Cryptography for pragmatic distributed trust and the role of blockchain

Quentin Santos

THÈSE DE DOCTORAT

de l'Université de recherche Paris Sciences et Lettres
PSL Research University

Préparée à l'École Normale Supérieure

Cryptography for Pragmatic Distributed Trust and the Role of Blockchain

École doctorale n°386

SCIENCES MATHÉMATIQUES DE PARIS CENTRE

Spécialité INFORMATIQUE FONDAMENTALE

COMPOSITION DU JURY :

M. Pierre-Alain Fouque
Université Rennes 1, Président du jury

M. Phan Dương Hiệu
Université de Limoges, Rapporteur

M. Fabien Laguillaumie
ENS de Lyon, Rapporteur

M. David Pointcheval
ENS, Directeur

M. Sébastien Canard
Orange Labs, Co-encadrant

M. Jacques Traoré
Orange Labs, Co-encadrant

Mme Aline Gouget
Gemalto, Membre du jury

M. Damien Stehlé
ENS de Lyon, Membre du jury

Soutenue par **Quentin Santos**
le 20 décembre 2018

Dirigée par **David Pointcheval**

ENS ÉCOLE NORMALE SUPÉRIEURE | PSL RESEARCH UNIVERSITY PARIS

Contents

1 Introduction **5**
- 1.1 Motivation . . . 5
- 1.2 The Rise of Modern Cryptography . . . 6
  - 1.2.1 The Information Age . . . 6
- 1.3 Modern Cryptography . . . 8
- 1.4 The New Vogue for Distributed System . . . 10
  - 1.4.1 Blockchain . . . 10
  - 1.4.2 Proof-of-Work . . . 12
  - 1.4.3 Smart Contracts . . . 14
  - 1.4.4 Blockchain-Free Solutions . . . 17
- 1.5 Cryptography in Distributed Networks . . . 20
  - 1.5.1 Provable Cryptography . . . 20
  - 1.5.2 Homomorphic Encryption (HE) . . . 22
  - 1.5.3 Zero-Knowledge Proofs (ZKPs) . . . 23
  - 1.5.4 Secure MultiParty Computation (MPC) . . . 24
- 1.6 Our Contribution . . . 25

2 Preliminaries **27**
- 2.1 Modular Arithmetic . . . 27
- 2.2 Assumptions . . . 31
  - 2.2.1 Axiomatic Frameworks . . . 31
  - 2.2.2 Common Assumptions . . . 32
  - 2.2.3 Pairings . . . 35
- 2.3 Signatures . . . 36
  - 2.3.1 Definition . . . 36
  - 2.3.2 Boneh-Lynn-Shacham . . . 37
  - 2.3.3 Pointcheval-Sanders . . . 38
- 2.4 Distributed Protocols . . . 38
  - 2.4.1 Shamir's Secret Sharing . . . 38
  - 2.4.2 Zero-Knowledge Proofs (ZKPs) . . . 40
- 2.5 Encryption . . . 45
  - 2.5.1 Asymmetric Encryption . . . 45
  - 2.5.2 Homomorphic Encryption . . . 48
  - 2.5.3 ElGamal Encryption . . . 49
  - 2.5.4 Paillier's Encryption . . . 50

3 A New Primitive for Pairwise Matching **53**
3.1 Organ Donation . . . 53
3.1.1 Compatibility for Organ Transplant . . . 53
3.1.2 Compatibility Matching by Equality Check . . . 55
3.2 Searchable and Comparable Encryption . . . 56
3.2.1 Related Work . . . 56
3.2.2 Our Contribution . . . 57
3.2.3 Organization . . . 58
3.3 Fingerprinting Scheme . . . 58
3.3.1 Description . . . 58
3.3.2 Security Model . . . 60
3.4 Assumptions . . . 62
3.5 Fingerprinting from Pointcheval-Sanders Signatures . . . 65
3.5.1 Fingerprinting Scheme with Public Plaintext-Equality Testing . . . 65
3.5.2 Security of the Basic Scheme . . . 65
3.5.3 Improving the Privacy of the User . . . 67
3.5.4 Verifiability . . . 69
3.5.5 Full Protocol . . . 69
3.6 Discussion . . . 71
3.6.1 Security Level . . . 71
3.6.2 Decentralized Setting . . . 71
3.6.3 Other Applications . . . 72
3.7 Conclusion . . . 72

4 Secure Strategy-Resistant Voting **75**
4.1 Electronic Voting Systems . . . 75
4.1.1 Voting Systems . . . 75
4.1.2 Electronic Voting . . . 77
4.1.3 Related Work . . . 78
4.1.4 Our Contributions . . . 79
4.1.5 Organization . . . 80
4.2 Majority Judgment . . . 81
4.2.1 Definition . . . 81
4.2.2 Removing Branching . . . 83
4.2.3 Expected Features for Encrypted Implementation . . . 84
4.3 Cryptographic Tools . . . 85
4.3.1 Batching Zero-Knowledge Proofs . . . 85
4.3.2 Proof of Private Multiplication . . . 85
4.4 Gate Evaluation with Multi-Party Computation . . . 86
4.4.1 Decryption Gate . . . 86
4.4.2 Conditional Gate . . . 87
4.4.3 Greater-Than Gate . . . 88
4.4.4 Mask Generation Gate . . . 89

4.5 Implementation 91
4.5.1 Ballot Encryption 91
4.5.2 Optimizations 92
4.5.3 Summary 92
4.5.4 Benchmarks 93
4.6 Discussion 94
4.7 Conclusion 95

5 Conclusion **97**

Acknowledgments

I would like to express my sincere gratitude to my academic advisor Prof. David Pointcheval as well as to my industrial advisors Dr. Sebastien Canard and Dr. Jacques Traoré for their inestimable support. Their guidance enabled me to conduct my research efficiently and grow as a scientist. I wish also to thank the other members of my thesis committee: Prof. Phan Dương Hiệu, Prof. Fabien Laguillaumie, Prof. Pierre-Alain Fouque, Dr. Aline Gouget, and Prof. Damien Stehlé. Their questions have encouraged me to enquire further and to consider alternative perspectives. I thank my fellow students with whom I exchanged many insights. Finally, I would like to thank my family for supporting me in general.

1 Introduction

"It is not in numbers, but in unity, that our great strength lies"
– Thomas Paine, *Common Sense*

1.1 Motivation

One of the main feats of modern civilization is being able to make a large number of people cooperate in the completion of a common goal. The most important aspect of this takes the aspect of social organization, and especially governments. Their role is to ensure that individuals cooperate despite natural incentives to do otherwise (essentially solving the Prisoner's Dilemma). Combining the contributions of many, and building upon past knowledge has enabled technological and economical progress at an accelerating pace for the last few millennia. For instance, the last few centuries have seen changes in governing that led to the competition of ideas and to more experimentation. Then, in the last few decades, telecommunications have accelerated the process considerably.

But, at this scale, social institutions and legal frameworks are not sufficient to rein the incentive of misbehaving. Here, technology can help in providing precise guarantees. Notably, cryptography makes it possible for large-scale cooperation through computers and telecommunication networks. This is achieved by ensuring the confidentiality and the integrity of the information exchanged.

More recently, a new technology that claims to coordinate many people without a government has surfaced, under the name of Blockchain. It is therefore legitimate to ask whether this technology will be able to improve our ability to cooperate. Indeed, it has attracted a lot of enthusiasm, and has often been compared to the information revolution. Billions of dollars have been invested into technologies and companies related to Blockchain. However, after a decade of investigation and this colossal funding, it remains unclear whether Blockchain have interesting use cases. Its original one is that of online currency, but it is also associated with a number of limitations and issues. It must be stressed, however, that Blockchain did bring something new to the table, and we should investigate whether it can help in large-scale cooperation.

In this thesis, we explore the problem of distributed trust, under practical constraints. More specifically, we consider use cases that require large-scale cooperation and investigate whether we can solve them with technology more than social institutions. In other words, we would like to avoid depending on a single trusted institution, and instead use technology to ensure that all parties behave. We look into details at potential solutions using modern techniques. In particular, we look at what cryptography can do, and the guarantees it provides, and consider what Blockchain can add.

1.2 The Rise of Modern Cryptography

1.2.1 The Information Age

Trust is easily achieved by humans on a small scale, such as that of a tribe or of a small village. This is made possible by various instincts, which command behavior in favor of cooperation. However, this mechanism does not work as well for larger settlements. The development of civilizations required *institutions* to widen the scale of cooperation. For example, the invention of fiat currencies to exchange goods and services between strangers. This means that trust stopped relying mainly on individuals instincts and shifted towards social mechanisms and *techniques*. The advances related to the industrial revolution and the application of the scientific methods were applied to communications and trust as well. Concretely, this refers to the combination of *telecommunications*, *computation*, and *cryptology* to devise secure means of communications and transfers of trust.

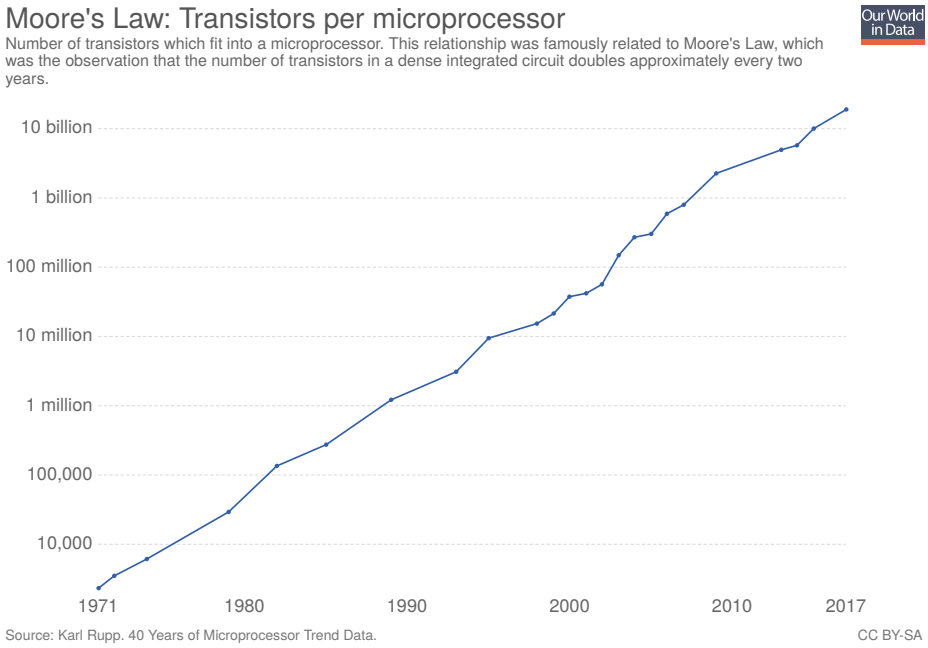


Figure 1.1: Moore's law: the number of transistors in an integrated circuit doubles every two years (from https://ourworldindata.org/technological-progress)

To understand how cryptography has become as common as it is today, we must look at the history of *payment cards*. A payment card (debit cards and credit cards) is a device that lets the user identify themselves for a transaction. In practice, a *payment terminal*, a device on the merchant's side, interacts with the payment card, contacts the bank through phone lines or the Internet, and completes the transaction. Initially, all the required information was stored on magnetic stripes. However, this approach provides little security, since the information can trivially be read by anyone and copied

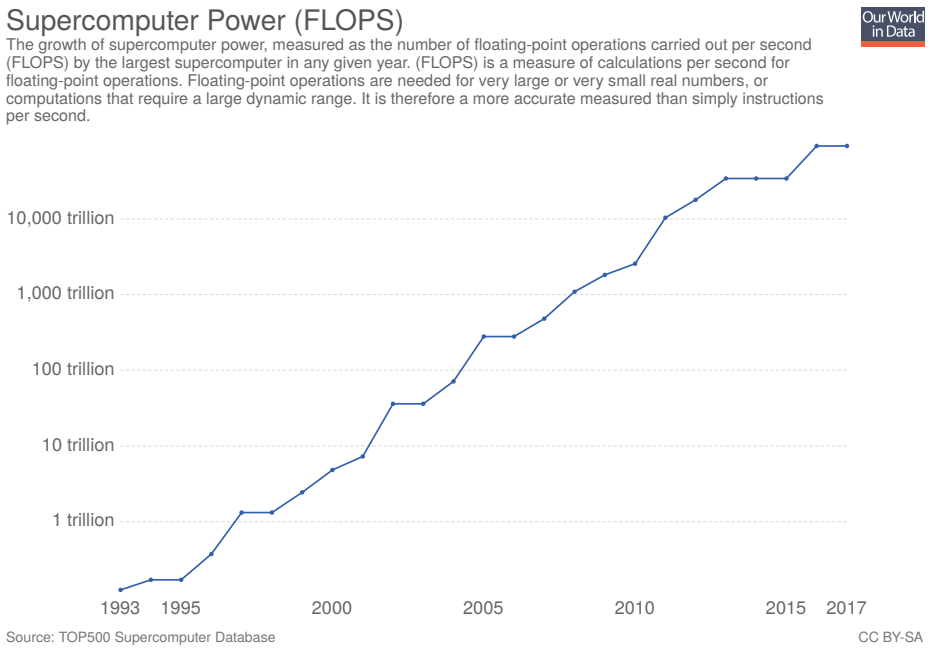


Figure 1.2: As a consequence of Moore's law, the available computing power grows exponentially (from https://ourworldindata.org/technological-progress)

to another card (cloning). During the 1970s, researchers developed *smart cards*, which embed integrated circuits, and present a pluggable interface. This device can thus authenticate the user by running an interactive cryptographic protocol with a server of the originating company. The first mainstream smart cards were sold in 1983 by France Telecom, then a division of the French Ministry of Posts and Telecommunications.

However, smart cards must be physically presented to a payment terminal on the merchant's side. This is incompatible with *card not present transactions* (i.e. mail or telephone orders). For a long time, the only solution was to have the user provide the information of the card over the phone or send it through the mail service. In parallel, the combined explosion of computation power due to Moore's law (see Figure 1.1 and Figure 1.2) and of communication capabilities related to Metcalfe's law enabled the World Wide Web. This commoditization of communication have led to the development of online retail, where users can easily browse department stores from their home. This both increased the demand for secure card not present transactions and enabled a solution. Effectively, online retail has brought cryptology to the general public. It is important to note that the democratization of cryptography was brought not by general privacy concerns, but by a very concrete use case.

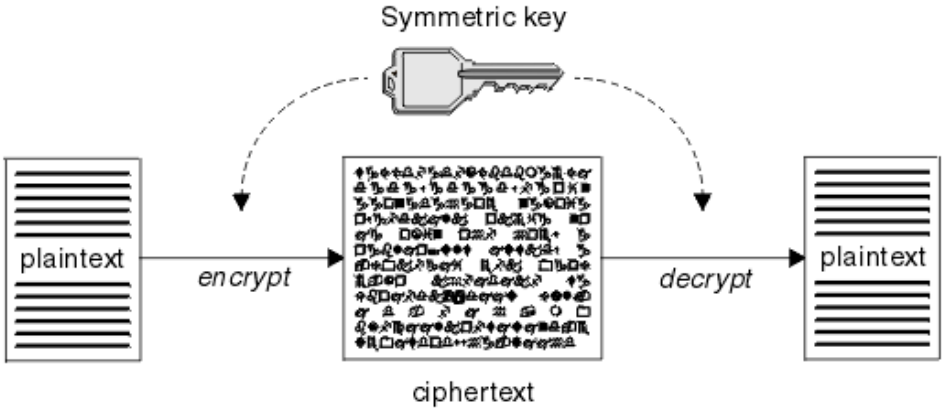


Figure 1.3: Encryption modifies a message such that it becomes unreadable to an eavesdropper. Only the receiver knows how to decrypt the ciphertext to obtain the original message. This enables secure communications even if messages are intercepted. Alternatively, this can be used for storing data securely, which is conceptually similar to sending a message to oneself (from https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_7.1.0/com.ibm.mq.doc/sy10500_.htm)

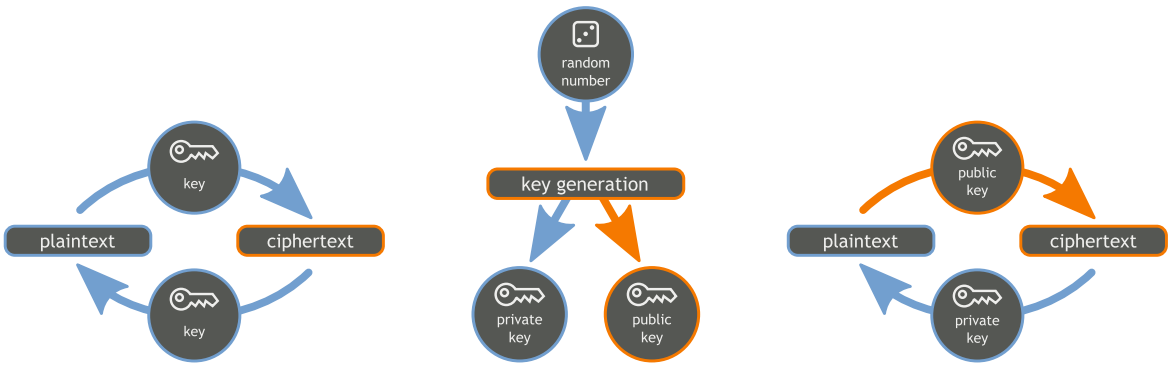
1.3 Modern Cryptography

Initially, *cryptography* ("crypto-" for "hidden" and "-graphy" for "message") was only concerned with *confidentiality* (see Figure 1.3). Historical ciphers include the scytale, Caesar's, Vigenere's, Vernam's and Alberti's ciphers, as well as many others. With the industrialization, more complex encryption methods were designed, such as the Enigma machine [HF45], the Lorenz cipher [MT45], or Delilah [TB45]. Following the computer revolution, even more elaborate schemes were introduced, and most notably DES [Nat77] and Rijndael [DR99] / AES [AES01]. The new computing abilities also enabled a new type of cryptographic scheme, where encryption and decryption are done by different processes: *asymmetric encryption* (see Figure 1.4).

However, practical considerations in the ways modern communications are used have quickly shown the importance of other properties.

- **Integrity:** ensuring that no attacker is able to alter the contents of the message during its transfer, neither in a predictable way, or randomizing part of the value;
- **Authentication:** ensuring that a particular person actually authored the message;
- **Non-repudiation:** ensuring that the author cannot pretend not to have originated the message after the fact.

Remark 1. *Authentication and non-repudiation look very similar. However, non-repudiation is actually a stronger guarantee than authentication: non-repudiation implies*



(a) symmetric encryption: the same process and secret (blue) value are used to encrypt or decrypt

(b) asymmetric key generation: from some secret (blue) value, a secret key and a corresponding public (orange) key are generated

(c) asymmetric encryption: using the public (orange) key, anyone can encrypt a message; however, decryption requires the secret (blue) key

Figure 1.4: Symmetric and asymmetric encryption. Figures by Wikipedia user Bananenfalter https://commons.wikimedia.org/wiki/User:Bananenfalter

that the receiver can convince other people that the sender indeed authored the message. With authenticity alone, the sender must still convince the receiver, but there is no guarantee that the receiver will be able to convince anyone else that the sender originated the message.

This has led to the quick expansion of this field of research, and the elaboration of various standards. In turn, this research has started exploring new domains of application for the techniques originally intended for communication.

For instance, it was observed early that one could modify an ElGamal or an RSA ciphertext with no knowledge of the secret key in such ways as to cause predictable changes to the plaintext. For classical applications, this *malleability* of the ciphertexts compromises security by sapping the integrity, and countermeasures had to be put in place. Yet, these *homomorphic* properties of the encryption method hinted at a cryptosystem that could be used in computations. While cryptography used to only concern itself with securing communications, new research have focused on providing security guarantees even during calculations.

Firstly, *Zero-Knowledge Proofs* (ZKP) provide ways to prove that a computation was performed correctly (e.g., decryption). Secondly, with homomorphic encryption, it becomes generally possible to design entire protocols that perform useful tasks without revealing any of the inputs. The idea of performing a *Secure MultiParty Computation* (MPC) between several actors that may not trust each other exploits the malleability of the ciphertexts to combine them and enforces good behavior using ZKPs.

These different approaches in general are currently at the focus of cryptology research. Initially, the results where mostly theoretical, as with Yao's garbled circuits, which first

proved that it was possible to run arbitrary algorithms on encrypted data. Throughout the years, the complexity costs have been greatly reduced. For instance, in 2012, Gentry et al. showed that it was possible to run an AES encryption over FHE in 36 hours [GHS12a], but the revised version from 2015 now gives running times of only 4 minutes [GHS12b]. Garbled circuits complete a similar operation in a few milliseconds [NST17], but are usually restricted to the two-party setting, or to the three-party setting [MRZ15]. However, recent advances in MPC have exploited more general variants of garbled circuits and scaled circuit evaluation to hundreds or even thousands of parties [HOSS18]. Large amounts of efforts are dedicated to expand the capability of secure computation by reducing this complexity further.

1.4 The New Vogue for Distributed System

1.4.1 Blockchain

Cryptography is a complex field of mathematics, so it might come as a surprise to learn of the recent surge of interest for this term by non-specialists. However, this phenomenon must be contextualized. Some users are indeed sensitive to the security of their communications, and have deliberately embraced the new computer and smartphone applications that provide end-to-end encryption. However, most discussions involving the topic are actually associated with a larger trend: *Blockchain*. We will use "Blockchain" (upper-case "B") to refer to the general concept, and "blockchain" (lower-case "b") to refer to specific implementations. A blockchain is a data structure conceived specially for *Bitcoin*, a fully decentralized electronic currency [Nak08]. The technical and speculative success of Bitcoin have brought in tremendous amount of interest in the academic and industrial world, but in the general public as well. Because Blockchain uses cryptographic hashes and Bitcoin relies on cryptographic signatures, this process was associated with the word "crypto". But, in this context, "crypto" is often short for *cryptocurrency*, which refers to Bitcoin and the other currencies based on a blockchain.

From a theoretical point of view, Bitcoin was an original, if expensive, solution to an intriguing problem. As discussed above, the problem of exchanging money through a digital medium has been long solved. However, until recently, all the solutions relied on some classical institution that the individual users must trust. With the recent development in science and engineering, it was only natural to wonder whether it was possible to replace these institutions entirely by technology. Cryptographers reached intermediate milestones, notably by allowing anonymous transactions in a digital setting (notably the electronic currency *e-cash* introduced by David Chaum [Cha82]).

Distributing trust while still being able to reach a consensus requires some mechanism to take mutual decisions, which usually takes the form of a voting system. However, in the digital realm, identity is a more fuzzy concept, since information can be duplicated without restriction. In practice, this corresponds to the problem of Sybil attacks, where an attacker creates many accounts to bias the result of an election (for instance). In general, online voting implies that some authority sources these identities by distributing

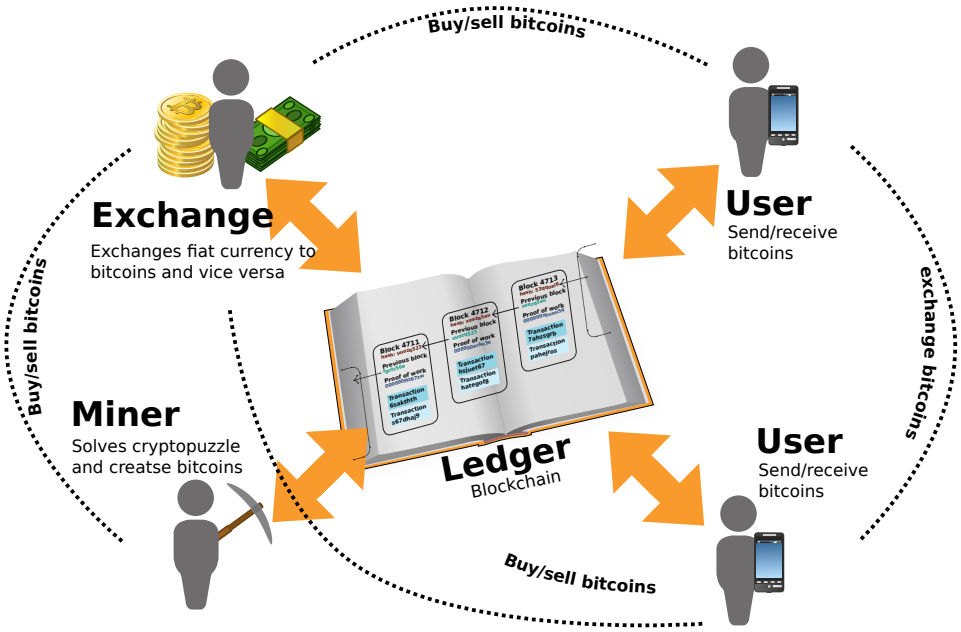


Figure 1.5: Bitcoin. A blockchain maintains a ledger containing all pasts transactions. Miners are the participants who ensure the integrity of the blockchain and update it. Users (and exchanges) submit to the miners candidate transactions, that only become valid once in the blockchain. The blockchain incentivizes miners to play their part by automatically rewarding them with new Bitcoin tokens. (from http://www.csg.uzh.ch/csg/en/research/blockchain.html)

credentials to each individual. But this is incompatible with a fully decentralized system. The solution offered by Blockchain is essentially to give up on controlled fairness (one vote per individual), so that votes can instead depend on computation power. Although some may find such a trade-off unacceptable from a point of view of the democratic process, since some individuals will have a higher influence, it does base itself on an objective metric. In other words, the influence of individuals is theoretically proportional to their involvement in the system.

Blockchain fits a very narrow class of use cases. Understanding this requires a sound technical knowledge of computer science, communication networks, cryptology and game theory. For many, however, Blockchain looks like a new silver bullet, and potentially a new technological revolution. This impression is influenced precisely because of the incentives put on its participants. This means that Blockchain has become a hammer, and enthusiasts are looking for anything that could look like a nail. Thus, many projects attempt to use this technology blindly, disregarding other solutions and historic best practices, and causing damage. On the other hand, the large amount of discussion did bring to the surface several use cases that could be solved with technologies that do not involve Blockchain. In some situations, it is simply that an industry has yet to take advantage of the capabilities of computers to automate tasks and transfer information.

Figure 1.6: Price of a single bitcoin (common unit of Bitcoin), ranging from 17 December 2016 to 17 December 2017. Although the technical aspects of Blockchain are often discussed, most of its popularity can probably be traced back to the speculative nature of Bitcoin and the other cryptocurrencies that followed. (from https://coinmarketcap.com/currencies/bitcoin/)

In others, solutions exist in the field of modern cryptography, but many of its techniques are still unknown in the broader community.

1.4.2 Proof-of-Work

Let us assume that Alice owns a token of some currency. If she sent the token to Bob, then Bob now owns it. If she sent the token to Charlie, then Charlie now owns it. But what if Alice somehow allows the token to be sent to both Bob and Charlie? It could be argued that Bob owns it, or that it should be Charlie's, and other interpretations are also possible. This is known as the double-spend problem. In the physical realm, currencies can assume that it is hard to copy tokens, and they can use various anti-counterfeiting mechanisms to ensure this. In the digital realm, everything can be easily duplicated by nature. There exist efforts to restrict this (DRMs), but their success is usually limited in time, and they often rely on tying a digital asset to a physical one. Other approaches focus on watermarking content to trace the individual who illegitimately redistributed the piece of intellectual property, but this does not help in our situation.

In 2008, Satoshi Nakamoto proposed Bitcoin [Nak08], a fully decentralized electronic currency. It solves the double-spend by considering that a transaction is confirmed only once it appears in a blockchain. The blockchain plays the role of global consensus so that the participants agree on which transactions are valid. More precisely, it waits about one hour to ensure that everyone realizes that the transaction is part of the blockchain. In the previous scenario, Bob would only accept the token as payment after seeing the corresponding transaction fully confirmed. If Alice then tried to use the same token as payment for Charlie, he would then notice that the token has already been spent by Alice and is now Bob's. By waiting one hour, Bob and Charlie ensure that no other

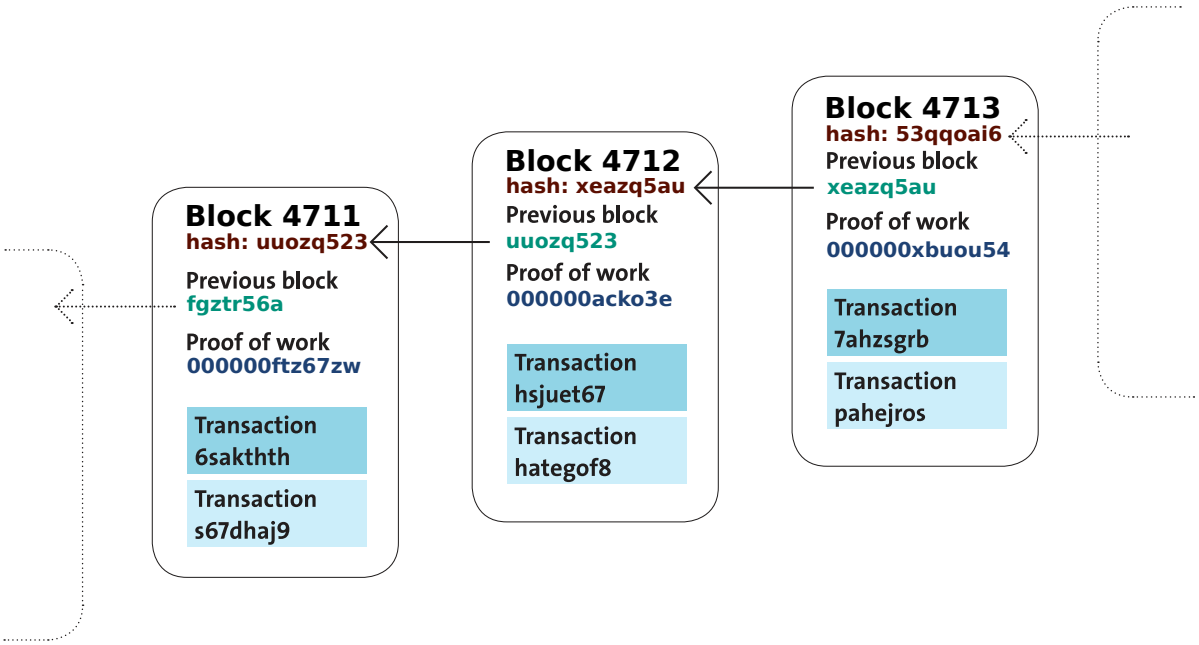


Figure 1.7: Blockchain: blocks are added left-to-right, each pointing to the previous one; the authorization to add a vote comes from the Proof-of-Work (from http://www.csg.uzh.ch/csg/en/research/blockchain.html)

transaction is trying to use the token.

Deciding which transactions should be incorporated in the blockchain is done by solving a puzzle. The participants who try to solve this puzzle are called *miners*. The first miner to solve the puzzle gets to update the blockchain with a new block containing the new transactions. Solving the puzzle is done by an exhaustive search, so it is trivially parallelized. This means that the probability of being the first to solve the puzzle is effectively proportional to the amount of computing effort one provides (and inversely proportional to the total amount of power provided by all the miners). Presenting a solution to the problem is thus thought of as a *proof-of-work*.

Effectively, this leverage the fact that the digital world is effectively limited by certain physical resources. The most convenient are computing power and memory storage. By using a proof-of-work, participants give a witness of their computing effort, which has a practical limit. HashCash uses this to limit the amount of unwanted emails [Bac97]. In this proposal, users who wish to send an email must prove to the server that they performed a small amount of computation. Although this would be transparent for legitimate users, spammers would need to dedicate significant resources to send many emails, having a deterring effect.

However, the amount of computation available to users varies greatly with the nature of the device. Most people have access to CPUs, which are optimized at executing a single task. Even within CPUs, there are several specializations, with desktop (good single-thread performance), laptop (various trade-offs), server (good single-thread and multi-thread performances) and smartphone CPUs (low power consumption). In the last

decades, the growth of the video game industry has led to the development of GPUs, which are optimized for massively parallel tasks. Finally, ASICs are integrated circuits designed for a specific case. Although many users have some sort of GPU (e.g., chipsets and, more recently, APUs), only a minority actually owns powerful ones (discrete GPUs on dedicated extension cards). They are much more expensive, and only a handful of people can acquire them. This gradation in computing power shows that a few individuals are disproportionately privileged when it comes to proof-of-work.

Alternatively, it is possible to rely on the rareness of computer memory. The idea is to require a proof-of-work whose computation occupies large amounts of memory. The Password Hashing Competition has selected Argon2 [Wet16] as a new password hash function, in particular for its memory-hardness. It succeeds scrypt, a memory-hard password hash function proposed in 2009. Several cryptocurrencies use scrypt (most notably Litecoin) or Argon2 to deter the use of ASICs by miners. Note that, with memory-hard functions, computing power can always be substituted for memory (by recalculating the value).

1.4.3 Smart Contracts

We now explain how transactions work in cryptocurrencies, and what a "smart contract" refers to. First, note that there is no concept of account in Bitcoin, where tokens, or the amount of currency would be stored. Instead, there are only used transactions and unused transactions. Essentially, a transaction contains the public key of its owner. To spend it, the owner signs a new transaction with the corresponding secret key. The new transaction contains the public key of the new owner. This is illustrated in Figure 1.8.

Thus, in a cryptocurrency such as Bitcoin, transactions are simple instructions that verify the signature of the sender and transfer a certain amount of currency from one cryptographic keypair to another. In fact, Bitcoin uses a basic scripting language with very limited functionality, named Script to give some flexibility to the transactions. It does not support loops and is thus not Turing-complete. It does however manage a stack, giving it some non-trivial capabilities. This restricted design is deliberate, as it ensures that transactions can be executed in a predictable amount of time, and gives less opportunity for bugs to crop up. Cryptocurrencies that use a Turing-complete language face the problem of handling execution costs. Because of the Halting Problem, such costs cannot be predicted for arbitrary programs. We show some examples of Bitcoin transactions in Figure 1.9, Figure 1.10 and Figure 1.11. See the wiki article at `https://en.bitcoin.it/wiki/Script` for a description of the possible instructions. A practical use of this feature is to implement escrow payments with transactions that require a threshold 2-out-of-3 signature.

