or whose wealth is not typically represented by their income level, the attractiveness of the monetary values will have to be adjusted as discussed in section 3.6.1. Finally, as people generally do not end up in jail for 100 Euros in the European Union, we have used the phrase "N/A" ("Not Available") for such combinations.

Type of a criminal act	Probability of getting caught (q)	Prob. of risking for € 100 (f)	Prob. of risking for € 10 000 (f)	Prob. of risking for € 1 000 000 (f)	Prob. of risking for € 1 000 000 000 (f)
No jail prospects	0	0.9	0.95	0.99	0.999
	0.001	0.01	0.05	0.5	0.9
	0.01	0.005	0.01	0.1	0.5
	0.1	0.0002	0.0005	0.001	0.005
	0.5 ... 0.9	0.0001	0.00012	0.0002	0.0005
Jail prospects	0	N/A	0.95	0.99	0.999
	0.001	N/A	0.01	0.05	0.9
	0.01	N/A	0.004	0.02	0.1
	0.1	N/A	0.001	0.002	0.001
	0.5 ... 0.9	N/A	0.00012	0.0002	0.0005

Table 5. Person's risk modifier r that represents a change to their probability of taking a risk depending on their social status.

Social status	Status-based risk modifier (r)
Top executives, high politicians	$1/3$
Experienced skilled professionals	$1/2$
Neutral	1
People with "nothing to lose", or in some cases, people who strongly fear or have already lost "everything".	10

values from table 5 to get a risk modifier corresponding to the person.

Finally, using the numbers from above we can calculate a person's likelihood of a dishonest act can then be calculated as follows:

$$p = w * [1 - (1 - f)^r]. \tag{3.1}$$

We can simplify the above formula for small values of f in the following way: $p = wfr$. Particularly, when rounding to a single digit, values $f \leq 0.1$ already work reasonably well with the simplified formula. The simplified formula is especially useful for quickly mentally estimating the probability of a dishonest act: one simply has to multiply the three values looked up from tables 3, 4 and 5.

3.6.2 Likelihoods of Success and Getting Caught When Enlisting New People

There are several ways people can enlist others into the attack conspiracy. This chapter calculates not only the probability of success for each act of enlistment (p), but also the probability of co-conspirators telling authorities about it (we represent that probability with q). Thus, from this section we get a tuple of values (p, q) that we can use for further analysis of attacks.

The importance of probability of reporting to authorities (q) depends on the case at hand. For some attacks, authorities finding out about the attack means the attack has failed (for instance, once a conspiracy is found, a security ceremony may be cancelled before its results are ever used for anything important); for such attacks, probability q signifies the probability of failure for the attack. For other attacks, notifying authorities is not even a major concern for attackers: perhaps they have stolen something and will escape to a foreign country with the item, and the authorities will not be able to get to the attacker.

In the following we list different ways to enlist people into conspiracies, each with their own sets of likelihood of success (p) but also a likelihood of being reported to authorities (q).

Proposing to or bribing other people.

This comes with a risk that other people will give up the proposer to the authorities (even if they initially appear to agree to the conspiracy). This probability of getting caught can depend greatly on how close the proposer and the proposed are to each other: close family members are far less likely to give up the other person, compared to people who do not know each other. Thus, smart attackers will tend to approach new people through friends as much as possible; under the rational attacker model (that is, an attacker that maximises their likely outcome) one would generally assume that at each step, a new person is approached by someone who is the closest to them among the people who already are part of the conspiracy.

We can use table 6 to estimate the probability of getting caught q when proposing to a person, depending on their level of closeness to the proposer. For instance, if a conspiracy will require 3 people, 2 of which are friends and another is a very close colleague, the probability of getting caught for all people in the conspiracy via these mutual proposals is $q_{\text{combined}} = 1 - (1 - 0.01) * (1 - 0.01) * (1 - 0.1) = 0.11791$. Proposing to each separate

Table 6. Likelihood of getting reported to authorities when proposing (q), depending on the closeness of the person being proposed to.

Closeness	Likelihood of getting caught when proposing (q)
Close family, the closest friend	0.001
Friends	0.01
Very close colleagues	0.1
A colleague, frequently met	0.9
Unknown auditor	0.999

person could be a separate node in an attack tree, for example; in that case, the aggregation of such values would be left to the mechanics of the attack tree construction.

We assume the relationships to be symmetric: that is, if person A considers person B to be a friend then person B will similarly consider person A to be a friend. In principle, this model could be extended to non-symmetric understanding of relationships as well, where every person would have a different perspective on how likely they are going to get caught, and would act accordingly.

Thus, in general, we can say that the likelihood of getting caught is multiplied by the probability q from the said table.

The likelihood of success p depends on two things: 1) whether the invited party is also willing to act dishonestly – specifics for calculating that are specified in section 3.6.1, and 2) the likelihood that the inviting party and invitee both trust each other not to give them up to authorities. This second part is indirectly already included in the calculations in the first part (that is, it is already included in the calculations in section 3.6.1) by the fact that the probability of getting caught increases during each such proposal (generally, the overall probability of getting caught needs to be multiplied by the value q from above), and the invited party's probability of acting dishonesty directly takes into account the scheme's overall probability of getting caught.

We assume that, unless the situation dictates otherwise, the proposals to different people can be considered independent probabilities, and their likelihoods of success can thus be multiplied as usual.

Managers can replace and issue orders to teams, including the chain of command all the way up to governments.

We assume that managers may have up to 100% control over people working under them as they often have the ability to replace the teams (unless they are too far in the chain of command, e.g. typically, democratic governments cannot arbitrarily replace teams in firms at will). As an example, the managers may replace teams under the guise of using a team they really trust; other people do not necessarily need to fully agree with the decision in order to go along with it eventually.

An important aspect for some systems can be the independence of countries from each-other's influence. For the purposes of national security, we may assume that even other honest people may "misbehave" against other countries, as ordered by their respective governments. (A related idea is that the meaning of honesty would always have to be compared to a local social context.) Thus, for some international systems, witnesses or validators may need to be present from multiple countries, and the trustworthiness of underlying hardware (and software) may also need to be considered from the same perspective.

If the managers wish to replace teams with malicious participants, they may nonetheless be limited by how many malicious well-trained experts they can find and dare to make contact with. Most well-trained experts already expect to have a reasonably high status in society and are thus far from being the people who would be desperate enough to risk loss of reputation or criminal charges. Further, even if the experts are indeed criminal, the managers may not risk proposing to them, as we saw in the previous point above. The table 7 contains sample guesses about available well-trained malicious IT professionals for attackers of different sizes.

Perhaps the most general defence against malicious managers could be independent auditing of procedures and hiring decisions, which could even be ordered by the shareholders of a company, without necessarily waiting for approvals by managers.

In many cases we might be able to make the simplification that managers higher up in the ladder are less likely to attack due to more social status to lose, as well as due to having more power and money, which makes them harder to coerce; if that is the case, then as a first approximation, we may not need to always analyse the risks regarding managers higher than the ones that are in the lowest ranks of these managers who are already capable of carrying out the attack. Obviously, the details vary on the case at hand.

At the same time, a manager's ability to issue orders generally does not extend to breaking the law; at that point, the manager should be considered a mere "proposer", albeit potentially a close one. The exact specifics would have to be judged on a case-by-case

basis.

Threats of violence.

Violence can be a very effective tool of coercion. Calculating the probability of performing or threatening with violent acts was discussed in section 3.6.1. The odds of getting caught, however, can differ. We propose the following crude estimates:

- For regular, white-collar extortion, the likelihood of a single victim turning to authorities may be $q = 0.5$.
- For more violent cases with threats of retaliation, perhaps the likelihood of turning to authorities can be $q = 0.1$ per victim.

Note that we are explicitly *not* considering large organised crime units here who may have credible means of threatening a large number of people at the same time, and who can carry out retaliatory acts long past some of their members have gotten caught. Information on such crime is partially classified and it is the work of governments to keep major threats under reasonable control. One potential defense measure for organisations could be to hide the identities of key personnel, although that might not always be enough against a large blanket-threat.

We are also excluding crimes of passion, as we assume that the attackers would very likely calm down before they could finalise a complex scheme involving multiple people. It is also assumed that such attackers do not necessarily behave according to the rational attacker model. However, such attacks cannot completely be ruled out in every situation; critical resources may need some protection against any single person's sudden misbehavior regardless.

Infiltration using legitimate channels.

We assume it to be relatively unlikely that an attacker can get a spy employed in a position that requires well-trained skills from the employees, for the following reasons. First, there are not a lot of such people who are willing to be spies and at the same time also would be able to have a well-trained career at another job. Secondly, one does not always know the full evaluation criteria for each job, the employers may decide partially based on their instincts or personal preferences. We provide some crude estimates for how many such spies might be employed by attackers with different resources in table 7.

For each such spy, the probability of getting employed in a specific place might be for

Table 7. Somewhat arbitrarily assumed number of well-trained IT professionals who also double as spies willing to infiltrate other organisations, depending on the size of the attacking organisation. The underlying assumption is that most well-trained professionals are not willing to accept the risks of being a spy, whereas spies in turn do not generally seek a second career at which they would be well-trained. Note that for smaller crimes – especially non-criminal offenses – the number of willing participants is likely much higher among all populations.

Attacker's size	Assumed well-trained IT professionals who double as spies, ready to infiltrate others
Nation state	100
Large organised crime unit	10
A high-ranking manager	1

example $p = 0.05$. For positions requiring less skill, the probability of employment could be for example $p = 0.5$ or more. However, the numbers will have to depend on the carefulness of the employer's background checks as well as the willingness of the attacker to prepare very thoroughly.

In the special case of using local people from a country by the hands of a foreign country in order to get employed at skilled positions, we may assume intuitively the likelihood of success to be $p < 0.001$.

The likelihood of getting caught will have to depend on the actions of the spy, which can be analysed separately; we assumed that the likelihood of getting caught during the hiring process (for example by background checks) can be quite low ($q < 0.01$, unless, obviously, there *is* something suspicious about the person's past that will fail to go through the background checks, like an existence of a criminal history). However, it the spy may be more likely to get caught later by their actions, depending on the situation and form of an attack.

Blackmail.

Blackmail is out of scope for our model. We offer no specific insight into blackmail, apart from saying that while blackmail can be a very effective tool, it may be less likely to work for more serious offenses; it can also be harder to use in a foreign country, where it can be harder to collect incriminating data about someone.

Additional notes.

Regardless of the attack at hand, we make the following additional observations:

- Co-conspirators may always have a change of heart and can give themselves up on purpose. We assume this to be happen at least with a probability ranging from of $q' = \frac{1}{4000}$ to $q' = \frac{1}{200000}$ per person per year involved with a conspiracy (depending on the ethics and incentives involved). That is, for a conspiracy of n people, the odds of at least one participant giving up the conspiracy in a year can be approximated in the following way: $q = 1 - (1 - q')^n$. For instance, with 2773 people, using the range of values for q' above, the odds of a participant speaking up in a year would roughly be in the range $0.01...0.5$. However, it should be noted that in many situations people can be more forthcoming, and in some, less so. Actual values will likely depend on many details, including ethics, legal, violent and monetary incentives. In practice, one may often skip this probability without noticeable loss of precision, as the other probabilities tend to overshadow it in magnitude. This value becomes more important when one is discussing conspiracies where the participants would otherwise assume that there is no chance of getting caught whatsoever. The range mentioned above is approximated from the data sets in [52]; the smaller value from that paper corresponds to the conspiracy over the NSA PRISM project, whereas the bigger value corresponds to the FBI forensics scandal that became public in 1998.
- As a reminder, when building for example attack trees, any other probabilities for getting caught need to be added as well (for example, an external monitoring service that occasionally sounds an alarm could be added as a factor in getting caught).

3.7 Applications of the Model for Centralised and Decentralised Architectures

3.7.1 Comments on The Trust Differences Between On-Site Ceremonies and Blockchain Consensus

In section 3.5 we discussed the security differences between blockchains and on-site ceremonies in terms of number of validators and witnesses required to achieve the same probability of honest behavior, given a fixed honesty probability for all members. In this section, we will build on that by commenting on how trust properties differ between the architectures by taking into account our new model on human and tool trustworthiness.

- First, in terms of probability of individuals acting dishonestly – section 3.6.1 – there is no difference between blockchains and on-site ceremonies. Indeed, if we consider the contents of the formula, none of the involved variables appear to be directly affected by this choice: the nature of the act remains the same whether or not we are using a blockchain; the likelihood of getting caught may differ, but only in ways we calculated in section 3.5 – that is, *the validation by regular full nodes is not performed in a ceremony, which may necessitate involving some more people as witnesses instead.* Also, the attractiveness of the goal does not likely change, neither does the person's social status.
- Second, an important difference with the on-site ceremony is that people will have to meet with each-other. This has the following effects for considerations from section 3.6.2:
  - When bribing or proposing to people, people who know one-another are more likely to conspire. It appears unlikely that meeting for a few times will make people friends with each-other, but to compensate for this effect, either *1) the team composition should be changed regularly (say, after a few months), or 2) the team members should not interact unless absolutely necessary (it might be possible to darken the room and specify them not to interact before or after meetings); a more firm alternative would be remote participation via cameras and/or other necessary sensors without the remote participants communicating to other participants; it could also be possible to watch a live-stream on a video streaming service or even watch the video later, as appropriate.* (The remote participants could even ask their own trusted parties to install their own cameras/sensors before the meeting and leave.)
  - Managers can still order their teams and change team contents in both cases.
  - Depending on the set-up, violence could still require threatening the same

number of people in both cases; however, an advantage of blockchain consensus would be that remote participants are harder to intimidate and locate. *This suggests that for parity, at least one (ideally more) on-site ceremony participants could also be joining remotely (or in some other way hide their identity well).* If their organisation also keeps their identity secret, the attackers would need to threaten additional people to find out the identities of the remote participants, which increases their chances of getting caught (potentially even getting caught before the attack succeeds). For the worst attackers, one could even imagine a corresponding defense protocol where even the managers do not know who among their teams participate remotely in a ceremony (and the same for blockchain validators). As always, one may also need to assume certain physical defenses to make sure that e.g. flying a mini-drone inside the organisation won't make it possible to easily see who the participant is (again, presumably the same with and without blockchains). Another obvious solution against some types of violence is the use of one or more security guards and various security devices. Even though violence can be relatively rare, when used it can be highly effective, at least in the short term; the more critical services often like to plan for such potential eventualities as well.
  - Infiltration using legitimate channels is not affected.
  - For blackmail attacks, there may be similar advantages to hiding identities as there are for violent attacks.

We also note that even if there is "feature-parity" between blockchains and remotely participated on-site ceremonies in terms of witnesses being the equivalents of validators, there can still be at least two differences: trusting tools (hardware and software) and cost differences. Trusting tools is briefly discussed in section 3.8. While comparing the costs of each solution is out of scope for the current work, we can offer some ideas for future study:

- Additional costs of on-site ceremonies.
  - The ceremony participants may need to work slower compared to regular computer administrators and may need a larger area for administrative tasks (perhaps several times slower and/or larger) owing to the fact that their actions will have to be done carefully so that witnesses can observe there is no foul play, particularly when handling computer hardware.
  - Costs of participation. For remote participants, camera(s) and other sensor(s) need to be bought, installed and maintained. (In the simplest case, we might imagine multiple smartphones plugged into wall sockets, communicating over Wi-Fi, utilising both their cameras as well as other sensors like depth sensors, perhaps Bluetooth for authenticating tamper-proof boxes and so forth.) For

local participants, additional travelling costs may need to be paid.
  - Physical defenses may or may not need to be more expensive for on-site ceremonies.
- Additional costs of blockchains.
  - The blockchain may need more computing power due to multiple computers re-verifying the same computations. Privacy techniques like secure multi-party computation (MPC) and zero-knowledge proofs can slow down the system further.
  - Blockchains may have a higher development cost due to more complicated protocols and less mature tooling. (On the flip side, the ceremonies also need tools and procedures to be developed and verified.)

3.7.2 Phishing and Other Cyberattacks.

Quoting from [53]: “Phishing is a form of social engineering in which an attacker attempts to fraudulently acquire sensitive information from a victim by impersonating a trustworthy third party. Phishing attacks today typically employ generalized “lures.” For instance, a phisher misrepresenting himself as a large banking corporation or popular on-line auction site will have a reasonable yield, despite knowing little to nothing about the recipient.”

Some phishing attempts direct users to malicious websites that use vulnerabilities in the users’ machines for hacking into them (for one example, refer to the High Roller attack in [21]). Other attacks may merely ask the users to log into a fake website instead of a real one in order to steal user credentials and/or their browser sessions.

The likelihood of success in impersonation attempts depends greatly on the type of an attack, preparedness of the attacker and whether the defenders are willing to let go of modern conveniences like clicking on links in e-mails (or at least willing to carefully inspect the links every single time), among other factors.

For higher-quality phishing attempts that are targeted towards specific victims (often called *spear phishing*), we propose the following rough likelihoods of success depending on the sophistication of the defenders: for average users, $p = 0.1$; for specifically trained IT professionals, $p = 0.001$; for environments like banks where people go to additional lengths including what some would call "draconian measures", $p = 0.0001$.

The likelihood of getting caught with a simple phishing attack or any other cyberattack can be low, especially if the attackers hide their tracks well and do not arose suspicion that would authorise a major search for them. The attackers may also live in different countries from their targets, which may require international cooperation to locate them.

3.7.3 Detecting Backdoors, Steganographic Data Infiltration and Exfiltration

The assumptions in this chapter are not strictly proven, but informal reasoning is presented to justify them as good candidates to represent the real world. The chapter starts with a more general discussion and then proceeds to describe specific assumptions.

When comparing centralised services to decentralised ones, one of the common questions that comes up is as follows: is a computer in one’s own possession more trustworthy than a

computer in the possession of a cloud company (which the user does not own)? There are of course questions of competence (regular users may not be as competent at cybersecurity as major cloud vendors), laws (the user may be in a different legal jurisdiction or simply different laws can apply for your own data that is in your own computer), etc.

However, this chapter focuses on the question of honesty. More specifically, a potential trust problem with the cloud is that the cloud operators can administer their computers at will, without the user directly noticing much of their misbehavior. As an example, the cloud owner could inspect the filesystem and processes running on their systems, modify them, use backdoored hardware and operating systems, and so forth. Nonetheless, our question must be whether that will make a real difference in practice: after all, the user did not build her own computer and thus her own computer could also have various backdoors inserted by manufacturers, potentially using steganographic covert communications channels to administer the user's computer beyond her ability to detect? If so, the situation would be equivalent between the external cloud machine and her own computer: in both cases there would be a trusted entity that she would have to simply trust not to abuse their (potential) administrative access.

To begin answering that question, let us look at the administrative powers that cloud operators have – regarding physical and remote access separately – and for each, compare the differences between a user's own computer and a computer in the cloud not operated by the user herself.

First, cloud employees can potentially tamper with hardware physically, even replace it with other hardware (temporarily or permanently), without the user directly seeing it. In a way, this seems somewhat analogous to what hardware manufacturers can do – they can ship their hardware with backdoors embedded in the factory, and thus have full control over what the hardware does; if they ever need to update the backdoor in hardware, they have the ability to authorise corresponding firmware updates (often via legitimate channels). Thus, hardware manufacturers do not necessarily need to physically tamper with user hardware after the time of purchase in order to insert a backdoor, but they may need to do more work than the cloud administrators (as presumably, the covert backdoors need to be built, whereas the honest administrative consoles already exist) and there exists at least a small probability that in the future someone will be able to find such backdoors in hardware (either accidentally or using systematic search; perhaps in a more distant future, there will be people scanning hardware under microscopes using futuristic AI-based backdoor scanners).

Assumption. *Even though in a cloud, it is theoretically also possible to get caught by*

an auditor, if the attackers working for the cloud company are confident they can escape auditors (perhaps a colleague will warn them in advance), they may assume their likelihood of getting caught is smaller.

Further, it is not enough to simply add backdoors: some attacks may be practically impossible to perform without the ability to communicate with the system under attack (for reconnaissance or other reasons), even if the attacker's goal is not data exfiltration *per se*; for instance, if the attacker cannot simply guess a configuration file format, she may not be able to configure a remote application to do her bidding; in that case, she might want to see a sample configuration or be able to download and reverse-engineer the configuration parsing software itself.

Thus, our second point is that cloud employees also have remote administrative access on the cloud computers: they can read, write and modify any data on the computer. Technically, this can also be achieved on a user's own computer using a backdoor with steganographic communication channels to keep the network traffic covert. If effectively used, it may allow users' own computers to be administered remotely by the attackers in a similar manner to a cloud, albeit perhaps at significantly slower speeds (steganographic channels may be slower to stay maximally covert, both in terms of bandwidth and latency). Presumably again such backdoors need to be built (if they do not exist yet), which requires additional work and dedication.