In the continuation of this basic scripting language, Ethereum [But13] sought a design where transactions would be programs of arbitrary complexity. This is done by introducing *smart contracts*. Smart contracts are simply complex transactions. In the case of Ethereum, these transactions can be Turing-complete (they can use loops). Since it is hard to determine the cost of running an arbitrary program, users pay for the time needed to execute the smart contract. More specifically, running a smart contract con-

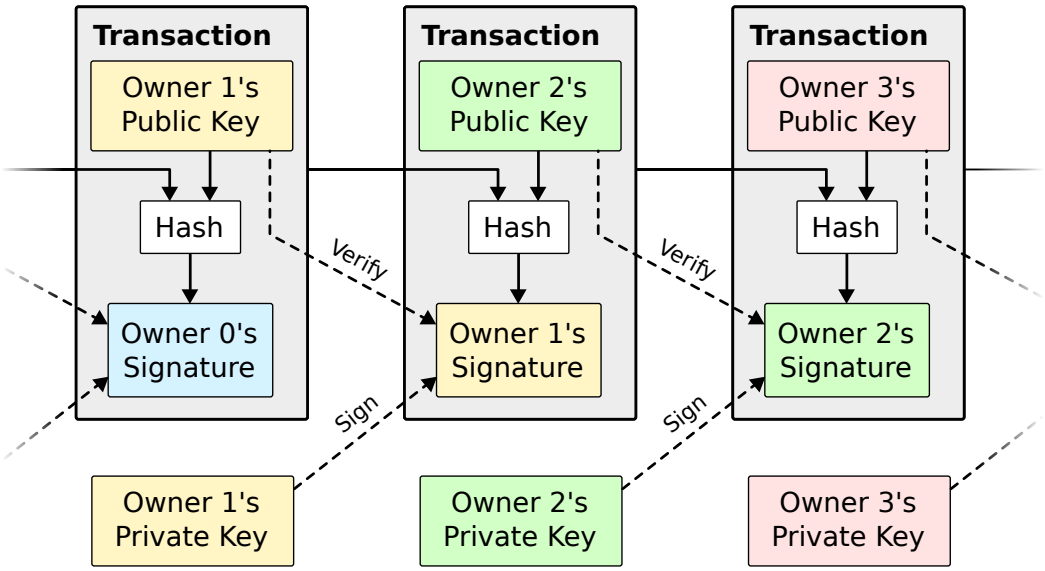


Figure 1.8: Transactions in a cryptocurrency. Owner 2 receives a transaction from Owner 1 which contains Owner 2's public key. Owner 2 can then transfer currency by signing a new transactions for Owner 3. (from https://fc.isima.fr/~mazenod/slides/privacy/bitcoin.html#/0/8)

sumes its "fuel", which is a value associated with the smart contract and paid in ether (the currency of Ethereum). Essentially, this transfers the problem of determining the running time of the smart contract from the device where it is evaluated to its authors. Since the authors write the smart contracts, they can design them in ways that ensure that they do not consume all of its fuel before finishing. This avoids the Halting Problem because the authors do not need to predict the running time of *arbitrary* programs.

As a generalization of escrow transactions, smart contracts were thought of as a way to implement decentralized institutions. In fact, the term Decentralized Autonomous Organization (DAO) refers to a system that uses smart contracts to take decisions. In effect, a DAO can hold funds in the form of ether, and a board committee takes votes to decide on how to use these funds. The most famous DAO, "The DAO", worked as a mutual fund made of Ethereum smart contracts, where investors could contribute some capital, and then vote on investments.

However, smart contracts highlight another difficulty with complex transactions, separate from that of the running time. Indeed, writing bug-free software is hard. The amount of issues present in a piece of software is often underestimated. For an extreme example, NASA is famous for reducing the number of errors to fewer than 1 per 10,000 single lines of code. However, the cost of the associated methodology is much higher than what would be acceptable for most software projects.

This fact proved itself again with smart contracts. In June 2016, a vulnerability of

```
scriptPubKey: <pubKey> OP_CHECKSIG
scriptSig: <sig>
```

Figure 1.9: How Bitcoin transactions initially worked. The "public key" (`scriptPubKey`) is actually a series of instructions indicating how to check a signature. The "signature" (`scriptSig`) contains a cryptographic signature and the corresponding cryptographic public key. The names between angle brackets are replaced by actual values in each transaction. Here, `scriptPubKey` verifies that `<sig>` is a valid signature under the cryptographic public key `<pubKey>`.

```
scriptPubKey: OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
scriptSig: <sig> <pubKey>
```

Figure 1.10: A modern standard Bitcoin transaction. This time, `scriptPubKey` first compares the hash of `<pubKey>` to `<pubKeyHash>`. Finally, the last instruction verifies that `<sig>` is a valid signature under the cryptographic public key `<pubKey>`.

the smart contract behind The DAO was exploited, allowing attackers to redirect the equivalent of about $50M at the time. The flaw stemmed from the behavior of the smart contract that allowed reentrant execution, but did not account for it. Since members of the Ethereum project had a significant stake in The DAO, they pushed for cancelling all the transactions corresponding to the attack. This reaction was considered hugely controversial, since it effectively changed a result that was supposed to be immutable and guaranteed by the blockchain. This event simultaneously illustrates the risks associated with relying on a fully automated system with no safeguards, and questions whether projects based on Blockchain actually decentralize trust as much as is expected.

```
scriptPubKey: OP_HASH256 6fe28c0ab6f1b372c1a6a246ae63f74f931e8365e15a0\
89c68d6190000000000 OP_EQUAL
scriptSig: <data>
```

Figure 1.11: A puzzle Bitcoin transaction. Here, `scriptPubKey` only checks that `<data>` hashes to the target value. This kind of transaction is intended as a playful challenge with an automatic reward. There are many such puzzles in the various cryptocurrencies.

Figure 1.12: Price of a single ether (common unit of Etehrem), from the announce of The DAO ot its demise. Because of the speculation in cryptocurrencies in general, the price of the ether varies with that of bitcoin. To isolate the effect of The DAO, we show the price of the ether (ETH) in bitcoins (BTC). (from https://coinmarketcap.com/currencies/ethereum/)

1.4.4 Blockchain-Free Solutions

As explained above, Blockchain only fits very specific use cases. Its popularity makes it tempting to use it to solve various problems, but it turns out that many proposed applications have little to do with what Blockchain actually provides.

What Blockchain does. First, Blockchain pertains to the digital realm. This is seldom useful because humans are still grounded in the physical, and so are most of our activities. In practice, this means that any use of a blockchain to track physical assets will rely on some mechanism to connect it to the physical world. For instance, in the case of delivering physical articles, tying the payment to the delivery requires trusting the delivery method itself. Similarly, Blockchain is often touted as a solution to track products along their supply chains. But the actual issue of the supply chain is collecting the correct information in the first place, which has little to do with Blockchain. However, this does not itself dismiss Blockchain as a way to track land tenures or property liens. The existence of a central authority and the poor usability do.

Second, Blockchain reaches a global consensus. This is not always useful since local consensus can be enough. For instance, the early World Wide Web used little consensus, and anyone was able to start a personal website without interacting with any particular institution. Granted, the Domain Name Service became quickly widespread. However, domain names are not strictly required, and quite a few websites could actually only be accessed directly by their IP addresses even in the 2000s. It could be argued that IP address themselves constitute a global consensus, but they are the basics for the Internet, and are almost a commodity. In fact, in decentralized systems based on Blockchain, such as Bitcoin and the other cryptocurrencies, miners require IP addresses to communicate.

But, since these addresses can be transient, there is not real need for a long-term global consensus on IP addresses.

Third, Blockchain works in a fully decentralized setting. This is rarely useful because it is possible to identify some form of central authority in most real-world coordination problems. In any situation involving a classical government (voting, land, economic policy), the center is obvious[1]. Within any company, the company is the center. Between several companies, the previously existing legal framework can often be used as a center. This leaves a single case: individuals interacting independently of an intermediate institution (international interplay, or ideal libertarianism).

For currencies, this is the original use case tackled by Bitcoin. In any case, it means that Blockchain did solve an interesting theoretical problem, but that it seems to bring little more in practice, bar a drastic shift in governance. There are practical reasons why such a shift is unlikely (although not impossible). To illustrate this, consider what happens when a mistake is done with a cryptocurrency: when users lose or leak their secret keys (this is extremely common), when they transfer to the wrong address, when an exchange fails (Mt. Gox, Coincheck, and many others), or when pump-and-dump schemes exit (cryptocurrencies bypass many regulations and safeguards). In many cases, users have no recourse; in some, they must fall back on classical institutions to initiate litigation.

Cryptocurrencies have also brought a new way to exploit compromised systems. Botnets are now used to mine cryptocurrencies for the profit of its administrators, and at the expense of the legitimate owners. Scripts can also be maliciously embedded in webpages to target a large number of computers [Mes17, ELMC18]. Several tools try to detect and block the execution of these scripts, but they mostly rely on black lists [WFX+18].

Permissioned Blockchain. Facing such a limited range of application, Blockchain experimenters have put forward the idea of deploying *permissioned blockchains*. In this variant, all the actors are clearly identified. This setting greatly reduces the cost of maintaining a blockchain. Indeed, once each participant has a secret cryptographic key, running votes and taking decisions becomes almost trivial. Theoretically, it would mean that Blockchain could actually fit several use cases (see Figure 1.13). However, one should keep in mind that permissioned Blockchain dismisses the innovative aspect of Blockchain. In fact, what a permissioned Blockchain does is simply maintain a log of activities between the participants secured by a traditional PKI (Public-Key Infrastructure).

With that said, the problem is actually not trivial. Although cryptography with identified actors effectively stops an attacker from altering arbitrarily the contents of a database, there still remain the problem of consistency. In fact, the CAP theorem states that the following three properties cannot hold simultaneously for a distributed system.

- **Consistency:** every read request returns the most recent written value (or fails);

[1] note that weak governments are not solved by Blockchain: you still need a physical way to enforce what could be called a "cryptolaw"

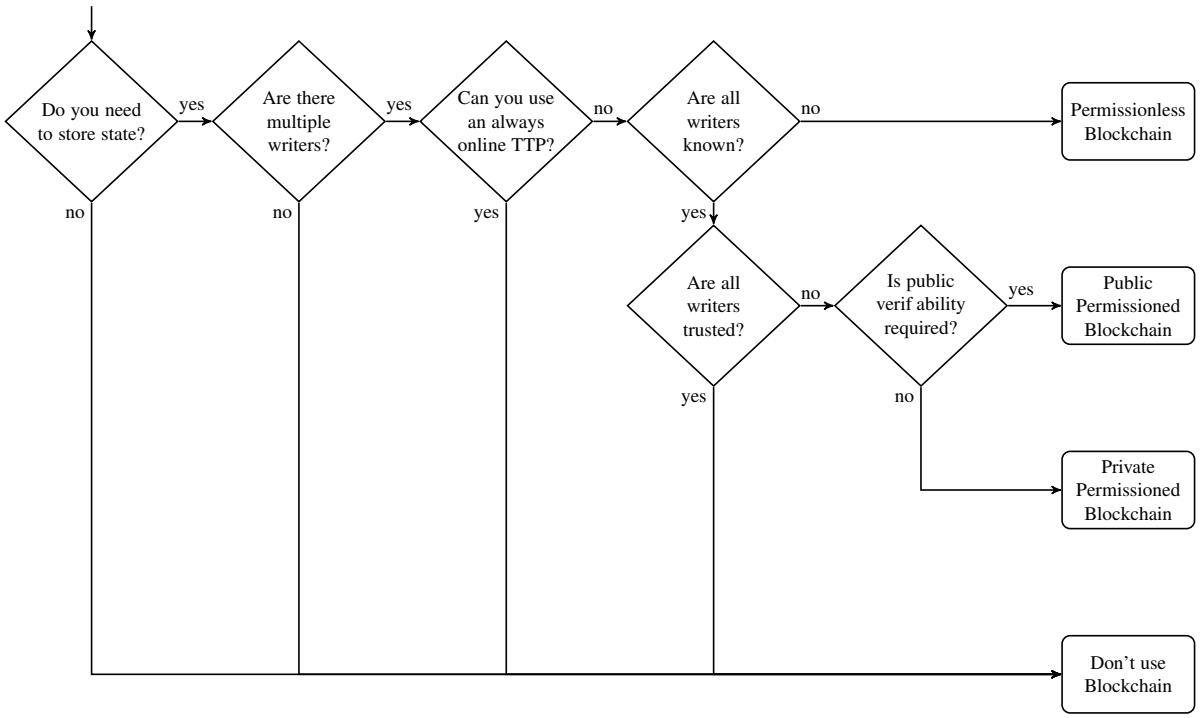


Figure 1.13: Do You Need a Blockchain? This decision flowchart helps determining whether Blockchain is pertinent for a given use case. It highlights multiple questions that filter out most real-world problems. However, this advises for the use of a permissioned blockchain in some situations (from [WG17])

- **Availability:** every request succeeds (a read request may not return the most recent written value);

- **Partition tolerance:** messages between nodes are dropped by the network.

In other words, when a network partition occurs, a distributed system must choose between either refusing requests, or serving obsolete results. This issue is well-known to IRC users in the form of netsplits, where servers from a same network drop consistency during partitions (see Figure 1.14).

Byzantine fault tolerance (BFT) is another description of the phenomenon. The usual example is that of two generals who must coordinate a simultaneous attack (see Figure 1.15). The general description considers the problem of keeping a distributed system consistent while nodes and messages may fail silently. Solving the BFT problem is of great importance for large scale databases and content delivery networks. Although no perfect solution exists, protocols using various trade-offs are well known. For instance, Paxos is a commonly used system to achieve consensus in such situations.

This shows that permissioned blockchain have little use. However, this brings us to the topic of trust in distributed systems, and how it can be achieved using cryptography.

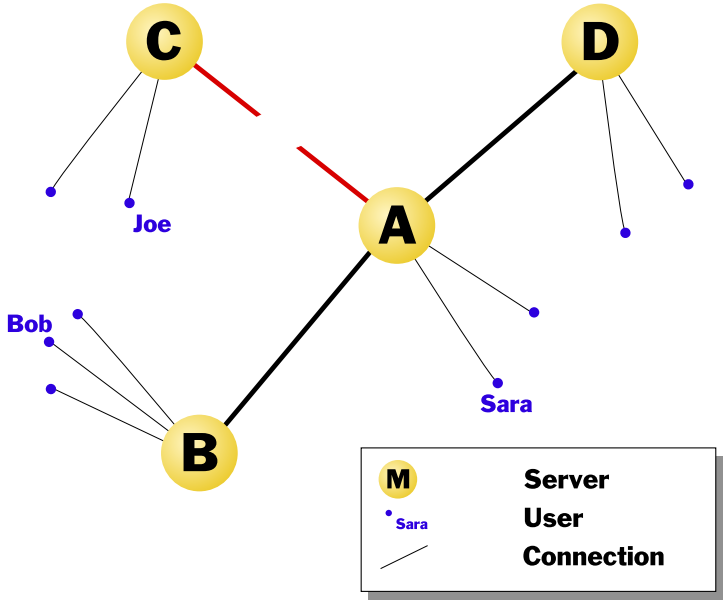


Figure 1.14: Servers A and C have lost sight of each other; Bob and Sara can still chat together, but Joe will not be able to communicate with them until the servers reconnect

1.5 Cryptography in Distributed Networks

As we mentioned in Section 1.2, modern cryptography brings new tools for administrating trust.

1.5.1 Provable Cryptography

With the development of computers, modern cryptography have become tightly connected with computer science and various mathematical fields. As a consequence, it has become common practice in cryptographic research to provide mathematical proofs of the security of proposed solutions. Ideally, this ensures that cryptographic schemes cannot be broken within the model of computation under consideration. The model of computation refers to the mathematical models [Fer09] used to idealize how computers work, although this can be applied to calculators and even human computation. However, this is seldom attained. As a concession to practicality, security is often reduced to a few assumptions instead. Although we do not know whether these assumptions are true, or even whether they are decidable, they are easier to inspect. This is the most pragmatic way to reduce the attack surface in the theoretical side. Finally, there are many cases where an attack is theoretically possible, but would take too long in the real world to be practical. Thus, many schemes assume that the computational abilities of the attackers are bounded.

Nevertheless, within this approach, we must consider whether the model of computation we use is actually accurate. For this, we need to rely on empirical evidence. Indeed,

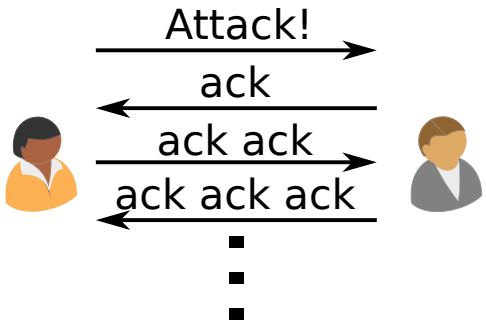


Figure 1.15: Assume that Alice and Bob are two Martian generals, who want to attack Earth simultaneously. If Alice sends a message to Bob and attacks, it might fail to arrive, and Bob will not join Alice in the attack. Adding an acknowledgement ensures that Alice does not attack alone, but if it does not arrive to Alice, Bob will be the one to attack alone. The problem switches back and forth as more acknowledgement messages are added. No deterministic solution exists, but it is possible to solve the problem probabilistically if a model for transmission failure is known.

it has been observed that this model is particularly good at predicting various behaviors when performing calculations. However, some edges cases do exist, and can be exploited to attack even theoretically secure cryptosystems that have been correctly implemented. Such attacks are referred to as *side-channel attacks*. These include notably:

- **timing attacks:** the time needed to run the computation might depend on secret information, and be measured by the adversary [Koc96, BB03];
- **power analysis:** the power drawn by the computation might depend on secret information, and be measured by the adversary [KJJ99, KJJR11];
- **electromagnetic analysis:** the electromagnetic field emitted by the system might depend on secret information, and be measured by the adversary [Age82, HSK97];
- **cold-boot attacks:** an attacker might directly access the memory of a system [HSH+09, RC11].

Finally, an attacker could use a different model of computation altogether. Specifically, it has been discovered that it was theoretically possible to build a computer with a fundamentally different set of basic operations using quantum mechanics [Ben80, Man80, Fey82, Deu85].

Various ways to ensure security guarantees over sensitive information have been designed within this context. However, it is possible to do more than simply secure the transmission of information.

1.5.2 Homomorphic Encryption (HE)

Homomorphic encryption (HE) is the general idea of performing computation over encrypted data, introduced in [RAD78]. More exactly, a HE scheme is a way to encrypt data so that it is then possible to perform operations on it. The end goal of homomorphic encryption is the ability to run arbitrary programs over encrypted data (see Figure 1.16).

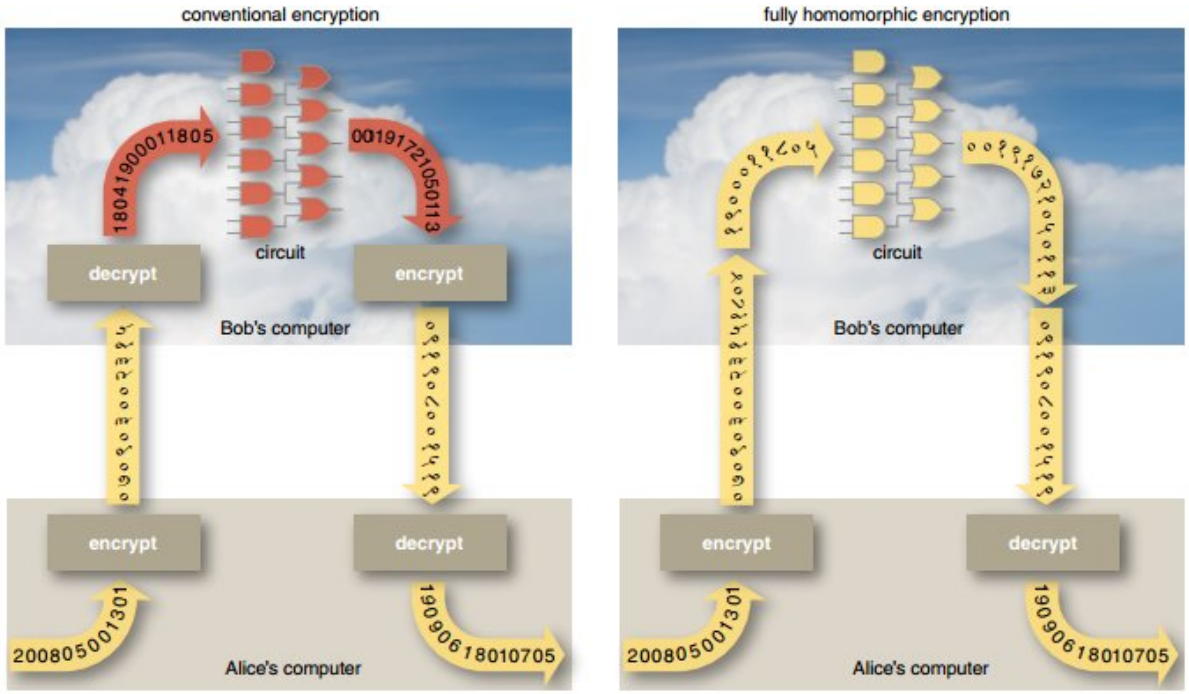


Figure 1.16: Ideally, Alice would like to execute arbitrary circuits on encrypted data and securely delegate computations to Bob. Traditionally, she would use normal encryption to send her data to Bob and retrieve the results, but this reveals information to Bob. Instead, she can use HE, and send encrypted data that he cannot decrypt, but that he can use for computations. Alice can then decrypt the result. From [Hay12].

Solutions for partially homomorphic encryption (PHE) schemes have been known for decades, such as the RSA [RSA78], or ElGamal [ElG84] cryptosystems. However, PHE only allows a single type of operation, such as either addition, or multiplication (but not both). This has some uses, but is not sufficient for most applications.

Somewhat Homomorphic Encryption (SHE). An intermediate stepstone was reached with somewhat homomorphic encryption (SHE). In a SHE scheme, both additions and multiplications can be performed, but there is a restriction on the number of multiplications that can be chained. For instance, the BGN cryptosystem [BGN05] allows an arbitrary number of additions, but only one multiplication. The execution time of this type of scheme rises quickly of the multiplicative depth of the evaluated expression. In

practice, this limits what can be done in a reasonable amount of time. This type of solution can cover some use cases, and provide partial solutions for others, but it is not yet fully satisfying.

Fully Homomorphic Encryption (FHE). In 2009, Craig Gentry proposed a way to first realize a fully homomorphic scheme (FHE) [Gen09]. The idea is to first take a SWE scheme, where each multiplication adds noise to the ciphertext. If too many multiplications are chained, the noise grows too much, and it becomes impossible to decrypt and recover the plaintext. The solution is to use a special operation, named *bootstrapping*, that reduces this noise. This way, bootstrapping is run every few multiplications, and arbitrary expressions can be evaluated.

However, bootstrapping is a costly operation, and it must be performed relatively regularly. In practice, it is essentially a re-encryption of the ciphertexts, itself done using the homomorphic properties of the encryption scheme. Re-encrypting removes the noise from the ciphertexts, but the fact that this operation is performed in the ciphertext domain adds back some noise. Because of this, FHE schemes are typically inefficient. However, new milestones are reached each year, and the efficiency of FHE is improving significantly.

1.5.3 Zero-Knowledge Proofs (ZKPs)

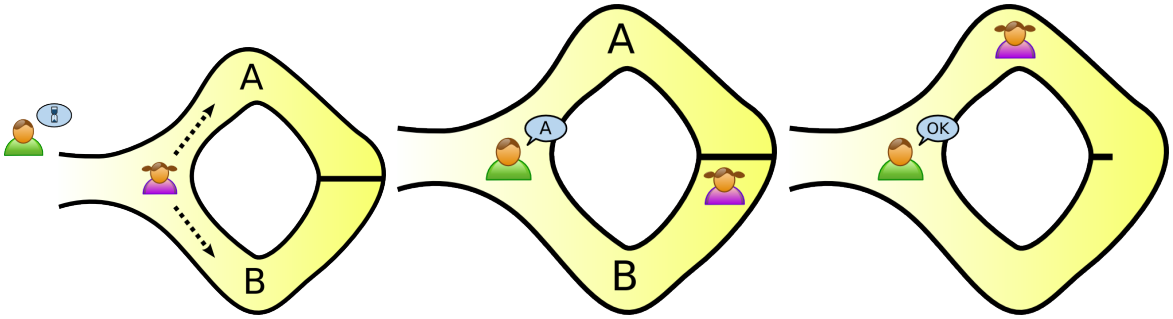
In the example of Figure 1.16, Alice might also want to ensure the correctness of the result. By combining confidentiality of the data and integrity of the computation, Alice can delegate heavy computations to any party. This relaxes the constraint of trusting the host of the calculations, and opens the door to lending CPU time more systematically.

In many situations however, secret information from both the prover and the verifier is combined. In this case, the prover has to convince the verifier that the computation was done correctly without revealing private information. The solution takes the form of *Zero-Knowledge Proofs*, or ZKPs. Here, "zero-knowledge" refers to the fact that the protocol leaks no secret information.

This technique relies on the idea that there are problems that are hard to solve, but for which it is easy to verify that a solution is valid (NP-complete problems). Such a solution is essentially a *witness* that the problem was solved. With ZKPs, the goal for the prover will be to demonstrate knowledge of this verifier without disclosing the solution (which would take a long time for the prover to find). Thus, ZKPs are often *Proofs of Knowledge*, or PoKs. Although there is yet to be a formal proof that NP-complete problems actually are hard to solve, they are usually assumed to be (e.g., 3-SAT or the travelling salesman problem). Indeed, after considerate effort, no efficient (polynomial-time) algorithm have been found to solve any of these (but heuristics often exist and are sometimes enough for real-world applications).

When the goal of the prover is only to convince the verifier that such a witness exists, we might use the expression of "proof of existence". The mere existence of a witness can be sufficient to guarantee that some assertion holds (e.g. relation between values). In a

ZKPoK, the verifier is convinced that the prover knows the witness, and therefore that it exists. A basic illustration of the concept is given in Figure 1.17.



(a) Alice enters the cave while Bob stays outside

(b) Bob chooses one of the two exists

(c) Alice exits by the selected path

Figure 1.17: Zero-Knowledge Proofs. The cave has two entrances, it is possible to get from one to the other, but the way is barred by a locked door. Only one knowing the secret codeword can pass. Alice can demonstrate to Bob that she knows the secret codeword by using the protocol illustrated here. Figures inspired by [QQQ+90], drawn by Wikipedia user Dake https://fr.wikipedia.org/wiki/Utilisateur:Dake

The concept was introduced in 1985 by Goldwasser, Micali and Rackoff in [GMR85]. Many protocols have been proposed, most notably the Schnorr protocol [Sch90], which can be generalized for various relations in the exponents [CS97]. A general approach also allows simplifying the protocol to a single exchange between the prover and the verifier [FS87]. Such protocols are said to be non-interactive.

1.5.4 Secure MultiParty Computation (MPC)

Once we know how to perform operations on secret data and how to convince that this was done correctly, we can consider entrusting secure computation to several entities. In effect, we will be exploring ways to split secret information among several parties.

Shamir's Secret Sharing. A well-known result due to Shamir offers a simple approach for distributing a secret value among several peers [Sha79]. For a given secret value s, each party gets a secret share s_i. When at least k parties agree, they can reconstruct s from their shares. This is already sufficient for some situations, but revealing the secret implies that it is essentially single-use. A more satisfying approach is to exploit the expression used to recover s. This means that it is possible to collectively compute values that depend on s, without ever revealing the secret s itself.

More generally, protocols that share secret values allow the execution of complex computations securely and relatively efficiently. This type of protocol is called Secure MultiParty Computation (MPC). However, not everything can be done for an arbitrary number of parties.

Secure Two-Party Computation (2PC) and Garbled Circuits (GC). If we restrict ourselves to two parties, more complete results exist. In fact, one of the main problems of Secure Two-Party Computation (2PC) is *Yao's Millionaires' problem*. The problem considers two wealthy parties, Alice and Bob, who want to determine which of them is the wealthier, without revealing the amount of their wealth. In other words, Alice knows some secret value a, Bob knows some secret value b, and they want to test whether $a \geq b$ without leaking information about a or b. The first solution to the problem was provided by Yao in 1982 [Yao82].

A logical circuit is a directed acyclic graph (DAG) where nodes are logical operators and edges connect the output of a logical operator to inputs of other logical operators. In this paper, he introduces *garbled circuits* (GC) which are a way to transform logical circuits so that they can be executed in a 2PC setting without revealing the inputs or the intermediate values. Such circuits can be used to represent arbitrary computations that run in a predefined amount of time.

In practice, we can simplify the problem as follows: Alice knows some circuit, and Bob some input. They would like to execute Alice's circuit on Bob's input without revealing either. What would be Alice's input can be incorporated in the circuit.

For each bit of the input of the circuit, Alice chooses an arbitrary representation for 0 and another for 1. She then garbles each gate of the circuit by providing a logical table where the output are encrypted by the inputs. It means that the circuit can only be executed for some specific inputs. When valid inputs are given, they allow decrypting the corresponding results of the first logical gates, which in turn allow decrypting the next level. This cascades all the way to the output of the circuit. Alice gives the resulting garbled circuit to Bob.

All that remains is for Bob to obtain the representation of each bit of its input. For this, Alice and Bob run an *oblivious transfer protocol*. In such a protocol, Alice offers two values x_0 and x_1 (the representations of 0 and 1), and Bob chooses $b \in \{0, 1\}$ (without seeing x_0 or x_1). The important property of this protocol is that Alice does not learn b, and Bob only learns x_b, but not x_{1-b}.

This result implies that there exist 2PC protocols for any computation. However, circuits can be large, and garbled circuits are expensive in general due to the sheer number of encryptions that Alice must do, and of decryptions for Bob. Thankfully, this result was only a first step and more efficient constructions are being discovered.

1.6 Our Contribution

In this thesis, we explore two different practical problems where secure computation in a distributed environment would prove pertinent. We show that, with proper consideration, it is already possible to use cryptography for concrete situations and provide working solutions.

First, we consider a situation that seems to directly contradict crucial conditions for confidentiality in cryptography. Organizations dedicated to organ donation search compatible donors and recipients among the medical records. Since we do want to find

such compatible records, the entropy of the input space is necessarily low. On first consideration, it seems to mean that any encryption of the medical records would imply some form of central authority to insert new medical records or compare them. But this is problematic in an international setting, where there might not be enough trust for such an authority. Therefore, we propose a practical solution to this problem by relying on a decentralized authority to provide strong confidentiality guarantees. Our solution does this while staying reasonably efficient.

Second, we explore the domain of electronic voting, in which deployed solutions are still unsatisfactory. Indeed, many solutions use cryptography only to secure the communication with a trusted authority. This trusted authority could trivially observe the choice of each voter. Although they are not yet widespread, solutions with stronger guarantees have been realized, such as Helios and Belenios [Adi08, CGGI14], Scytl [CHI+08], or Prêt à Voter [CRS05, RBH+09, XSHT08]. However, all of the existing solutions either cover only the simplest voting systems, or reveal aggregates and complete their execution in the clear. Thus, we propose an implementation of a cryptographic protocol for an advanced voting system which only reveals the winner and keep all intermediate values secret. The proof-of-concept gives acceptable running times, and they could be improved easily with larger resources.

Through these two examples, we show that the current knowledge of cryptography already enables us to solve many more problems without having to wait for generic schemes to become efficient.

2 Preliminaries

Our contributions are built upon preexisting knowledge. In this chapter, we present some cryptographic primitives and assumptions that we use to introduce our own constructions. For completeness, we also provide the definitions of the basic mathematical objects that are commonly used in cryptography.

2.1 Modular Arithmetic

Group Theory. Groups are the basic tool for cryptology, and entires branches of mathematics in general.

Definition 1 (Group)**.** *Let G be a set and $+$ be a binary operator. We say that $(G, +)$ is a group when the following properties are verified.*

- ***Closure:*** $\forall a, b \in G, a + b \in G$*;*
- ***Associativity:*** $\forall a, b, c \in G, a + (b + c) = (a + b) + c$*;*
- ***Identity element:*** $\exists 0, \forall a \in G, a + 0 = 0 + a = a$*;*
- ***Inverse element:*** $\forall a \in G, \exists b \in G, a + b = b + a = 0$.

For the last property, b is the inverse of a, and noted $-a$. This is the additive notation.

Definition 2 (Abelian group)**.** *When the following additional property is also met, we say that the group is abelian.*

- ***Commutativity:*** $\forall a, b \in G, a + b = b + a$.

Definition 3 (Field)**.** *When we have two laws $+$ and $\times$, we say that $(G, +, \times)$ is a field when:*

- $(G, +)$ *is a group;*
- $(G \backslash \{0\}, \times)$ *is a group;*
- ***distributivity:*** $\forall a, b, c \in G, a \times (b + c) = a \times b + a \times c$ *($\times$ is distributive over $+$).*

To avoid confusion we use the multiplicative notation for $\times$: the identity element is written 1 (note that $1 \neq 0$), and the inverse of a is noted a^{-1} or $\frac{1}{a}$. Finally, the $\times$ operator might be omitted when doing so does not introduce ambiguity.

Definition 4 (Order)**.** *The order of a group $(G, +)$ is the cardinality of the set G.*

Arithmetic. Cryptographic constructs are usually defined over generic groups associated with some properties or assumptions. However, we often need at least the natural numbers to describe relations between the elements of these groups. The set of naturals numbers is usually represented by the character $\mathbb{N}$, and defined by the zero element 0, the equality relation $=$ and the successor function S. There are themselves defined by the Peano axioms.

1. $0 \in \mathbb{N}$;
2. $\forall x \in \mathbb{N}, x = x$ (reflexivity of equality);
3. $\forall x, y \in \mathbb{N}, x = y \Rightarrow y = x$ (symmetry of equality);
4. $\forall x, y, z \in \mathbb{N}, x = y \wedge y = z \Rightarrow x = z$ (transitivity of equality);
5. $\forall x, y, (x \in \mathbb{N} \wedge x = y \Rightarrow y \in \mathbb{N})$ (closure of equality);
6. $\forall x \in \mathbb{N}, S(x) \in \mathbb{N}$ (closure of successor function);
7. $\forall x, y \in \mathbb{N}, S(x) = S(y) \Rightarrow x = y$ (injectivity of successor function);
8. $\forall x \in \mathbb{N}, S(x) \neq 0$.

In short, 0 is an element of $\mathbb{N}$, $=$ is an equivalence relation, and $S : \mathbb{N} \to \mathbb{N}$ is injective such that $S^{-1}(0) = \varnothing$. As an example, the fourth element of $\mathbb{N}$ is thus written as $S(S(S(0)))$. The explicit expression for the answer to the ultimate question of life, the universe and everything is a bit unwieldy. Instead, we can use the decimal representation for clarity (although sexagesimal has some nice properties). However, this requires the principle of induction:

9. $\forall P, P(0) \wedge (\forall n \in \mathbb{N}, P(n) \Rightarrow P(S(n))) \Rightarrow \forall n \in \mathbb{N}, P(n)$.

The most natural model for the Peano axioms is perhaps given by set theory where the zero element is defined as the empty set, and the successor adds to x a singleton containing x:

- $0 = \varnothing$;
- $S(x) = x \cup \{x\}$.

From this, it becomes easy to define our usual operators $+$, $-$, $\times$, $\div$, and then the corresponding closures $\mathbb{Z}$ and $\mathbb{Q}$.

Prime Numbers. In particular, we say that b divides a when their quotient is integral: $\forall a, b \in \mathbb{N}, b|a \Leftrightarrow \exists q \in \mathbb{N}, a = b \times q$. This is where things start to get interesting. Indeed, we know of no closed-form expression to determine how many divisors a given number has. We define the slightly more general divisor function $\sigma_x(n) = \sum_{d|n} d^x$. Then, $\sigma_0(n)$ is the number of divisors of n. At least, some properties are obvious:

- $\sigma_0(0) = \infty$;
- $\sigma_0(1) = 1$;
- $\forall n \geq 2, \sigma_0(n) \geq 2$ since $1|n$ and $n|n$;
- $\forall n \geq 2, \sigma_0(n) \leq n$.

Numbers n such that $\sigma_0(n) = 2$ are particularly interesting. Since they work as the basic blocks for multiplication (we cannot split them in smaller numbers), we call them prime numbers and note their set $\mathbb{P}$. A very useful remark is that any natural number may be written as the product of prime numbers.