Steganography is a common method of covertly infiltrating and exfiltrating data. It appears to be agreed that clever use of steganography can be difficult to detect. For an example, Dr. Simon R Wiseman writes the following in A Defenders' Guide to Steganography [54]:

> Only in limited cases can Discoverable Steganography be discovered with any degree of certainty, whereas by definition Undetectable Steganography cannot be discovered. Thus, the strategy of detecting messages that carry a hidden message in order to block their use in a cyber-attack does not work. Attackers generally find ways of evading attempts to discover their activities, and this will surely be the case with detecting steganography.

Since detection can be very hard, the same paper also suggests a better strategy of dealing with steganography – transforming and normalising data in order to reduce the attack surface for steganography [54]:

> "The only effective way of destroying hidden messages is to take the informa-

> tion that is found in the carrier data and build entirely new normalised data to carry it forward. This is the strategy behind Content Threat Removal [ctr]. It transforms the way information is carried, converting potentially unsafe data to known clean safe data.

Essentially, what such content threat removal methods accomplish is reducing the degrees of freedom available to a user of a system, thus making it harder to effectively embed large amounts of covert data. However, even such methods are not perfect, as practical systems often must allow their users at least some degrees of freedom. For example, a Bitcoin user must be allowed to choose how much money they send to whom and when. Such data could already be used to encode hidden information. And it does not help that such data must also be used very precisely (users won't accept a money amount that is only roughly correct), and in a format that the known Bitcoin client code will be able to understand (that is, humans cannot be in the loop of verifying the details of the Bitcoin blockchain due to efficiency reasons); this also makes it easier for a backdoor to recognise the data steganographically. Such covert channels may even work if the service is technically hidden from the rest of the world (no access is allowed from the outside networks) as some data would often still go in, and some data would still come out (for instance, an internal payment gateway in the bank will still see the payment amounts and account numbers, and will be able to issue errors for the payments which the attackers may be able to see, at least to an extent).

Further, for example Bitcoin explicitly allows adding encrypted custom data and data hashes to its blockchain; as such data is expected to be computationally indistinguishable from random numbers, it can be a very efficient location for encoding steganographic content: the attacker can simply encrypt covert data and it will look indistinguishable from regular content. Similarly, many web services allow clients to send custom data like messages to customer service, comment fields, profile images, etc, and it is expected that the service does not change the data significantly. (Although, it is useful that if there is a service that doesn't need to process some data – it just needs to pass it through – then the data could be encrypted in transit by the data sender itself, only to be decrypted at the last possible moment; this can reduce the attack surface of steganography and can sometimes be applied to data stored in databases and even web services.)

As alluded to above, a problem for steganography can be the low data rate; establishing an exact data rate for each situation is out of scope for this work. However, a potential help for attackers might be that network traffic numbers will probably go up in the future, not down, and as such, if users send more data to services, there may also be more potential for covert data along with it.

There is, however, one generic class of situations where it may be possible to almost completely remove any degrees of freedom: one can make all of their computations deterministic (including timing and random numbers), and then log all the inputs and outputs of the system (for example, this logging action could be done using another device which is less likely to be compromised at the same time); this allows one to verify the entire computation with another computer. If either of the computers attempts to send steganographic communication on any of the main channels (as opposed to side-channels like power usage or sound), the responses from computers will not match byte-by-byte anymore and can be detected. In that case, data exfiltration attempts on the main channel can be detected on one's own system, but this does not necessarily apply to data infiltration attempts (as one cannot typically ask all of their clients to make their systems deterministic as well).

Although the above analysis is not a proof, it is hoped that it is enough to motivate the following assumptions, which will be used as a basis in the rest of this work.

Assumption. *Reasonably undetectable steganographic methods may exist and may be used by attackers so long as there is any way for attackers to send any information to the compromised system or receive any information from it, for incoming and outgoing data respectively. However, if all transmissions are monitored by a correctly functioning system, these transmissions that are deterministic – for which there can be no alternatives – (perhaps due to the fact that there is exactly one way to respond to each incoming request and there is even one specific response time), the monitoring system must necessarily be able to detect any covert alterations to the communications stream.*

For instance, even if a payments engine inside a bank is technically an internal system and is cut off from internet traffic, it probably still needs to receive the payments that customers authorise; if these payments or their metadata contain any covert commands, they could be processed and recognized by a backdoor in the payments processor. Similarly, that payments processor will need to send data out to the outside world – it may have the ability to modify payment descriptions (especially for payments that get routed to other banks); even flagging a certain set of payments to be investigated for fraud may result in effects that a customer can notice (perhaps the payments will be slower or the bank will contact the customer for clarifications). Clever attackers may invent schemes to pack surprising amounts of information into such covert channels.

On the other hand, if the above mentioned payment processor only modifies payment data deterministically – perhaps it merely converts between different payment data formats, and does so in a canonical way – and even always processes payments in exactly 10 seconds

(to eliminate time as a side-channel for communication as well), then someone (correctly) monitoring the entire communications history of the payment processor would be able to notice if steganographic communications had altered the outgoing traffic in any way; however, to do so, the monitoring party may have to perform the same computations as the payment processor itself (for example, to convert between payment data formats to verify that the payment processor did the same correctly); this has many similarities to a Byzantine fault-tolerant consensus, with an important exception that the parties receiving payments do not necessarily know about the result of consensus (they get a response from just one machine).

It should be noted that a simple method exists that can help against *targeted* backdoors that are installed only for specific users: a buyer can go to an unexpected store or warehouse, perhaps at an unexpected time, and perhaps purchase a random (unpredictable) instance among the products being sold. For software, it can be even simpler to merely compare the cryptographic hash of a downloaded binary against similar hashes from other people. This would remove the attacker's ability to aim precisely: their remaining options would be 1) aiming "blindly" and often missing their target, 2) stopping the attack, or 3) backdoor a large percentage of their products to make sure that a randomly selected one would end up going to the right place, which in turn can make it more likely for the backdoor to be discovered (even if the discovery takes a long time).

3.7.4 Software and Hardware Backdoors

It may be possible to predict the number of malicious backdoors in libraries and other third-party software, hardware and tools using estimates from section 3.6.1. We might decide for example that roughly one software developer in 10000 would be willing to insert a backdoor into the code for a given purpose, and then we could take into account the review processes in the company (that is, how many people would review the code) – so long as there is at least one reviewer then the probability of dishonest acts per the two people would be the probability for one person squared; that number could then be "multiplied" by the number of developers using the following formula: $1 - [1 - (10^{-4})^2]^n$.

Even with 10 000 developers this would then give the likelihood of backdoors as roughly 10^{-4}. However, in practice, on many less mission-critical teams it may be possible for developers to fake a code review (perhaps they lie to other team members that they already did a code review on their own machine with another person). In that case, the real number of developers who can attack may be 1, not 2, and with 10 000 developers the likelihood that someone would be willing to attack would then be around 0.63 per year.

The precise mechanics of this will have to be reserved for future studies, as the results may technically depend on the number of developers who have ever existed in the world and who have built different tools that in turn have built other tools and so forth; for instance, in principle, it is possible to have a self-replicating backdoor in a compiler that is not even present in any source code anywhere [55]. At the same time, the attacks get harder the longer they have to stay covert; some backdoors may need code updates as well over time, to adjust for changing unpredictable circumstances.

3.8 Statistics on Software and Hardware Security

We attempt to roughly estimate the number of exploitable 0-day vulnerabilities in common software and hardware by analysing known history of 0-day vulnerabilities in a few representative components that we assume are most dangerously exposed to network attackers: for software, we include web servers (Microsoft IIS and Oracle WebLogic), web browsers (Google Chrome, Internet Explorer and Mozilla Firefox) and operating systems (Microsoft Windows, Linux, Apple MacOS, Apple iOS); for hardware we include the Intel CPUs and their chipsets as a representative. This analysis is not meant to be exhaustive, but rather a starting point for further study in the future.

The data sets in this section lack strong empirical validation and should be treated with care. For example, we did not establish any error bars, we did not analyse how many secret 0-day vulnerabilities as estimated to exist that are not publicly known, and how different vulnerabilities can be used together to create more powerful combined attacks. Rather we focused on the simple case of remote code execution (RCE) vulnerabilities as these are the most critical ones to defend against.

For the above software components we took a look at the known and exploited 0-day vulnerabilities between the years 2015 and 2020 in The Zero-day Vulnerability Database [56] (refer to appendix on page 80 for more details). For vulnerabilities in the Intel CPUs and their chipsets in the same time frame we used the CVE Details database [57].

The Zero-day Vulnerability Database only lists "0-day" software vulnerabilities that are known to have been actively exploited "in the wild". Thus, the actual number of vulnerabilities being exploited is even larger and this provides a conservative estimate.

The time period was chosen to be 5 years, 2015 - 2020. In appendix on page 80 we list only the remote code exploit vulnerabilities (that is, these that allow remote attackers to directly execute arbitrary code on the target machines), but the complete number of 0-days (including all types of 0-days) is also provided for each software.

No backdoors were found in Linux, Windows, MacOS, iOS, Android and popular web browsers and web servers (except for third party browser plug-ins).

Counting the above-mentioned remote code execution vulnerabilities, we can see from the dataset that:

- The application servers in the list appear to have roughly 1 remote code execution

vulnerability per 5 years, or about 0.2 per year.

- Although the operating systems of Windows, Linux, maxOS and iOS have $2 + 1 + 0 + 2 = 5$ combined remote code exploits in 5 years, none of these appear to be readily exploitable from a web server, rather from a client's web browser. Thus, for client-side exploits we might estimate $(2 + 1 + 0 + 2)/4/5 \approx 0.3$ per year per operating system, whereas for server-side, we hesitate to use the number 0 (especially given our knowledge of earlier exploits like Shellshock [58] and Heartbleed [59]) and rather estimate 1 per 10 years per operating system, or 0.1.
- For web non-mobile browsers, we get an average of $(4 + 10 + 5)/3/5 \approx 1$ remote code exploits per browser per year.

When building applications, one must also consider any vulnerabilities in our own application code, as well as various other tools. For reference, Bitcoin and Ethereum are generally considered relatively secure blockchain platforms; we did not find any comparable 0-day vulnerabilities for either. Here is what we found when we searched for 0-days in Bitcoin and Ethereum:

- Using different databases we found at least one serious vulnerability for Bitcoin between the years 2015 and 2020 – a dangerous inflation bug was discovered in 2018 which allowed breaking a strong property of Bitcoin – namely, allowing Bitcoins to be created out of thin air [60]. Technically, this does not meet the same criteria for a 0-day which is actively exploited, as there is no evidence that it was ever exploited by attackers. Nonetheless, we have included it here for reference.
- We also did not find similar 0-days for Ethereum's main client (Geth) in the specified timeline.