Theorem 1 (Fundamental theorem of arithmetic)**.** $\forall n, \exists! \alpha : \mathbb{P} \to \mathbb{N}, n = \prod_{p \in \mathbb{P}} p^{\alpha(p)}$.

From this property, it comes that we can describe phenomena on prime numbers, and deduce the results for all of the natural numbers. This is handy because prime numbers verify many convenient properties.

Modular Arithmetic. The operation $+$ implies a group over the set $\mathbb{Z}$. But $(\mathbb{Z}, +)$ is infinite and working in finite groups is often more convenient in practical applications. To construct simple finite groups, we take the successors of 0, but warp back to zero at some value $n \in \mathbb{N}^*$. For instance, $\mathbb{Z}_5$ corresponds to the set of values $\{0, 1, 2, 3, 4\}$. In $(\mathbb{Z}, +)$, $4 \times 4 = 16$. If we count from 0 to 16, we encounter 5 three times, and count one more, so, in $\mathbb{Z}_5$, $4 \times 4 = 1$. To be clearer, we will use $x \equiv y \mod n$ to indicate the group in which we are working, thus writing $4 \times 4 \equiv 1 \mod 5$. Here, "mod" is short for "modulo".

How can we describe the map from $\mathbb{Z}$ to $\mathbb{Z}_n$ more explicitly? For any number $m \in \mathbb{Z}$, we warp from n to 0 a certain number of times q, and count a few more values r. In other words: $m = q \times n + r$ with $q, r \in \mathbb{Z}$ and $0 \leq r < n$. This corresponds to Euclidean division and the following property:

Theorem 2 (Euclidean division)**.** $\forall m, n \in \mathbb{Z}, \exists! (q, r) \in \mathbb{Z}^2, m = q \times n + r \wedge 0 \leq r < n$.

Furthermore, we also know how to compute q and r efficiently. Since we do not usually care about q, we use the notation $m \bmod n$ to refer to this r. With this, we can extend our notation to $\mathbb{Z}$: $16 \equiv 1 \mod 5$ because $16 \bmod 5 = 1$.

We can check that this is consistent for the various operations on $\mathbb{Z}$. Take $a = q_a \times n + r_a$ and $b = q_b \times n + r_b$ with $q_a, q_b, r_a, r_b \in \mathbb{Z}$ and $0 \leq r_a, r_b < n$. Clearly, $a \equiv r_a \mod n$ and $b \equiv r_b \mod n$. Further, we have:

$$a + b = (q_a + q_b) \times n + (r_a + r_b)$$

Thus, $a + b \equiv r_a + r_b \mod n$. More generally, all of the operations $+$, $-$ and $\times$ work as expected in $\mathbb{Z}_n$. For $\div$, let us try to find the multiplicative inverse of $a \in \mathbb{Z}_n$. That is to say, we want b such that $a \times b \equiv b \times a \equiv 1 \mod n$. In other words, such that there exists $q, r \in \mathbb{Z}$ for which $a \times b = q \times n + 1$. However, this is not always possible. For instance, when $a = 2$ and $n = 4$, the left-hand side is always even, and the right-hand side is always odd. To describe the existence of an inverse in the general case, we need to introduce the notion of greatest common divisor.

Greatest Common Divisor. We define the greatest common divisor of two natural numbers.

Definition 5 (Greatest common Divisor)**.** $\gcd(a, b) = \max\{q : q|a \wedge q|b\}$.

This notion is sometimes extended to a set of natural numbers: $\gcd(S) = \max\{q : \forall n \in S, q|n\}$. However, it is mostly used for pairs of natural numbers. We can notice that $\gcd(ma, mb) = m \gcd(a, b)$. This raises the question of "basic pairs", which cannot be divided into smaller elements. Similarly as for primes, we say that a and b are coprime when $\gcd(a, b) = 1$. Although we again know no closed-form expression, we count them with Euler's Totient Function:

Definition 6 (Euler's Totient)**.** $\varphi(n) = \#\{k : 0 \leq k \leq n \Rightarrow \gcd(k, n) = 1\}$.

Obviously, for any prime number p, we have $\gcd(k, p) = 1$ for all $0 \leq k < p$ so $\varphi(p) = p - 1$. In parallel, we can also define the least common multiple:

Definition 7 (Least Common Multiple)**.** $\mathrm{lcm}(a, b) = \min\{q : a|q \wedge b|q\}$.

It is easy to switch back and forth between gcd and lcm with the following property.

Theorem 3. $\gcd(a, b) \times \mathrm{lcm}(a, b) = a \times b$

Returning to the problem of finding an inverse for a modulo n, let us note $d = \gcd(a, n)$. Then, if $d \neq 1$, the left-hand side of the equation $a \times b = q \times n + 1$ is clearly divided by d while the right-hand side is not, so it has no solution. When $d = 1$ however, a does admit an inverse.

Theorem 4 (Bezout's Identity.)**.** $\forall a, b \in \mathbb{Z}, \exists x, y, ax + by = \gcd(a, b)$

For the particular case where $\gcd(a, b) = 1$, this gives us the assertion we need for inverses modulo n. Indeed, if $\gcd(a, n) = 1$, then Bezout's identity tells us that there exists b and q such that $a \times b + q \times n = 1$. This means that b is the inverse of a modulo n. Since, for a prime number p, $\gcd(k, p) = 1$ for $0 \leq k < p$, it means that all the elements of $\mathbb{Z}_p$ have inverses, so $(\mathbb{Z}_p, \times)$ is a group. Because $\times$ is still distributive over $+$, $(\mathbb{Z}_p, +, \times)$ is a field. In other cases, we can restrict ourselves to the invertible elements of $\mathbb{Z}_n$, noted $\mathbb{Z}_n^\times$, which is also a group for $\times$. Note that $\mathbb{Z}_n^\times$ is the set of the integers that are coprime with n.

Cyclic Groups. We say that a group G is cyclic when it is generated by a single element. That is, there exists an element g such that $G = \{g^k : k \in \mathbb{N}\}$. It means that any element $a \in G$ can be written as $a = g^k$ for some natural number k. This is convenient because this allows describing relations between elements of G directly with arithmetic equations.

If p is prime, then $(\mathbb{Z}_p, \times)$ is a cyclic group of order $p - 1$. We can construct a cyclic group of prime order by finding $p = qr + 1$ with p and q prime and finding h such that $h^r \not\equiv 1 \mod p$. Then, the subgroup of $(\mathbb{Z}_p, \times)$ generated by $g = h^r \bmod p$ is of prime order q. It is common to set $r = 2$; in that case, the subgroup is the squares of $\mathbb{Z}_p$. Working with a cyclic group of prime order let us manipulate the element with exponents that lie in the arithmetic field $(\mathbb{Z}_p, +, \times)$.

Carmichael Function. A less frequently used function is the Carmichael function, which gives the least common order of $\mathbb{Z}_n^\times$:

Definition 8 (Carmichael)**.** $\lambda(n) = \min\{m \in \mathbb{N}^* : \forall a, \gcd(a, n) = 1 \Rightarrow a^m \equiv 1 \mod n\}$

Equivalently, it is the least common multiple of the orders of the elements of the group $(\mathbb{Z}_n^\times, \times)$:

$$\lambda(n) = \operatorname{lcm}\{\operatorname{ord}_n(k) : 0 \leq k < n \wedge \gcd(k, n) = 1\}$$

where

$$\operatorname{ord}_n(k) = \min\{m \in \mathbb{N}^* : k^m \equiv 1 \mod n\}$$

2.2 Assumptions

2.2.1 Axiomatic Frameworks

Few cryptographic properties (e.g., IND–CPA or IND–CCA presented in Subsection 2.5.1) are proven from the common axioms, without additional assumptions. The most notable cryptographic primitives that gives such a guarantee is Shannon's one-time pad, which ensures perfect secrecy, but is trivially malleable. Such strong proofs are generally referred to as information-theoretic security.

Unconditional security adds the pragmatic assumption that an adversary has limited time and computation abilities. In practice, it means that unconditional security is achieved when no adversary can break the scheme in a polynomial time. That is to say, the running time is polynomial in a security parameter. Then, it is assumed than an attack would take a disproportionate amount of time, making it practically unfeasible.

This assumption might have to be updated to account for quantum cryptanalysis. Quantum computers are devices that operate on qubits. A single qubit holds a complex probability distribution over $\mathbb{F}_2$. Combining qubits is much more interesting: n qubits store a complex probability distribution over $\mathbb{F}_2^n$. In effect, quantum computers can be thought of as Turing machine with special oracles for some operations on complex

probability distributions[1]. These oracles give them an edge for some problems, which can be solved using quantum algorithms.

Regarding cryptanalysis, if quantum computers ever scale to a few thousands qubits, Shor's algorithm might break existing schemes efficiently. On the other hand, quantum properties can be used for cryptography as well, for instance with quantum key distribution (QKD). Although quantum entanglement does not bring faster-than-light communications, it does provide a synchronous source of randomness that cannot be tampered with. This is sufficient to enable parties to create cryptographic keys and communicate securely. In 2016, QUESS demonstrated QKD through an artificial satellite, and enabled the first intercontinental video call secured by quantum cryptography.

However, most of the main cryptographic primitives actually use much more specific assumptions. Although we know some constructions that reduce to a more general problem or are information-theoretic secure, they are also much less efficient.

2.2.2 Common Assumptions

Provable cryptography attempts to reduce the security properties of cryptosystems to as few security assumptions as possible. In practice however, few reductions of these assumptions are known, and many hypotheses must be studied separately. For instance, RSA uses a very specific assumption, which essentially states that RSA is secure. It is however well-known that it is easier than the factorization problem.

Factorization Problem (FP). Many cryptographic schemes rely on the hardness of factorizing composite numbers. The problem is stated below.

Definition 9 (FP). *Given $n \in \mathbb{N}^*$ determine the unique k, $(p_i)_i \in \mathbb{P}^k$, $(\alpha_i)_i \in \mathbb{N}^{*k}$ such that $n = \prod_i p_i^{\alpha_i}$.*

The existence and unicity of the solution are proven, but no efficient algorithm to find it is known. More specifically, no classical polynomial-time algorithm is known to solve the problem. Concretely, the largest semi-prime (product of two primes) ever factored consists of 768 bits [KAF+10], and the factorization required 2,000 CPU-years (see Figure 2.1).

To increase the difficulty of factoring $n = pq$, some conditions are usually put on p and q. First, cryptographic primes are "large": RSA requires primes of thousands of bits (NIST and ANSSI recommend n to be 2,048 bit long). Safe primes are prime numbers of the form $p = ap' + 1$ with p' itself a cryptographic prime. Such numbers are more resistant to factoring algorithms, and embed cyclic groups of prime order p'.

RSA Problem. However, the RSA encryption scheme does not rely just on factorization. As a reminder, the RSA encryption scheme is defined as follows:

[1] the catch is that the output is eventually a single draw from the final distribution, so quantum algorithms usually involve repeating the main procedure a set amount of times

$$
\begin{aligned}
& 1230186684530117755130494958384962720772853569595334792197322452151726400507263657518745202199786469389956474942774063845925192557326303453731548268507917026122142913461670429214311602221240479274737794080665351419597459856902143413 \\
= \; & 3347807169895689878604416984821269081770479498371376856891243138898288379387800228761471165253174308773781446799948 \\
\times \; & 3674604366679959042824463379962795263227915816434308764267603228381573966651127923337341714339681027009279873630891 7
\end{aligned}
$$

Figure 2.1: RSA-768, the largest semi-prime ever factored [KAF+10]

- KeyGen(): draw p and q two safe prime numbers, set $n = p \times q$, $e = 65{,}537$ (or some other convenient value), find d such that $ed \equiv 1 \mod n$ and set $\mathsf{pk} = (n, e)$ and $\mathsf{sk} = d$;
- Encrypt(pk, M): return $M^e \mod n$;
- Decrypt(sk, C): return $C^d \mod n$.

Instead, we must use a specific problem:

Definition 10 (RSA)**.** *Given a public RSA key (n, e) and a ciphertext c, compute m such that $c \equiv m^e \mod n$.*

This problem is not harder than the factorization problem, but the most efficient methods we know of solving the RSA problem require to first compute the factorization of n.

Discrete Logarithm Problem (DLP). Many cryptographic assumptions stem from the fact that there are groups in which it is easy to compute an exponentiation, but hard to compute a logarithm. In its most basic form, this fact gives us the discrete logarithm problem:

Definition 11 (DLP)**.** *Let G be a group. Given $g, h \in G$, compute, if it exists, x such that $h = g^x$.*

Such groups include most notably the multiplicative groups of integer modulo some prime number p, and the elliptic curves. As for FP, constraints are put on the prime numbers used, with a length of the order of 200 bits.

Diffie-Hellman Problem (DH). However, few cryptographic schemes actually rely on the DLP assumption directly. For instance, consider the Diffie-Hellman key exchange between Alice and Bob, where G is a cyclic group of prime order p:

- Alice draws $x \xleftarrow{\$} \mathbb{Z}_p$ and sends $X \leftarrow g^x \bmod p$ to Bob;
- Bob draws $y \xleftarrow{\$} \mathbb{Z}_p$, sets $k \leftarrow X^y \bmod p$ and sends $Y \leftarrow g^y \bmod p$ to Alice;
- Alice sets $k \leftarrow Y^x \bmod p$.

This gives Alice and Bob a common cryptographic key k assuming only a communication channel that guarantees integrity (but not confidentiality). This scheme is the basis for the ElGamal encryption scheme. A similar technique is used in the Digital Signature Algorithm (DSA). We do not know a reduction of its security of to the DLP. Instead, we usually derive its security, either from the hardness assumption of the Computational Diffie-Hellman (CDH) problem:

Definition 12 (CDH)**.** *Let G be a group. Given $g \in G$, g^a and g^b, compute g^{ab}.*

Or that of the Decisional Diffie-Hellman (DDH) problem (which gives indistinguishability):

Definition 13 (DDH)**.** *Let G be a group. Given $g \in G$, g^a, g^b and h, determine whether $h = g^{ab}$.*

The decisional variant is easier than the computational one which, in turn, is easier than the Discrete Logarithm Problem (DLP).

Decisional Composite Residuosity Assumption (DCRA). In fact many cryptographic schemes use specific assumptions in general. For Paillier's cryptosystem, we must introduce the concept of residues.

We say that x is a k-residue modulo n when there exists y such that $x \equiv y^k \mod n$. The most common form of residue is the quadratic residue, which in particular yields the quadratic residuosity problem (QR):

Definition 14 (QR)**.** *Given an RSA modulus $n = p \times q$ for p and q safe primes and $x \in \mathbb{Z}_n$, decide whether x is a quadratic residue modulo n.*

This is essentially equivalent to computing the Jacobi symbol $\left(\frac{x}{n}\right)$. Since Paillier's cryptosystem works modulo n^2, we use a different definition for the DCRA:

Definition 15 (DCRA)**.** *Given an RSA modulus $n = p \times q$ for p and q safe primes and $x \in \mathbb{Z}_n$, decide whether x is a n-residue modulo n^2.*

In other words, we must decide whether there exists y such that $x \equiv y^n \mod n^2$.

2.2.3 Pairings

The groups defined on elliptic curves brought an intermediate model where the DDH is easy but the CDH is hard. Such groups are referred to as Gap Diffie-Hellman (GDH) groups. In practice, this takes the form of a new operation, described as a bilinear map: the pairing.

Let us define pairings formally. For this, let $\mathbb{G}$, $\tilde{\mathbb{G}}$ and $\mathbb{G}_T$ be multiplicative groups, let g be a generator of $\mathbb{G}$ and $\tilde{g}$ be a generator of $\tilde{\mathbb{G}}$ and let $e : \mathbb{G} \times \tilde{\mathbb{G}} \to \mathbb{G}_T$. We say that e is a pairing when the following conditions are met:

- **bilinearity:** $\forall a, b \in \mathbb{Z}, e(g^a, \tilde{g}^b) = e(g, \tilde{g})^{ab}$
- **non-degeneracy:** $e(g, \tilde{g}) \neq 1_{\mathbb{G}_T}$
- **computability:** e is efficiently computable

Pairings are classified in three types.

1. **Type 1** pairings correspond to the case where there are efficiently computable morphisms $\mathbb{G} \to \tilde{\mathbb{G}}$ and $\tilde{\mathbb{G}} \to \mathbb{G}$. In this case, $\mathbb{G}$ and $\tilde{\mathbb{G}}$ can be seen as the same group. Although they are simpler to describe and more flexible, they are insecure over fields of small characteristic, and they are too costly to be practical in fields of large characteristic.

2. **Type 2** pairings correspond to the case where there are efficiently computable morphisms only from $\tilde{\mathbb{G}}$ to $\mathbb{G}$. This asymmetry means that there is still some relation between $\mathbb{G}$ and $\tilde{\mathbb{G}}$ aside the pairing itself, but it only goes one way. These pairings can be transformed into type 3 pairings [CM09].

3. **Type 3** pairings correspond to the case when there is not efficiently computable morphism either from $\mathbb{G}$ to $\tilde{\mathbb{G}}$ or from $\tilde{\mathbb{G}}$ to $\mathbb{G}$, making the two groups effectively separate.

Because type 1 pairings are either insecure or inefficient, and type 2 pairings can be reduced to type 3 pairings, we only use type 3 pairings in this document. The most common form of pairing is the Weil pairing, but the Tate pairing also exists.

The existence of pairings enables some cryptographic schemes. Although the value returned by the pairing is rarely used by itself, comparing the results of several evaluations of e can be useful. In particular, pairings allow testing relations in the exponents, without having to compute the exponents themselves (which would require solving the DLP).

Strong Diffie-Hellman (q-SDH). The Strong Diffie-Hellman assumption was introduced by Boneh and Boyen in [BB04] as the basis for the security of a signature scheme. The Strong Diffie-Hellman assumption is noted q-SDH, where q refers to the number of elements provided, not to be confused with the Static Diffie-Hellman assumption, noted SDH.

Definition 16 (q-SDH)**.** *Let $(p, \mathbb{G}, \mathbb{G}_T, e)$ define a bilinear group setting of type 1, with g a generator of $\mathbb{G}$. Given $(g^{x^i})_{0 \leq i \leq q}$ no adversary can output a pair $(w, g^{\frac{1}{x+w}})$ for $w \in \mathbb{Z}_p^*$.*

For completeness, we also provide the definition of the Static Diffie-Hellman assumption (SDH).

Definition 17 (SDH)**.** *Let $\mathbb{G}$ be a group, $g, h \in \mathbb{G}$ and $a \in \mathbb{Z}$. Given g, g^a and h, no adversary can output h^a.*

In [PS18], Pointcheval and Sanders introduced a modified version of this assumption for their signature scheme (see Subsection 2.3.3). We reproduce the first variant of the Modified Strong Diffie-Hellman, which we use in our own construction, here:

Definition 18 (q-MSDH-1)**.** *Let $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e)$ define a bilinear group setting of type 3, with g (respectively $\tilde{g}$) a generator of $\mathbb{G}$ (respectively $\tilde{\mathbb{G}}$). Given $(g^{x^i}, \tilde{g}^{x^i})_{0 \leq i \leq q}$ along with $(g^a, \tilde{g}^a, \tilde{g}^{a \cdot x})$ for $a, x \xleftarrow{\$} \mathbb{Z}_p^*$, no adversary can output a tuple $(w, P, h^{\frac{1}{x+w}}, h^{\frac{a}{P(x)}})$ for some $h \in \mathbb{G}^*$ where P is a polynomial of degree at most q and w is a scalar such that $(X + w)$ and $P(X)$ are relatively prime.*

2.3 Signatures

2.3.1 Definition

A cryptographic signature scheme with signatures $\mathcal{S}$ for messages $\mathcal{M}$ over secret keys $\mathcal{SK}$ and public keys $\mathcal{PK}$ is made of:

- $\mathsf{KeyGen}()$: draw and return $\mathsf{pk} \xleftarrow{\$} \mathcal{PK}$ and $\mathsf{sk} \xleftarrow{\$} \mathcal{SK}$;
- $\mathsf{Sign}(\mathsf{sk}, M)$: for $\mathsf{sk} \in \mathcal{SK}$ and $M \in \mathcal{M}$, return a signature $\sigma \in \mathcal{S}$ of M under key sk;
- $\mathsf{Verify}(\mathsf{pk}, M, \sigma)$: for $\mathsf{pk} \in \mathcal{MK}$, $M \in \mathcal{M}$ and $\sigma \in \mathcal{S}$, return 1 is σ is a valid signature of M under key pk, otherwise 1.

It is valid if, for any keypair $(\mathsf{pk}, \mathsf{sk})$ created by $\mathsf{KeyGen}()$, and for any message M, $\mathsf{Verify}(\mathsf{pk}, M, \mathsf{Sign}(\mathsf{sk}, M)) = 1$.

Intuitively, a signature scheme is secure when one cannot forge new signatures without the secret key. Levels of security are described by attack models, where an adversary must play a game: For instance, we might ask the adversary:

- to sign a specific message (giving universal unforgeability, or UUF);
- to choose a message before being knowing the public key, and then forging a signature (giving selective unforgeability, or SUF);
- to output any forged signature (giving existential unforgeability, or EUF).

We formally define the strongest of these guarantees, EUF–CMA.

Definition 19 (EUF–CMA)**.** *Let* $\Pi = (\mathit{KeyGen}, \mathit{Sign}, \mathit{Verify})$ *be a signature scheme for messages* $\mathcal{M}$*. For an adversary* $\mathcal{A}$ *(a probabilistic Turing Machine), we define*

$$\mathit{Adv}^{\mathit{EUF-CMA}}_{\Pi}(\mathcal{A}) = \Pr\left((\mathit{pk}, \mathit{sk}) \xleftarrow{\$} \mathit{KeyGen}(), (M^\star, \sigma^\star) \leftarrow \mathcal{A}^{\mathcal{S}}(\mathit{pk}) : \mathit{Verify}(\mathit{pk}, M^\star, \sigma^\star) = 1\right)$$

where $\mathcal{S}$ *is an oracle that, on query* $M \in \mathcal{M}$ *returns* $\mathcal{S}(M) \leftarrow \mathit{Sign}(\mathit{sk}, M)$*. We insist that* $M^\star$ *is never queried. We say that* Π *is* (t, ε)*-EUF–CMA-secure (or secure in the sense of EUF–CMA) when, for any such adversary* $\mathcal{A}$ *running in time* t*,* $\mathit{Adv}^{\mathit{EUF-CMA}}_{\Pi}(\mathcal{A}) \leq \varepsilon$*.*

Remark 2. *This definition is compatible with randomizable signature schemes because the adversary cannot query the oracle with* $M^\star$*. For other schemes, the definition might be relaxed to only requiring that* $\sigma^\star$ *not be an answer of the oracle to query* $M^\star$*.*

A slightly weaker version requires the adversary to query all the messages before the setup (the adversary is non-adaptative):

Definition 20 (EUF–wCMA)**.** *Let* $\Pi = (\mathit{KeyGen}, \mathit{Sign}, \mathit{Verify})$ *be a signature scheme for messages* $\mathcal{M}$*. For an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *(a pair of probabilistic Turing Machines), we define*

$$\mathit{Adv}^{\mathit{EUF-wCMA}}_{\Pi}(\mathcal{A}) = \Pr\left(\begin{array}{l}(M_i)_i \leftarrow \mathcal{A}_1(), (\mathit{pk}, \mathit{sk}) \xleftarrow{\$} \mathit{KeyGen}(), \sigma_i \leftarrow \mathit{Sign}(\mathit{sk}, M_i), \\ (M^\star, \sigma^\star) \leftarrow \mathcal{A}_2(\mathit{pk}, (\sigma_i)_i) : \mathit{Verify}(\mathit{pk}, M^\star, \sigma^\star) = 1\end{array}\right)$$

We say that Π *is* (t, ε)*-EUF–wCMA-secure (or secure in the sense of EUF–wCMA) when, for any such adversary* $\mathcal{A}$ *running in time* t*,* $\mathit{Adv}^{\mathit{EUF-wCMA}}_{\Pi}(\mathcal{A}) \leq \varepsilon$*.*

2.3.2 Boneh-Lynn-Shacham

One of the signature schemes with the simplest signing procedures was proposed by Boneh, Lynn and Shacham in [BLS01]. Let $\mathbb{G}$ a group of prime order p, g a generator of $\mathbb{G}$, $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ a bilinear pairing, and $\mathcal{H}$ a random oracle with image in $\mathbb{G}$. Then, the BLS signature scheme is defined by:

- $\mathsf{KeyGen}()$: draw $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p^*$, and set $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$;
- $\mathsf{Sign}(\mathsf{sk}, M)$: return $\mathcal{H}(M)^{\mathsf{sk}}$;
- $\mathsf{Verify}(\mathsf{pk}, M, \sigma)$: return 1 if $e(\sigma, g) = e(\mathcal{H}(M), g^x)$, else 0.

Unforgeability can be reduced to the difficulty of the CDH problem in $\mathbb{G}$. This scheme has been revised for asymmetric pairings in [CHKM09]. The security in the multi-user setting has been studied in [Lac18].

It has the advantage of having both a very simple key generation and signing process. Like with RSA, Sign, involves a single exponentiation. However, key generation for RSA is rather involved, and is hard to do in a distributed manner. Additionally, BLS has much shorter keys, and is thus more efficient (CPU and memory-wise). With BLS, it suffices to select a random element from $\mathbb{Z}_p^*$. On the other hand, BLS requires stronger assumptions because of its use of pairings, and the Verify procedure is more expensive.

2.3.3 Pointcheval-Sanders

The Pointcheval-Sanders signature scheme [PS16, PS18] has various interesting properties. We reproduce here the definition for the single-message version. Let $e : \mathbb{G} \times \tilde{\mathbb{G}} \to \mathbb{G}_T$ be a type-3 pairing with $\mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T$ of prime order p, and $\mathbb{G}^* = \mathbb{G} \backslash \{1_{\mathbb{G}}\}$. Then, the following procedures define the PS signature scheme.

- $\mathsf{KeyGen}()$: draw $(g, \tilde{g}, x, y) \xleftarrow{\$} \mathbb{G} \times \tilde{\mathbb{G}} \times \mathbb{Z}_p^2$, return $\mathsf{sk} = (x, y)$ and $\mathsf{pk} = (g, \tilde{g}, \tilde{X}, \tilde{Y})$ where $\tilde{X} = \tilde{g}^x$ and $\tilde{Y} = \tilde{g}^y$;
- $\mathsf{Sign}(\mathsf{sk}, M)$: draw $h \xleftarrow{\$} \mathbb{G}^*$ and return $\sigma = (h, h^{x+yM})$;
- $\mathsf{Verify}(\mathsf{pk}, M, \sigma)$: return 1 if $\sigma_1 \neq 1_{\mathbb{G}}$ and $e(\sigma_1, \tilde{X}\tilde{Y}^M) = e(\sigma_2, \tilde{g})$, else 0.

It is unforgeable in the sense of EUF–CMA under the interactive PS assumption [PS16], and in the sense of EUF–wCMA (non-adaptative) under the q-MSDH-1 assumption (see Subsection 2.2.2) [PS18]. The same levels of security are achieved when elements $g \in \mathbb{G}$ and $Y = g^y$ are included in the public key pk, as long as $X = g^x$ is kept private.

2.4 Distributed Protocols

We now present techniques that are used to design distributed protocols. Such protocols involve several parties, who can hold different roles, but who do not trust a single entity. In a centralized protocol, the trusted authority is able to single-handedly execute operations that depend on secret values. In a distributed protocol, such operations require the coordination of several parties, who share secret information. We first present a simple way that information can be shared among several parties, before describing how one party can verify the behavior of another. These two basic building blocks enable us to construct more elaborate protocols with strong correctness and confidentiality guarantees.

2.4.1 Shamir's Secret Sharing

In some settings, each party might know a secret value, and operations can allow us to combine these values to compute useful information (for instance, their sum). However, it is often useful to have a single secret value, generally the secret key of a cryptographic primitive, shared among several parties. But no single party must be able to actually know the secret value, since it would allow that party to decrypt arbitrary ciphertexts, for instance. Instead, each party should hold a *share* of the secret, and several parties should need to cooperate to reconstruct the secret. This ensure that no party knows the secret, but that, together, they can perform useful operations. Finally, explicitly reconstructing the secret exposes it again. We would generally prefer that the secret is never assembled, and that cooperating parties only reveal specific information (such as the decryption of a single ciphertext).

A common solution is Shamir's Secret Sharing [Sha79]. It splits a secret s among k parties and s can only be reconstructed when at least t parties cooperate, for $t \leq k$. Since only t shares out of k are needed, we say that this is a threshold scheme.

Let f by a random polynomial over a finite field of degree $t-1$ so that $f(0) = s$ and let us define the i-th share as $s_i = f(i)$ for $i = 1 \dots k$. Given $t-1$ shares $(s_i)_i$, no information about s is revealed. Indeed, for each possible s_0 value of s, there exists a unique polynomial g such that $g(i) = s_i$ for $i = 0 \dots k$. However, given t shares, we can recover $f(0)$ by using Lagrangian interpolation (see Figure 2.2):

$$s = \sum \lambda_i s_i \qquad \text{where} \qquad \lambda_i = \prod_{j \neq i} \frac{j}{j-i}$$

We can use this idea to conduct cryptographic operations without revealing s at any point. If each party i knows a share s_i of sk, then they can cooperate to sign a message. For instance, the BLS signature scheme from Subsection 2.3.2 uses $\mathsf{Sign}(\mathsf{sk}, M) = \mathcal{H}(M)^{\mathsf{sk}}$ where M is an element of $\mathbb{G}$, a group of prime order p, and $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p^*$. To do this, each party i computes $\sigma_i = \mathcal{H}(M)^{s_i}$ and publishes the result. Once all the parties have computed and published their shares of the signature, anyone can reconstruct $\sigma = \prod {\sigma_i}^{\lambda_i} = \mathcal{H}(M)^s$.

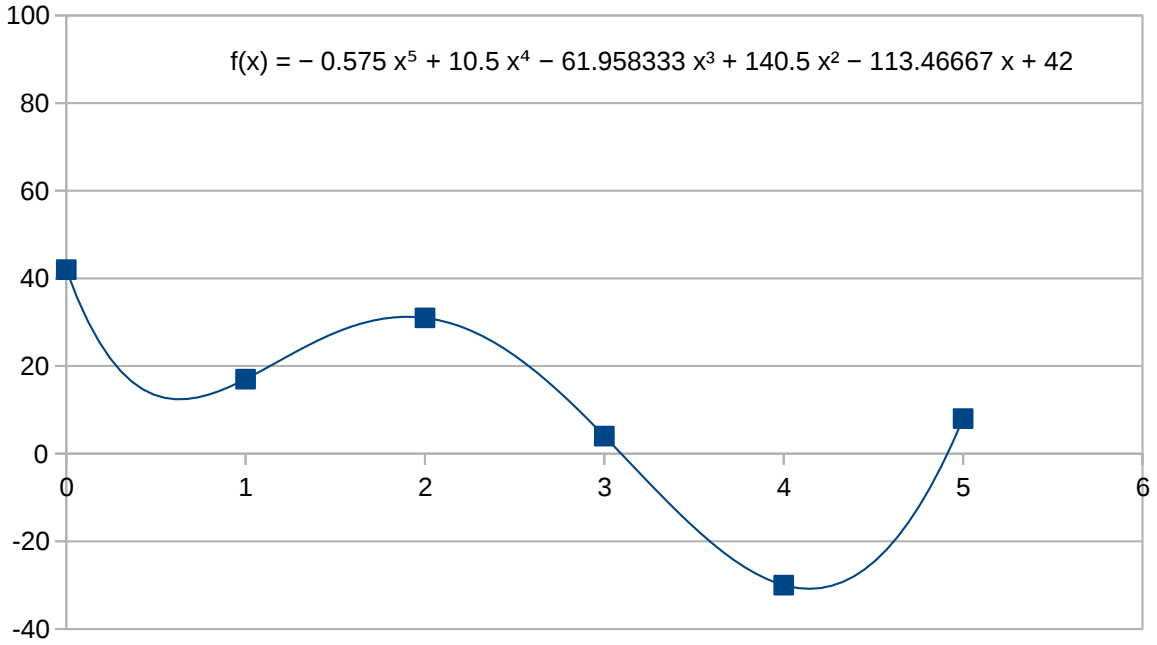


Figure 2.2: Shamir Secret's Sharing: $f(0) = 42$, $t = 6$ so six points are enough to reconstruct f, but we can pick any of them

Shamir's secret sharing is particularly useful because it is simple to understand and efficient to deploy. In general, sharing schemes are convenient in that they allow us to design protocols that require a cryptographic secret value without giving it to any one

authority. Threshold schemes are also more resilient, because they can accommodate for the case where one or several of the parties are corrupted, fail or are simply not available.

2.4.2 Zero-Knowledge Proofs (ZKPs)

Definition. A Zero-Knowledge Proof (ZKP) is a protocol between two parties, the prover $\mathcal{P}$ and the verifier $\mathcal{V}$. The parties are modelized as interactive probabilistic Turing machines running in polynomial time. The goal of the prover is to convince the verifier that some statement x belongs to some language $\mathcal{L} \in \mathsf{NP}$. A witness w is an efficiently verifiable proof that $x \in \mathcal{L}$. A ZKP verifies:

- **correctness:** if $\mathcal{P}$ knows such a w, the protocol succeeds;
- **soundness:** if $x \notin \mathcal{L}$, then no prover will succeed;
- **zero-knowledge:** the protocol leaks no information about w.

Simulation and Rewinding. A simulator is an algorithm that replaces one or several parties in a protocol and uses the Turing machines of the other parties to test their behaviors. In particular, a simulator has an advantage compared to the participants of the protocol: the simulator can rewind parties, that is to say reset them to a previous state. However, a simulator cannot directly inspect the internal state of a participant.

Extractability. A ZKP is further said to be extractable when there exists an extractor for the witness. An extractor is a simulator that runs $\mathcal{P}$ and obtains w. Extractability implies soundness, because it requires that $\mathcal{P}$ actually knows w, and thus that $x \in \mathcal{L}$. Intuitively, from the point of view of $\mathcal{P}$, the protocol executes normally, but the simulator verified that $\mathcal{P}$ knows w.

Proving Zero-Knowledge. If a simulator is able to convince $\mathcal{V}$ without knowing w, then the protocol is zero-knowledge. Intuitively, from the point of view of $\mathcal{V}$, the protocol executes normally, but $\mathcal{V}$ learned nothing about w.

Example with Graph Coloring. Let us consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is some set of *vertices* and $\mathcal{E} \subseteq \mathcal{V}^2$ is the set of the edges. The problem of graph-coloring is to find a k-coloring for $\mathcal{G}$, that is a map $c : \mathcal{V} \to \mathbb{Z}_k$ such that $\forall (a, b) \in \mathcal{E}, c(a) \neq c(b)$. See Figure 2.3 for an example.

Determining whether a graph (an instance x of the above problem) admits a k-coloring ($x \in \mathcal{L}$) is an NP-complete problem. Actually, even the problem of finding the smallest k for which $\mathcal{G}$ admits a k-coloring (its chromatic number) is also NP-complete. Let us see how we can design a ZKP around graph coloring. For this, let us assume that Alice knows some 3-coloring of $\mathcal{G}$ (a witness w that $\mathcal{G}$ admits a 3-coloring) and wants

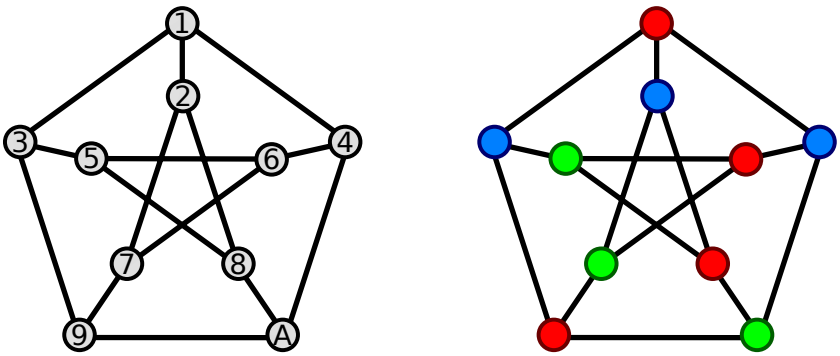


Figure 2.3: This graph admits a 3-coloring, shown on the right. We can check that, if we take any edge, its vertices of different colors. Alternatively, pairs of vertices of the same color are not in the edge set $\mathcal{E}$. The problem of 3-coloring arbitrary graphs is NP-complete. Since it is in NP, we can run a ZKP of knowledge of the coloring. Since it is NP-hard, finding the coloring is generally non-trivial.

to prove so to Bob, without revealing said 3-coloring w. Figure 2.4 rephrases the very clear description of the protocol given by Matthew Green[2].

In practice, putting hats on the colors corresponds to publishing cryptographic commitments. When Bob requests an edge to be revealed, Alice opens the two corresponding commitments. Alice repeats the protocol by combining her 3-coloring by a random permutation of the colors. For the security proofs, we can think of rewinding as traveling through time, since it corresponds to resetting the state of the other party.

- **Correctness.** If Alice indeed knows w, then she will succeed every time.
- **Soundness.** If Bob can travel through time (rewind Alice), then he can query every vertex of the graph and extract the coloring. This gives us a rewinding extractor for the witness, so the protocol is sound.
- **Zero-knowledge.** If Alice can travel through time (rewind Bob), then she can convince Bob without knowing the coloring (she chooses a random invalid coloring, and resets the protocol every time Bob asks for an edge whose vertices have the same color). This gives us a rewinding simulator that convinces the verifier, so the protocol is zero-knowledge.

Proof-of-Existence and Proof-of-Knowledge We can actually interpret a ZKP in two different ways:

1. $\mathcal{P}$ wants to convince $\mathcal{V}$ that $x \in \mathcal{L}$
2. $\mathcal{P}$ wants to convince $\mathcal{V}$ that $\mathcal{P}$ knows a witness w

[2]see https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs/

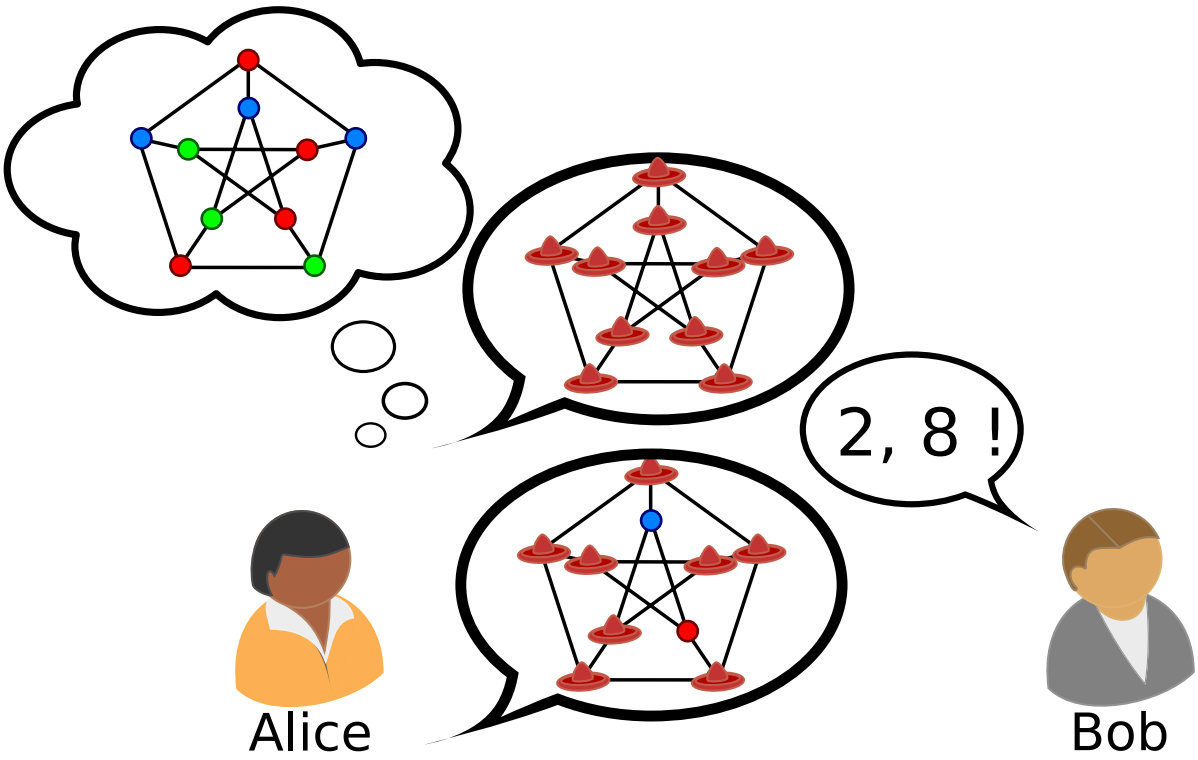


Figure 2.4: ZKP of knowledge of a graph coloring. Alice knows a 3-coloring of the graph $\mathcal{G}$ shown in Figure 2.3. She hides the color of each vertex under a hat and presents the result to Bob. He then selects an edge (see Figure 2.3 for numbering) and Alice reveals the corresponding vertices. Bob then checks that the colors of the two vertices are actually different. By repeating the protocol many times (Alice shuffles the colors at every round), Bob can be convinced that Alice indeed knows a 3-coloring. Because every time Alice only proves that the two neighboring vertices have different colors, Bob learns nothing more.

In the first case, $\mathcal{P}$ might have chosen some secret values and given commitments to $\mathcal{V}$, but $\mathcal{V}$ will only accept them if they verify some conditions, which $\mathcal{P}$ can ensure using a ZKP. If we assume the hardness of the DLP, then ZKPs can help prove relations between values. For instance, we might want to prove that y_1 and y_2 have the same discrete logarithms, that is to say that $y_1 = g_1^x$ and $y_2 = g_2^x$ for some integer x, without revealing x. Here, it might not even be important that $\mathcal{P}$ knows x, as long as it exists. Thus, such proofs are sometimes referred to as Zero-Knowledge Proofs-of-Existence, or ZKPoEs.

In the second case, $\mathcal{V}$ might already know that $x \in \mathcal{L}$ (i.e., $\mathcal{V}$ knows a witness w'), but checks that $\mathcal{P}$ also knows this (i.e., $\mathcal{P}$ knows a witness w) by running a ZKP (without using w'). An ingenious way that this is used is to ensure that $\mathcal{P}$ generates random values correctly. Again using the DLP, assume that $\mathcal{P}$ is supposed to draw a random $x \xleftarrow{\$} \mathbb{Z}_p$, and to output $y \leftarrow g^x$. *A priori*, it is hard to distinguish between an honest $\mathcal{P}$, and a

prover who directly draws $y \xleftarrow{\$} G$. However, we can test that $\mathcal{P}$ knows x by running a ZKP. Here, x is the witness of the ZKP. Because computing the discrete logarithm of y is hard, $\mathcal{P}$ will need to have first computed x, thus giving a solid guarantee on the way y was generated. In this type of situation, we will talk about Zero-Knowledge Proof of Knowledge, or ZKPoK.

In cryptographic protocols, the most common properties that must be proven involve discrete logarithms. For instance, it can be useful to prove knowledge of the discrete logarithm of a public value (yielding a signature scheme). Even more often, we need to prove relations between two discrete logarithms. Below, we present several protocols for this class of problem.

Schnorr Protocol [Sch90]. Let us consider the first case, where a prover $\mathcal{P}$ wants to prove to some verifier $\mathcal{V}$ knowledge of x such that $y = g^x$ for some $g \in G$ of prime order p. The Schnorr protocol works as follows.

1. $\mathcal{P}$ picks $r \xleftarrow{\$} \mathbb{Z}_p$ and sends the commitment $t = g^r$ to $\mathcal{V}$;
2. $\mathcal{V}$ picks a challenge $c \xleftarrow{\$} \mathbb{Z}_p$ and sends it to $\mathcal{P}$;
3. $\mathcal{P}$ computes the proof $s \leftarrow r + cx \bmod p$ and sends it to $\mathcal{V}$;
4. $\mathcal{V}$ verifies that $g^s = t \times y^c$.

If $\mathcal{P}$ does not know such a x, the Schnorr protocol above can only be completed with negligible probability. As it is, the Schnorr protocol only provides the following guarantees:

- **correctness:** as long as $\mathcal{P}$ knows x, the protocol succeeds;
- **soundness:** by rewinding $\mathcal{P}$, the simulator can obtain $s_1 = r + c_1 x$ and $s_2 = r + c_2 x$ for two different challenges c_1 and c_2, then $x = \frac{s_2 - s_1}{c_2 - c1} \mod p$;
- **zero-knowledge:** *if we assume that* $\mathcal{V}$ *is honest*, we can expect that c is actually chosen uniformly at random, that is, it depends only on random bits provided by the simulator; with this, the simulator can obtain c, rewind $\mathcal{V}$ to set $t = g^s y^{-c}$ for some arbitrary value s and convince $\mathcal{V}$.

The assumption done for proving zero-knowledgeness is crucial: the Schnorr Protocol is only known to be Honest Verifier Zero-Knowledge (HVZK). Fortunately, the values provided by the verifier in the protocol are only random values. This means that we can actually force the verifier to be honest by simulating it with a random oracle.

Fiat-Shamir Heuristic [FS87, PS96]. The Fiat-Shamir heuristic proposes the idea of removing some interactions in cryptographic protocols by having random values be generated by random oracles. Such variants are said to be non-interactive, and we use the prefix "NI-" for short. There are NI-ZKPs, NI-ZKPoKs and NI-ZKPoEs. To ensure that the value is not derived from previous values (essentially what the simulator did to convince $\mathcal{P}$ when we proved the HVZK of the Schnorr protocol), the random oracle should be given the previous transcript as parameter. Let us make the Schnorr protocol presented above non-interactive by using the Fiat-Shamir heuristic with a random oracle $\mathcal{H}$:

1. $\mathcal{P}$ picks $r \xleftarrow{\$} \mathbb{Z}_p$ and sets $t = g^r$;
2. $\mathcal{P}$ computes $c \leftarrow \mathcal{H}(g|y|t)$;
3. $\mathcal{P}$ computes the proof $s \leftarrow r + cx \bmod p$ and publishes (y, t, s);
4. $\mathcal{V}$ verifies sets $c \leftarrow \mathcal{H}(g|y|t)$ and checks that $g^s = t \times y^c$.

What guarantees do we have this time?

- **correctness:** again, as long as $\mathcal{P}$ knows x, the protocol succeeds;
- **soundness:** the simulator controls $\mathcal{H}$; it rewinds $\mathcal{P}$, which thus queries $\mathcal{H}$ twice, and the simulator returns two different challenges c_1 and c_2, making $\mathcal{P}$ publish again $s_1 = r + c_1 x$ and $s_2 = r + c_2 x$, then $x = \frac{s_2 - s_1}{c_2 - c1} \mod p$;
- **zero-knowledge:** again, the simulator controls $\mathcal{H}$ and can trivially choose an arbitrary value s and then set c conveniently to convince $\mathcal{V}$.

Remark 3. *As an optimization, $\mathcal{P}$ can send the scalar value c instead of the group element t. The former can often be more compact than the later. Then, $\mathcal{V}$ recovers t as $g^s y^{-c}$ and checks that $c = \mathcal{H}(g|y|t)$.*

Chaum-Pedersen Protocol [CP93]. Now, let us consider the second case, where a prover $\mathcal{P}$ wants to prove to some verifier $\mathcal{V}$ that $\log_{g_1} y_1 = \log_{g_2} y_2$. In fact, we simply need to adapt the Schnorr protocol to prove knowledge of x such that both $y_1 = g_1^x$ and $y_2 = g_2^x$. This protocol works in a cyclic group G of known prime order q. Thus, $x \in \mathbb{Z}_q$ and $g_1, g_2, y_1, y_2 \in G$. The definition is very similar to that of the Schnorr protocol:

- $\mathcal{P}$ picks $u \xleftarrow{\$} \mathbb{Z}_q$ and sends commitments $t_1 \leftarrow g_1^u$ and $t_2 \leftarrow g_2^u$;
- $\mathcal{V}$ sends a challenge $h \xleftarrow{\$} \mathbb{Z}_q$;
- $\mathcal{P}$ returns $w \leftarrow u - hx \bmod q$;
- $\mathcal{V}$ checks that $g_1^w = t_1 y_1^{-h}$ and that $g_2^w = t_2 y_2^{-h}$.

As for the Schnorr protocol, this protocol is correct and sound, but is only HVZK. Again, we can fortunately replace the interactions with the verifier by a random oracle. Here, h should be computed as $\mathcal{H}(g_1|g_2|y_1|y_2|t_1|t_2)$. When we use the Fiat-Shamir heuristic, the protocol becomes zero-knowledge.

Remark 4. *Again, as an optimization, $\mathcal{P}$ can send c instead of t_1 and t_2.*

Girault-Poupard-Stern Protocol [GPS06]. Until here, we assumed that the order of the group was known. This works well for many settings, such as working modulo a prime number. However, in this work, we make use of the Paillier cryptosystem, which works modulo a composite number with a secret factorization. This means that the order of the group is actually not know by the participants, unless they know the secret key.

Following Girault's [Gir91] work, Poupard and Stern [PS98, PS99] considered the problem of working in groups of unknown order. This resulted in the Girault-Poupard-Stern protocol that we present below. The general idea is to use larger groups. The difference in size should be enough to drown any information in a cryptographic amount of noise. Let us assume that a prover $\mathcal{P}$ wants to prove to some verifier $\mathcal{V}$ that $\log_{g_1} y_1 = \log_{g_2} y_2$. We work in G of unknown order q, and $\mathcal{P}$ actually proves knowledge of $x \in \mathbb{Z}_q$ such that $y_1 = g_1^x$ and $y_2 = g_2^x$ for $g_1, g_2, y_1, y_2 \in G$. Additionally, we assume that there is a security parameter κ and a publicly known upper bound Q of q. That is: $Q > q$. Then, the Chaum-Pedersen protocol can be adapted as follows.

- $\mathcal{P}$ picks $u \xleftarrow{\$} \mathbb{Z}_{2^{2\kappa}Q}$ and sends commitments $t_1 \leftarrow g_1^u$ and $t_2 \leftarrow g_2^u$;
- $\mathcal{V}$ sends a challenge $h \xleftarrow{\$} \mathbb{Z}_{2^\kappa}$;
- $\mathcal{P}$ returns $w \leftarrow u - hx$;
- $\mathcal{V}$ checks that $0 < w < 2^{2\kappa}Q$, and both $g_1^w = t_1 y_1^{-h}$ and $g_2^w = t_2 y_2^{-h}$.

Since w falls outside of the correct set with negligible probability, the prover simply restarts the proof when w is wrong. As usual, this protocol is correct, sound and HVZK, and the protocol can be made non-interactive with the Fiat-Shamir heuristic. Here, $h = \mathcal{H}(g_1|g_2|y_1|y_2|t_1|t_2)$.

Remark 5. *Still, as an optimization, $\mathcal{P}$ can send c instead of t_1 and t_2.*

2.5 Encryption

2.5.1 Asymmetric Encryption

A symmetric encryption scheme from messages $\mathcal{M}$ to ciphertexts $\mathcal{C}$ over secret keys $\mathcal{SK}$ is made of:

- $\mathsf{KeyGen}()$: draw and return $\mathsf{sk} \xleftarrow{\$} \mathcal{SK}$;

- $\mathsf{Encrypt}(\mathsf{sk}, M)$: for $\mathsf{sk} \in \mathcal{SK}$ and $M \in \mathcal{M}$, encrypt M into ciphertext C;
- $\mathsf{Decrypt}(\mathsf{sk}, C)$: for $\mathsf{sk} \in \mathcal{SK}$ and $C \in \mathcal{C}$, decrypt C into message M.

It is valid if, for any secret key sk created by $\mathsf{KeyGen}()$, and for any message M, $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{Encrypt}(\mathsf{sk}, M)) = M$.

An asymmetric encryption scheme from messages $\mathcal{M}$ to ciphertexts $\mathcal{C}$ over public keys $\mathcal{PK}$ and secret keys $\mathcal{SK}$ is made of:

- $\mathsf{KeyGen}()$: draw and return $\mathsf{pk} \xleftarrow{\$} \mathcal{PK}$ and $\mathsf{sk} \xleftarrow{\$} \mathcal{SK}$;
- $\mathsf{Encrypt}(\mathsf{pk}, M)$: for $\mathsf{pk} \in \mathcal{PK}$ and $M \in \mathcal{M}$, encrypt M into ciphertext C;
- $\mathsf{Decrypt}(\mathsf{sk}, C)$: for $\mathsf{sk} \in \mathcal{SK}$ and $C \in \mathcal{C}$, decrypt C into message M.

Implicitly, sk includes pk. It is valid if, for any keypair $(\mathsf{pk}, \mathsf{sk})$ created by $\mathsf{KeyGen}()$, and for any message M, $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{Encrypt}(\mathsf{pk}, M)) = M$.

Intuitively, an encryption scheme is secure when C gives no information about M. Levels of security are described by attack models, where an adversary must play a game: the adversary interacts with a simulator in a probabilistic protocol and aims to fulfill a particular condition. If no adversary can succeed in the given time (usually, polynomial) with non-negligible probability, then the scheme is considered secure for this particular security level.

In its most basic form, the adversary must guess M given C. However, partial information might leak even though no adversary can recover the complete message. To address this, we aim for the security guarantee of indistinguishability. This is usually described by letting the adversary $\mathcal{A}$ choose some messages M_0 and M_1, giving $\mathcal{A}$ $C \leftarrow \mathsf{Encrypt}(\mathsf{pk}, M_b)$ for $b \xleftarrow{\$} \{0, 1\}$, and asking $\mathcal{A}$ to guess b. If no adversary can guess b with non-negligible probability, then the scheme is said to be indistinguishable (IND).

Ciphertext-Only Attacks bring the notion of $\mathsf{IND–COA}$. In this attack model, the adversary must guess b given only C. This attack model already puts some constraints on the encryption scheme. If ciphertexts can be tested for plaintext-equality (this is the case for deterministic encryption schemes), an attacker can always attempt an exhaustive search. More specifically, the adversary can try to compute $C' \leftarrow \mathsf{Decrypt}(\mathsf{sk}, M)$ for every $M \in \mathcal{M}$, and compare C' to C. This can be done efficiently when $\mathcal{M}$ is small. Thus, encryption schemes must be probabilistic and counter comparisons between ciphertexts, or have a large message-space (more specifically, the min-entropy of the message-space must be high).

However, encryptions of other messages can reveal partial information about the secret key sk, which in turn can leak information about M. Thus, in Known-Plaintext Attacks ($\mathsf{IND–KPA}$), the adversary is given a list of ciphertexts and their underlying messages. However, an attacker is sometimes able to actually choose, or at least influence, which message-ciphertext pairs are leaked. This is trivially the case for asymmetric encryption schemes (the adversary can encrypt arbitrary messages). This brings us to chosen attack models.

In such attacks, the adversary is given access to an oracle, a function that gives specific abilities. When the adversary can encrypt messages (through an oracle for symmetric encryption), we obtain the notion of Chosen-Plaintext Attacks (IND–CPA). For an even stronger security level, Chosen-Ciphertext Attacks (IND–CCA) give an oracle to decrypt arbitrary ciphertexts (except the target). Finally, letting the adversary query the oracles after being given M_b or not corresponds to IND–CCA1 and IND–CCA2 respectively. We define IND–CPA and IND–CPA formally below.

Definition 21 (IND–CPA)**.** *Let* $\Pi = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *be an asymmetric encryption scheme from* $\mathcal{M}$ *to* $\mathcal{C}$*. For an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *(a pair of probabilistic Turing Machines), we define*

$$\mathsf{Adv}^{\mathsf{IND\text{–}CPA}}_{\Pi}(\mathcal{A}) = \left| \Pr \begin{pmatrix} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(), (m_0, m_1) \leftarrow \mathcal{A}_1(\mathsf{pk}), \\ b \xleftarrow{\$} \{0,1\}, C_b \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m_b), b' \leftarrow \mathcal{A}_2(\mathsf{pk}, C_b) : b' = b \end{pmatrix} - \frac{1}{2} \right|$$

We say that Π *is* (t, ε)*-IND–CPA-secure (or secure in the sense of IND–CPA) when, for any such adversary* $\mathcal{A}$ *running in time* t*,* $\mathsf{Adv}^{\mathsf{IND\text{–}CPA}}_{\Pi}(\mathcal{A}) \leq \varepsilon$*.*

Definition 22 (IND–CCA1)**.** *Let* $\Pi = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *be an asymmetric encryption scheme from* $\mathcal{M}$ *to* $\mathcal{C}$*. For an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, we define*

$$\mathsf{Adv}^{\mathsf{IND\text{–}CCA1}}_{\Pi}(\mathcal{A}) = \left| \Pr \begin{pmatrix} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(), (m_0, m_1) \leftarrow \mathcal{A}_1^{\mathcal{D}}(\mathsf{pk}), \\ b \xleftarrow{\$} \{0,1\}, C_b \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m_b), b' \leftarrow \mathcal{A}_2(\mathsf{pk}, C_b) : b' = b \end{pmatrix} - \frac{1}{2} \right|$$

where $\mathcal{D}$ *is an oracle that, on query* $C \in \mathcal{C}$ *returns* $\mathcal{D}(C) \leftarrow \mathsf{Decrypt}(\mathsf{sk}, C)$*. We say that* Π *is* (t, ε)*-IND–CCA1-secure (or secure in the sense of IND–CCA1) when, for any such adversary* $\mathcal{A}$ *running in time* t*,* $\mathsf{Adv}^{\mathsf{IND\text{–}CCA1}}_{\Pi}(\mathcal{A}) \leq \varepsilon$*.*

Definition 23 (IND–CCA2)**.** *Let* $\Pi = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *be an asymmetric encryption scheme from* $\mathcal{M}$ *to* $\mathcal{C}$*. For an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, we define*

$$\mathsf{Adv}^{\mathsf{IND\text{–}CCA2}}_{\Pi}(\mathcal{A}) = \left| \Pr \begin{pmatrix} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(), (m_0, m_1) \leftarrow \mathcal{A}_1^{\mathcal{D}}(\mathsf{pk}), \\ b \xleftarrow{\$} \{0,1\}, C_b \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m_b), b' \leftarrow \mathcal{A}_2^{\mathcal{D}}(\mathsf{pk}, C_b) : b' = b \end{pmatrix} - \frac{1}{2} \right|$$

where $\mathcal{D}$ *is an oracle that, on query* $C \in \mathcal{C}$ *returns* $\mathcal{D}(C) \leftarrow \mathsf{Decrypt}(\mathsf{sk}, C)$*. We insist that* C_b *is never queried (note that* Adv *has access to* $\mathcal{D}$ *even after getting the challenge* C_b*). We say that* Π *is* (t, ε)*-IND–CCA2-secure (or secure in the sense of IND–CCA2) when, for any such adversary* $\mathcal{A}$ *running in time* t*,* $\mathsf{Adv}^{\mathsf{IND\text{–}CCA2}}_{\Pi}(\mathcal{A}) \leq \varepsilon$*.*

These attack models give relatively strong security guarantees, but practical attacks have been found that bypass them by violating the model. For example, these models assume completely unrelated keys, so that information about one keypair gives no information about another. However, there are situations where encryption schemes are used with several related keys. In these cases, attackers were able to attack on practical implementations of these schemes due to theoretical security flaws. Infamously,

Input: $x \in G$ and $e \in \mathbb{N}$
Output: x^e
$y \leftarrow 1$;
while $n > 0$ **do**
  `/* multiply */`
  if $n \bmod 2 = 1$ **then**
    $y \leftarrow x \times y$;
  end
  `/* square */`
  $x \leftarrow x \times x$;
  $n \leftarrow \lfloor \frac{n}{2} \rfloor$;
end
return y

Algorithm 1: Square-and-Multiple

this has brought the complete compromise of the security algorithm WEP for Wi-Fi communications, which can now be broken easily.

Alternatively, side-channel attacks break the Turing machine abstraction and observe the behavior of the devices that run the cryptographic schemes. In their simplest versions, such attacks measure the amount of time required to run the computations (timing attack). Because the various operations and different data patterns can lead to slightly different execution times, attackers are able to deduce information about the internal state. When this information is correlated with the secret key, they are able to derive the secret key. More sophisticated attacks will measure the power consumption of the device throughout its operation (power-monitoring attack). Even more advanced techniques will observe other physical properties, such as the radiated electromagnetic field (electromagnetic attack), or even sound variations (acoustic attack). Timing attacks can be countered by using algorithms whose running times do not depend on secret values. For instance, for exponentiation the square-and-multiply algorithm has a running time that depends on the exponent (see Algorithm 1); Montgomery offers a solution whose running time is independent from the exponent (see Algorithm 2).

2.5.2 Homomorphic Encryption

A homomorphic encryption scheme is an encryption scheme such that $\mathsf{Encrypt}$ is a group homomorphism: $\mathsf{Encrypt}(m_1) \oplus \mathsf{Encrypt}(m_2) = \mathsf{Encrypt}(m_1 + m_2)$ for some binary operations $+$ over $\mathcal{M}$ and $\oplus$ over $\mathcal{C}$. We will focus on asymmetric homomorphic encryption schemes. As common examples, RSA encryption is endomorphic over $(\mathbb{Z}_N, \times)$ and the Paillier cryptosystem is homomorphic from $(\mathbb{Z}_n, +)$ to $(\mathbb{Z}_{n^2}, \times)$.

In a fully homomorphic encryption (FHE) scheme, $\mathsf{Encrypt}$ is a ring homomorphism (in addition to being a group homomorphism). In other words, we again have the property $\mathsf{Encrypt}(m_1) \oplus \mathsf{Encrypt}(m_2) = \mathsf{Encrypt}(m_1 + m_2)$ but we additionally have that $\mathsf{Encrypt}(m_1) \otimes \mathsf{Encrypt}(m_2) = \mathsf{Encrypt}(m_1 \times m_2)$. Here, $+$, $\times$, $\oplus$ and $\otimes$ are binary

Input: $x \in G$ and $e \in \mathbb{N}$
Output: x^e
$x_1 \leftarrow x$;
$x_2 \leftarrow x \times x$;
while $n > 0$ **do**
  if $n \bmod 2 = 1$ **then**
    $x_1 \leftarrow x_1 \times x_2$ `// multiply`
    $x_2 \leftarrow x_2 \times x_2$ `// square`
  else
    $x_2 \leftarrow x_1 \times x_2$ `// multiply`
    $x_1 \leftarrow x_2 \times x_2$ `// square`
  end
  $n \leftarrow \lfloor \frac{n}{2} \rfloor$;
end
return x_1

Algorithm 2: Montgomery's Ladder [Mon87], or Square-and-Multiply-Always: in both conditional branches, the operations are the same; only the variable used to store the result changes

operations such that $\times$ is distributive with respect to $+$ and $\otimes$ is distributive with respect to $\oplus$. With FHE, we can implement the NAND gate, and thus execute arbitrary circuits.

In existing FHE schemes, ciphertexts are associated with some level of noise. This noise increases significantly at each execution of the $\otimes$ operation ($\oplus$ introduces little noise). If too much noise is added, the message cannot be recovered. Fortunately, a method was found to remove this noise: *bootstrapping*. However, this is an extremely costly operation, which must be used regularly in FHE schemes. Thus, currently known solutions are still particularly slow. Although an overhead is expected when switching from the plaintext domain to the ciphertext domain, such a significant slowdown can discourage people considering using of cryptography to secure information.

2.5.3 ElGamal Encryption

The ElGamal encryption scheme was published in 1984 [ElG84] and remains convenient for its relative efficiency and interesting properties. Let us consider a group G of order p generated by some element g, and let $x \in \mathbb{Z}_p$. Then, the ElGamal encryption scheme is defined by the following algorithms.

- $\mathsf{KeyGen}()$: draw $x \xleftarrow{\$} \mathbb{Z}_p$, set $\mathsf{pk} = g^x$, $\mathsf{sk} = x$;
- $\mathsf{Encrypt}(\mathsf{pk}, M)$: draw $r \xleftarrow{\$} \mathbb{Z}_p$, return $C \leftarrow (g^r, \mathsf{pk}^r \cdot M)$;
- $\mathsf{Decrypt}(\mathsf{sk}, C)$: return $M \leftarrow C_2 \cdot C_1^{-\mathsf{sk}}$.

One can check that this is valid encryption scheme. Its security relies on the Diffie-Hellman assumptions. More specifically, under the CDH assumption, Encrypt is a one-way function. Also, under the DDH assumption, the ElGamal encryption scheme is secure in the sense of IND–CPA. However, the very fact that it is homomorphic makes it malleable, and thus, it is not secure under IND–CCA.

Multiplicatively Homomorphic ElGamal. To see that this scheme is homomorphic, let us consider two ciphertexts $C = \mathsf{Encrypt}(\mathsf{pk}, M)$ and $C' = \mathsf{Encrypt}(\mathsf{pk}, M')$. Then, we define the following operation over $\mathcal{C}$: $C'' = C \cdot C' = (C_1 \cdot C'_1, C_2 \cdot C'_2)$. Under these assumptions, we can verify that $C'' = (g^{r+r'}, g^{x(r+r')} \cdot (M \cdot M'))$ is a valid encryption for $M'' = M \cdot M'$. So, the ElGamal encryption scheme is homomorphic from $(G, \cdot)$ to $(G^2, \cdot)$.

Additively Homomorphic ElGamal. This multiplicative homomorphic property is interesting, but computations rely on addition more often than on multiplication. A common variant of the ElGamal encryption scheme is thus the following.

- $\mathsf{KeyGen}()$: draw $x \xleftarrow{\$} \mathbb{Z}_p$, set $\mathsf{pk} = g^x$, $\mathsf{sk} = x$;
- $\mathsf{Encrypt}(\mathsf{pk}, M)$: draw $r \xleftarrow{\$} \mathbb{Z}_p$, return $C \leftarrow (g^r, \mathsf{pk}^r \cdot g^M)$;
- $\mathsf{Decrypt}(\mathsf{sk}, C)$: search M such that $g^M = C_2 \cdot C_1^{-\mathsf{sk}}$ and return M.

The Decrypt operation needs to perform an exhaustive search over $\mathcal{M}$ to break the DLP. Thus, this only works when we can make the assumption that the message space is small. This can work in various situations, but usually prohibits decrypting masked values. In any case, we can easily verify that this scheme is homomorphic from $(G, +)$ to $(G^2, \cdot)$ using the definition of $\cdot$ in G^2 used above.

2.5.4 Paillier's Encryption

Although the ElGamal encryption scheme is efficient, it is unsatisfying in certain situations that require a complete additively homomorphic property. Paillier's encryption, introduced in 1999 [Pai99], offers a "natively" additively homomorphic scheme. We need this scheme in our implementation of an online voting system to be able to mask values and decrypt the results in our Secure Multiparty Computation protocols.

Encryption Scheme. For clearer notation, we identify $\lambda = \lambda(n)$ and $\varphi = \varphi(n)$. Paillier's encryption scheme is defined by:

- $\mathsf{KeyGen}()$: p, $q \xleftarrow{\$}$ safe primes, $n \leftarrow pq$, $g \in \mathbb{Z}^*_{n^2}$, set $\mathsf{pk} = (n, g)$, $\mathsf{sk} = \lambda$;
- $\mathsf{Encrypt}(\mathsf{pk}, M)$: draw $r \xleftarrow{\$} \mathbb{Z}^*_n$, return $C \leftarrow g^M r^n \bmod n^2$;
- $\mathsf{Decrypt}(\mathsf{sk}, C)$: return $M \leftarrow \frac{L(C^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \bmod n$ where $L(x) = \frac{x-1}{n}$.

Seeing that this is a valid encryption scheme is less obvious than for the ElGamal encryption scheme, so we refer to the original paper [Pai99] for the complete description. The general idea is that knowing the secret value λ gives the ability to compute discrete logarithms in $\mathbb{Z}_{n^2}$. Again, this scheme cannot be secure in the sense IND–CCA because of its homomorphic properties, but it is secure in the sense of IND–CPA under DCRA (see Subsection 2.2.2).

Homomorphic Properties. Working in $\mathbb{Z}_{n^2}$ is much more expensive than working in $\mathbb{Z}_p$, but Pailler's cryptosystem is homomorphic from $(\mathbb{Z}_n, +)$ to $(\mathbb{Z}_{n^2}, \times)$ without restriction on $\mathcal{M}$. Indeed, for ciphertexts $C_1 \leftarrow \mathsf{Encrypt}(\mathsf{pk}, M_1) \leftarrow g^{M_1} r_1^n \bmod n^2$ and $C_2 \leftarrow \mathsf{Encrypt}(\mathsf{pk}, M_2) \leftarrow g^{M_2} r_2^n \bmod n^2$, then:

$$C_1 \times C_2 \equiv g^{M_1+M_2}(r_1 \times r_2)^n \mod n^2$$

Thus, $C_1 \times C_2$ is a valid Paillier encryption for $M_1 + M_2$ under key pk. The result might need to be re-randomized before publication by drawing $r_3 \xleftarrow{\$} \mathbb{Z}_n^*$ and using $C_1 \times C_2 \times r_3^n \bmod n^2$. Otherwise, information might leak with the ciphertext. For a simple example, if $C_1 \leftarrow \mathsf{Encrypt}(\mathsf{pk}, a)$ and we publish C_1 and $C_2 = C_1^a$, then it is possible to conduct an exhaustive search to find a.

Threshold Decryption. We might want to share the secret value λ among several parties so that no one entity knows sk. However, with the traditional scheme, it means that the parties must compute and publish both C^λ and g^λ, which reveals more information than needed. Instead, we consider the particular case for $g = 1 + n$. In this situation, $\mathsf{Encrypt}(\mathsf{pk}, M)$ returns $C \leftarrow r^n \bmod n$ for some random value r. So we can set $\mathsf{sk} \leftarrow s \leftarrow n^{-1} \bmod \varphi$ and instead define $\mathsf{Decrypt}$ as:

- $\mathsf{Decrypt}(\mathsf{sk}, C)$: set $R \leftarrow C^s \bmod n \leftarrow r \bmod n$, and return $M \leftarrow \frac{(CR^{-n} \bmod n^2)-1}{n}$.

With this definition, s is drawn from $[0, \varphi)$, which is statistically indistinguishable from $[0, n)$ [Sho00].