For our representative hardware – the Intels CPU and their chipsets – we found from [57] only 1 confirmed remote code exploit 0-day vulnerability between 2015 and 2020: identified as CVE-2017-5689, its description starts in the following way [57]: "An unprivileged network attacker could gain system privileges to provisioned Intel manageability SKUs: Intel Active Management Technology (AMT) and Intel Standard Manageability (ISM)." Thus, the remote attacker could completely overtake the target system. However, this vulnerability required Intel Management Technology to be explicitly turned on for it to be exploitable. We make the somewhat arbitrary assumption that on a critical server, all components have been turned off as much as possible for security reasons, which then leads us to the figure 0 exploits; even looking further back to 2009 in the same database, we fail to find other similarly critical vulnerabilities. Thus, we will, as a first approximation, use the number 0 to represent such CPU vulnerabilities.

It also matters how widely known and exploited a 0-day vulnerability is. Bilge et al. wrote in 2012 that “Zero-day attacks last between 19 days and 30 months, with a median of 8 months and an average of approximately 10 months.” [61]. Loosely based on the 10 months figure and acknowledging the increasing imprecision in our estimates, we could make the simplification of assuming that a zero-day will stay exploitable for a whole year. However, discussions with a security expert lead us to believe that "sophisticated" entities like nation states and expert security research firms tend to know about exploitable 0-days for longer than a year. The expert was not able to claim any specific numbers. Therefore, we assume the following instead, subject to further study in the future:

- "Sophisticated" parties (including nation states and expert security research firms) know of a 0-day for an average of 5 years.
- "Non-sophisticated" parties occasionally also get access to advanced 0-day exploits, but we assume that to be 10 times less likely and their knowledge of the 0-day could last for about 2 weeks before the vulnerability is patched in up to date systems (admittedly, leaving some systems behind that have not been updated on time).

(As an aside on terminology, in security literature it is common to call someone an "attacker" even if they are performing the said "attack" for legitimate purposes such as law enforcement and intelligence gathering; this does not in any way pass judgement on morality of the actions, which can be subject for separate study.)

Our results are summarised in table 8 where we also have calculated the likelihood that the vulnerability happens to be exploitable at least once during a year for both the "sophisticated" and "non-sophisticated" parties (we multiplied the frequencies for sophisticated entities by 5 and divided them for non-sophisticated entities by 10; the probability calculations for turning frequencies to probabilities were done using the binomial distribution and taking each day as a separate event).

From the data in table 8 we can estimate the likelihood that clients and servers have 0-day remote code execution vulnerabilities in them in a given year. Then, we can perform a second calculation assuming the use of Byzantine fault-tolerant consensus between multiple computers to see how the reliability would differ.

For both server and client we will perform two sets of calculations. First we will calculate assuming the "sophisticated" attackers who may know about a 0-day exploit for a year before it gets patched. Second we calculate assuming the attackers know about the attacks for two weeks before they get fixed by the vendors.

Table 8. The numbers listed are all computed as estimates for the frequency and probability of 0-day vulnerabilities per year that allow remote attackers to execute arbitrary code on attacked machines. The corresponding probabilities are calculated using a binomial distribution to reflect the probability that there is at least one vulnerability in a year. Column f_0 represents the number of such vulnerabilities discovered per year, p_0 is the corresponding probability per year; f_s represents an estimate for a number of such vulnerabilities known by "sophisticated" parties per year, p_s is the corresponding probability that they known at least one per year; f_n is the same frequency for "non-sophisticated" parties and p_n their corresponding probability for at least one per year.

Component	f_0	p_0	f_s	p_s	f_n	p_n
CPU and chipset	0	0	0	0	0	0
Application servers	0.2	0.18	1	0.63	0.02	0.02
Operating systems on servers	0.1	0.095	0.5	0.39	0.01	0.01
Operating systems on clients	0.3	0.26	1.5	0.78	0.03	0.03
Web browsers	1	0.63	5	0.99	0.1	0.095

- Server side (application server + operating system)
  - "Sophisticated" attackers who know about a 0-day potentially for a several years before it gets patched.
    - The combined probability of 0-days on hardware, application servers and operating systems per year (that is, the probability that at least one of them has a 0-day) is $1 - (1 - 0)(1 - 0.63)(1 - 0.39) \approx 0.77$.
    - The likelihood that two different server stacks have such vulnerabilities at the same time would thus be roughly $0.77^2 \approx 0.59$
    - The likelihood for three different server stacks would be $0.77^3 \approx 0.46$, or roughly once every two years.
  - "Non-sophisticated" attackers who use publicly known attacks which stay public for about 2 weeks before getting patched.
    - The likelihood of at least one attack happening in a year is analogously to the "sophisticated" case $1 - (1 - 0)(1 - 0.02)(1 - 0.01) \approx 0.03$.
    - The likelihood of two different server stacks having vulnerabilities at the same time is considerably smaller, as the two week windows may not match as likely (we divide yearly probability with 26, roughly half the number of weeks in a year): $(0.03/26)^2 = 1 \times 10^{-6}$.
    - For three server stacks, the probability is lower still: $(0.03/26)^3 = 2 \times 10^{-9}$.
- Client side (web browser + operating system)
  - "Sophisticated" attackers.
    - The likelihood that there would be at least one exploitable 0-day per year would be $1 - (1 - 0.78)(1 - 0.99) \approx 0.998(!)$.
    - For two services in consensus: $0.998^2 = 0.996$.

    - For three services in consensus: $0.998^3 = 0.994$.
  - "Non-sophisticated" attackers.
    - The likelihood that there would be at least one 0-day per year would be $1 - (1 - 0.03)(1 - 0.095) \approx 0.12$.
    - For two services in consensus: $(0.12/26)^2 \approx 2 \times 10^{-5}$.
    - For three services in consensus: $(0.12/26)^3 \approx 1 \times 10^{-7}$.

Let us analyse the results. According to our model, for “sophisticated” attackers infiltrating on the server side, subjectively, it does not seem to make a lot of difference whether one uses 1 server stack or 3 server stacks together in a BFT consensus (corresponding probabilities of knowing a 0-day vulnerability are 0.77 and 0.46). However, we do note this result is sensitive to exact numbers in an exponential way, therefore with an improved understanding of the number of 0-days discovered this number could change significantly. Subjectively, it is hard to say whether the result is correct or not.

For non-sophisticated attackers aiming for the server side, there does appear to be a large difference: they could compromise a single server stack roughly once per 30 years, whereas compromising already two stacks at the same time would be extremely unlikely, or about once in a million years. It is likely that real systems would thus be compromised using alternative means — phishing attacks, bribing, and so forth.

For client-side attacks, in our model, it is virtually impossible to defend against sophisticated attackers, whether or not someone is using multiple devices. This is largely due to the numerous 0-day vulnerabilities discovered in web browsers. An obvious solution would be to stop using the browsers or to cleverly restrict which websites are being visited. The feasibility of these and other measures will have to be weighed depending on the situation.

Once again, there is a difference for non-sophisticated attackers between how many different devices are used in parallel, although the situation is also not hopeless for a single device (an attack roughly 10 years; for two devices, it an attack would happen once every 50 000 years.

It may be wise for client users, especially with devices that connect to critical systems, to only visit minimally suspicious websites (even though that does not guarantee success: popular legitimate websites have been known to be infected by attackers occasionally); subjectively, this is good advice. However, particular attackers may still not be able to infect multiple devices of the same user – for example, the user may visit different websites on the phone and the computer, or the phone could have better protection mechanisms

between different applications in such a way that the same attackers might not be able to infect the entire phone at a time.

For many client computers, it is also possible that they are generally not even receiving all the latest security updates quickly, making the 0-days last even longer on these computers. The European Union Agency for Cybersecurity (ENISA) has advised in 2012: "For a bank, in the current situation it is safer to assume that all of its customers' PCs are infected – and the banks should therefore take protection measures to deal with this." They follow that up with more specific advice [62]:

> Many online banking systems, some with one-time transaction codes, calculators or smartcard readers, work based on the assumption that the customer's PC is not infected. Given the current state of PC security, this assumption is dangerous. Banks should instead assume that PCs are infected, and still take steps to protect customers from fraudulent transactions. For example, a basic two factor authentication does not prevent man-in-the-middle or man-in-the-browser attacks on transactions. Therefore, it is important to cross check with the user the value and destination of certain transactions, via a trusted channel, on a trusted device (e.g., an SMS, a telephone call, a standalone smartcard reader with screen). Even smartphones could be used here, provided smartphone security holds up.

It should be noted that according to their advice above, using 2 devices together (a PC and a mobile phone) is deemed secure enough for banking purposes. (Which in itself, lacking further clarifications, neither confirms nor refutes our model.)

3.9 Sample Analysis of a Cryptocurrency Use-Case

3.9.1 Introduction

Cryptocurrencies were the first popular use-case of blockchain technology, they are relatively easy to understand and explain as applications, and provide enough interesting functionality to make risk analysis useful in our framework.

We describe a simple permissioned cryptocurrency that has two operations. First, the owner of an account can transfer money to any other account (thus potentially creating a new account in the process) using the TRANSFER operation: TRANSFER(from,to,amount). Every such operation is digitally signed by the transaction creator. Account numbers "from" and "to" are simply public keys of the account owners. The "from" field can only be the public key of the signer. The cryptocurrency amounts are represented by 64-bit integers and there is a fixed amount of the cryptocurrency. Every account has a computed balance.

The second operation is a lottery. Everyone can send money to a specific lottery account, and once per month the system randomly picks a winner from all the people who sent the money, and the lucky winner gets everything that was sent that month. The odds of winning are weighed by how much money someone has sent. The winner is picked by a random number that, for fairness, is composed of three random numbers that the three key people (validators or witnesses) send – all numbers are hashed together with a cryptographically strong hash function and the result is taken as an input to the random number generator; a commit protocol is used so they cannot peek at the numbers of other people before sending their own.

All lottery bids are read in the order as they were received, by their weight, and added to an array data structure (the more money was sent, the more slots in the array) and the random number simply gives the index of the winning slot.

The system starts in a "genesis state" which was determined during system launch – different parties sent money to the bank accounts of system creators, and in exchange for how much money they spent, the genesis state contained the right amounts of cryptocurrency on each of their accounts, assigned to the public keys of their respective owners. We assume that this startup process was correctly executed.

For the purposes of handling user load, the system only needs one computer. However, for

availability reasons, there need to be more machines.

We analyse the attack tree of an insider attacker for each of the two configurations: 1) an on-site ceremony with three witnesses (with 3 computers for availability), compared to 2) a Tendermint permissioned blockchain with three validators (each running two Tendermint consensus nodes for better availability). Naturally, end-users of the system can also run a Tendermint regular fully validating node. Two of these people are close friends who started the company; the third one is an auditor from a well-respected auditing company. We estimate both trustworthiness and operational costs for both with and without using a blockchain.

The scenario is as follows. One of the two trusted friends working for the company really wants to steal money from the system. We consider different options for him to do so, cooperating with other people and attempting to do it alone as well.

We will calculate the odds of succeeding for such attacks; we will utilise data from sections 3.6.1 and 3.6.2, as well as many other sections throughout this work.

We assume everyone's social status modifier in table 5 is "experienced skilled professionals" and that if caught, they expect to go to jail.

We represent attack trees as nested bullet points, where each node will have its probability of success specified. For instance, a choice between two attacks where one has a chance of success 0.1 and another has a chance of success 0.5 will be represented as follows:

- "OR". p = 0.5.
  - Option 1. p = 0.5.
  - Optopn 2. p = 0.1.

Nodes that both have to be implemented are represented in the following way:

- "AND". p = 0.05.
  - Option 1. p = 0.5.
  - Optipn 2. p = 0.1.

We consider getting caught as a failure as well. Probability of getting caught is represented by the letter q and probability of success is represented by p. We do not calculate the probability of getting caught q for every node if it does not appear useful for evaluating the model in a given case.

The scenario begins with the assumption that one of the close friends has already decided to attack; he now needs to convince his friend or anyone else as necessary to get the attack implemented.

3.9.2 Case 1: Printing and Stealing Money

"Printing" money or creating an invalid transaction that pays money from "nowhere" is technically not stealing anybody else's money, but in practice, as more money is created there tends to be an inflationary effect, which essentially means that everybody else looses some value on their holdings. Technically, this would account to stealing from everybody and would thus break the social contract they established with their users in the beginning.

- "OR" $p = 5 \times 10^{-4}$
  - $p = 6.4 \times 10^{-9}$. The attacker would invite his close friend and the auditor to come and break blockchain consensus by issuing an invalid transaction that creates money "out of thin air" by sending money to their own accounts from a non-existing account. Immediately, the other blockchain nodes would notice it, thus the odds of getting caught are nearly 1, and since they can even go to jail over this action, then using using equation 3.1 for the other two actors the likelihood of attacking for both of them is $0.8 * 0.0002 * 1/2 = 8 \times 10^{-5}$, and the likelihood that they both agree to perform the attack is even smaller: $(8 \times 10^{-5})^2 = 6.4 \times 10^{-9}$.
  - "AND" $p = 0.9 * 0.5 * 0.001 \approx 5 \times 10^{-4}$.
    The attacker could insert a backdoor into the binary build of the blockchain code such that when a rich person sends money to someone, the code would actually send more money to the attacker instead, and then the attacker would also send money to the intended receiver (about which the hacked client code notifies him); the receiver might not mind which account his money came from. The rich person would not see the attack as his blockchain client would display the amount he is supposed to see. So long as he never spends all of his money he may never notice. However, the attacker may not be able to hack every node in the future; at some point the victim may be using a node that shows him his real balance.
    - $p = 0.9$. Volunteer to build a binary and insert a backdoor into the binary build to perform the attack.
    - $p = 0.5$. When the money arrives, make sure to send the right amount to the actual receiver as well, and hope the receiver does not notice the money coming from a different account and the sender does not look at his balance via another mechanism.

        * $p = 0.001$ Long-haul maintenance: build every next version of the binary as well, insert and maintain the hacked code there, hoping that the victim would keep installing that hacked version of the binary in the future and that the victim would not look at the balance using other tools.
    - "AND" $2 \times 10^{-7} * 0.46 \approx 9 \times 10^{-8}$

      The attacker could pay a huge amount of money to purchase a specialised attack kits that could break into all blockchain nodes using rare zero-day vulnerabilities of each operating system. He would compromise all blockchain nodes (both consensus machines and regular full nodes) – at which point he could install persistent malware kits to these computers and hack the blockchain software in these computers. (We note that paying this much money for the attack kits would likely be rational only if he expected to steal and use more money than the expensive attack kits were worth, which could cost many millions of Euros over the years.)
        * $p = 0.46$ Wait for a time when 3 different operating systems and web server stacks are exploitable at the same time (we assume all operating systems and web server stacks have good representation throughout the blockchain infrastructure). The probability from section 3.8 for 3 different server-side stacks being compromised at the same time is 0.46.
        * Make sure that when customers buy new computers these can be compromised as well, or otherwise some users will notice that they do not have the right amount of money. Assume this will continue for 20 years straight – $0.46^{20} \approx 2 \times 10^{-7}$.

Thus, even with the most likely attack to succeed, the attack of altering the blockchain binaries, the likelihood of success is low, even owing to the simple fact that people remember how much money the are supposed to have. Let us now compare the above to a centralised version of an architecture, the on-site security ceremony involving just three people, the two close friends and an external auditor. A difference here is that there are no outside observers who would verify the results (other than the fact that people generally remember the history of their spending and how much money they think they should have). Let us investigate whether the probabilities of success will increase for the attackers.

- "OR" $p = 0.009$
    - "AND". $p = 8 \times 10^{-5} * 8 \times 10^{-5} * 0.99 \approx 6 \times 10^{-9}, q = 1 - (1 - 0.001)(1 - 0.01)(1 - 0.999) \approx 0.999$.

      The attacker would invite his close friend and the auditor to come and break on-site ceremony's security, which requires all 3 participants to agree. The objective is to issue an invalid transaction that would create money "out of thin

air" and send that money to their own accounts. Without blockchain, outside parties do *not* notice it easily anymore.

  * $p = 4 \times 10^{-5}, q = 0.001$ The attacker would invite his close friend to participate in the attack. Using equation 3.1 we calculate the likelihood of joining in the following way, knowing from the upper leaves of the tree that the likelihood of getting caught is 0.999: $p = 0.8 * 0.0002 * 1/2 = 8 \times 10^{-5}$.
  * $p = 4 \times 10^{-5}, q = 0.999$. The attacker would invite the independent auditor to join in the attack. From equation 6 we estimate that the auditor will give them up with a likelihood of $q = 0.999$. Knowing from the upper leaves of the tree that the overall likelihood of getting caught is 0.999, thus $p = 0.8 * 0.0002 * 1/2 = 8 \times 10^{-5}$
  * $p = 0.99, q = 0.01$. The attacker(s) need to pass any audits.

- "AND". $p = 0.9 * 0.01 = 0.009$

  The attacker could insert a backdoor into the binary build of the code used in the ceremony such that when a rich person sends money to someone, the code would actually send more money to the attacker instead, as in a case before; the rich person (and everyone else) would not see anything different. The advantage here is that the attacker only has to control one computer; the shared disadvantage is that eventually a correctly working computer will come, and then the attack may become apparent, as either someone will end up with less money than they think they have or the total number of coins available will be too high, if a simple audit is launched that adds up everyone's coins.

  * $p = 0.9$. Volunteer to build a binary and insert a backdoor into the binary build to perform the attack.
  * $p = 0.01$ Long-haul maintenance: build every next version of the binary as well for 20 years, insert and maintain the hacked code there.

■ "AND". $p = 0.77 * 5 \times 10^{-3} \approx 4 \times 10^{-3}$.

The attacker could pay a lot of money to purchase a specialised attack kit that could break into the centralised ceremony computers using rare zero-day vulnerabilities. (Again we note an extreme cost of this attack as well, due to the cost of such attack kits.)

- $p = 0.77$. Wait for a time when the used operating system is exploitable, then buy the exploit kit. The probability from section 3.8 is 0.77 per year.
- $p = 5 \times 10^{-3}$. Make sure that when the on-site set-up gets new computers then these can be compromised as well, or otherwise some users will notice that they do not have the right amount of money. Assume this will continue for 20 years straight – $p = 0.77^2 0 = 5 \times 10^{-3}$.

While there is a difference of roughly 18 times of a more probable attack in the centralised case, there would also be a simple fix for this attack: if the team was using reproducible builds and if every binary was verified by more than one key participant, that would fix these attacks both for the centralised as well as the decentralised architecture.

3.9.3 Case 2: Compromising a Random-Number Generator

The above-described random number generation protocol has a weakness: not only is it easy to compromise by the colluding 3 parties, the collusion does not necessarily leave any evidence to find, other than the altered binary that other people may also stumble upon and investigate. These facts can be taken advantage of by the attacker. First, let us describe the blockchain variant.

- "AND". $p = 0.9 * 0.5 * 0.8 = 0.36$
  The attacker could single handedly hack the binary of their blockchain download, knowing that both of his two colleagues use the official blockchain client binary from the website to send their random numbers. Since the other two do not suspect anything, this will likely work well. However, as he cannot differentiate between different downloaders on the download site (he does not administer the site), his main concern could be someone else downloading the same binary and inspecting it to find the hacking code inside; also, once he wins, other people may want to investigate even harder.
  - $p = 0.9$. The attacker programs and tests the additions to send specific random numbers; since he knows the numbers in advance, he can pick his own numbers ("brute-forcing" the hash function) accordingly.
  - $p = 0.5$. The attacker volunteers to build a binary of the client, then builds it with his own additions, and uploads to the download site. We estimate a likely one-time probability of getting caught of 0.5 which comes from people disassembling the binary; thus, the odds of success of this step are $1-0.5 = 0.5$.
  - $p = 0.8$. Once the lottery is finished successfully, he has to successfully convince the tax office, his friends and a wider community that he did not steal the money. This is likely to work.

Then, the centralised variant.

- "AND". $p = 0.9 * 0.95 * 0.8 \approx 0.68$
  Here, there is no need to worry about outside downloaders, unless there is a process for involving them.

- $p = 0.9$. The attacker programs and tests the additions to send specific random numbers; since he knows the numbers in advance, he can pick his own numbers ("brute-forcing" the hash function) accordingly.
- $p = 0.95$. The attacker volunteers to build a binary of the client, then builds it with his own additions, and uploads to the download site. We estimate a likely one-time probability of getting caught of 0.05; thus, the odds of success of this step are $1 - 0.05 = 0.95$.
- $p = 0.8$. Once the lottery is finished successfully, he has to successfully convince the tax office, his friends and a wider community that he did not steal the money. This is likely to work.

Here, as with the previous use-case, the main issue was lack of a repeatable build system. With that fixed, there likely would not have been such a likely attack vector.

3.9.4 Discussion of Results

The model behaved expectedly, as the results form the initial use-case analysis seem realistic. The biggest attack vector against the system was found to be a compromised insider working on a critical task that nobody else was verifying. Getting people involved in a conspiracy appeared unlikely, which makes sense considering how high the chance of getting caught was.