Since the secret value is only used in a single exponentiation, it makes it much easier to use it in a distributed way. So s can be distributed over k authorities by Shamir Secret Sharing method. However, we do require another trick, since the parties do not individually know φ, and thus cannot efficiently compute inverses in the exponents. The idea is simply to multiply exponents by $\Delta = k!$ to always work with integers. Let P of degree $t-1$ in $\mathbb{Z}_\varphi$ so that $P(0) = s$, and set $s_i = P(i)$ for the authority i. With a set S of t authorities, we recover the secret by computing $\Delta \cdot s = \sum_{i \in S} \lambda_i^S \cdot s_i$ where:

$$\lambda_i^S = \left(\prod_{j \in S \setminus \{i\}} j \right) \times \left(\frac{\Delta}{\prod_{j \in S \setminus \{i\}} (j - i)} \right) \text{ an integer}$$

Given C, each authority computes $R_i \leftarrow C^{s_i} \bmod n$. Together, they combine these shares into $R' \leftarrow \prod_{i \in S} R_i^{\lambda_i^S} = C^{\Delta s} \bmod n$. Because Paillier's encryption scheme is additively homomorphic, if we set $C' \leftarrow C^\Delta \bmod n^2$ and $M' \leftarrow (C' R'^{-n} \bmod n^2 - 1)/n$, we have that $M' = M\Delta \mod n$. From this, any subset of t authorities can compute M.

Threshold Verifiability. To ensure correctness, it is necessary that each authority proves that $R_i \equiv C^{s_i} \mod n$. For this, we choose v in the setup, a generator of Q_n, the cyclic group of the quadratic residues in $\mathbb{Z}_n^*$. Each authority initially publishes $v_i \leftarrow v^{s_i} \mod n$. When decrypting, they provide a NIZK proof of knowledge of s_i such that $R_i \equiv C^{s_i} \mod n$ and $v_i \equiv v^{s_i} \mod n$ using the Girault-Poupard-Stern Protocol (see Subsection 2.4.2).

3 A New Primitive for Pairwise Matching

As a practical starting point, let us consider the problem of matching organ donors with recipients. This use case seems challenging to solve using classical cryptographic tools due to its constraints. In this chapter, we present a new type of cryptographic primitive tailored for this type of application. These results were presented in [CPST18b].

3.1 Organ Donation

3.1.1 Compatibility for Organ Transplant

Senescence, injury or illness can deteriorate tissues to the point of causing organ failure. This can lead to discomfort, pain, and a dramatically reduced lifespan. Sometimes, people die in such ways that their organs, or some of their organs, are preserved. Then, an organ can be removed from the body, and transplanted to a patient who needs it. This can be the case for cardiac arrests, on the condition that the removal be done within a few hours after circulatory death. Alternatively, the circulatory system of brain dead individuals is still perfectly functioning. It makes it easier to collect the organs since the allowed time frame is large in this case.

However, the body of the recipient might reject the graft. To avoid this, organ transplants are only done under some conditions, such as between donors and recipients of the same blood type.

To treat as many people as possible, people in need for a new organ are registered on recipient lists along with their medical records. Whenever someone dies in a way that enables some organs to be transplanted (good medical condition, and consent), the medical record of the deceased is compared against these lists. If there exists a recipient with a compatible medical record, then a transplant can be organized.

Moreover, the human body contains two kidneys, but can live with one. There is also the possibility of only collecting part of an organ (pancreas, lungs, intestines). For these situations, live transplants are possible. They are simpler regarding the time frame, but they also let the donor join the recipient to simplify long-distance transplants. Most people would oppose giving an organ while they are alive, unless if it were for the benefit of a close relative. This greatly restricts the pool for potential matches (see Figure 3.1).

To relax these constraints, the common approach is to pair a recipient with a donor on the sole criterion of consent, disregarding medical incompatibility. Then, the problem consists in finding two donor/recipient pairs that are mutually compatible (the donor of

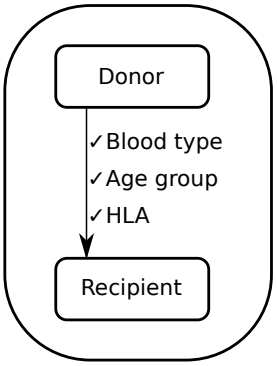


Figure 3.1: Single Matching: the donor is willing to give one kidney to the recipient and they are medically compatible

a pair is compatible with the recipient of the other pair). This cross-matching makes it so that each donor accepts to give an organ to the other recipient (see Figure 3.2). This construction aligns the interests of both donors and considerably enlarges the pool of potential matches.

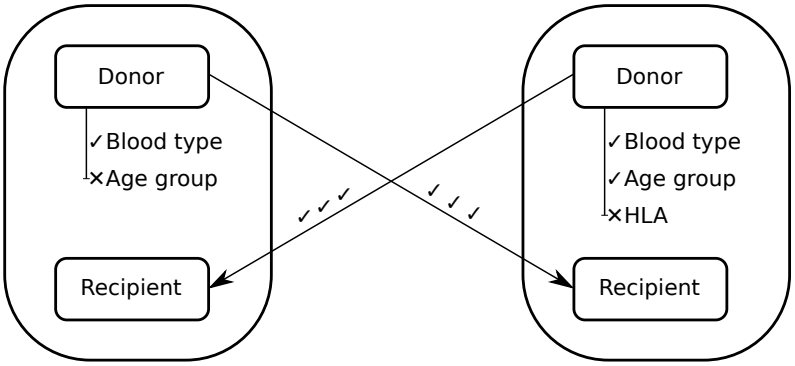


Figure 3.2: Pair Matching. The donor from the left pair is willing to give a kidney to the recipient from the left pair. The donor from the right pair is willing to give a kidney to the recipient from the right pair. However they are not medically compatible. Instead, the donor from the left pair can give to the recipient of the right pair and the donor from the right to the recipient from the left. By doing so, each recipient receives a kidney from the other's donor thanks to their own donors.

We can improve upon this idea by considering the directed graph over donor-recipient pairs, where we draw an edge from a donor to a compatible recipient (see Figure 3.3). Then, when we finding simple cycles in this graph, we can align the incentives of all the corresponding donors.

Such solutions are currently deployed, but require access to medical records. This sensitive information is usually managed by non-profit entities (e.g., Organ Procurement Organizations). To maximize the number of matches, we should aim to aggregate information from as many sources as possible. However, this poses a threat to the con-

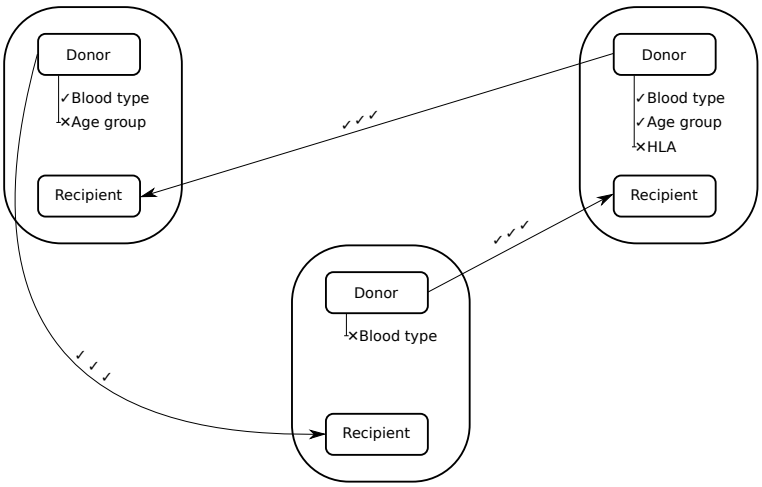


Figure 3.3: Graph Matching. More generally, we can draw directed edges from a donor to a compatible recipient and look for a simple cycle in the corresponding graph. This can enable even more live organ transplants.

fidentiality of the participants. Our goal is thus to encrypt records such as these to guarantee confidentiality but be able to test for compatibility.

3.1.2 Compatibility Matching by Equality Check

In this chapter, we will provide a solution that can help find compatible donor and recipient for organ donation. We mainly focus on the case of live donor and recipients (e.g., kidney donation), since they are the most interesting case. Additionally, geographical constraints are the weakest in these situation, since the donor can easily travel to meet the recipient for the transplant.

First, we show that we can reduce this problem to equality testing with a viable overhead. This means that we can avoid relying on fully homomorphic encryption and such expensive constructs.

Blood Type. Donor and recipient can each be O, A, B or AB. The compatibilities are shown on Figure 3.4. The (simple) trick is to generate a recipient record for each of the compatible blood types. For instance, consider a recipient of blood type A, who can accept a donor with blood type A or O. We generate two records for the recipient: one whose field `Blood Type` is set to O, and another where it is set to A.

Remark 6. *Records should not be linkable one to another; however, if they are, the number of compatible blood types can be hidden by padding with filler values. This can be done by constant values that always make donors and receivers incompatible.*

Age Group. Individuals of similar ages should match. This criterion is soft: we do not want to discriminate them into separate age groups. Instead, we let the age groups

		Donor			
		O	A	B	AB
Recipient	O	✓			
	A	✓	✓		
	B	✓		✓	
	AB	✓	✓	✓	✓

Figure 3.4: Blood Compatibility

overlap by letting the recipient list acceptable age groups for the donor (see Figure 3.5). In practice, the age groups would be of varying sizes and provide more precision where needed.

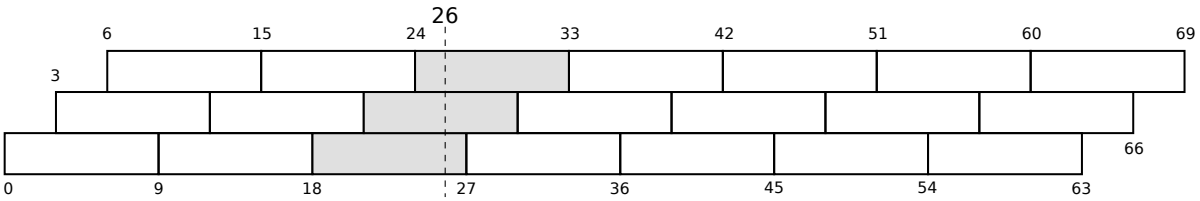


Figure 3.5: Age Groups: here, each age group spans 9 years and they are offset by 3 years; a 26 year old would list three age groups

Antigens. As a simplification, each individual is associated with six variables: HLA-A, HLA-B, HLA-C, HLA-DR, HLA-DQ and HLA-DP. Two individuals are considered to be HLA-wise compatible when at least three of these variables match. Again, we generate several records for the recipient, one for each of the 20 combinations of three antigens (binomial of 3 out of 6).

All Together. By combining these brute-force-inspired solutions, we expect to generate about a hundred records per recipient, on average. The overhead is non-trivial but is compensated by the fact of using a simpler matching procedure. It remains to show how we can implement such a procedure while keeping confidentiality.

3.2 Searchable and Comparable Encryption

3.2.1 Related Work

Comparing ciphertexts that were deterministically encrypted under the same key is equivalent to plaintext equality [BBO06]. A pair containing a digest and a probabilistic encryption of the message also allows testing against a plaintext. Public Key Encryption

Patient A	Patient B	Records	
A=HLA-A*02:12	A=HLA-A*02:02	A, B, C	B, C, R
B=HLA-B*7	B=HLA-B*42	A, B, R	B, C, Q
C=HLA-C*04	C=HLA-C*04	A, B, Q	B, C, P
R=HLA-DR12	R=HLA-DR12	A, B, P	B, R, Q
Q=HLA-DQ7	Q=HLA-DQ7	A, C, R	B, R, P
P=HLA-DPA1*01:06	P=HLA-DPA1*01:06	A, C, Q	B, Q, P
		A, C, P	C, R, Q
		A, R, Q	C, R, P
		A, R, P	C, Q, P
		A, Q, P	R, Q, P

Figure 3.6: Human Leukocyte Antigens: patients A and B have the same HLA-C, HLA-DR, HLA-DQ and HLA-DP so, several of the records match (shown in red)

with Equality Test (PKEET) allows comparing ciphertexts against both ciphertexts and plaintexts [YTHW10]. So do deterministic public key encryption schemes.

In searchable encryption [SWP00, BDOP04], each word w from the input m is encrypted as s. Anyone with some trapdoor value T_w can search for a w in m by testing each s. A variant works without a trapdoor [CFGL12]. Interactive protocols compute private matching and set intersection [FNP04].

3.2.2 Our Contribution

Fingerprinting and testing keys. When anyone can freely test ciphertexts against plaintexts, the cost of an exhaustive search depends on the min-entropy of the message distribution [LZL13]. Since we have low entropy, we exclude PKEETs. We introduce fingerprints: a cryptographic scheme with controlled encryption (fingerprinting key) and controlled testing between ciphertexts (testing key). Our model covers the four combinations of public or private fingerprinting and testing. Private testing is generally worse than private fingerprinting, since users test more often than they encrypt. We give an instantiation with private encryption but public non-interactive testing.

Blind and threshold fingerprinting. The *fingerprinter*, who holds the fingerprinting key, controls the number of queries to avoid exhaustive search. We present blind fingerprinting, which hides the queries from the fingerprinter. This concept is similar to blind signing [Cha82], but we do not require unlinkability. Finally, we fully decentralize the system by having several parties hold shares of the fingerprinting key (threshold scheme), as first presented in [CPST18b].

3.2.3 Organization

Section 3.3 presents the generic model for fingerprinting. Section 3.4 introduces a new assumption used by our construction, which we prove in the generic bilinear group model. We propose an instantiation in Section 3.5, with the blind and threshold variants, and show its security. Finally, we discuss the relevance of our results, and their relation to distributed technologies in Section 3.6.

3.3 Fingerprinting Scheme

We now introduce the general model for fingerprinting. In this generic definition, both fingerprint generation and equality testing may require keys. Later, we will present our instantiation, where fingerprint generation is private but equality testing is public.

3.3.1 Description

We consider three categories of players (see Figure 3.7):

- the **fingerprinter** generates the fingerprints of messages using the fingerprinting key;
- the **tester** checks whether two fingerprints correspond to the same message or not, using the testing key;
- the **users** have access to fingerprints and send queries to the fingerprinter or to the tester.

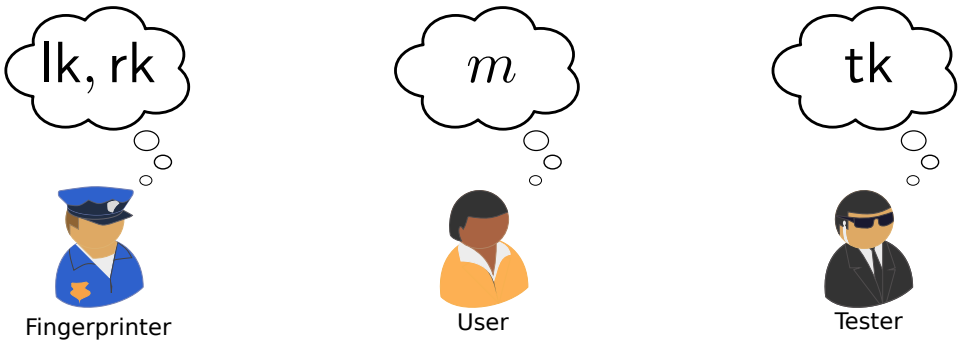
We stress however that the fingerprinting and testing keys may each be public or private. When a key is secret, the users have to interact with the owner of the key to benefit from the corresponding functionality; when it is public, the users can act on behalf of the fingerprinter or the tester. The choice of publishing a key or keeping it private will depend on the scenario under consideration.

Our model is built upon the honest-but-curious framework, assuming good behavior of the players. In our instantiation, we eventually split the fingerprinter into several parties, allowing us to police their actions.

Finally, we take advantage of the asymmetric nature of our use case: we never test the equality between two donors or two recipients. So, we will manipulate two kinds of fingerprints: "left" and "right" fingerprints in this generic specification.

We thus define four protocols:

- $\mathsf{KeyGen}()$ creates the global parameters and the left and right fingerprinting keys lk and rk as well as the testing key tk, for security parameter k;
- $\mathsf{LFingerprint}(\mathsf{lk}, m)$, given a left-fingerprinting key lk and a message m, outputs a left-fingerprint f_L;



f_1, 1P2F, Cleveland Clinic, US
f_2, JLR6, Assistance Publique Hôpitaux de Paris, FR
f_3, RV0I, Buddhist Tzu Chi General Hospital, TW
⋮

Public database of fingerprints

Figure 3.7: Fingerprinting Actors: the public database contains fingerprints and anonymous information to contact the relevant people; the fingerprinter and the tester hold the fingerprinting and testing keys respectively; the user has a message m to insert, or want to test two fingerprints

- $\mathsf{RFingerprint}(\mathsf{rk}, m)$, given a right-fingerprinting key rk and a message m, outputs a right-fingerprint f_R;
- $\mathsf{Test}(\mathsf{tk}, f_L, f_R)$, given a testing key tk, a left-fingerprint f_L and a right-fingerprint f_R, reveals whether the fingerprints correspond to the same message or not.

As already noted above, these procedures can be either private or public, and they can be either offline algorithms, or interactive protocols. Various situations can be envisioned according to the secrecy of the fingerprinting and testing keys:

- Testing and fingerprinting keys public: security solely rely on the high entropy of the inputs (message-locked encryption, as in PKEETs);
- Fingerprinting keys private only: our use case, where we want to limit the generation of fingerprints, but allow anyone to test freely for compatibility;
- Testing key private only: this can be relevant if the message space is very constrained, when even a few tests could leak too much information;
- Testing and fingerprinting keys private: this has the highest security guarantee, but is usually impractical unless performing very few queries is enough.

Remark 7. *We can choose to have one of the fingerprinting keys private, and the other public. This setup can give some flexibility for specific use cases.*

3.3.2 Security Model

Let us now detail the security notions we want to achieve. Secret information can include the fingerprinting keys lk and rk, the testing key tk, and the users' input messages. We consider the following security properties.

1. unforgeability of fingerprinting (even against the tester[1]);
2. one-more indistinguishability of testing (even against the fingerprinter[2]);
3. privacy of the user w.r.t. the tester;
4. privacy of the user w.r.t. the fingerprinter.

Authentication of the Fingerprinter. The *raison d'être* of the fingerprinter is to generate fingerprints, so unforgeability guarantees that no one else can do so: not even a collusion between the tester (access to the testing key) and users (queries to the fingerprinter) can generate a fresh valid fingerprint. In particular, this implies that the fingerprinting key is not leaked during this game. We formally define Fingerprint-Unforgeability ($\mathsf{FP\text{–}UF}$).

Definition 24 ($\mathsf{FP\text{–}UF}$)**.** *Let* $\Pi = (\mathsf{KeyGen}, \mathsf{LFingerprint}, \mathsf{RFingerprint}, \mathsf{Test})$ *be the scheme presented above, and let* $\mathcal{A}$ *be a polynomial-time adversary. Let*

$$\mathsf{Adv}^{\mathsf{FP\text{–}UF}}_{\Pi,L}(\mathcal{A}) = \Pr\left(\begin{array}{l}(\mathsf{lk},\mathsf{rk},\mathsf{tk}) \xleftarrow{\$} \mathsf{KeyGen}(), (m^\star, f_L^\star) \leftarrow \mathcal{A}^{\mathcal{L}}(\mathsf{rk},\mathsf{tk}),\\ f_R \leftarrow \mathsf{RFingerprint}(\mathsf{rk}, m^\star) : \mathsf{Test}(\mathsf{tk}, f_L^\star, f_R) = 1\end{array}\right)$$

where $\mathcal{L}$ *refers to the left-fingerprinting oracle, which answers to queries on message* m_i *with* $f_{L,i} = \mathsf{LFingerprint}(\mathsf{lk}, m_i)$*. We insist that* $m^\star$ *is fresh (distinct from any queried* m_i*).*

We similarly define $\mathsf{Adv}^{\mathsf{FP\text{–}UF}}_{\Pi,R}$*, with the left-fingerprinting key but access to the right-fingerprinting oracle. We say that* Π *is* (t,ε)*-*$\mathsf{FP\text{–}UF}$*-secure when both* $\mathsf{Adv}^{\mathsf{FP\text{–}UF}}_{\Pi,L}(\mathcal{A}) \leq \varepsilon$ *and* $\mathsf{Adv}^{\mathsf{FP\text{–}UF}}_{\Pi,R}(\mathcal{A}) \leq \varepsilon$ *for any* $\mathcal{A}$ *running within time* t*.*

Authentication of the Tester. The purpose of the tester is to help the user test plaintext equality between fingerprints: not even a collision between the fingerprinter (access to the fingerprinting key) and users (queries to the tester), can guess the result of a fresh test. In particular, this implies that the testing key is not leaked. We formally define Testing-Indistinguishability ($\mathsf{T\text{–}IND}$).

[1] the testing key should give no advantage in generating fingerprints
[2] the fingerprinting key should give no advantage in equality testing

Definition 25 (T–IND). *Let* $\Pi = (\mathsf{KeyGen}, \mathsf{LFingerprint}, \mathsf{RFingerprint}, \mathsf{Test})$ *be the scheme presented above, and* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *a polynomial-time adversary. Let*

$$\mathsf{Adv}^{\mathsf{T\text{–}IND}}_{\Pi,L}(\mathcal{A}) = \left| \Pr \begin{pmatrix} (\mathsf{lk}, \mathsf{rk}, \mathsf{tk}) \xleftarrow{\$} \mathsf{KeyGen}(), \\ (m_0, m_1, s) \leftarrow \mathcal{A}_1^{\mathcal{T}}(\mathsf{lk}, \mathsf{rk}), f_L \leftarrow \mathsf{LFingerprint}(\mathsf{lk}, m_0), \\ b \xleftarrow{\$} \{0,1\}, f_R \leftarrow \mathsf{RFingerprint}(\mathsf{rk}, m_b), \\ b' \leftarrow \mathcal{A}_2^{\mathcal{T}}(s, f_L, f_R) : b' = b \end{pmatrix} - \frac{1}{2} \right|$$

where $\mathcal{T}$ *refers to the testing oracle, who answers to queries on fingerprints* f_L, f_R *with* $\mathcal{T}(f_L, f_R) = \mathsf{Test}(\mathsf{tk}, f_L, f_R)$*. We require that the attacker does not submit the challenge fingerprint* f_R *to the testing-oracle.*

We define $\mathsf{Adv}^{\mathsf{T\text{–}IND}}_{\Pi,R}(\mathcal{A})$ *in a similar fashion. We say that* Π *is* (t, ε)*-*$\mathsf{T\text{–}IND}$*-secure if both* $\mathsf{Adv}^{\mathsf{T\text{–}IND}}_{\Pi,L}(\mathcal{A}) \leq \varepsilon$ *and* $\mathsf{Adv}^{\mathsf{T\text{–}IND}}_{\Pi,R}(\mathcal{A}) \leq \varepsilon$ *for any adversary* $\mathcal{A}$ *running within time* t*.*

One can note that for such a strong notion of indistinguishability, which only excludes the challenge fingerprints from being queried to the testing-oracle, the fingerprints must be non-malleable.

Privacy of the User. This security notion adapts semantic security to our scheme: not even a collusion between the tester (access to the testing key) and users (queries to the fingerprinter) can distinguish a fingerprint of a message m_0 from a fingerprint of a message m_1 (unless they know a fingerprint of m_0 or of m_1). Furthermore, the collusion could include one of the two fingerprinting keys (but not both): we give attacker the left-fingerprinting key when proving the semantic security of left-fingerprinting, and the right-fingerprinting key when proving the semantic security of right-fingerprinting. We formally define Fingerprint-Indistinguishability (FP–IND).

Definition 26 (FP–IND). *Let* $\Pi = (\mathsf{KeyGen}, \mathsf{LFingerprint}, \mathsf{RFingerprint}, \mathsf{Test})$ *be the scheme presented above, and let* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *be a polynomial-time adversary. Let*

$$\mathsf{Adv}^{\mathsf{FP\text{–}IND}}_{\Pi,L}(\mathcal{A}) = \left| \Pr \begin{pmatrix} (\mathsf{lk}, \mathsf{rk}, \mathsf{tk}) \xleftarrow{\$} \mathsf{KeyGen}(), (m_0, m_1, s) \leftarrow \mathcal{A}_1^{\mathcal{R}}(\mathsf{lk}, \mathsf{tk}), \\ b \xleftarrow{\$} \{0,1\}, f_L \leftarrow \mathsf{LFingerprint}(\mathsf{lk}, m_b), \\ b' \leftarrow \mathcal{A}_2^{\mathcal{R}}(s, f_L) : b' = b \end{pmatrix} - \frac{1}{2} \right|$$

where $\mathcal{R}$ *refers to the right-fingerprinting oracle, which answers to queries on message* m'_i *with* $\mathcal{R}(m'_i) = \mathsf{RFingerprint}(\mathsf{rk}, m'_i)$*. We insist that* $m'_i \notin \{m_0, m_1\}$ *for any queries to* $\mathcal{R}$*.*

We define $\mathsf{Adv}^{\mathsf{FP\text{–}IND}}_{\Pi,R}(\mathcal{A})$ *similarly. We say that* Π *is* (t, ε)*-*$\mathsf{FP\text{–}IND}$*-secure if both* $\mathsf{Adv}^{\mathsf{FP\text{–}IND}}_{\Pi,L}(\mathcal{A}) \leq \varepsilon$ *and* $\mathsf{Adv}^{\mathsf{FP\text{–}IND}}_{\Pi,R}(\mathcal{A}) \leq \varepsilon$ *for any adversary* $\mathcal{A}$ *running within time* t*.*

Note that fingerprinting generation itself reveals nothing: the view of the fingerprinter does not depend on the message. Like in blind signatures [Cha82], no adversary playing the role of fingerprinter can distinguish a fingerprinting of m_0 from a fingerprinting of m_1. However, if the fingerprinter sees the resulting fingerprint and locally generates

another fingerprint for m_0, it becomes trivial to distinguish between the two cases. This can be avoided by splitting the fingerprinter into several parties who need to cooperate to create a new fingerprint. This last security notion thus suggest the use of a blind protocol and a threshold scheme.

Remark 8. *Contrary to blind signatures, we do not require user anonymity. In our use-case, plain contact information (e.g., referent doctor) is attached to the fingerprint.*

3.4 Assumptions

Our construction adapts the randomizable signature presented in Subsection 2.3.3, which relies on q-MSDH-1 presented in Subsection 2.2.2. Our scheme additionally requires indistinguishability, which implies another assumption; for this, we introduce q-DMSDH-1, which is decisional variant of q-MSDH-1, and prove it to hold in the generic bilinear group model. For comparison, we reproduce the definition of q-MSDH-1 below.

Definition 27 (q-MSDH-1)**.** *Let $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e)$ define a bilinear group setting of type 3, with g (respectively $\tilde{g}$) a generator of $\mathbb{G}$ (respectively $\tilde{\mathbb{G}}$). Given $(g^{x^i}, \tilde{g}^{x^i})_{0 \leq i \leq q}$ along with $(g^a, \tilde{g}^a, \tilde{g}^{a \cdot x})$ for $a, x \xleftarrow{\$} \mathbb{Z}_p^*$, no adversary can output a tuple $(w, P, h^{\frac{1}{x+w}}, h^{\frac{a}{P(x)}})$ for some $h \in \mathbb{G}^*$ where P is a polynomial of degree at most q and w is a scalar such that $(X + w)$ and $P(X)$ are relatively prime.*

Now, our decisional variant q-DMSDH-1 only changes the goal of the adversary.

Definition 28 (q-DMSDH-1)**.** *Let $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e)$ define a bilinear group setting of type 3, with g (respectively $\tilde{g}$) a generator of $\mathbb{G}$ (respectively $\tilde{\mathbb{G}}$). Given $(g^{x^i}, \tilde{g}^{x^i})_{0 \leq i < q}$ along with $(g^a, g^{a \cdot x}, \tilde{g}^a)$ for $a, x \xleftarrow{\$} \mathbb{Z}_p^*$, and for any (w, P) where P is a polynomial of degree at most q and w is a scalar such that $(X + w)$ and $P(X)$ are relatively prime, no adversary can distinguish $(h^{\frac{1}{x+w}}, h^{\frac{a}{P(x)}})$ for some $h \in \mathbb{G}^*$ from a random pair of elements of $\mathbb{G}$.*

Theorem 5. *q-DMSDH-1 holds in the generic bilinear group model.*

Proof. The computational assumption q-MSDH-1 from [PS18] gives $\tilde{g}^{a \cdot x} \in \tilde{\mathbb{G}}$ and expects the forged pair in $\mathbb{G}$, whereas the decisional version q-DMSDH-1 gives $g^{a \cdot x} \in \mathbb{G}$ and the challenge pair in $\mathbb{G}$. So, the security guarantee we obtain (unforgeability from q-MSDH-1 or indistinguishability from q-DMSDH-1) depends on whether the oracle gives elements from $\mathbb{G}$ or from $\tilde{\mathbb{G}}$. Thus, the reasoning for q-DMSDH-1 is very similar to that for q-MSDH-1.

We prove that the q-DMSDH-1 assumption holds in the generic bilinear group model. The generic group model (not bilinear) was used by Victor Shoup in [Sho97] to assess more tightly the difficulty of computing the discrete logarithm and related problems. A vastly clarified introduction to this technique can be found in [Jag12]. The case for the bilinear setting (the generic bilinear group model) is presented in appendix A of [BBG05]. It is essentially a formal way to enumerate all the values that an adversary

can potentially compute from a restricted number of inputs, using only the group laws, as black boxes.

We use the classical approach of simulating group operations by an oracle $\mathcal{G}$, which operates on arbitrary representations $(\xi_{1,i})_i$, $(\xi_{2,i})_i$, $(\xi_{T,i})_i$ of the elements of $\mathbb{G}$, $\tilde{\mathbb{G}}$ and $\mathbb{G}_T$ (respectively). The oracle is built such that all interactions are done without relation to the secret values, hence reducing the attack to a guess.

For instance, $\mathcal{G}(\times, \xi_{1,i}, \xi_{1,j})$ is the result of querying the oracle $\mathcal{G}$ for the $\times$ operation with operands $\xi_{1,i}$ and $\xi_{1,j}$. It is a representation of the product of the underlying values in $\mathbb{G}$. The oracle $\mathcal{G}$ similarly allows the adversary $\mathcal{A}$ to compute products in $\tilde{\mathbb{G}}$ and $\mathbb{G}_T$, evaluate the pairing e, and test two representations for the equality of the underlying values.

To simulate the operations, the oracle $\mathcal{G}$ stores the values known to the adversary $\mathcal{A}$ (at beginning, and following a request) into lists L_1, L_2 and L_T (for each group). To track how the adversary $\mathcal{A}$ obtained these values, we save with each representation $\xi_{\square,i}$ a polynomial $p_{\square,i}$ corresponding to the operations used to compute the value The representations used are not important, and the reader must simply remember that a new random representation is generated for each new computed value; testing whether the value is fresh or not is done by searching the polynomial in the relevant list L_1, L_2 or L_T.

The values initially provided to the adversary $\mathcal{A}$ are:

- in $\mathbb{G}$: $(g^{x^i})_{0\leq i\leq q}$, g^a, $g^{a\cdot x}$, $h^{\frac{1}{x+w}}$, $h^{\frac{a}{P(x)}}$
- in $\tilde{\mathbb{G}}$: $(\tilde{g}^{x^i})_{0\leq i\leq q}$, $\tilde{g}^a$

To simulate operations over these elements, we set r such that $h = g^r$ and introduce the indeterminate values $\bar{x}$, $\bar{a}$, $\bar{r}$. Then, we initialize $L_1 = \{\bar{x}^i\}_i \cup \{\bar{a}, \bar{a}\bar{x}, \frac{\bar{r}}{\bar{x}+w}, \frac{\bar{a}\cdot\bar{r}}{P(\bar{x})}\}$, $L_2 = \{\bar{x}^i\}_i \cup \{\bar{a}\}$ and $L_T = \varnothing$ (along with arbitrary representations), and set:

- $\mathcal{G}(\times, \xi_{\square,i}, \xi_{\square,j})$: append $p_{\square,i} + p_{\square,j}$ to $L_\square$
- $\mathcal{G}(=, \xi_{\square,i}, \xi_{\square,j})$: return whether $p_{\square,i} = p_{\square,j}$
- $\mathcal{G}(e, \xi_{1,i}, \xi_{2,j})$: append $p_{1,i} \times p_{2,j}$ to L_T

Remark 9. *Comparing the representations directly is equivalent to calling the group oracle for testing, because the representations are generated so as to be equal when the corresponding polynomials are equal*

We now have to show two things: the simulation does not allow the adversary to distinguish between $(h^{\frac{1}{x+w}}, h^{\frac{a}{P(x)}})$ and a pair of random elements from $\mathbb{G}$ the simulation is indistinguishable from the initial game.

Indistinguishability in simulation Since representations are opaque, the adversary can only obtain information from testing two values for equality (either of representations or through the group oracle $\mathcal{G}$).

Comparing elements of $\mathbb{G}$. Consider a comparison of $\xi_{1,i}$ to $\xi_{1,j}$; the difference of their polynomials, $p_{1,i} - p_{1,j}$, is of the form:

$$\sum_i \left(C_x^{(i)} \bar{x}^i + C_a \bar{a} + C_{ax} \bar{a}\bar{x} + C_1 \frac{\bar{r}}{\bar{x} + w} + C_2 \frac{\bar{a} \cdot \bar{r}}{P(\bar{x})} \right)$$

as a polynomial in $\bar{r}$, the linear term implies that, if this polynomial were equal to zero, then:

$$C_1 P(\bar{x}) + C_2 \bar{a}(\bar{x} + w) = 0$$

as a polynomial in $\bar{a}$, this implies $C_1 = C_2 = 0$. Thus, the polynomial does not depend on the challenge pair.

Comparing elements of $\tilde{\mathbb{G}}$. Elements in $\tilde{\mathbb{G}}$ do not depend on the challenge pair.

Comparing elements of $\mathbb{G}_T$. Since L_T starts out empty, a comparison of $\xi_{T,i}$ to $\xi T, j$ will correspond to polynomials whose difference $p_{T,i} - p_{T,j}$ is the sum of products of one element from $\mathbb{G}$ and one element from G_2, thus of the form:

$$\sum_i \left(Q(\bar{x}) + C_{i,a} \bar{a} + C_{i,ax} \bar{a}\bar{x} + C_{i,1} \frac{\bar{r}}{\bar{x} + w} + C_{i,2} \frac{\bar{a} \cdot \bar{r}}{P(\bar{x})} \right) \times \left(R(\bar{x}) + \tilde{C}_{i,a} \bar{a} \right)$$

where Q and R are polynomials of degrees at most q. As a polynomial in $\bar{r}$, if this were the zero polynomial, then the linear term would imply that:

$$\sum_i \left(C_{i,1} P(\bar{x}) + C_{i,2} \bar{a}(\bar{x} + w) \right) \times \left(R(\bar{x}) + \tilde{C}_{i,a} \bar{a} \right) = 0$$

as a polynomial in $\bar{a}$, then the linear term would imply that:

$$\sum_i \left(C_{i,1} P(\bar{x}) \tilde{C}_{i,a} + C_{i,2} (\bar{x} + w) R(\bar{x}) \right) = 0$$

that is, $CP(\bar{x}) + S(\bar{x})(\bar{x} + w) = 0$ for C a constant and S a polynomial. Since $P(\bar{x})$ and $(\bar{x} + w)$ are relatively prime, this means that $C = 0$ and $S = 0$ and thus that the original equation does not depend on the challenge pair.