Sophisticated exploit kits appeared usable as means for attacking, but sustaining the attacks over time appeared once again unlikely, which also feels realistic. Also unsurprisingly, compared to the blockchain system, the centralised system appeared less robust against exploit kits due to a single application stack being used (although we note that during a 20-year attack, the likelihood of succeeding all the way through was only $p = 0.004$). However, caution must be expressed over the capabilities of the exploit kits, as the particular 20-year formula scales exponentially, thus potentially exaggerating smaller inaccuracies in the model. Further studies are needed to determine more accurately the capabilities of such exploit kits.

4. Results

4.1 Summary of Results

The main result of this work is the proposal of a new model for assessing socio-technical aspects of security in heterogeneous systems. The model estimates the probability of humans committing dishonest acts in various situations, depending on a set of parameters including the seriousness of a crime (violent and non-violent), seriousness of criminal punishment (jailable and non-jailable offenses), attractiveness of the goal as expressed in money, the social status of the person, various coercion methods being applied (extortion, managers changing team contents, infiltrating an organisation through legitimate hiring channels, bribing and generally proposing a conspiracy). The model further assesses the likelihood that a person will be given up to authorities by another person for suggesting a conspiracy and takes into account how that affects the other person's willingness towards suggesting the conspiracy in the first place.

The result can be applied to estimate the likelihood of criminal acts and corresponding conspiracies both in individuals as well as in groups of several people; the latter is particularly helpful for comparing aspects of security between blockchain systems and their centralised counterparts, including for on-site security ceremonies.

The model was also validated against two use-cases and worked as expected according to subjective evaluation of the results.

Additionally, using the model, permissioned blockchain architectures were compared to centralised on-site security ceremonies for their security properties. The main differences that were found based on the model were as follows (section 3.7.1 has more details):

- In a typical centralised architecture, users' computers cannot directly validate the results, as opposed to the case with blockchain systems which often support what is called a "full node" verification method for thoroughly verifying the computations of the blockchain on the client side (section 3.5). For the cases where this leads to sub-optimal security for a given use-case, some additional people can be added to the ceremony as witnesses to compensate.

- In regular on-site ceremonies, people meet in order to perform the ceremony, which has the following two disadvantages: 1) for acts of bribery and proposing conspiracies to each-other, people who know one-another well are on average significantly more likely to conspire (section 3.6.2), and 2) violent acts of coercion are more difficult to carry out when people are attending remotely and/or hiding their identities. Violence may be relatively rare, but when it is used it can be highly effective, at least in the short term. Luckily, the blockchain validators do not have a strong need to meet and may even be able to hide their identities from each-other completely, whereas in a typical ceremony, people have to meet in a common location. The following improvements were suggested for ceremonies accordingly:
  - One or more ceremony participants could attend remotely, without communicating with other participants. They can use whatever cameras and other sensors they trust and need in order to record or observe the live transmissions from the ceremony room. Different users may have their own sensors that they trust (section 2.2).
  - Ceremony participants could be rotated on a regular basis (this only applies to people becoming overly familiar with each-other and may not help against violence in itself).
  - An obvious solution against some types of violence (but not necessarily against people becoming overly familiar with one-another) is the use of one or more security guards and/or various security devices.

  Nonetheless we note an important difference – a Byzantine fault tolerant (BFT) consensus may be needed if hardware and/or software cannot be trusted without such consensus (though utilising consensus for such purposes is apparently a rare occurrence in business). Technically, such consensus systems can be used both with blockchains and with the centralised ceremonies, although typically they tend to signify a blockchain solution.

4.2 Novelty of Results

To our knowledge, a novel part of this work is the combination of using a model for human behavior with the properties comparable to that of the proposed model together with the task of comparing different heterogeneous systems like blockchains and centralised on-site ceremonies, and our results thereof.

4.3 Application of Results

The resulting model of human behavior can be applied across a wide range of systems in order to estimate aspects of security in socio-technical systems. The model can help assess the number of people needed to securely perform different tasks and can help inform people on picking the appropriate mix of insiders and outsiders for security in different situations.

It can also bring some additional clarity to discussions around blockchains and their alternatives, where especially people mostly coming from the technology sector (such as the author of this work) may be curious about analysing human behaviour in the context of their work.

The potential use of security ceremonies as replacements for permissioned blockchains can be particularly interesting for expensive computations (perhaps even for supercomputers) for which it would be prohibitively expensive to have a large number of computers verifying the computations. The economical aspects of such arrangements are left for a future study.

5. Conclusions and Future Work

5.1 Conclusions

A new model was proposed for assessing socio-technical aspects of security, and in particular, the probability of humans committing dishonest acts in various situations. The model was validated with sample use-cases and was applied to perform a generalised initial comparison on the security aspects of permissioned blockchains to centralised on-site ceremonies.

The model suggests that in order to bring the security of centralised systems closer to the security of permissioned blockchains with Byzantine fault-tolerant (BFT) consensus, one could consider the possibility for remote participation in centralised security ceremonies. Nevertheless, if trusting computer hardware or software requires a BFT consensus, then a remote ceremony alone will not be sufficient. A combined approach can be used as well: ceremonies can be used together with BFT consensus, which could be particularly useful if the number of computers participating in consensus needs to be low for economic or performance reasons and yet there is a need for more people to act as witnesses for the correct operation of the computers.

5.2 Future Work

Several areas of research can be interesting for further study.

First, the models presented here could be used to study many more use-cases. In doing so, the models could also receive further refinement and/or validation. The addition of strong empirical studies to validate various aspects of human behavior would also be particularly interesting.

Second, economic and performance differences between blockchains and centralised ceremonies are also left for future study, as well as more concrete specifications for such ceremonies.

Further study would also be warranted for more accurate estimates for the occurrence

of 0-day vulnerabilities in hardware and software and the potential damage different vulnerabilities are likely to cause. This could help further refine the decisions on when a single hardware or software component can be trusted without consensus with other components.

Estimating the likelihood of backdoors in all of our tools is also somewhat of a nuanced problem, owing to the fact that as a society we build smart tools that we then use to create the next generation of smart tools and so forth, in countless successive iterations, and in principle, viruses and backdoors can spread over time into many different seemingly unrelated systems. More precise estimates for such likelihoods could also help decide how much to rely on multiple different hardware and software stacks when developing software in different situations.

Integrating our model with public permissionless blockchains like Bitcoin and Ethereum, especially with their properties of censorship resistance, could also be an interesting study to continue with.

References

[1] Łukasz Goczek and Ivan Skliarov. "What drives the Bitcoin price? A factor augmented error correction mechanism investigation". In: *Applied Economics* 51.59 (2019), pp. 6393–6410. DOI: 10.1080/00036846.2019.1619021. eprint: https://doi.org/10.1080/00036846.2019.1619021. URL: https://doi.org/10.1080/00036846.2019.1619021.

[2] Kaidong Wu. *An Empirical Study of Blockchain-based Decentralized Applications*. 2019. arXiv: 1902.04969 [cs.DC].

[3] *Digital Disruption Profile: Blockchain's Radical Promise Spans Business and Society*. URL: https://www.gartner.com/en/doc/3855708-digital-disruption-profile-blockchains-radical-promise-spans-business-and-society. (Accessed: 28 December 2020).

[4] Wikipedia contributors. *Distributed ledger — Wikipedia, The Free Encyclopedia*. [Online; accessed 30-December-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Distributed_ledger&oldid=995824149.

[5] *Distributed Ledger Technology: beyond block chain*. URL: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf. (Accessed: 29 August 2016).

[6] Claudio Scardovi. *Restructuring and Innovation in Banking*. Springer, 2016, p. 36. ISBN: 978-331940204-8.

[7] Ethan Buchman. "Tendermint: Byzantine Fault Tolerance in the Age of Blockchains". In: (2016). University of Guelph, Guelph, Ontario, Canada.

[8] K. Wüst and A. Gervais. "Do you Need a Blockchain?" In: *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. 2018, pp. 45–54. DOI: 10.1109/CVCBT.2018.00011.

[9] Guy Zyskind, Oz Nathan, and Alex Pentland. *Enigma: Decentralized Computation Platform with Guaranteed Privacy*. 2015. arXiv: 1506.03471 [cs.CR].

[10] Jörg Becker et al. “Can We Afford Integrity by Proof-of-Work? Scenarios Inspired by the Bitcoin Currency”. In: *The Economics of Information Security and Privacy*. Ed. by Rainer Böhme. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 135–156. ISBN: 978-3-642-39498-0. DOI: 10.1007/978-3-642-39498-0_7. URL: https://doi.org/10.1007/978-3-642-39498-0_7.

[11] Christian Badertscher et al. “Ouroboros Genesis: Composable Proof-of-Stake Blockchains with Dynamic Availability”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’18. Toronto, Canada: Association for Computing Machinery, 2018, pp. 913–930. ISBN: 9781450356930. DOI: 10.1145/3243734.3243848. URL: https://doi.org/10.1145/3243734.3243848.

[12] *The ‘Hot Swap’ Plan to Switch Ethereum to Proof-of-Stake Explained*. URL: https://www.coindesk.com/the-hot-swap-plan-to-switch-ethereum-to-proof-of-stake-explained. (Accessed: 29 December 2020).

[13] Qin Wang et al. “Preserving transaction privacy in bitcoin”. In: *Future Generation Computer Systems* 107 (2020), pp. 793–804.

[14] *The trust machine*. 2015. URL: https://www.economist.com/leaders/2015/10/31/the-trust-machine. (Accessed: 4 January 2021).

[15] J. Nicholas Hoover. *Amazon Launches Cloud Services For Government*. [Online; accessed 30-December-2020]. 2011. URL: https://www.informationweek.com/cloud/amazon-launches-cloud-services-for-government/d/d-id/1099599.

[16] *AWS launches a Secret region for the U.S. intelligence community*. URL: https://techcrunch.com/2017/11/20/aws-launches-a-secret-region-for-the-u-s-intelligence-community/. (Accessed: 29 December 2020).

[17] *The DNSSEC Root Signing Ceremony*. URL: https://www.cloudflare.com/dns/dnssec/root-signing-ceremony/. (Accessed: 29 December 2020).

[18] *DNSSEC Practice Statement for the Root Zone KSK Operator*. URL: https://www.iana.org/dnssec/icann-dps.txt. (Accessed: 29 December 2020).

[19] “Security Is a Weakest-Link Problem”. In: *Beyond Fear: Thinking Sensibly About Security in an Uncertain World*. New York, NY: Springer New York, 2003, pp. 103–117. ISBN: 978-0-387-21712-3. DOI: 10.1007/0-387-21712-6_8. URL: https://doi.org/10.1007/0-387-21712-6_8.

[20] Victor Costan and Srinivas Devadas. *Intel SGX Explained*. Cryptology ePrint Archive, Report 2016/086. https://eprint.iacr.org/2016/086. 2016.

[21] *Dissecting Operation High Roller*. URL: https://www.wired.com/images_blogs/threatlevel/2012/06/rp-operation-high-roller.pdf. (Accessed: 5 January 2021).

[22] *What is Blockchain and DLT?* URL: https://www.wto.org/english/res_e/reser_e/01_a_virginia_cram-martos_final_wto_2019-1202.pdf. (Accessed: 30 December 2020).

[23] Miguel Castro, Barbara Liskov, et al. "Practical Byzantine fault tolerance". In: *OSDI*. Vol. 99. 1999. 1999, pp. 173–186.

[24] Ethan Buchman, Jae Kwon, and Zarko Milosevic. "The latest gossip on BFT consensus". In: *arXiv preprint arXiv:1807.04938* (2018).

[25] Mark Silver, M. Markus, and Cynthia Beath. "The Information Technology Interaction Model: A Foundation for the MBA Core Course". In: *MIS Quarterly* 19 (Sept. 1995), pp. 361–390. DOI: 10.2307/249600.

[26] Ahto Buldas et al. "Rational choice of security measures via multi-parameter attack trees". In: *International Workshop on Critical Information Infrastructures Security*. Springer. 2006, pp. 235–248.

[27] Aivo Jürgenson and Jan Willemson. "Processing multi-parameter attacktrees with estimated parameter values". In: *International Workshop on Security*. Springer. 2007, pp. 308–319.

[28] Aivo Jürgenson and Jan Willemson. "Serial model for attack tree computations". In: *International Conference on Information Security and Cryptology*. Springer. 2009, pp. 118–128.

[29] Aivo Jürgenson and Jan Willemson. "On fast and approximate attack tree computations". In: *International Conference on Information Security Practice and Experience*. Springer. 2010, pp. 56–66.

[30] Ahto Buldas and Roman Stepanenko. "Upper bounds for adversaries' utility in attack trees". In: *International Conference on Decision and Game Theory for Security*. Springer. 2012, pp. 98–117.

[31] Ahto Buldas and Aleksandr Lenin. "New efficient utility upper bounds for the fully adaptive model of attack trees". In: *International Conference on Decision and Game Theory for Security*. Springer. 2013, pp. 192–205.

[32] Aleksandr Lenin and Ahto Buldas. "Limiting adversarial budget in quantitative security assessment". In: *International Conference on Decision and Game Theory for Security*. Springer. 2014, pp. 155–174.

[33] Ahto Buldas et al. “Simple infeasibility certificates for attack trees”. In: *International Workshop on Security*. Springer. 2017, pp. 39–55.

[34] Ahto Buldas et al. “Attribute evaluation on attack trees with incomplete information”. In: *Computers & Security* 88 (2020), p. 101630.

[35] Konstantinos Christidis and Michael Devetsikiotis. “Blockchains and smart contracts for the internet of things”. In: *Ieee Access* 4 (2016), pp. 2292–2303.

[36] Christian Decker, Jochen Seidel, and Roger Wattenhofer. “Bitcoin meets strong consistency”. In: *Proceedings of the 17th International Conference on Distributed Computing and Networking*. 2016, pp. 1–10.

[37] Marko Vukolic. “Eventually Returning to Strong Consistency.” In: *IEEE Data Eng. Bull.* 39.1 (2016), pp. 39–44.

[38] Elli Androulaki et al. “Hyperledger fabric: a distributed operating system for permissioned blockchains”. In: *Proceedings of the thirteenth EuroSys conference*. 2018, pp. 1–15.

[39] A.M. Antonopoulos. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. O’Reilly Media, 2014. ISBN: 9781491902646. URL: https://books.google.ee/books?id=IXmrBQAAQBAJ.

[40] Wikipedia contributors. *Blockchain — Wikipedia, The Free Encyclopedia*. [Online; accessed 30-December-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Blockchain&oldid=997158758.

[41] Carl M Ellison. “Ceremony Design and Analysis.” In: *IACR Cryptol. ePrint Arch.* 2007 (2007), p. 399.

[42] Jean Everson Martina, Túlio Cícero Salavaro de Souza, and Ricardo Felipe Custodio. “Ceremonies formal analysis in pki’s context”. In: *2009 International Conference on Computational Science and Engineering*. Vol. 3. IEEE. 2009, pp. 392–398.

[43] Carsten Schürmann et al. “Framing Electoral Transparency: A comparative analysis of three e-votes counting ceremonies”. In: (2016).

[44] Roger C Mayer, James H Davis, and F David Schoorman. “An integrative model of organizational trust”. In: *Academy of management review* 20.3 (1995), pp. 709–734.

[45] Jae Kwon and Ethan Buchman. “Cosmos whitepaper: A network of distributed ledgers, 2019”. In: *URL: https://cosmos.network/cosmos-whitepaper.pdf* (2019).

[46] Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. “Fraud proofs: Maximising light client security and scaling blockchains with dishonest majorities”. In: *arXiv preprint arXiv:1809.09044* (2018).

[47] Vitalik Buterin and Virgil Griffith. “Casper the friendly finality gadget”. In: *arXiv preprint arXiv:1710.09437* (2017).

[48] *Nodes and clients*. URL: https://ethereum.org/en/developers/docs/nodes-and-clients/. (Accessed: 1 January 2021).

[49] Wenbo Wang et al. "A survey on consensus mechanisms and mining management in blockchain networks". In: *arXiv preprint arXiv:1805.02707* (2018), pp. 1–33.

[50] Vitalik Buterin. *Timeliness detectors and 51% attack recovery in blockchains*. URL: https://ethresear.ch/t/timeliness-detectors-and-51-attack-recovery-in-blockchains/6925/1. (Accessed: 2 January 2021).

[51] Bruce Schneier. "Attack trees". In: *Dr. Dobb's journal* 24.12 (1999), pp. 21–29.

[52] David Robert Grimes. "On the viability of conspiratorial beliefs". In: *PLoS One* 11.1 (2016), e0147905.

[53] Tom N Jagatic et al. "Social phishing". In: *Communications of the ACM* 50.10 (2007), pp. 94–100.

[54] Simon R Wiseman. *Defenders Guide to Steganography*. Tech. rep. Deep Secure Technical Report DS-2017-2. DOI: https://doi. org/10.1314 0/rg . . ., 2017.

[55] Ken Thompson. "Reflections on trusting trust". In: *Communications of the ACM* 27.8 (1984), pp. 761–763.

[56] *Zero-day Vulnerability Database*. URL: https://www.zero-day.cz/database/. (Accessed: 6 January 2021).

[57] Serkan Özkan. *CVE Details*. URL: https://www.cvedetails.com/. (Accessed: 11 January 2021).

[58] A Caroline Mary. "Shellshock attack on linux systems–bash". In: *International Research Journal of Engineering and Technology* 2.8 (2015), pp. 1322–1325.

[59] Zakir Durumeric et al. "The matter of heartbleed". In: *Proceedings of the 2014 conference on internet measurement conference*. 2014, pp. 475–488.

[60] *Inflation Bug Still a Danger to More Than Half of All Bitcoin Full Nodes*. URL: https://cointelegraph.com/news/inflation-bug-still-a-danger-to-more-than-half-of-all-bitcoin-full-nodes. (Accessed: 11 January 2021).

[61] Leyla Bilge and Tudor Dumitraş. "Before we knew it: an empirical study of zero-day attacks in the real world". In: *Proceedings of the 2012 ACM conference on Computer and communications security*. 2012, pp. 833–844.

[62] *Flash note: EU cyber security agency ENISA; "High Roller" online bank robberies reveal security gaps*. URL: https://www.enisa.europa.eu/news/enisa-news/copy_of_eu-cyber-security-agency-enisa-201chigh-roller201d-online-bank-robberies-reveal-security-gaps. (Accessed: 8 January 2021).

Acknowledgements

This work would not have been possible without the generous support I have received from many people over the years. First, I would like to thank my advisor Ahto Buldas who repeatedly pointed me in the right direction and shared his wisdom that was crucial to the success of this work. I would also like to thank professor Jaan Ginter for his tireless answers to my questions on criminology and human behaviour, without which this work would not have been possible.

Next I would like to thank many of my colleagues and friends who have helped me learn about computer security and blockchains, including Andres Ojamaa, Risto Laanoja, Ahto Truu, Andres Kroonmaa and Peeter Omler. I would also like to thank my girlfriend Liisi who patiently tolerated my busy schedule as I was finishing the work.

Appendices

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis

I Risto Alas

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Using permissioned blockchains for security risk mitigation: an analysis framework and case studies", supervised by Ahto Buldas
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

11.01.2021

Appendix 2 – Derivations of the Tendermint Blockchain Fork Threshold Under Delayed Network Packets

Let us calculate for the Tendermint BFT consensus algorithm the threshold of nodes that need to be compromised in order to successfully finalize (commit to) an invalid parallel fork of the blockchain, under the assumption that the attacker can delay the arrival of network packets long enough that the honest consensus nodes which are on different parallel forks do not learn about the existence of each-others forks before they have already finalized on their own fork. Thus, the attacker can separate the consensus nodes from each other and the attacker's nodes can equivocate, that is, the attacking nodes can secretly participate on both forks.

Let n be the number of consensus nodes. We know that it takes $c_{\text{finalise}} = \lfloor \frac{2}{3} n \rfloor + 1$ nodes to finalise a block, which means that the other leftover nodes could be sent by the attacker to another fork: $c_{\text{leftover}} = n - c_{\text{finalise}} = n - (\lfloor \frac{2}{3} n \rfloor + 1)$. However, this number is clearly less than what it takes to finalize another fork, thus the attacker will need to enlist some nodes that would vote for both parallel chains. The minimum number of required double-voters is equal to the number of votes required for finalising a block minus the number of "leftover" nodes from the other parallel block:

$$
\begin{aligned}
c_{\text{malicious}} &= c_{\text{finalise}} - c_{\text{leftover}} \\
&= c_{\text{finalise}} - (n - c_{\text{finalise}}) \\
&= 2(c_{\text{finalise}}) - n \\
&= 2(\lfloor \frac{2}{3} n \rfloor + 1) - n \\
&= 2(\lfloor \frac{2}{3} n \rfloor + 1) - 2n + n \\
&= 2(\lfloor \frac{2}{3} n \rfloor - n + 1) + n \\
&= 2(-(n - \lfloor \frac{2}{3} n \rfloor) + 1) + n \\
&= 2(-(n + \lceil -\frac{2}{3} n \rceil) + 1) + n \\
&= 2(-(\lceil n - \frac{2}{3} n \rceil) + 1) + n \\
&= 2(-(\lceil \frac{1}{3} n \rceil) + 1) + n \\
&= 2(-\lceil \frac{1}{3} n \rceil + 1) + n \\
&= 2(1 - \lceil \frac{1}{3} n \rceil) + n \\
&= -2(\lceil \frac{1}{3} n \rceil - 1) + n \\
&= n - 2(\lceil \frac{1}{3} n \rceil - 1).
\end{aligned}
$$

For instance, if the number of consensus nodes is $n = 10$, then the resulting value $c_{\text{malicious}} = 4$; if $n = 9$ then $c_{\text{malicious}} = 5$.