Undistinguishability of simulation Let $q_{\mathcal{G}}$ be the number of queries to the group oracle $\mathcal{G}$. The simulation is undistinguishable from the original game unless the adversary assembles two distinct polynomials (p, q) with $(p - q)(x, a, r) = 0$.

The adversary can adaptively test whether (x, a, r) is a root of one of the at most $q' = (5 + 2q + q_{\mathcal{G}})^2/2$ differences of polynomials of degrees at most $d = 2q$. Per the Schwartz-Zippel lemma, which states that a multivariate polynomial of degree d has at most d roots, this is equivalent to testing whether (x, a, r) pertains to one of q' subsets of $\mathbb{Z}_p^3$ of sizes at most d. Finally, the probability of adaptively finding such subsets is bounded above by $\frac{q' \cdot d}{p^3}$, which is negligible. □

3.5 Fingerprinting from Pointcheval-Sanders Signatures

In the following, we focus on our initial scenario with secret fingerprinting and public testing of plaintext-equality, for low-entropy messages. Our approach is heavily influenced by the assumption that it is possible to efficiently enumerate all the valid messages. Our construction derives from the Pointcheval-Sanders signature scheme presented in Subsection 2.3.3.

3.5.1 Fingerprinting Scheme with Public Plaintext-Equality Testing

We propose a fingerprinting scheme with private fingerprinting keys lk or rk, while the testing key $\mathsf{tk} = \varepsilon$ is public. Let $\mathcal{H}$ be a random oracle:

- $\mathsf{KeyGen}()$: randomly draw $(g, \tilde{g}, x, y) \xleftarrow{\$} \mathbb{G}\times\tilde{G}\times\mathbb{Z}_p^2$, set $(X, \tilde{X}, Y, \tilde{Y}) \leftarrow (g^x, \tilde{g}^x, g^y, \tilde{g}^y)$, return $\mathsf{lk} = X$, $\mathsf{rk} = \tilde{X}$, and $\mathsf{pk} = (g, Y, \tilde{g}, \tilde{Y})$;
- $\mathsf{LFingerprint}(\mathsf{lk}, m)$: draw $u \xleftarrow{\$} \mathbb{Z}_p^*$, return $f_L = (g^u, (XY^{\mathcal{H}(m)})^u)$;
- $\mathsf{RFingerprint}(\mathsf{rk}, m)$: draw $u \xleftarrow{\$} \mathbb{Z}_p^*$, return $f_R = (\tilde{g}^u, (\tilde{X}\tilde{Y}^{\mathcal{H}(m)})^u)$;
- $\mathsf{Test}(f_L, f_R)$: return 1 if $f_{L,1}, f_{R,1} \neq 1_{\mathbb{G}}$ and $e(f_{L,1}, f_{R,2}) = e(f_{L,2}, f_{R,1})$, else 0.

We show below that this scheme is secure in the honest-but-curious model defined in Subsection 3.3.2. Then, we propose improvements to maintain the guarantees when the actors are dishonest.

3.5.2 Security of the Basic Scheme

Theorem 6. *Our fingerprinting scheme is FP–UF under q-MSDH-1 in the random oracle model, where q corresponds to the number of queries to the random oracle or to the fingerprinting oracles.*

Proof. We define the extended Pointcheval-Sanders signature scheme (EPS) as a variant of the PS signature scheme where pk includes Y, i.e. $\mathsf{pk} = (Y, \tilde{g}, \tilde{X}, \tilde{Y})$. Lemma 1 holds that EPS is EUF–wCMA secure under q-MSDH-1, and Lemma 2 reduces the FP–UF security of our fingerprinting scheme to the EUF–wCMA security of EPS. □

Lemma 1. *If q-MSDH-1 holds, then EPS is EUF–wCMA where q is the number of queries to the signing oracle.*

Proof. We adapt the proof of from [PS18, Section 5.1], by setting $Y_1 \leftarrow g^a$.

Let $\mathcal{A}$ be an EUF–wCMA adversary against EPS. That is to say that $\mathcal{A}$ submits the list of queries $(m_i)_i$ before the challenger generates the key. Given signatures $(\sigma_i)_i$ for these messages, $\mathcal{A}$ then returns a valid signature σ^* on a fresh m^* $(\forall i, m^* \neq m_i)$ with non-negligible probability.

For our reduction, we solve q-MSDH-1 with non-negligible probability given such an $\mathcal{A}$. So, for $\gamma \in \mathbb{G}$, $\tilde{\gamma} \in \tilde{\mathbb{G}}$ and some scalars x and a, we are given a q-MSDH-1 instance

$(\gamma^{x^i})_{i\leq q}$, $(\tilde{\gamma}^{x^i})_{i\leq q}$, γ^a, $\tilde{\gamma}^a$ and $\tilde{\gamma}^{ax}$. Our goal is to assemble $(w, P, h^{\frac{1}{x+w}}, h^{\frac{a}{P(x)}})$ for some w, h and P such that $P(X)$ and $X + w$ are relatively prime.

First, let us set $P(X) = \prod_i (X + m_i)$, $g \leftarrow \gamma^{P(x)}$ and $\tilde{g} \leftarrow \tilde{\gamma}^{P(x)}$. We can compute g by expanding $P(x)$ and using $(\gamma^{x^i})_i$, and similarly for $\tilde{g}$. With this, we can now set $\mathsf{pk} = (g, Y, \tilde{g}, \tilde{X}, \tilde{Y})$ with $Y \leftarrow \gamma^a$, $\tilde{X} \leftarrow \tilde{\gamma}^{ax}$ and $\tilde{Y} \leftarrow \tilde{\gamma}^a$. This implies that $\mathsf{sk} = (x', y')$ where $x' = \frac{ax}{P(x)}$ and $y' = \frac{a}{P(x)}$.

These choices allow us to generate valid EPS signatures under pk for each m_i as $\sigma_i \leftarrow ((\gamma^{\prod_{i\neq j}(x+m_i)})^{t_j}, \gamma^{at_j})$, using the same trick as for g. Finally, $\mathcal{A}$ returns m^* and σ^* such that $e(\sigma_1^*, \tilde{X}\tilde{Y}^{m^*}) = e(\sigma_2^*, \tilde{g})$. This implies that $e(\sigma_1^*, \gamma^{a(x+m^*)}) = e(\sigma_2^*, \gamma^{P(x)})$, so σ^* is of the form $(h^{\frac{1}{x+m^*}}, h^{\frac{a}{P(x)}})$, for some h. Note that $P(X)$ and $X + m^*$ are relatively prime, since $\forall i, m^* \neq m_i$. Thus, $(m^*, P, \sigma_1^*, \sigma_2^*)$ is a valid answer to the q-DMSDH-1 challenge. □

Lemma 2. *If EPS is EUF–wCMA, then our fingerprinting scheme is FP–UF. The number of queries to the signing oracle in EPS maps to the number of queries to the random oracle or to the fingerprinting oracles in our scheme.*

Proof. Let $\mathcal{A}$ be an adversary that breaks the FP–UF security of our scheme. Then, we create an adversary $\mathcal{B}$ that breaks the EUF–wCMA security of EPS. By symmetry of the left and right games, we assume that $\mathsf{Adv}^{\mathsf{FP\text{–}IND}}_{\Pi,L}(\mathcal{A})$ is non-negligible without loss of generality.

We will use $\mathcal{H}$ to "redirect" the queries from $\mathcal{A}$ towards predetermined values: $\mathcal{B}$ first draws $(m_i)_i \xleftarrow{\$} \mathbb{Z}_p^q$, submits the list of messages $(m_i)_i$ to the signing challenger, and will answer to the i-th original query to $\mathcal{H}$ (for some message M_i) with m_i.

In return, our adversary $\mathcal{B}$ is given $\mathsf{pk} = (Y, \tilde{g}, \tilde{X}, \tilde{Y})$ as well as signatures $(\sigma_i)_i$ for $(m_i)_i$, i.e. values such that $e(\sigma_{i,1}, \tilde{X}\tilde{Y}^{m_i}) = e(\sigma_{i,2}, \tilde{g})$ We need to output $(m^\star, \sigma^\star)$ such that $e(\sigma_1^\star, \tilde{X}\tilde{Y}^{m^\star}) = e(\sigma_2^\star, \tilde{g})$ where $m^\star$ is distinct from any queried m_i.

For this, we simulate the FP–UF game for $\mathcal{A}$ with $\mathsf{pk}' \leftarrow (g, Y, \tilde{g}, \tilde{Y})$, $\mathsf{rk} \leftarrow \tilde{X}$ as well as access to an oracle $\mathcal{L}$ which answer to queries M_i with σ_i. Then, $\mathcal{A}$ should output $(M^\star, f_L^\star)$ where $M^\star$ is distinct from any queried M_i. We also require that $\mathsf{Test}(\mathsf{tk}, f_L^\star, f_R)$ for some $f_R \leftarrow \mathsf{RFingerprint}(\mathsf{rk}, m^\star) = 1$, i.e. such that $f_{L,1} \neq 1_{\mathbb{G}}$ and:

$$e\left(f_{L,1}, \left(\tilde{X}\tilde{Y}^{\mathcal{H}(M^\star)}\right)^u\right) = e\left(f_{L,2}, \tilde{g}^u\right)$$

for some u. Thus, $\sigma^\star = f_L^\star$ is a valid PS signature for $m^\star = \mathcal{H}(M^\star)$ with $m^\star$ distinct from any queried m_i. □

Theorem 7. *Our fingerprinting scheme is FP–IND under q-DMSDH-1 in the random oracle model, where q corresponds to the number of queries to the random oracle or to the fingerprinting oracles.*

Proof. Let $\mathcal{A}$ be an adversary against FP–IND, then we provide an adversary $\mathcal{B}$ against q-DMSDH-1. By symmetry, we assume that $\mathsf{Adv}^{\mathsf{FP\text{–}IND}}_{\Pi,L}(\mathcal{A})$ is non-negligible.

First, $\mathcal{B}$ is given $(g^{x^i})_{0\leq i\leq q}$, $(g^a, g^{a\cdot x}, \tilde{g}^a)$. Then, it draws $(m_i)_i \xleftarrow{\$} \mathbb{Z}_p^q$, $m \xleftarrow{\$} \mathbb{Z}_p$, sets $P = \prod_i (X + m_i)$, and submits (m, P) to the challenger, who answers with a pair σ

which is either random or of the form $(h^{\frac{1}{x+m}}, h^{\frac{a}{P(x)}})$ for some $h \in \mathbb{G}$. $\mathcal{B}$ will run $\mathcal{A}$ while simulating the game for FP–IND by setting $g' \leftarrow g^{\prod_i (x+m_i)}$ and $\tilde{g'} \leftarrow \tilde{g}^{\prod_i (x+m_i)}$, using $(g^{x^i})_i$ and $(\tilde{g}^{x^i})_i$, as well as $X \leftarrow g^{a \cdot x}$, $Y \leftarrow g^a$, and $\tilde{Y} \leftarrow \tilde{g}^a$ to define the public key $\mathsf{pk} = (g', Y, \tilde{g'}, \tilde{Y})$ and the left-fingerprinting key $\mathsf{lk} = X$. This implicitly sets $x' = \frac{a \cdot x}{\prod_i (x+m_i)}$ and $y' = \frac{a}{\prod_i (x+m_i)}$.

To generate fingerprints for the q queried fingerprints, $\mathcal{B}$ sets the random oracle $\mathcal{H}$ to map the j-th original query M_j to m_j, and the right-fingerprinting oracle $\mathcal{R}$ to return $((\tilde{g'}^{\prod_{i \neq j} (x+m_i)})^{u_j}, (\tilde{g'}^a)^{u_j})$. One may verify that this is a valid right-fingerprint for M_j.

Finally, $\mathcal{A}$ outputs (M'_0, M'_1), and $\mathcal{B}$ draws $b \leftarrow \{0, 1\}$. We would now like to set $\mathcal{H}(M'_b)$ to m, but $\mathcal{A}$ may have queried the random oracle on this value before. Thus, on any query M_j, $\mathcal{H}$ will additionally guess with probability $\frac{1}{q}$ that $M_j = M'_b$ and accordingly set $\mathcal{H}(M_j)$ to m instead of m_j. $\mathcal{B}$ can then check its guess when $\mathcal{A}$ outputs (M'_0, M'_1), and abort if it was incorrect; this implies a penalty of a factor q to the probability that $\mathcal{B}$ wins the q-DMSDH-1 game.

Now, since $\mathcal{H}(M'_b) = m$, if σ is of the form $(h^{\frac{1}{x+m}}, h^{\frac{a}{P(x)}})$, then it is a valid left-fingerprint for M_b. Otherwise, it provides no information about b to the adversary. Thus, $\mathcal{B}$ answers the final request of $\mathcal{A}$ with σ, and, if $\mathcal{A}$ guesses b correctly, then $\mathcal{B}$ guesses that σ is of the form $(h^{\frac{1}{x+m}}, h^{\frac{a}{P(x)}})$; otherwise, that it is a random pair. □

3.5.3 Improving the Privacy of the User

Although we have proven the security of our scheme in the model introduced earlier, the guarantees are unsatisfactory. Indeed, if the fingerprinter were to act dishonest, we would lose any semblance of security. To actually provide credible security, we make the protocol interactive and introduce several changes. By symmetry, we only show the protocols for left fingerprinting.

Against the Fingerprinter without the Final Fingerprint. In the naive fingerprinting protocol from Subsection 3.5.1, the user sends the message in the clear. We propose a blinded version similar to that of [PS16] to extend privacy and protect the user against the fingerprinter:

1. the user draws $r \xleftarrow{\$} \mathbb{Z}_p$ and sends $C \leftarrow Y^m g^r$;
2. the user runs a ZKPoK of m, r such that $C = Y^m g^r$;
3. the fingerprinter draws $u \xleftarrow{\$} \mathbb{Z}_p$ and sends back $\alpha \leftarrow (g^u, (XC)^u)$;
4. the user sets $f_1 \leftarrow \alpha_1$ and $f_2 \leftarrow \alpha_2 \cdot \alpha_1^{-r}$.

This protocols leaks no information, since the fingerprinter only sees a perfectly hiding Pedersen commitment [Ped92] and a ZKP. The security of this blinded version can be proved by extracting the values from the ZKPoK, as done in [PS16].

Against the Fingerprinter with the Final Fingerprint. The protocol above still do not protect the user if the fingerprinter gains access to the final fingerprint f. In such a situation, the fingerprinter could search exhaustively by creating fingerprints for arbitrarily many messages. We amend the protocol to prevent this by splitting the fingerprinter into n parties $(F_i)_i$, of which k must cooperate to create a fingerprint. Even $k-1$ fingerprinters can do no more than a common user.

This threshold version uses Shamir's secret sharing (see Subsection 2.4.1) to split the secret fingerprinting key $\mathsf{lk} = x$ into n shares $\mathsf{lk}_i = x_i$, and we note $X_i = g^{x_i}$. This way, for any k fingerprinters $(\mathsf{F}_i)_i$, there exists publicly known coefficients $(\lambda_i)_i$ such that $x = \sum \lambda_i x_i$, and thus $\prod X_i^{\lambda_i} = X$. They interact with the user as follows:

1. the user draws $r \xleftarrow{\$} \mathbb{Z}_p$ and broadcasts $C \leftarrow Y^m g^r$;
2. the user sends a NI-ZKPoK of m, r such that $C = Y^m g^r$;
3. each F_i draws $u_i \xleftarrow{\$} \mathbb{Z}_p$ and broadcasts $\alpha_{i,1} \leftarrow g^{u_i}$;
4. each F_i computes $G \leftarrow \prod \alpha_{j,1}^{\lambda_j}$, and sends back $\alpha_{i,2} \leftarrow G^{x_i} C^{u_i}$;
5. the user sets

$$f_1 \leftarrow G = g^u \qquad\qquad f_2 \leftarrow G^{-r} \prod \alpha_{i,2}^{\lambda_i} = (XY^m)^u$$

which implicitly defines $u = \sum \lambda_i u_i$.

As with the previous version, this protocol leaks no information. However, we add the property that no subset of fewer than k fingerprinters can conduct an exhaustive search.

Theorem 8. *The above protocol leaks no private information of honest fingerprinters to corrupted ones.*

Proof. For this, we show that the view of any (static) subset of corrupted fingerprinters can be simulated from the output $\alpha = (\alpha_1, \alpha_2)$ of the single fingerprinter in the centralized protocol.

Let us assume that the corrupted fingerprinters are $\mathsf{F}_1, \ldots, \mathsf{F}_c$, and the honest ones are $\mathsf{F}_{c+1}, \ldots, \mathsf{F}_k$ (where $c < k$), and the simulator has drawn $v_i \xleftarrow{\$} \mathbb{Z}_p$ for $i = c+1, \ldots, k$: the adversary sends $\alpha_{i,1}$ for $i = 1, \ldots, c$, and the simulator draws $u_i \xleftarrow{\$} \mathbb{Z}_p$ and generates $\alpha_{i,1} \leftarrow g^{u_i}$ for $i = c+1, \ldots, k-1$, while $\alpha_{k,1} \leftarrow (\alpha_1 / \prod_{i=1}^{k-1} \alpha_{i,1}^{\lambda_i})^{1/\lambda_k}$. Finally, the simulator sets $G \leftarrow \alpha_1$, and $\alpha_{i,2} \leftarrow G^{v_i} C^{u_i}$ for $i = c+1, \ldots, k-1$, while $\alpha_{k,2} \leftarrow (\alpha_2 / \prod_{i=1}^{k-1} \alpha_{i,2}^{\lambda_i})^{1/\lambda_k}$.

Since no information is known about the actual secret values x_i, and the values v_i are indistinguishable from the real secret, all the simulated elements are perfectly indistinguishable from a real execution, under the condition that the corrupted fingerprinters are honest-but-curious (and the subset of honest players remains the same: static corruptions). □

Again, this last property requires that the fingerprinters behave correctly, even if they try to learn more information.

3.5.4 Verifiability

We need to verify the correctness of the protocol to protect against malicious fingerprinters. We call this private verifiability since it is only available to the participants of a given execution of the protocol. We also consider public verifiability, to avoid random data or copies of fingerprints posted by malicious users in the database In both cases, fingerprinters publish a commitment $C_i = g^{x_i}Y^{t_i}$ of their secret shares x_i during the initial key generation. This is a perfectly hiding commitment, and the binding property relies on secrecy of the discrete logarithm of Y in basis g.

Private Verifiability. We simply have each F_i provide a NI-ZKPoE of (x_i, u_i, t_i) such that $C_i = g^{x_i}Y^{t_i}$ and $\alpha_{i,1} = g^{u_i}$ and $\alpha_{i,2} = G^{x_i}C^{u_i}$. The proofs can be efficiently realized with Schnorr-like proofs.

Public Verifiability. Similarly, the user publishes a NI-ZKPoK of m and r such that $\alpha_2/f_2 = \alpha_1^r$ and $C = Y^m g^r$. To guarantee non-malleability and avoid replay attacks, the user includes his own identity in the challenge computation (signature of knowledge).

3.5.5 Full Protocol

To sum up, we now give a description of the resulting protocol.

Setup. The following is executed once:

1. $(F_i)_{i<n}$ jointly share a random secret x so that each F_i knows only x_i;
2. each F_i publishes a commitment $C_i = g^{x_i}Y^{t_i}$ for a random scalar t_i.

Fingerprinting protocol. Now, when a user Id wants a fingerprint for a message m, the user contacts a subset of k fingerprinters $(F_i)_{i<k}$ and executes the following protocol:

1. the user draws $r \xleftarrow{\$} \mathbb{Z}_p$ and broadcasts $C \leftarrow Y^m g^r$;
2. the user sends an NI-ZKPoK of m, r such that $C = Y^m g^r$;
3. each F_i draws $u_i \xleftarrow{\$} \mathbb{Z}_p$ and sends back $\alpha_{i,1} \leftarrow g^{u_i}$;
4. each F_i computes $G \leftarrow \prod \alpha_{j,1}^{\lambda_j}$, and sends back $\alpha_{i,2} \leftarrow G^{x_i}C^{u_i}$;
5. each F_i starts a NI-ZKP for u_i, x_i and t_i such that

$$C_i = g^{x_i}Y^{t_i} \qquad \alpha_{i,1} = g^{u_i} \qquad \alpha_{i,2} = G^{x_i}C^{u_i}.$$

   More precisely, F_i draws $u_i', x_i', t_i' \xleftarrow{\$} \mathbb{Z}_p$, broadcasts $\alpha_{i,1}' \leftarrow g^{u_i'}$, assembles $G' \leftarrow \prod {\alpha_{i,1}'}^{\lambda_i}$ as above and sends

$$C_i' = g^{x_i'}Y^{t_i'} \qquad \alpha_{i,1}' = g^{u_i'} \qquad \alpha_{i,2}' = G^{x_i'}C^{u_i'};$$

6. the user assembles

$$\Gamma \leftarrow \prod {C_i}^{\lambda_i} = XY^t \qquad \alpha_1 \leftarrow G = \prod \alpha_{i,1}^{\lambda_i} = g^u \qquad \alpha_2 \leftarrow \prod \alpha_{i,2}^{\lambda_i} = (XC)^u$$

where $u = \sum \lambda_i u_i$ and $t = \sum \lambda_i t_i$, and deduces the fingerprint

$$f_1 \leftarrow \alpha_1 \qquad\qquad f_2 \leftarrow G^{-r}\alpha_2.$$

Additionally, the user aggregates the elements from the NI-ZKPs:

$$\Gamma' \leftarrow \prod {C'_i}^{\lambda_i} = X'Y^{t'} \qquad \alpha'_1 \leftarrow \prod {\alpha'_{i,1}}^{\lambda_i} = g^{u'} \qquad \alpha'_2 \leftarrow \prod {\alpha'_{i,2}}^{\lambda_i} = (X'C)^{u'},$$

where $u' = \sum \lambda_i u'_i$, $x' = \sum \lambda_i x'_i$, $X' = g^{x'}$ and $t' = \sum \lambda_i t'_i$. Before generating the challenge, the user starts another NI-ZKP for m and r such that $\alpha_2/f_2 = \alpha_1^r$ and $C = Y^m g^r$, by drawing r' and m', setting:

$$\beta_1 \leftarrow \alpha_1^{r'} \qquad\qquad \beta_2 \leftarrow Y^{m'} g^{r'}$$

and the common challenge as $e = \mathcal{H}(\mathsf{Id}, C, \Gamma, f_1, f_2, \Gamma', \alpha'_1, \alpha'_2, \beta_1, \beta_2)$;

7. each F_i completes the corresponding NI-ZKP by publishing

$$t''_i \leftarrow t'_i - et_i \qquad u''_i \leftarrow u'_i - eu_i \qquad x''_i \leftarrow x'_i - ex_i;$$

8. the user again aggregates the NI-ZKs

$$u'' \leftarrow \sum \lambda_i u''_i \qquad x'' \leftarrow \sum \lambda_i x''_i \qquad t'' \leftarrow \sum \lambda_i t''_i$$

which satisfy

$$g^{x''}Y^{t''} = \Gamma'\Gamma^{-e} \qquad g^{u''} = \alpha'_1\alpha_1^{-e} \qquad G^{x''}C^{u''} = \alpha'_2\alpha_2^{-e}$$

and completes the corresponding NI-ZKP with

$$m'' \leftarrow m' - em \qquad\qquad r'' \leftarrow r' - er$$

which satisfy

$$\alpha_1^{r''} = \beta_1(\alpha_2/f_2)^{-e} \qquad\qquad Y^{m''}g^{r''} = \beta_2 C^{-e}.$$

The final fingerprint $f = (f_1, f_2)$ is published along with the intermediate values (Γ', α_2, C), the challenge e and the exponents $(u'', x'', t'', m'', r'')$. One verifies this proof by checking that $e = \mathcal{H}(\mathsf{Id}, C, \Gamma, f_1, f_2, \Gamma', \alpha'_1, \alpha'_2, \beta_1, \beta_2)$, where Γ is publicly computable, and the missing elements can be recomputed as

$$\begin{aligned} \alpha'_1 &\leftarrow g^{u''} f_1^e & \alpha'_2 &\leftarrow f_1^{x''} C^{u''} \alpha_2^e \\ \beta_1 &\leftarrow f_1^{r''}(\alpha_2/f_2)^e & \beta_2 &\leftarrow Y^{m''} g^{r''} C^e. \end{aligned}$$

This is an optimization of the Fiat-Shamir heuristic for Schnorr-like proofs.

3.6 Discussion

3.6.1 Security Level

On the one hand, we require the ability to insert new records, and to compare the records against each other. On the other hand, we want to keep the contents of these records secret and to protect the confidentiality of the patients. This means that confidentiality must rely on the entropy of the records themselves.

As we highlighted when presenting our use case, the situation presents a contradictory objective. The nature of the problem implies that we actually want a low entropy! Indeed, the usefulness of our system is related to its ability to find compatible matches. Additionally, we prefer false positives to false negatives. False positives can be detected in the next step of the process, while false negatives remain hidden. Thus, our system should be used as a coarse filter to reduce the work required to find compatibles matches. However, this reduces the available entropy further.

The solution we propose involves an authority to limit the number of requests that can be performed. Yet, we ensure that this authority can be distributed into several parties, in a resilient manner. We also provide a solution that still gives as much control to the users as possible, since compatibility tests can be run by anyone. Finally, we make sure that the confidentiality of the records is fully preserved.

These considerations are of particular importance when implementing this solution. In particular, the policies of the fingerprinters to accept a request or not must be designed with great care. In normal function, they should simply accept all requests. However, they should keep track of how many fingerprints were created. At a certain threshold, they should either reduce the rate of fingerprinting, or refuse new queries altogether.

To avoid such blocking situations, it is possible to reset the database. In effect, after a set period of time has passed, new keys should be generated, and active records should be fingerprinted anew. This refresh would prune stale and obsolete records from the database, and stop any ongoing exhaustive search attack. It is important to note that the new fingerprint of a record should not be linkable with its old fingerprint. Otherwise, the previous exhaustive searches can be resumed on the new fingerprints. In particular, it means that the public information associated to the fingerprint used to identify the record should change as well (new record number for instance).

3.6.2 Decentralized Setting

This problem was initially considered as a potential use-case for Blockchain. Indeed, several of the constraints match:

- we are considering a decentralized solution because the international setting makes it hard to completely rely on a central authority;
- all of the objects can be considered to be digital assets, since we are only looking for information, not taking binding decisions.

Also, one might think that Blockchain is good at storing information, but it is not actually the case. Nevertheless, the practical constraints led us to design a cryptographic protocol that relies on the existence of various entities. Since these entities are identified, the corresponding PKI gives us the center we need to coordinate the system. So, combining our cryptographic primitive with a blockchain adds no value to the solution. On the other hand, we might imagine using Blockchain to limit the number of queries performed (i.e., the global consensus consists in which fingerprints were done). However, it remains unclear how this can be used effectively to protect confidentiality (once a fingerprint is validated, who encrypts it?).

We want to stress that Blockchain does not actually address the core issue of our use case, that is confidentiality. By running scripts, or *smart contracts*, it is trivial to compare plaintext records for compatibility. Still, the difficulty lies in actually encrypting them. There exists settings where users can encrypt the information themselves (for instance, publishing a cryptographic hash for timestamping). However, in this situation, the cryptographic keys must be accessible to all but known to none.

Regarding the required PKI to determine who plays the roles of fingerprinters, we assume that an international organization such as the World Health Organization (WHO) can help in setting up the system. Yet, relying on a central authority to collect all the medical records and search compatible matches would be a much stronger assumption. Instead, we think that a solution that runs independently from a central authority after the key setup is more realistic, and more resilient.

3.6.3 Other Applications

Finally, it should be noted that our use case provides a motivation for the scheme we presented in this chapter, but that this scheme can be used in other cases as well. Essentially, what we present is a distributed and scalable system to find matches between individuals while always keeping the information used for the matches private. For instance, it seems like this approach would adapt well for matching system for dating, especially with the bipartite aspect that we exploit in our solution.

Still, the cost of protecting the confidentiality might be non-trivial. Indeed, remember that we simplified the matching process to simple equality testing by generating several records for each individual. This cost matters for highly dimensional problems. In other words, using fine-grained parameters might not increase the number of records too much, but using many parameters might.

3.7 Conclusion

We considered a concrete use case raised by the problem of organ donation and the conflict between confidentiality and functionality. We then provided a solution by introducing a new type of cryptographic primitive tailored for this class of problem. This shows that specific constructions can be used to solve practical problems more efficiently

then generic approaches. We hope that this type of design can help spread cryptography further and provide solutions for new use cases.

4 Secure Strategy-Resistant Voting

We consider another practical situation, in the form of electronic and online voting. Although some solutions exist, they are generally limited in functionality or in the level of confidentiality provided. In this chapter, we present a cryptographic protocol with advanced functionality (strategy-resistance), and strong confidentiality (only the result is revealed), while remaining efficient. These results were first published in [CPST18a].

4.1 Electronic Voting Systems

4.1.1 Voting Systems

As discussed in Section 1.2 and Section 1.4, voting systems are among the most basic elements for managing trust.

Referendums are the simplest voting system. Each voter $\mathcal{V}_k$ casts a ballot containing a choice, either "Yes" or "No" regarding a specific proposal. Several referendums can be run at once, with a ballot containing a vote for each, but the principle remains the same. They are relatively simple, both for voters, who express their preferences intuitively, and for talliers, who count the "Yes" and "No" votes.

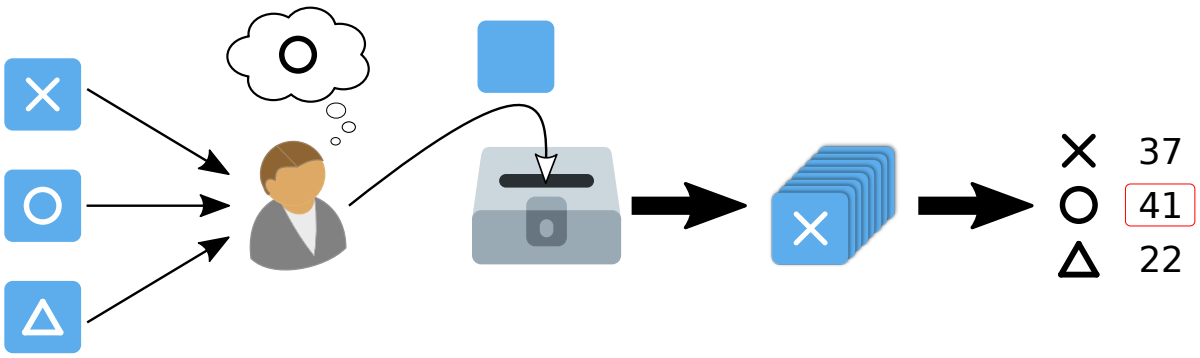


Figure 4.1: Plurality Voting. The voter casts a ballot corresponding to one of the options (cross, circle or triangle) by secretly putting it in the urn. When the election is finished, all the ballots are extracted of the urn. The votes are counted and the option with the most votes wins (here, circle). Because all the ballots are mixed together, only the voters themselves know what options they chose. Referendums are a particular case with only two options: "Yes" or "No".

Plurality voting generalizes this to more than two options. Each voter casts a ballot containing a single choice. For single-seat elections, the option with the most votes

wins (first-past-the-post); for multi-seat elections, the options with the most votes win (plurality-at-large) (see Figure 4.1). They are simple to tally but encourage voters to vote for candidates with better chances to maximize the influence of their ballots. If the favorite candidate of a given voter has little chance of winning the election in the first place, then the voter might feel that voting for this candidate is of little use. As a consequence, this voter might prefer voting for one of the main parties. This behavior reinforces itself, since it lowers the chances of the small candidate even further. This results in few parties getting most of the votes (2 in the US).

This issue is partly addressed in runoff voting, where voters cast ballots on several occasions: if they vote for a lesser-known candidate in the first round, they can still choose between the remaining candidates in the second round (see Figure 4.2). However, voting for smaller parties is still disincentivised. For multi-seat elections, a more representative result can be achieved with party-list proportional representation: each party gets a number of seats proportional to the number of votes obtained.

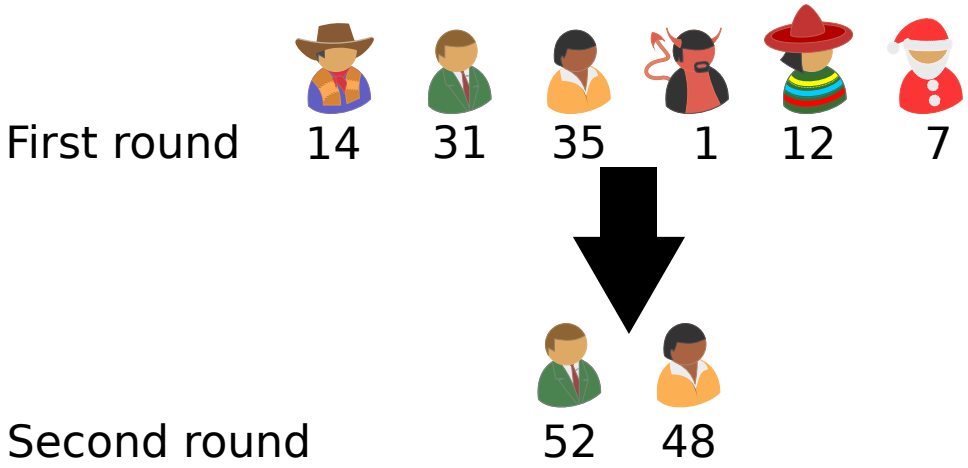


Figure 4.2: Runoff Voting. The election consists in two rounds. In the first round, voters may vote for any candidate; the ballots are counted and the two candidates with the most votes are selected. During the second round, voters cast new ballots, but only for one of these two options. This way, a voter may vote for Santa Claus in the first round even though he has little chance of winning the election, and still choose between the two main candidates during the second round.

Alternative voting systems have been explored. In ranked voting, the voters cast an ordering of the candidates (see Figure 4.4); in cardinal voting, they rate each candidate (see Figure 4.3). Single Transferable Vote (STV) is a ranked voting system proposed in 1819. It is essentially a runoff system which eliminates a single candidate at each round so that voters' preferences are always taken into account (see Figure 4.6 and Figure 4.7). In single-seat elections, it is called Instant-Runoff Voting. Majority Judgment is a cardinal voting system from 2007 which considers the median of the ratings (see Figure 4.5). While STV requires processing each ballot individually several times, Majority Judgment aggregates the ballots during the tally.

Majority Judgment is more efficient than STV (linear complexity instead of quadratic), but it also offers stronger strategy resistance. Notably, it verifies the following properties, while STV does not:

- **independence of irrelevant alternatives:** the global preference between choices A and B depends only on the individual preferences between A and B;
- **monotonicity criterion:** ranking a candidate lower on some of the ballots cannot help in making them elected;
- **no favorite betrayal:** voters do not need to rank a candidate above their favorite to obtain a preferable result.

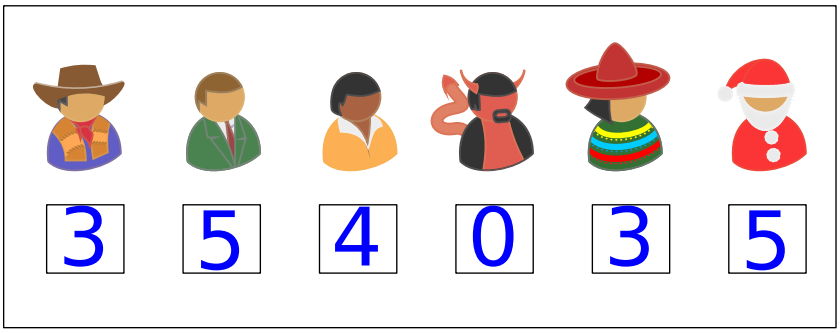


Figure 4.3: Cardinal Voting. A voter grades each candidates from 0 to 5. There are various ways to tally the ballots. The simplest way would simply average all the grades, but this encourages the voters to always put 0 or 5 to increase their influence. Instead, Majority Judgment considers the median grade of each candidate. From the point of view of the voter, cardinal voting is very similar to ranked voting, since grades are usually appreciated relatively one to another.

4.1.2 Electronic Voting

Electronic voting runs elections on computers. This makes it easer to use STV or Majority Judgment, which improves the legitimacy of political elections by giving each voter the feeling of being heard. Additionally, the lower effort required for online voting might increase the turnout. However, privacy of voters and integrity of the election are a central issue.

Indeed, with electronic urns, the voters must trust that the manufacturer of the votes:

- does not read their ballot;
- actually takes their ballot into account;
- functions appropriately;

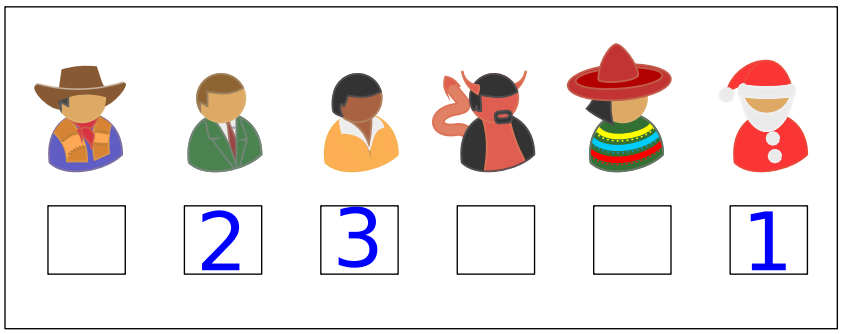


Figure 4.4: STV Ballot. A voter ranks candidates in order of preference. Here, the voter prefers Santa; otherwise, Bob and otherwise, Alice. The voter is not interested in the other candidates. This type of ballot will make sure that the voter's preference are taken into account as long as at least one of Santa, Bob or Alice remains. Note that the voters do not need to rank all the candidates. This can be an important feature when there are many candidates and voters may not care about most of them.

- does not induce the user in error.

In the real world, each guarantee was violated in various cases. From an economic point of view, this is particularly problematic: implementing voting urns is a niche market, making it harder to impose strong guarantees on the product without incurring proportionally large costs.

In contrast, general computers are widespread, so that intentional and unintentional malfunctions are *relatively* unlikely, and are usually highly visible and easily detectable. Additionally, the idea of polling the population on a regular basis for various topics (not just for presidential elections) have been considered multiple times. This leads to the idea of online voting, where voters cast their ballots from their own devices, through the Internet.

This is easily achieved with a central authority to collect the ballots. However, in these cases, we still need to trust the urn not to look at the ballots, and to run the tally properly. In consequence, the idea of creating a trust-less urn is being investigated.

Referendums are easily realized with additively homomorphic encryption: if voters encrypt 1 for "Yes" and 0 for "No", the tally corresponds in simply combining the ciphertexts and decrypting the result. For integrity, the voters provide cryptographic proofs that their ballots contains either 0 or 1 (without revealing which). First-past-the-post voting can be implemented in a similar fashion (but in less efficient way, since cryptographic proofs become more complicated).

4.1.3 Related Work

Secure implementations of referendum and first-past-the-post voting have been regularly studied in cryptographic literature. In particular, [DK05] proposes a solution for running referendums without revealing the exact count (only whether "Yes" or "No" won) that

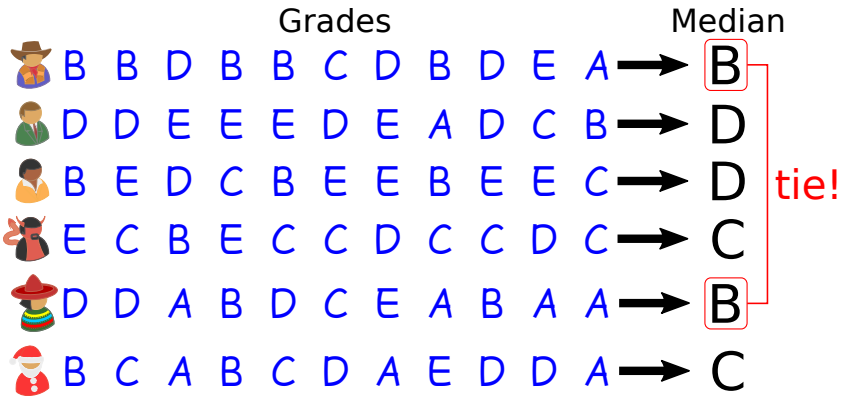


Figure 4.5: Cardinal Voting Tally. The grades of a candidate are listed horizontally. A column can represent a single ballot from Figure 4.3, but it is not necessary (grades are independent). The median grade of each candidate is computed, as shown on the right. The candidate who has the best median grade wins. However, several candidates may have the same median grade. If several candidates have the best median grade, then a tiebreak is required. Each ballot is represented vertically, with the candidates ranked by the voter.

does not rely on MixNets or on any trusted server. Helios [Adi08] and Belenios [CGGI14] offer solutions for plurality voting.

Majority Judgment and STV are more complex, especially for the tallying phase. One possibility is to aggregate all the encrypted ballots, decrypt them, and perform the final steps in the clear. Although this approach ensures ballot secrecy, it reveals the scores of all the candidates. This might not be desirable.

For strategy-resistant voting systems, [TRN08, BMN+09] explore the case of privacy-preserving STV using a mixing protocol. As far as we know, no such study nor implementation, where intermediate values are kept secret, has been done for Majority Judgment.

4.1.4 Our Contributions

In this chapter, we propose a protocol for Majority Judgment using a restricted number of logical gates. This protocol can then be extended to reveal the ordered winners for multi-seat elections. Our solution uses Paillier cryptosystem to distribute trust while keeping performance manageable. We then benchmark this protocol with a Python implementation. This protocol was initially presented in [CPST18a].

We stress that our approach provides strong cryptographic guarantees regarding confidentiality and integrity. Additionally, we support multi-seat elections, where several winners can be determined. All of this is done while revealing no intermediate value and with practical results: for 5 candidates and 1000 voters, the tallying phase gives the winner in less than 10 minutes.

We think that our work also serves to demonstrate that cryptography can provide strong guarantees even in complex settings while remaining practical. We hope it can

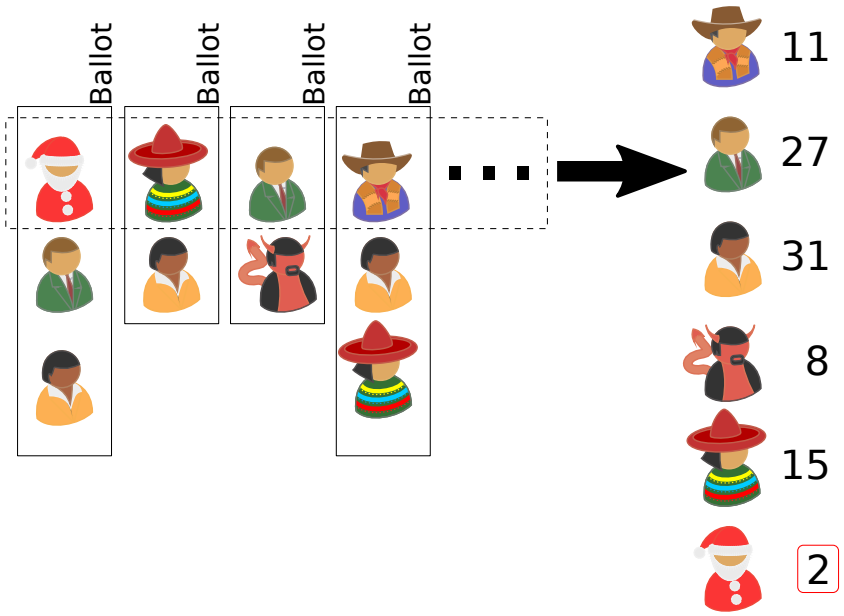


Figure 4.6: STV Tally, first round. Each ballot is represented vertically, with the candidates ranked by the voter. The first ballot (left) correspond to the one from Figure 4.4. During this first round, only the first choice of each ballot is considered. We count the number of votes for each candidate as usual. As for runoff-voting, we eliminate some candidates. Here, only the one with the least votes (Santa Claus) is eliminated.

help bridge the gap that continues to exist between theoretical cryptography (strong guarantees) and industrial practices (high efficiency) and encourage a more widespread use of cryptography to improve users' privacy in many applications.

4.1.5 Organization

Section 4.2 presents how Majority Judgement works, and how we can adapt the tallying to be encryption-friendly. This tells us what properties are needed for the encryption scheme. Section 4.3 explain what cryptographic tools will be used to solve the problem. Most cryptographic primitives have been introduced in Section 2.4 and Section 2.5. We then go through the different MPC gates used by the protocol in Section 4.4. Section 4.5 coalesces the building blocks from the previous sections into a full protocol, and offers benchmarks for a proof-of-concept implementation. Finally, we discuss the relevance of our results, and their relation to distributed technologies in Section 4.6.

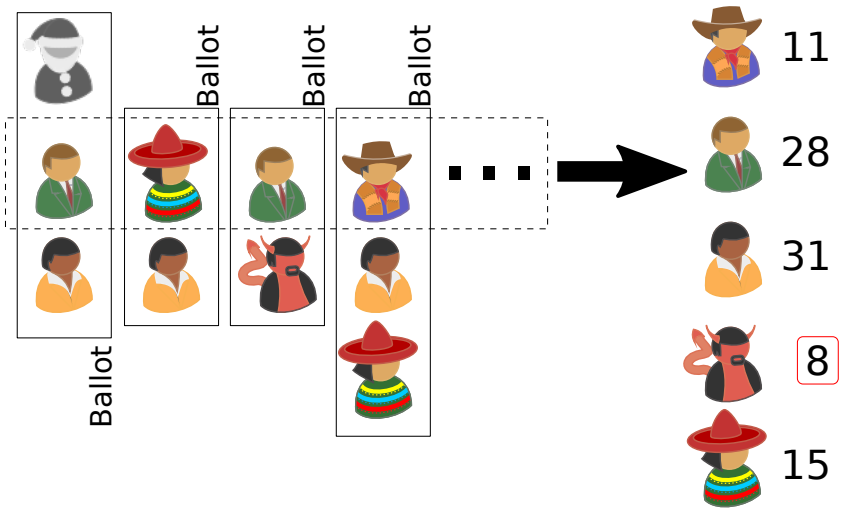


Figure 4.7: STV Tally, second round. After the first round, one candidate has been eliminated, voters must again cast their opinions. Instead of calling the voters back to the polls, we take advantage of the ranking on the ballots. Ballots whose top choice is not eliminated are kept as is. However, ballots which listed an eliminated candidate (Santa Claus here) as their first choices are shifted, and their second choices are counted. Here, one ranked Cowboy in second place and another Businessman. Again, the candidate with the least votes (Satan) is eliminated. This process repeats until there is only one candidate left.

4.2 Majority Judgment

4.2.1 Definition

Principle. Majority Judgment is a cardinal voting mechanism presented by Michel Balinski and Rida Laraki in [BL07, BL10] that claims to improve the legitimacy of the elected candidates.

In Majority Judgment, each voter $\mathcal{V}_k$ attributes a grade to each candidate $\mathcal{C}_i$. These grades have to be ordered values but are not necessarily numbers (e.g., A > B > C > D > E). The ballot $\mathcal{B}_k$ is structured as a matrix where each row represents a candidate and each column represents a grade. The voter writes a 1 in the chosen grade for each candidate and 0 in other cells. For instance, if voter k gives C to candidate Alice, B to candidate Bob and D to candidate Charlie, k casts the following ballot:

$$\begin{array}{c} \\ \text{Candidates} \end{array} \begin{array}{l} \\ \left\{ \begin{array}{l} \text{Alice} \\ \text{Bob} \\ \text{Charlie} \end{array} \right. \end{array} \begin{array}{c} \begin{array}{ccccc} + & & \leftarrow \text{Grades} \rightarrow & & - \\ \text{A} & \text{B} & \text{C} & \text{D} & \text{E} \end{array} \\ \left(\begin{array}{ccccc} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{array} \right) \end{array} = \mathcal{B}_k.$$

After combining all the ballots into an Aggregate Matrix $A = \sum_k \mathcal{B}_k$, the median

grade (or "majority-grade") of each candidate is computed: it is to the grade for which there are as many votes for worse grades as for better grades. Then, the candidate with the best median grade is elected. Continuing on our example, we may obtain the following Aggregate Matrix:

$$\begin{array}{rl} & \begin{array}{ccccc} + & \leftarrow \text{Grades} \rightarrow & & & - \\ \text{A} & \text{B} & \text{C} & \text{D} & \text{E} \end{array} \\ \text{Candidates} \left\{ \begin{array}{l} \text{Alice} \\ \text{Bob} \\ \text{Charlie} \end{array} \right. & \left(\begin{array}{ccccc} 31 & 151 & \mathbf{529} & 254 & 35 \\ 312 & 90 & \mathbf{76} & 107 & 305 \\ 101 & 7 & 2 & 86 & \mathbf{804} \end{array} \right) = A. \end{array}$$

Each cell represents the number of voters who gave this grade to that candidate. In bold are the candidates' median grades. Here, Alice and Bob have the same median grade, C, while Charlie has the median grade E. We can already eliminate Charlie, but cannot decide the winner yet.

Solving Ties. As in our example, several candidates will probably obtain the best median grade. In that case, we consider the grades lower than the median grade and the grades greater than the median grade to make a decision. We construct the Tiebreak Matrix T by aggregating grades to the left and to the right of the median grade:

$$\text{Candidates} \left\{ \begin{array}{l} \text{Alice} \\ \text{Bob} \end{array} \right. \left(\begin{array}{cc} 31+151 & 254+35 \\ 312+90 & 107+305 \end{array} \right) = \left(\begin{array}{cc} 182 & 289 \\ 402 & \mathbf{412} \end{array} \right) = T.$$

To make a decision, the largest of these values is considered. If it is in right column (many low grades), then the corresponding candidate is eliminated; if it is in left column (many high grades), then the corresponding candidate wins. In the Tiebreak Matrix of the example, the largest value, in bold, rejects Bob, so Alice is elected.

Formal definition of ordering. The result of each candidate is summed up as (p, α, q) where α is the median grade, p is the number of votes above α, and q, of those below. For two candidates $\mathcal{C}_A$ and $\mathcal{C}_B$ with results (p_A, α_A, p_B) and (p_B, α_B, q_B) respectively, $\mathcal{C}_A$ wins against $\mathcal{C}_B$ when either:

1. $\alpha_A > \alpha_B$ (better median grade);
2. $\alpha_A = \alpha_B \wedge p_A > q_A \wedge p_B < q_B$ ("stronger" median grade);
3. $\alpha_A = \alpha_B \wedge p_A > q_A \wedge p_B > q_B \wedge p_A > p_B$ (more secure median grade);
4. $\alpha_A = \alpha_B \wedge p_A < q_A \wedge p_B < q_B \wedge q_A < q_B$ (less insecure median grade).

This defines a total ordering on the candidates with high probability.

Goal statement. Our aim is to implement Majority Judgment in the encrypted domain: output the above ordering while leaking no additional information. We select a suitable encryption scheme and adapt the algorithm above to obtain the best possible efficiency/security trade-off.

4.2.2 Removing Branching

At first sight, it seems like Majority Judgment requires branching instructions (conditions and variable loops). This would be costly when implemented in the encrypted domain. However, it is possible to avoid branches and most of this overhead.

Early Elimination of Candidates. The first remark is that we can avoid explicitly checking condition 1 (comparing median grades): we build the Tiebreak Matrix with all candidates. In the previous use-case, C is the best median grade (for Alice and Bob), which leads to the following Tiebreak Matrix

$$T = \begin{pmatrix} 31+151 & 254+35 \\ 312+90 & 107+305 \\ 101+7 & 86+804 \end{pmatrix} = \begin{pmatrix} 182 & 289 \\ 402 & 412 \\ 108 & \mathbf{892} \end{pmatrix}$$

Charlie still gets eliminated, during tie-breaking, since he holds the highest count, in the right column.

More generally, let $\mathcal{C}_W$ and $\mathcal{C}_L$ be candidates with results (p_W, α_W, q_W) and (p_L, α_L, q_L) respectively, and such that $\alpha_W < \alpha_L$. Let us define p'_L (resp. q'_L) the number of votes for L that are better (resp. worse) than the best median grade, α_W (instead of the candidate's, α_L). Because $\alpha_L < \alpha_W$, $q'_L > \frac{1}{2}$, but we also have $q_W < \frac{1}{2}$, and thus $q_W < q'_L$. As a consequence, we only need to compute the p' and q' values, defined around the best median grade (rather than each candidate's). We are left with the following conditions to determine whether candidate $\mathcal{C}_A$ wins against candidate $\mathcal{C}_B$:

2'. $p'_A > q'_A \wedge p'_B < q'_B$;

3'. $p'_A > q'_A \wedge p'_B > q'_B \wedge p'_A > p'_B$;

4'. $p'_A < q'_A \wedge p'_B < q'_B \wedge q'_A < q'_B$.

Building the Tiebreak Matrix T. We need to detect which columns of the Aggregate Matrix $A = (a_{i,j})$ are to the left (resp. right) of the best median grade. For this, we introduce the Candidate Matrix $C = (c_{i,j})$, such that $c_{i,j} = 1$ when the j-th grade is better than the candidate's median grade:

$$c_{i,j} = \begin{cases} 1 & \text{if } 2 \times \sum_{k<j} a_{i,k} < \sum_k a_{i,k} \\ 0 & \text{otherwise.} \end{cases}$$

In our example, C is as follows, where the zeroes in bold correspond to the median grade for each candidate:

$$C = \begin{pmatrix} 1 & 1 & \mathbf{0} & 0 & 0 \\ 1 & 1 & \mathbf{0} & 0 & 0 \\ 1 & 1 & 1 & 1 & \mathbf{0} \end{pmatrix}.$$

Then, we can compute the Grade Vector $G = (g_j)$, such that $g_j = 1$ when the j-th grade is better than the best median grade: $g_j = \prod c_{i,j}$. In the example:

$$G = \begin{pmatrix} 1 & 1 & \mathbf{0} & 0 & 0 \end{pmatrix}.$$

The zero in bold corresponds to the best median grade.

Once we have this Grade Vector G, we can build the two columns of the Tiebreak Matrix $T = (t_{i,k})$, from the Aggregate Matrix $A = (a_{i,j})$, as:

$$t_{i,1} = \sum_{\substack{j \\ g_j=1}} a_{i,j} \text{ and } t_{i,2} = \sum_{\substack{j \\ g_{j-1}=0}} a_{i,j}.$$

Note that the votes for the best median grade are discarded. This can be written as $t_{i,1} = \sum_j g_j \times a_{i,j}$ and $t_{i,2} = \sum_j (1 - g_{j-1}) \times a_{i,j}$.

Identifying the Winner. From T, we can define the Winning Vector $W = (w_i)$ such that $w_i = 1$ when candidate i wins the election:

$$w_i = \bigwedge_{\substack{j \\ j \neq i}} \begin{pmatrix} (t_{i,1} > t_{i,2} \wedge t_{j,1} < t_{j,2}) \\ \vee (t_{i,1} > t_{i,2} \wedge t_{j,1} > t_{j,2} \wedge t_{i,1} > t_{j,1}) \\ \vee (t_{i,1} < t_{i,2} \wedge t_{j,1} < t_{j,2} \wedge t_{i,2} < t_{j,2}) \end{pmatrix}$$

This is a translation of conditions 2', 3' and 4' between each pair of candidates (i, j). The elected candidate corresponds to the unique w_i equal to 1.

For multi-seat elections, we remove from A and C the line corresponding to the winner and compute G, T and W again.

4.2.3 Expected Features for Encrypted Implementation

To run the algorithm above with encrypted ballots, the encryption scheme must allow the following operations:

- *addition of two plaintexts*, to compute the Aggregate Matrix A and the Tiebreak Matrix T;
- *comparison of two plaintexts*, to compute the Candidate Matrix C and the Winning Vector W;
- *multiplication of two plaintexts*, to compute the Grade Vector G and the Tiebreak Matrix T;
- *AND/OR between two plaintexts*, to compute the Winning Vector W.

In the end, a decryption of the Winning Vector W reveals the result.

Notice that all the multiplications have at least one operand restricted to $\{0, 1\}$. This relaxes the requirement from a full multiplication gate to a "*Conditional Gate*": $\mathsf{CondGate}(x, y) = x \times y$ for $y \in \{0, 1\}$.

When $(\mathsf{False}, \mathsf{True})$ maps to $(0, 1)$, the Conditional Gate also gives us Boolean AND. With an additively homomorphic encryption scheme, we have the logical NOT gate for free, since $\neg x = 1 - x$. Eventually, the AND and NOT gates let us construct the missing gates.

To sum up, we want an additively homomorphic encryption scheme, together with efficient Multi-Party Computation (MPC) protocols for distributed decryption, distributed evaluation of the Conditional Gate, and distributed comparison. Multi-party computation requires zero-knowledge proofs to ensure correctness. In the next section, we implement these building blocks.

4.3 Cryptographic Tools

The secret key of the election is needed to decrypt the ballots at the end of the tally and for the gates that help obtain the intermediate values. Because no one entity should hold the power to decrypt the ballots, we use several authorities that need to cooperate to use the secret key. Our MPC protocols are run the talliers $(T_i)_i$ who share a secret using Shamir's method (see Subsection 2.4.1).

4.3.1 Batching Zero-Knowledge Proofs

We use many ZKPs, introduced in Subsection 2.4.2. Because of their high cost, they represent a large part of the total running time of the protocol. To reduce their cost, we can run batch them together. That is, we combine the values of several ZKPs so that a single computation can be used to verify all of the proofs simultaneously.

We aggregate the proofs of valid decryption when several ciphertexts are decrypted by the same tallier, as done in [APB+04, Appendix C]. For instance, let us consider that T_i is participating in the decryption of ciphertexts $(C_j)_j$. Then, T_i publishes $R_j \leftarrow C_j^{s_i} \bmod n$, for each j. The verifier wants to run a protocol with T_i for each R_j to verify that they were computed correctly. Instead, the verifier generates random scalars $\alpha_j \xleftarrow{\$} \mathbb{Z}_{2^\kappa}$ and share these values with T_i. The proof proceeds on the aggregate ciphertext $C^* \leftarrow \prod_j C_j^{\alpha_j} \bmod n$ and plaintext $R^* \leftarrow \prod_j R_j^{\alpha_j} \bmod n$. With this approach, T_i only need to prove that R^* is a valid partial decryption of C^*, and this convinces the verifier that all of the R_j are correctly computed.

Remark 10. *It is important to ensure that T_i publishes $(R_j)_j$ before the verifier gives $(\alpha_j)_j$. However, it is possible to make this part non-interactive by using a random oracle and including $(R_j)_j$ in the request. Then, the tallier can also provide a non-interactive zero-knowledge proof of the equality of the discrete logarithms. This way, the whole proof can be made non-interactive.*

4.3.2 Proof of Private Multiplication

$\mathcal{P}$ privately multiplies $[\![z]\!] \leftarrow [\![x]\!]^y$. For correctness, $\mathcal{P}$ proves that $[\![x]\!]$, $[\![y]\!]$, $[\![z]\!]$ are encryptions of some x, y, and z such that $z \equiv x \times y \mod n$:

- $\mathcal{P}$ draws $u \xleftarrow{\$} \mathbb{Z}_n$, $r_u, r_{yu} \xleftarrow{\$} \mathbb{Z}_n^*$ and sends $[\![u]\!] \leftarrow g^u r_u^n \bmod n^2$ and $[\![yu]\!] \leftarrow [\![y]\!]^u r_{yu}^n \bmod n^2$ to $\mathcal{V}$;

- $\mathcal{V}$ draws a challenge $e \xleftarrow{\$} \mathbb{Z}_{2^\kappa}$ and sends it to $\mathcal{P}$;
- $\mathcal{P}$ sends $w \leftarrow u - xe \bmod n$, $r_w \leftarrow r_u r_x^{-e} \bmod n$ and $r_{yw} \leftarrow r_{yu} r_z^{-e} \bmod n$ to $\mathcal{V}$;
- $\mathcal{V}$ checks that $[\![u]\!] \equiv g^w r_w^n [\![x]\!]^e \mod n^2$ and $[\![yu]\!] \equiv [\![y]\!]^w r_{yw}^n [\![z]\!]^e \mod n^2$.

The Fiat-Shamir heuristic (see Section 2.4.2) makes it non-interactive.

The proofs for several private multiplications of the same x with different y_i's can be batched: $x \times (\sum_i \alpha_i y_i) \equiv \sum_i \alpha_i (x \times y_i) \mod n$. This does not work for independent pairs (x_i, y_i).

4.4 Gate Evaluation with Multi-Party Computation

We now present MPC protocols that implement the operations listed at the end of Subsection 4.2.3:

- Decryption: our peculiar use of the Paillier cryptosystem simplifies this tasks, since each tallier only needs to compute a simple exponentiation; this gate is also used by the other two protocols below;
- Conditional Gate: this is realized by randomizing the Boolean operand, decrypting it, and doing a private multiplication;
- Comparison: requires individually encrypted bits, which we obtain by decrypting the masked the operands, encrypting their bits individually, and removing the mask by adding bitwise.

These protocols are secure in the malicious setting thanks to proofs of correct execution. We modularize our implementation into several gates, which can be visualized in Figure 4.8.

4.4.1 Decryption Gate

As seen in Subsection 2.5.4, we can configure our setup so that the Decrypt algorithm for the Paillier cryptosystem is essentially a single exponentiation. We must compute $R \equiv C^s \equiv r \mod n$, then a simple operation with only public values gives us back the plaintext message M. Since the secret value is only used in the exponent, this operation can easily be distributed by using Shamir's secret sharing (see Subsection 2.4.1). To be more specific, each tallier T_i provides the share $R_i \leftarrow C^{s_i} \bmod n$ of the decryption and a non-interactive zero-knowledge proof. Once all the decryption shares have been computed, they can be combined into:

$$\prod R_i^{\lambda_i} \equiv C^{\lambda_i s_i} \equiv C^s \equiv R \mod n$$

where λ_i are the Lagrange coefficients for the selected talliers normally used to reconstruct the secret in Shamir's secret sharing.

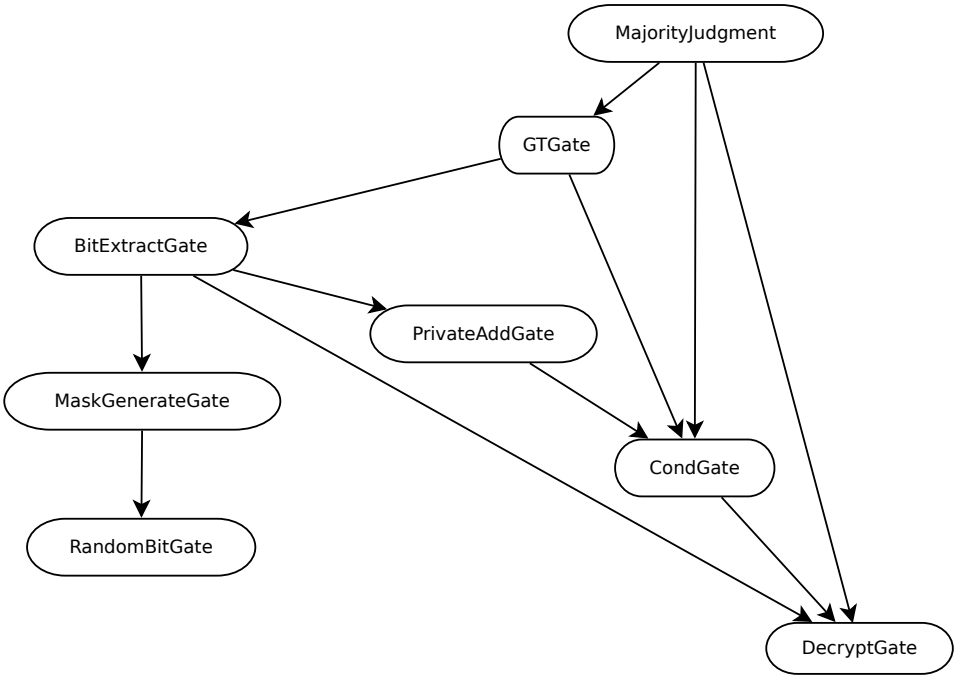


Figure 4.8: Call Graph: an edge indicates that the originating gate calls the target gate; note the three main gates, directly used by MajorityJudgment

4.4.2 Conditional Gate

Schoenmakers and Tuyls introduced the Conditional Gate in [ST04]. Given $x \in \mathbb{Z}_n$ and $y \in \{-1, 1\}$, it computes $x \times y \in \{x, -x \bmod n\}$. Adapting this into a gate taking $y \in \{0, 1\}$ is trivial.

Conditional Gate. First, each tallier masks y in turn, by multiplying it by -1 or 1. They do the same transformation on x to keep $x \times y$ unchanged. Then, they decrypt the final y and do a private multiplication. In Algorithm 3, decryption of $[\![y]\!]$ is performed in a distributed way as shown before.

For the malicious setting, the talliers give proofs of equality of discrete logarithm between $[\![x]\!]$ and $[\![y]\!]$ (i.e., both values were raised to the same o). As explained in [ST04], no proof that $o \in \{-1, 1\}$ is needed, as long as the decrypted y is in $\{-1, 1\}$.

Mapping to $\{0, 1\}$. The additive property let us adapt the Conditional Gate to accept its second operand from $\{0, 1\}$: $y \in \{0, 1\} \Rightarrow 2y - 1 \in \{-1, 1\}$. With input $(x, 2y - 1)$, Algorithm 3 outputs $z = 2xy - x$. We get the desired result by outputting $(z + x) \times 2^{-1} \bmod n$

Input: $[\![x]\!]$ and $[\![y]\!]$, encryptions of $x \in \mathbb{Z}_n$ and $y \in \{-1, 1\}$ respectively
Output: $[\![z]\!]$, the encryption of $z = x \times y$
foreach *tallier* **do**
  $o \xleftarrow{\$} \{-1, 1\}$;
  $[\![x]\!] \leftarrow [\![x]\!]^o$; `/* private multiplication` $x \leftarrow x \times o$ `*/`
  $[\![y]\!] \leftarrow [\![y]\!]^o$; `/* private multiplication` $y \leftarrow y \times o$ `*/`
  pass $[\![x]\!]$, $[\![y]\!]$ to the next tallier;
end
$y \leftarrow \mathsf{Decrypt}([\![y]\!])$;
assert $y \in \{-1, 1\}$;
return $[\![z]\!] = [\![x]\!]^y$; `/* private multiplication` $z \leftarrow x \times y$ `*/`

Algorithm 3: CondGate: Conditional Gate

4.4.3 Greater-Than Gate

We use a comparison circuit on the bitwise encryptions (each of their bits encrypted separately) of the operand. The hard part is converting an encrypted integer into the encryptions of its bits. This is done by the Bit-Extraction Gate.

In short, this gate masks the input integer, decrypts the result, encrypts the individual bits of this value, and removes the mask on the bitwise encryption by using a binary addition circuit (see Figure 4.9). The binary addition circuits, or Private Addition Gate, takes two operands: the masked bitwise encryption of the input, and the unencrypted mask. The fact the one operand is not encrypted saves several executions of the Conditional Gate. To prevent any part from knowing the mask, it is generated using another Multiparty Computation gate, GenerateMaskGate.

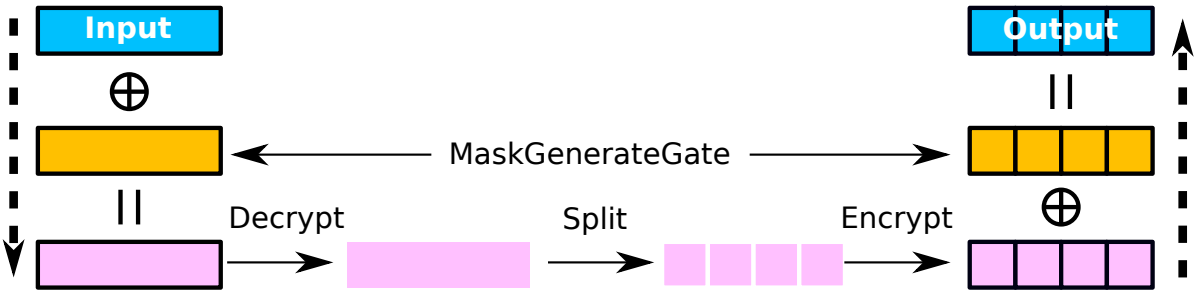


Figure 4.9: Bit-Extraction Gate: the input is combined with a random value, decrypted, split, encrypted bitwise, and the random value is removed

Private Addition Gate. In the Algorithm 4, x_i is known as an encrypted value $[\![x_i]\!]$, while y_i is a plaintext value.

Remark 11. *[DFK+06, Section 6] offers a constant round circuit for addition but the constant is a 37. Since y is given in the clear, we avoid the round of line 1 of* CARRIES.

Input: $([\![x_i]\!])$ the bitwise encryption of $x \in \mathbb{Z}_n$, (y_i) the bits of $y \in \mathbb{Z}_n$
Output: $([\![z_i]\!])$ the bitwise encryption of $z = x + y$
$[\![c]\!] \leftarrow \mathsf{Encrypt}(0)$; /* carry */
foreach *index i, from 0 — the least significant bit—* **do**
  $[\![x_i \oplus y_i]\!] \leftarrow [\![x_i]\!] \times [\![y_i]\!] \div [\![x_i]\!]^{2y_i}$; /* $x_i \oplus y_i = x_i + y_i - 2x_iy_i$ */
  $[\![z_i]\!] \leftarrow [\![x_i \oplus y_i]\!] \times [\![c]\!] \div \mathsf{CondGate}([\![x_i \oplus y_i]\!], [\![c]\!])^2$; /* $(x_i \oplus y_i) \oplus c$ */
  $[\![c]\!] \leftarrow ([\![x_i]\!] \times [\![y_i]\!] \times [\![c]\!] \div [\![z_i]\!])^{\frac{1}{2}}$; /* $(x_i + y_i + c - z_i)/2$ */
end
return $([\![z_i]\!])$;

Algorithm 4: PrivateAddGate: Private Addition Gate

Input: $[\![x]\!]$, the encryption of $x \in \mathbb{Z}_n$
Output: $([\![x_i]\!])$, the bitwise encryption of x
$[\![y]\!], ([\![y_i]\!]) \leftarrow \mathsf{GenerateMaskGate}(\ell, \kappa)$;
$[\![z]\!] \leftarrow [\![x]\!] \div [\![y]\!]$; /* mask $z = x - y$ */
$z \leftarrow \mathsf{DecryptGate}([\![z]\!])$; /* decrypt as number in $[-n/2, n/2]$ */
$([\![x_i]\!]) \leftarrow \mathsf{PrivateAddGate}(([\![y_i]\!]), (z_i))$; /* unmask $(x_i) = (y_i) + (z_i)$ */
return $([\![x_i]\!])$;

Algorithm 5: BitExtractGate: Bit-Extraction Gate

In contrast, our straight-forward Private Addition Gate implies ℓ rounds for up to $2^\ell - 1$ votes, so less than 36 for practical cases.