Appendix 3 – Statistics On 0-day Vulnerabilities in Chosen Software Components Between 2015 - 2020

The following is a list of known to be actively exploited 0-day remote code exploits, as well as other types of vulnerabilities from the years 2015 to 2020, for select software. We summarise the results in section 3.8.

- Google Chrome. Remote code executions: 4, total: 7.
  - Multiple vulnerabilities in Google Chrome. CVE-2020-16017. Use-after-free
  - Remote code execution in Google Chrome. CVE-2019-13720. Use-after-free
  - Remote code execution in Google Chrome. CVE-2019-5786. Use-after-free
  - Multiple vulnerabilities in Google Chrome. CVE-2020-6418. Type Confusion.
- Google Chrome for Android. Remote code executions: 1, total: 1.
  - Remote code execution in Google Chrome for Android. CVE-2020-16010. Heap-based buffer overflow.
- Microsoft Internet Explorer. Remote code executions: 10, total: 17.
  - Remote code execution in Microsoft Internet Explorer. CVE-2020-1380. Buffer overflow
  - Remote code execution in Microsoft Internet Explorer. CVE-2020-0674. Buffer overflow
  - Remote code execution in Microsoft Internet Explorer. CVE-2019-1429. Buffer overflow
  - Remote code execution in Microsoft Internet Explorer. CVE-2019-1367. Use-after-free
  - Remote code execution in Microsoft Internet Explorer. CVE-2018-8373. Use-after-free
  - Remote code execution in Microsoft Internet Explorer. CVE-2015-2502. Memory corruption.
  - Multiple vulnerabilities in Microsoft Internet Explorer. CVE-2017-0222. Memory corruption
  - Multiple vulnerabilities in Microsoft Internet Explorer. CVE-2017-0149. Memory corruption.

  - Multiple vulnerabilities in Microsoft Internet Explorer and Edge. CVE-2016-3351. Memory corruption
  - Multiple vulnerabilities in Microsoft Internet Explorer. CVE-2015-2425. Memory corruption
- Mozilla Firefox. Remote code executions: 5. Total: 7.
  - Remote code execution in Mozilla Firefox and Firefox ESR. CVE-2020-6820. Use-after-free
  - Remote code execution in Mozilla Firefox and Firefox ESR. CVE-2020-6819. Use-after-free
  - Remote code execution in Mozilla Firefox and Firefox ESR. CVE-2019-17026. Type Confusion
  - Remote code execution in Mozilla Firefox and Firefox ESR. CVE-2019-11707. Type Confusion
  - Remote code execution in Mozilla Firefox. CVE-2016-9079. Use-after-free
- Oracle WebLogic. Remote code executions: 1. Total: 1.
  - Remote code execution in Oracle WebLogic Server. CVE-2019-2729. Deserialization of Untrusted Data
- Microsoft IIS. Remote code executions: 1. Total: 1.
  - Remote code execution in Microsoft IIS 6.0. CVE-2017-7269. Buffer overflow
- Oracle Java SE. Remote code executions: 1. Total: 2.
  - Remote code execution in Oracle Java SE. CVE-2015-2590. Remote code execution
- Microsoft Windows. Remote code executions: 2. Total: 41.
  - Remote code execution in Microsoft Windows. CVE-2015-2426. Buffer overflow. A remote attacker can create a specially crafted document or website with embedded malicious OpenType font, trick the victim into opening it, cause memory corruption and execute arbitrary code on the target system.
  - Multiple vulnerabilities in Microsoft Windows. CVE-2016-3393. Arbitrary code execution.
- Linux. Remote code executions: 0. Total: 3.
- Apple MacOS. Remote code executions: 1. Total: 3.
  - Multiple vulnerabilities in Apple macOS. CVE-2020-27930. Memory corruption
- Apple iOS. Remote code executions: 2. Total: 6.
  - Remote code execution in Apple iOS. Out-of-bounds write. 2020-04-22. The vulnerability exists due to a boundary error when processing email in the iOS MobileMail.
  - Multiple vulnerabilities in Apple iOS. CVE-2016-4657. Memory corruption

Appendix 4 – Effect of Re-Shuffling Members by an Individual Request on the Resilience of Ceremonies

Although our main analysis does not use this configuration, merely for informative reasons we analyse the case of an on-site ceremony where we allow every single ceremony participant to veto a ceremony, thus causing a ceremony to be re-scheduled, and if the same team is vetoed and rescheduled too many times then eventually we would have to assign new team members (perhaps the team members randomly assigned across a larger pool, thus the term "re-shuffling" for members). One of the reasons for composing a new team is that otherwise the team members could eventually get too familiar with each other, which, according to our model, would make them slightly more likely to conspire. Another reason for re-shuffling the team could be that the veto represents something negative about the team, in which case the next team could hopefully do better. However, if we allow such reshuffling by a mere request of a single member, we run a risk of the following attack: if malicious parties always keep vetoing every time they can see that the team does not consist of their co-conspirators, they would increase the likelihood of ending up on the same team with their co-conspirators. Let us calculate the likelihood of that happening.

To be conservative, let us assume that re-shuffling can be done an infinite number of times, and that all dishonest participants know each other and can tell whether all other participants in their ceremony happen to be dishonest or not (if they are dishonest, they could safely conspire together).

Let the likelihood of the entire group being dishonest be D and the likelihood of at least one member out of that group (but not all of them) being dishonest d. Without reshuffle, the probability of arriving at a dishonest group would the simply be:

$$P(\text{DishonestyNoShuffle}) = D.$$

With infinite numbers of reshuffles (even when the same people end up on asking for reshuffles as they eventually end up on the team again), the probability of ending up with

a dishonest team would be a geometric series, where the first addendum represents the likelihood of the group being dishonest at the first attempt and every next addendum represents the likelihood of the group being dishonest after the next shuffle:

$$P(\text{DishonestyWithShuffle}) = D + dD + d^2D + d^3D + ... = \sum_{k=0}^{\infty} Dd^k$$

The above represents a geometric series, often generally represented with the following formula: $\sum_{k=0}^{\infty} ar^k$. The series is known to converge to the value computed by the formula $a/(1-r)$ under the assumption that the common ratio r stays in the range $|r| < 1$. Substituting our numbers to the last formula, we can calculate the dishonesty likelihood to converge to:

$$P(\text{DishonestyWithShuffle}) = \frac{D}{1-d}, 0 \leq d < 1 \tag{1}$$

Thus we can calculate how much the "infinite" amount of shuffling allowed will make ceremonies less secure. Thus, we can calculate the honesty of the group with and without shuffling for various group sizes n and members' probabilities for dishonest acts p (for simplicity, we assume here that the probabilities p are equal for all group members).

We assume in the following example that so long as there is one person in the ceremony that is honest, the ceremony will not approve a malicious change, which means that the overall group honesty *without* shuffling is simply

$$P(\text{DishonestyNoShuffle}) = D = p^n. \tag{2}$$

With shuffle, we can use the equation 1, where D can be calculated again as p^n. Calculating d (the likelihood that some, but not all group members are dishonest) is slightly trickier, but not by much: $d = 1 - p^n - (1-p)^n$, which leads to the final formula for dishonesty using shuffle:

Table 9. Sample values to illustrate the odds of the entire group ending up as dishonest, depending on whether or not shuffling group members is allowed at the request of any single group member at all times, and depending on the group size n and the probability of a dishonest act of a single member p.

Group size n	Member dishonesty p	Probability that everyone is dishonest *without* shuffle $D = p^n$	Probability that everyone is dishonest *with* shuffle $P(\text{DishonestyWithShuffle})$
1	0.001	$1.000\,000 \times 10^{-3}$	$1.000\,000 \times 10^{-3}$
1	0.010	$1.000\,000 \times 10^{-2}$	$1.000\,000 \times 10^{-2}$
1	0.100	$1.000\,000 \times 10^{-1}$	$1.000\,000 \times 10^{-1}$
1	1.000	$1.000\,000$	$1.000\,000$
4	0.001	$1.000\,000 \times 10^{-12}$	$1.004\,010 \times 10^{-12}$
4	0.010	$1.000\,000 \times 10^{-8}$	$1.041\,020 \times 10^{-8}$
4	0.100	$1.000\,000 \times 10^{-4}$	$1.523\,926 \times 10^{-4}$
4	1.000	$1.000\,000$	$1.000\,000$
7	0.001	$1.000\,000 \times 10^{-21}$	$1.007\,028 \times 10^{-21}$
7	0.010	$1.000\,000 \times 10^{-14}$	$1.072\,886 \times 10^{-14}$
7	0.100	$1.000\,000 \times 10^{-7}$	$2.090\,751 \times 10^{-7}$
7	1.000	$1.000\,000$	$1.000\,000$
10	0.001	$1.000\,000 \times 10^{-30}$	$1.010\,055 \times 10^{-30}$
10	0.010	$1.000\,000 \times 10^{-20}$	$1.105\,727 \times 10^{-20}$
10	0.100	$1.000\,000 \times 10^{-10}$	$2.867\,972 \times 10^{-10}$
10	1.000	$1.000\,000$	$1.000\,000$

$$\begin{aligned} P(\text{DishonestyWithShuffle}) &= \frac{p^n}{1 - (1 - p^n - (1-p)^n)} \\ &= \frac{p^n}{p^n + (1-p)^n}. \end{aligned} \tag{3}$$

To tell the difference of probability of honesty between not shuffling and shuffling, we thus need to compare the results of equation 2 and equation 3. Analysing the formulae is out of scope for the current work, but when designing a ceremony, the exact parameters would have to be calculated by the designers. For reference, we show some values for multiple realistic situations for which the difference between the reliability of the shuffled and not shuffled version is relatively small in table 9.