Bit-Extraction Gate. Schoenmakers and Tuyls present the Bit-Extraction Gate (Algorithm 5) in [ST06, LSBs gate]. $\mathsf{GenerateMaskGate}(\ell, \kappa)$ returns the scalar and bitwise encryptions of a random secret mask for ℓ bits with security κ. We apply this mask to the scalar input, and then remove it bitwise after decryption.

Remark 12. *Drawing the mask from $\mathbb{Z}_N$ could cause wrapping when adding with x. Instead, we draw from $[0, 2^{\ell+\kappa})$ for security parameter κ. See Subsection 4.4.4 for implementation.*

Greater-Than Gate. Finally, we compare two integers by applying the comparison circuit from [ST04] on their bitwise encryptions. The circuit updates $t_{i+1} \leftarrow (1 - (x_i - y_i)^2)t_i + x_i(1 - y_i)$, starting from $t_0 = 0$. See Algorithm 6.

Remark 13. *[DFK+06] also has a 19-round circuit for comparison.*

4.4.4 Mask Generation Gate

The Mask Generation Gate returns a random encrypted value known by no one. We need a bitwise encryption to remove the mask using the Private Addition Gate. Drawing

Input: $([\![x_i]\!])$, $([\![y_i]\!])$ the bitwise encryptions of $x, y \in \mathbb{Z}_n$
Output: $[\![x > y]\!]$, encryption of 1 if $x > y$, and 0 otherwise
$[\![t]\!] \leftarrow \mathsf{Encrypt}(0)$; /* temporary result */
foreach *index i, from 0 — the least significant bit—* **do**
  $[\![x_i \wedge y_i]\!] \leftarrow \mathsf{CondGate}([\![x_i]\!], [\![y_i]\!])$;
  $[\![a]\!] \leftarrow [\![1]\!] \div [\![x_i]\!] \div [\![y_i]\!] \times [\![x_i \wedge y_i]\!]^2$; /* $1 - (x_i - y_i)^2$ */
  $[\![b]\!] \leftarrow \mathsf{CondGate}([\![a]\!], [\![t]\!])$; /* $(1 - (x_i - y_i)^2)t_i$ */
  $[\![t]\!] \leftarrow [\![b]\!] \times [\![x_i]\!] \div [\![x_i \wedge y_i]\!]$; /* $(1 - (x_i - y_i)^2)t_i + x_i(1 - y_i)$ */
end
return $[\![t]\!]$;

Algorithm 6: GTGate: Greater-Than Gate

Input: -
Output: $[\![b]\!]$ the encryption of secret $b \xleftarrow{\$} \{0, 1\}$
$[\![b]\!] \leftarrow \mathsf{Encrypt}(0)$;
foreach *tallier T_i* **do**
  $b_i \xleftarrow{\$} \{0, 1\}$;
  $[\![b_i]\!] \leftarrow \mathsf{Encrypt}(b_i)$;
  $[\![b]\!] \leftarrow [\![b]\!] \times [\![b_i]\!] \div [\![b]\!]^{b_i}$; /* $b \leftarrow b \oplus b_i = b + b_i - b \times b_i$ */
  T_i sends $[\![b]\!]$ to T_{i+1} and a ZKP of $b_i \in \{0, 1\}$ verifying the relation;
end
return $[\![b]\!]$;

Algorithm 7: RandomBitGate: Random Bit Gate

the mask from $\mathbb{Z}_N$ could cause wrapping so we draw from $[0, 2^{\ell+\kappa})$ for security parameter κ.

The Random Bit Gate of Algorithm 7 returns the encryption of a random bit known by no one. Each tallier T_i XORs the current encrypted bit b with a locally drawn bit b_i. For correctness, T_i provides a ZKPoK of b_i such that $b_i \in \{0, 1\}$ and that $b' = b \oplus b_i$ where b' is the new value of b.

The Basic Random Mask Gate of Algorithm 8 combines generated bits into a scalar encryption of the mask.

Optimized Random Mask Gate of Algorithm 9 avoids generating the last κ bits since

Input: ℓ the size of the mask, κ the security parameter
Output: $[\![r]\!]/([\![r_i]\!])$, scalar/bitwise encryptions of secret $r \xleftarrow{\$} [0, 2^{\ell+\kappa})$
$\forall 0 \leq i < \ell + \kappa, [\![r_i]\!] \xleftarrow{\$} \mathsf{RandomBitGate}()$;
$[\![r]\!] \leftarrow \prod_i [\![r_i]\!]^{2^i}$; /* $r \leftarrow \sum r_i 2^i$ */
return $[\![r]\!]$, $([\![r_i]\!])$;

Algorithm 8: GenerateMaskGate: Basic Random Mask Gate

Input: ℓ the size of the mask, κ the security parameter
Output: $[\![r]\!]/([\![r_i]\!])$, scalar/bitwise encryptions of secret $r \xleftarrow{\$} [0, 2^{\ell+\kappa})$
$\forall 0 \leq i < \ell, [\![r_i]\!] \xleftarrow{\$} \mathsf{RandomBitGate}()$;
foreach *tallier* T_i **do**
  T_i draws $r_{*,i} \xleftarrow{\$} [0, 2^{\ell+\kappa})$;
  $[\![r_{*,i}]\!] \leftarrow \mathsf{Encrypt}(r_{*,i})$;
  T_i broadcasts $[\![r_{*,i}]\!]$ with range proof;
end
$[\![r_*]\!] \leftarrow \prod_i [\![r_{*,i}]\!]$; /* $r_* \leftarrow \sum r_{*,i}$ */
return $[\![r]\!]$, $([\![r_i]\!])$;

Algorithm 9: GenerateMaskGate: Optimized Random Mask Gate

their bitwise encryptions are not needed. Instead, it uses a random value in $[0, 2^\kappa)$. For correctness, each tallier provides a ZKPoK of $r_{*,i*} \in [0, 2^{l+\kappa})$ (range-proof).

Remark 14. *In the optimized version, r can actually be larger than $2^{\ell+\kappa}$. This does not affect the correctness since $\ell + \kappa[\![|N|$.*

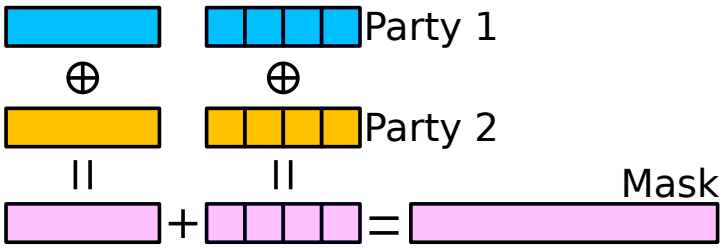


Figure 4.10: Mask Generation Gate: the parties choose random bits and a random scalar; they combine the scalars and the bits individually; then they add the bits (which make the least significant part), and the scalar (which makes the most significant part)

4.5 Implementation

Using the various building blocks presented in the sections above, we implemented a cryptographic protocol for Majority Judgment. We realized a proof-of-concept and measured the running time of its execution to provide realistic benchmarks. We now explain how the encryption of the ballots works, present the optimizations that were used to improve the running time, and list the results of our measurements.

4.5.1 Ballot Encryption

Each voter $\mathcal{V}_k$ encrypts ballot $\mathcal{B}_k$ element-wise and proves that each element is either 0 or 1 (OR-proof), and that there is one 1 per row (sum decrypts to 1). Then $A = \sum_k \mathcal{B}_k$

using homomorphic addition. At this point, we could decrypt A and finish tallying in the clear, while keeping ballot confidentiality. Instead, we stay in the encrypted domain and implement the approach described in Subsection 4.2.2.

4.5.2 Optimizations

Avoiding Final Logical Gates. Boolean OR cannot be implemented exactly with homomorphic addition. However, for the final value, it only matters whether the result is 0 or not. This saves a few Conditional Gates in the final part of the protocol:

$$\neg w_i = \bigvee_{j \neq i} \begin{pmatrix} \neg(t_{i,1} > t_{i,2} \wedge t_{j,1} < t_{j,2}) \\ \wedge \neg(t_{i,1} > t_{i,2} \wedge t_{j,1} > t_{j,2} \wedge t_{i,1} > t_{j,1}) \\ \wedge \neg(t_{i,1} < t_{i,2} \wedge t_{j,1} < t_{j,2} \wedge t_{i,2} < t_{j,2}) \end{pmatrix}$$

becomes

$$\ell_i = \sum_{j \neq i} \begin{pmatrix} \neg(t_{i,1} > t_{i,2} \wedge t_{j,1} < t_{j,2}) \\ \wedge \neg(t_{i,1} > t_{i,2} \wedge t_{j,1} > t_{j,2} \wedge t_{i,1} > t_{j,1}) \\ \wedge \neg(t_{i,1} < t_{i,2} \wedge t_{j,1} < t_{j,2} \wedge t_{i,2} < t_{j,2}) \end{pmatrix}.$$

Candidate i wins if and only if $\ell_i \neq 0$. To avoid leaking information, each tallier multiplies l_i by a secret non-zero value before decryption.

Batching. Our software implementation batches operations as much as possible to reduce the number of modular exponentiations. More specifically, each gate receives a list of inputs to process and callers try to regroup inputs before calling gates. When possible, the gates then batch the proofs of valid decryption or of private multiplication over their inputs. Unfortunately, we can only batch the proofs of multiplication in some specific cases.

Pipelining. The most critical part of the protocol is the Conditional Gate. A straightforward implementation would have only one tallier active at any given time. Instead, for α tallier, we split the input batch in α sub-batches, give one sub-batch to each tallier, and rotate them. This way, each tallier can still negate any of the inputs, but most of the idle CPU time vanishes.

Remark 15. *After candidate $\mathcal{C}_W$ is elected, it is possible to reveal the next candidate by removing the line W of A and repeating the algorithm. This allows for multi-seat elections, but does reveal the order of the elected candidates.*

4.5.3 Summary

Find the full protocol for Majority Judgment in Algorithm 10. To improve readability, $\mathsf{CondGate}(\llbracket x_i \rrbracket)$ is the encryption of $\prod x_i$ for two values or more. We defend against malicious adversary by using ZKPs when randomizing ℓ_i and in the basic gates (see

Input: encrypted Aggregate Matrix ($[\![a_{i,j}]\!]$)
Output: index of elected candidate
/* Candidate matrix $c_{i,j} = \sum_{k<j} a_{i,k} < \frac{1}{2} \times \sum_k a_{i,k}$ */
$[\![c_{i,j}]\!] \leftarrow \mathsf{GTGate}(\prod_k [\![a_{i,k}]\!], (\prod_{1 \le k \le j} [\![a_{i,k}]\!])^2)$;
/* Grade vector $g_j = \wedge_i c_{i,j}$ */
$[\![g_j]\!] \leftarrow \mathsf{CondGate}([\![c_{i,j}]\!])$;
/* Tiebreak matrix left column $t_{i,1} = \sum_j g_j \times a_{i,j}$ */
$[\![t_{i,1}]\!] \leftarrow \prod_j \mathsf{CondGate}([\![a_{i,j}]\!], [\![g_j]\!])$;
/* Tiebreak matrix right column $t_{i,2} = \sum_j (1 - g_{j-1}) \times a_{i,j}$ */
$[\![t_{i,2}]\!] \leftarrow \prod_j \mathsf{CondGate}([\![a_{i,j}]\!], ([\![1]\!] \div [\![g_{j-1}]\!]))$;
/* Each assignment maps to an inner parenthesis of ℓ_i */
$[\![p^1_{i,j}]\!] \leftarrow \mathsf{CondGate}(\mathsf{GTGate}([\![t_{i,1}]\!], [\![t_{i,2}]\!]), \mathsf{GTGate}([\![t_{j,2}]\!], [\![t_{j,1}]\!]))$;
$[\![p^2_{i,j}]\!] \leftarrow \mathsf{CondGate}(\mathsf{GTGate}([\![t_{i,1}]\!], [\![t_{i,2}]\!]), \mathsf{GTGate}([\![t_{j,1}]\!], [\![t_{j,2}]\!]), \mathsf{GTGate}([\![t_{i,1}]\!], [\![t_{j,1}]\!]))$;

$[\![p^3_{i,j}]\!] \leftarrow \mathsf{CondGate}(\mathsf{GTGate}([\![t_{i,2}]\!], [\![t_{i,1}]\!]), \mathsf{GTGate}([\![t_{j,2}]\!], [\![t_{j,1}]\!]), \mathsf{GTGate}([\![t_{j,2}]\!], [\![t_{i,2}]\!]))$;

/* Losing Vector $\ell_i = \sum_{j \neq i} \neg p^1_{i,j} \wedge \neg p^2_{i,j} \wedge \neg p^3_{i,j}$ */
$[\![\ell_i]\!] \leftarrow \prod_{j \neq i} \mathsf{CondGate}([\![1]\!] \div [\![p^1_{i,j}]\!], [\![1]\!] \div [\![p^2_{i,j}]\!], [\![1]\!] \div [\![p^3_{i,j}]\!])$;
foreach *tallier* **do**
  $o \xleftarrow{\$} \mathbb{Z}_n$;
  $[\![\ell_i]\!] \leftarrow [\![\ell_i]\!]^o$;
end
$\ell_i \leftarrow \mathsf{DecryptGate}([\![\ell_i]\!])$;
return unique i such that $\ell_i = 0$;

Algorithm 10: Full Protocol

Section 4.4). The talliers ensure the consistency of the computation by all running the same protocol, and verifying that the others behave correctly in each gate. They can verify the proofs asynchronously, but must complete the checks before decrypting (ℓ_i).

4.5.4 Benchmarks

Our Python implementation uses `gmpy2` (GMP's `powmod` is faster than CPython's `pow`). For simplification, we use a central point of coordination (security does not rely on it), and basic secret sharing. We use standard RSA keys of 2,048 bits (keys are 1,024 bits and Paillier ciphertexts are 4,096 bits). Run times are shown in Figure 4.11, but exclude ballot encryption (on voters' devices), verification, and aggregation (on-the-fly). Most of it is for private multiplication proofs. We could do better with many CPU cores, but we are below 20 minutes for a 5-candidate election and a million voters on a single CPU!

Regarding ballot encryption and the associated proofs, note that this can be almost

	3 candidates	5 candidates
Up to $2^{10} - 1$ voters	4' 26"	09' 53"
Up to $2^{20} - 1$ voters	8' 53"	19' 05"

Figure 4.11: Tallying for 3 talliers/5 grades, run on 2 CPU cores (i5-4300U)

entirely pre-computed on the voters' devices: only the order of ciphertexts in each row depends on the choice of the voter. In total, ballot generation takes about one second without parallelization.

Our results shows that our protocol can truly be used in a real-world election.

4.6 Discussion

In cryptographic research, large efforts are currently focused on improving generic constructs for secure computation. Some frameworks do show interesting promises and might some day give us a silver bullet to protect most computations. However, we want to emphasize that the current state of cryptography and of computing already allows us to envision complex systems protected by cryptography.

There have been numerous initiatives aiming to implement electronic voting using Blockchain ([Aye17], [MSH17], [Lai18], Votem[1], Follow My Vote[2], TIVI[3], Voatz[4]). Indeed, elections are a critical component of democratic systems, and should not be entrusted to anyone. Thus, it seems logical to use a blockchain to implement a fully decentralized voting system. However, such initiatives often fail to take several aspects into consideration.

Firstly, blockchains themselves do not use a fair voting system, but instead usually rely on proof-of-work. This is because the notion of digital identity in a fully distributed setting is unreliable. In fact, this is a well-known problem for online communities, where it is generally easy to conduct a Sybil attack and bias the results. Such systems rely on the low stakes of the vote, and the non-null amount of effort required to conduct the attack. However, this is not acceptable for more important polls and for elections. In such situations, it is necessary to authenticate the voters (potentially anonymously) to ensure that no one can vote twice. As of now, we rely on institutions to validate the identity of a person. In consequence, any system that would source identity information from these institutions would already have a centralized consensus. This consideration alone challenges the relevance of Blockchain for this use case.

Secondly, we have observed that it is commonly believed that Blockchain itself provides "security". However, this vague concept is rarely scrutinized, and some projects expect private information stored on a blockchain to be confidential. Other projects do understand that sensitive information should be encrypted, but think that Blockchain

[1] https://votem.com/
[2] https://followmyvote.com/
[3] https://tivi.io/
[4] https://voatz.com/

can efficiently decide which ciphertexts can be decrypted. But the difficult part is not to identify the ciphertexts that should be decrypted, but rather decrypting them without revealing the decryption key. In a fully distributed setting, we cannot clearly identify several parties among which a secret key can be shared. Thus, we cannot strictly avoid the situation where a party collects several secret shares. Instead, we would again need to rely on proof-of-work to use the stronger assumption that no party can provide 51 % of the effort.

Thirdly, there are often other layers of protection than cryptography for particularly sensitive data. For instance, electronic voting systems in France must be hosted on the French territory[5]. Using a "public" blockchain would necessarily imply surrendering the control of the data. As we discussed, permissioned blockchains seem to bring little compared to normal, or "public", blockchains. Furthermore, Blockchain relies on economic incentives for the miners in the form of a cryptocurrency. This kind of setup might become problematic in the context of elections. In contrast, our solution can let the users inspect all of the encrypted ballots to verify the integrity of the election, but it can also restrict this function to identified parties.

We conclude that using Blockchain is not a sound approach for this use case, given the current implementation of identity, and the requirements on privacy.

4.7 Conclusion

In this chapter, we have explored the problematic of online voting both from a security viewpoint and with a practical aim. The voting system we have considered includes a relatively more involved tallying phase than in the traditional ones. However, we were able to design a problem-specific solution which seems to run in reasonable time. Again, this shows that specific constructions can be used to solve practical problems more efficiently then generic approaches. As for Chapter 3, we hope that this type of design can help spread cryptography further and provide solutions for new use cases.

[5] https://www.legifrance.gouv.fr/affichCnil.do?id=CNILTEXT000023174487

5 Conclusion

In this document, we explored two problems with the aim of providing strong security guarantees with a practical solution.

For our first use case, presented in Chapter 3, we have considered the concrete problem of organ donation and taken into account the associated practical constraints. In particular, the requisites bring a paradox, where the confidentiality of private medical records must rely on the entropy of the inputs, but we actually hope for low entropy (finding matches). We have shown that it was possible to sidestep this issue by establishing a stricter control of the encrypted data, in the form of fingerprinting. Then, we have searched for a type of cryptographic primitive that would let us achieve this goal. For this, we have introduced a formal model describing how such a primitive would work various settings, which can map to different use cases. Then, we have provided a construction connected to the state-of-the-art and relying on reasonable cryptographic assumptions. This approaches enables us to provide a balance between security and practicality in this use case.

For our second use case, presented in Chapter 4, we have looked at the existing literature for voting systems, electronic voting and cryptographic voting, and have extended what was possible. Concretely, we have decided to implement Majority Judgment, the voting system which provides the most interesting properties regarding strategy-resistance and the legitimacy of the result. For this, we combined various techniques from MPC, notably multiparty gates designed for ElGamal's encryption scheme that we adapted for Pailler's encryption scheme. Once we attained a complete theoretical design with strong guarantees for both confidentiality of the ballots and integrity of the election, we realized a software implementation. This software implementation confirmed the correctness of the protocol and provided us with rough estimations of the running times required to use this approach in a real election. Although we are convinced that these benchmarks could be improved significantly, notably by being run on many CPUs in parallel, the results already show that this solution is viable.

In both of these use cases, we have considered the relevance of Blockchain and similar technologies inspired by Bitcoin. We have found that, like in many purported use cases, using a blockchain would not help in solving the problem. Instead, we support that it is more sensible to design a cryptographic protocol adapted to the problem, so as to balance functionality, security, and efficiency.

The progress in generic cryptographic computation such as FHE, GCs and MPC are promising. However, our results demonstrate that cryptography can already be used to distribute trust among several actors while meeting pragmatic constraints. Thus, it could contribute to many more areas today while combining security with efficiency. We hope that this can help spread cryptography further and improve the guarantees given

to users in many applications.

Finally, we underline that there remain many open questions, both in general, and for the use cases we considered. First, the potential improvements to come for generic cryptographic computations, might continue to extend what is possible. Second, Blockchain currently have little use cases besides cryptocurrencies, and cryptocurrencies themselves have limited functionality, but organizational changes that could happen in the future might help distribute some features, such as authentication, and make Blockchain relevant for other use cases. Thirdly, the model we introduced in Subsection 3.3.2 might be pertinent for other use cases than the one we considered; then, the construction we proposed in Section 3.5 might be adapted, or others may be designed to better fit the constraints. Fourthly, our proof-of-concept implementation from Section 4.5 gave us interesting results, but it is far from a deployable solution; further refinement in the design might improve its efficiency, its security, or even its properties regarding voting.

Bibliography

[Adi08] Ben Adida. Helios: Web-based open-audit voting. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, pages 335–348, Berkeley, CA, USA, 2008. USENIX Association.

[AES01] Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce, November 2001.

[Age82] National Security Agency. Nacsim 5000 tempest fundamentals, 1982.

[APB+04] Riza Aditya, Kun Peng, Colin Boyd, Ed Dawson, and Byoungcheon Lee. Batch verification for equality of discrete logarithms and threshold decryptions. In Markus Jakobsson, Moti Yung, and Jianying Zhou, editors, *ACNS 04*, volume 3089 of *LNCS*, pages 494–508. Springer, Heidelberg, June 2004.

[Aye17] Ahmed Ben Ayed. A conceptual secure blockchain-based electronic voting system, 2017.

[Bac97] Adam Back. A partial hash collision based postage scheme, 1997.

[BB03] David Brumley and Dan Boneh. Remote timing attacks are practical. In *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, SSYM'03, pages 1–1, Berkeley, CA, USA, 2003. USENIX Association.

[BB04] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 56–73. Springer, Heidelberg, May 2004.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. Cryptology ePrint Archive, Report 2005/015, 2005. http://eprint.iacr.org/2005/015.

[BBO06] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. Cryptology ePrint Archive, Report 2006/186, 2006. http://eprint.iacr.org/2006/186.

[BDOP04] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 506–522. Springer, Heidelberg, May 2004.

[Ben80] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of Statistical Physics*, 22(5):563–591, May 1980.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 325–341. Springer, Heidelberg, February 2005.

[BL07] Michel Balinski and Rida Laraki. A theory of measuring, electing, and ranking. *Proceedings of the National Academy of Sciences*, 104(21):8720–8725, 2007.

[BL10] Michel Balinski and Rida Laraki. *Majority Judgment: Measuring Ranking and Electing*. MIT Press, 2010.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, December 2001.

[BMN+09] Josh Benaloh, Tal Moran, Lee Naish, Kim Ramchen, and Vanessa Teague. Shuffle-sum: coercion-resistant verifiable tallying for STV voting. *IEEE Trans. Information Forensics and Security*, 4(4):685–698, 2009.

[But13] Vitalik Buterin. A next-generation smart contract and decentralized application platform, 2013.

[CFGL12] Sébastien Canard, Georg Fuchsbauer, Aline Gouget, and Fabien Laguillaumie. Plaintext-checkable encryption. In Orr Dunkelman, editor, *CT-RSA 2012*, volume 7178 of *LNCS*, pages 332–348. Springer, Heidelberg, February / March 2012.

[CGGI14] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Election verifiability for helios under weaker trust assumptions. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *ESORICS 2014, Part II*, volume 8713 of *LNCS*, pages 327–344. Springer, Heidelberg, September 2014.

[Cha82] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982.

[CHI+08] Michael Clarkson, Brian Hay, Meador Inge, Alec Yasinsac, Abhi Shelat, and David Wagner. Software review and security analysis of scytl remote voting software. 2008.

[CHKM09] Sanjit Chatterjee, Darrel Hankerson, Edward Knapp, and Alfred Menezes. Comparing two pairing-based aggregate signature schemes. Cryptology

ePrint Archive, Report 2009/060, 2009. http://eprint.iacr.org/2009/060.

[CM09] Sanjit Chatterjee and Alfred Menezes. On cryptographic protocols employing asymmetric pairings – the role of ψ revisited. Cryptology ePrint Archive, Report 2009/480, 2009. http://eprint.iacr.org/2009/480.

[CP93] David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer, Heidelberg, August 1993.

[CPST18a] Sébastien Canard, David Pointcheval, Quentin Santos, and Jacques Traoré. Practical strategy-resistant privacy-preserving elections. In Javier López, Jianying Zhou, and Miguel Soriano, editors, *ESORICS 2018, Part II*, volume 11099 of *LNCS*, pages 331–349. Springer, Heidelberg, September 2018.

[CPST18b] Sébastien Canard, David Pointcheval, Quentin Santos, and Jacques Traoré. Privacy-preserving plaintext-equality of low-entropy inputs. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18*, volume 10892 of *LNCS*, pages 262–279. Springer, Heidelberg, July 2018.

[CRS05] David Chaum, Peter Y. A. Ryan, and Steve A. Schneider. A practical voter-verifiable election scheme. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *ESORICS 2005*, volume 3679 of *LNCS*, pages 118–139. Springer, Heidelberg, September 2005.

[CS97] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. Technical report, 1997.

[Deu85] Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 400(1818):97–117, 1985.

[DFK+06] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 285–304. Springer, Heidelberg, March 2006.

[DK05] Yvo Desmedt and Kaoru Kurosawa. Electronic voting: Starting over? In Jianying Zhou, Javier Lopez, Robert H. Deng, and Feng Bao, editors, *ISC 2005*, volume 3650 of *LNCS*, pages 329–343. Springer, Heidelberg, September 2005.

[DR99] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael, 1999.

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Heidelberg, August 1984.

[ELMC18] Shayan Eskandari, Andreas Leoutsarakos, Troy Mursch, and Jeremy Clark. A first look at browser-based cryptojacking. *CoRR*, abs/1803.02887, 2018.

[Fer09] Maribel Fernndez. *Models of Computation: An Introduction to Computability Theory*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[Fey82] Richard P. Feynman. Simulating physics with computers. *Int. J. Theor. Phys.*, 21:467–488, 1982. [,923(1981)].

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 1–19. Springer, Heidelberg, May 2004.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[GHS12a] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 850–867. Springer, Heidelberg, August 2012.

[GHS12b] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. Cryptology ePrint Archive, Report 2012/099, 2012. http://eprint.iacr.org/2012/099.

[Gir91] Marc Girault. An identity-based identification scheme based on discrete logarithms modulo a composite number (rump session). In Ivan Damgård, editor, *EUROCRYPT'90*, volume 473 of *LNCS*, pages 481–486. Springer, Heidelberg, May 1991.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985.

[GPS06] Marc Girault, Guillaume Poupard, and Jacques Stern. On the fly authentication and signature schemes based on groups of unknown order. *Journal of Cryptology*, 19(4):463–487, October 2006.

[Hay12] Brian Hayes. Alice and Bob in cipherspace. *American Scientist*, 100, 2012.

[HF45] Erich Hüttenhain and Walther Fricke. TICOM I-45: Cryptanalytic research on Enigma, Hagelin and cipher teleprinter machines, 1945.

[HOSS18] Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. TinyKeys: A new approach to efficient multi-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 3–33. Springer, Heidelberg, August 2018.

[HSH+09] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, 2009.

[HSK97] Takashi Harada, Hideki Sasaki, and Yoshio Kami. Investigation on radiated emission characteristics of multilayer printed circuit boards, 1997.

[Jag12] Tibor Jager. *Black-Box Models of Computation*. Vieweg+Teubner Verlag, Wiesbaden, 2012.

[KAF+10] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman J. J. te Riele, Andrey Timofeev, and Paul Zimmermann. Factorization of a 768-bit RSA modulus. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 333–350. Springer, Heidelberg, August 2010.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Heidelberg, August 1999.

[KJJR11] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, Apr 2011.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, Heidelberg, August 1996.

[Lac18] Marie-Sarah Lacharité. Security of bls and bgls signatures in a multi-user setting. *Cryptography and Communications*, 10(1):41–58, Jan 2018.

[Lai18] W.-J. Lai Ja-Ling Wu. An efficient and effective Decentralized Anonymous Voting System. *ArXiv e-prints*, April 2018.

[LZL13] Yao Lu, Rui Zhang, and Dongdai Lin. Stronger security model for public-key encryption with equality test. In Michel Abdalla and Tanja Lange, editors, *PAIRING 2012*, volume 7708 of *LNCS*, pages 65–82. Springer, Heidelberg, May 2013.

[Man80] IUriĭ Ivanovich Manin. *Vychislimoe i nevychislimoe*. 1980.

[Mes17] Andrey Meshkov. Cryptojacking surges in popularity growing by 31month. 2017.

[Mon87] Peter Lawrence Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48:243–264, 1987.

[MRZ15] Payman Mohassel, Mike Rosulek, and Ye Zhang. Fast and secure three-party computation: The garbled circuit approach. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 591–602. ACM Press, October 2015.

[MSH17] Patrick McCorry, Siamak F. Shahandashti, and Feng Hao. A smart contract for boardroom voting with maximum voter privacy. In Aggelos Kiayias, editor, *FC 2017*, volume 10322 of *LNCS*, pages 357–375. Springer, Heidelberg, April 2017.

[MT45] Jack Good Donald Michie and Geoffrey Timms. General report on Tunny, with emphasis on statistical methods, 1945.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[Nat77] National Institute of Standards and Technology. Data encryption standard, 1977.

[NST17] Jesper Buus Nielsen, Thomas Schneider, and Roberto Trifiletti. Constant round maliciously secure 2PC with function-independent preprocessing using LEGO. In *NDSS 2017*. The Internet Society, February / March 2017.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Heidelberg, May 1996.

[PS98] Guillaume Poupard and Jacques Stern. Security analysis of a practical "on the fly" authentication and signature generation. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 422–436. Springer, Heidelberg, May / June 1998.

[PS99] Guillaume Poupard and Jacques Stern. On the fly signatures based on factoring. In *ACM CCS 99*, pages 37–45. ACM Press, November 1999.

[PS16] David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 111–126. Springer, Heidelberg, February / March 2016.

[PS18] David Pointcheval and Olivier Sanders. Reassessing security of randomizable signatures. In Nigel P. Smart, editor, *CT-RSA 2018*, volume 10808 of *LNCS*, pages 319–338. Springer, Heidelberg, April 2018.

[QQQ+90] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis C. Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, Soazig Guillou, and Thomas A. Berson. How to explain zero-knowledge protocols to your children (rump session). In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 628–631. Springer, Heidelberg, August 1990.

[RAD78] Ronald Linn Rivest, Leonard Adleman, and Michael Leonidas Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[RBH+09] Peter Y. A. Ryan, David Bismark, James Heather, Steve A. Schneider, and Zhe Xia. The prêt à voter verifiable election system. 12 2009.

[RC11] M. Salois R. Carbone, S. Bean. *An in-depth analysis of the cold boot attack*. 2011.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997.

[Sho00] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 207–220. Springer, Heidelberg, May 2000.

[ST04] Berry Schoenmakers and Pim Tuyls. Practical two-party computation based on the conditional gate. In Pil Joong Lee, editor, *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 119–136. Springer, Heidelberg, December 2004.

[ST06] Berry Schoenmakers and Pim Tuyls. Efficient binary conversion for Paillier encrypted values. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 522–537. Springer, Heidelberg, May / June 2006.

[SWP00] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55. IEEE Computer Society Press, May 2000.

[TB45] Alan Mathison Turing Turing and Donald Bayley. Report on speech secrecy system DELILAH, a technical description compiled by A. M. Turing and Lieutenant D. Bayley REME, 1945–1946, 1945.

[TRN08] Vanessa Teague, Kim Ramchen, and Lee Naish. Coercion-resistant tallying for STV voting. In *2008 USENIX/ACCURATE Electronic Voting Workshop, EVT 2008, July 28-29, 2008, San Jose, CA, USA, Proceedings*, 2008.

[Wet16] Jos Wetzels. Open sesame: The password hashing competition and Argon2. Cryptology ePrint Archive, Report 2016/104, 2016. http://eprint.iacr.org/2016/104.

[WFX+18] Wenhao Wang, Benjamin Ferrell, Xiaoyang Xu, Kevin W. Hamlen, and Shuang Hao. SEISMIC: SEcure in-lined script monitors for interrupting cryptojacks. In Javier López, Jianying Zhou, and Miguel Soriano, editors, *ESORICS 2018, Part II*, volume 11099 of *LNCS*, pages 122–142. Springer, Heidelberg, September 2018.

[WG17] Karl Wüst and Arthur Gervais. Do you need a blockchain? Cryptology ePrint Archive, Report 2017/375, 2017. http://eprint.iacr.org/2017/375.

[XSHT08] Zhe Xia, Steve A. Schneider, James Heather, and Jacques Traoré. Analysis, improvement, and simplification of prêt à voter with paillier encryption. In *USENIX/ACCURATE Electronic Voting Workshop*, 2008.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

[YTHW10] Guomin Yang, Chik How Tan, Qiong Huang, and Duncan S. Wong. Probabilistic public key encryption with equality test. In Josef Pieprzyk, editor, *CT-RSA 2010*, volume 5985 of *LNCS*, pages 119–131. Springer, Heidelberg, March 2010.

Résumé

Au cours cette thèse, nous explorons des méthodes cryptographiques pour résoudre des problèmes concrets. D'abord, nous étudions la technologie « Blockchain », introduite récemment par la cryptomonnaie « Bitcoin ». En particulier, nous demandons si cette technologie peut aider à la résolution de nouveaux problèmes, où à la mise en place de solutions plus efficaces. Nous clarifions quelles propriétés sont apportées par la cryptographie, et lesquelles sont particulières à Blockchain. Nous prenons en considération le cas d'usage du don d'organe et les conditions que les donneurs et receveurs doivent remplir. Nous offrons ensuite une solution qui trouve un équilibre entre rapidité et sécurité. Elle prend la forme d'une nouvelle primitive cryptographique adaptée aux particularités du problème. Additionnellement, nous considérons le cas d'usage du vote électronique ou en ligne, et les possibilités offertes par les systèmes de vote plus avancés. Nous concevons ensuite un protocole cryptographique qui garantit la confidentialité et l'intégrité pour implémenter le Jugement Majoritaire. En parallèle avec la construction théorique, nous implémentons une preuve de concept et fournissons des valeurs chiffrées qui démontrent des temps d'exécution raisonnables. Finalement, il semble que Blockchain ne soit pas adéquat pour ces cas d'usage, mais que les techniques cryptographiques actuelles sont capables de faire davantage que ce qui est attendu par les non-spécialistes.

Mots Clés

cryptographie blockchain distribué voting vie privée intégrité

Abstract

In this thesis, we explore cryptographic methods to solve concrete problems. First, we study the technology “Blockchain”, recently introduced by the cryptocurrency “Bitcoin”. In particular, we ponder whether this technology can help in solving new problems or in providing more efficient solutions. We clarify which properties are brought by cryptography, and which are particular to Blockchain. We consider the use case of organ donation and the conditions that donors and recipients must meet. We then offer a solution that makes a reasonable trade-off between speed and security. This takes the form of a new cryptographic primitive tailored to the specificities of the problem. Furthermore, we consider the use case of online and electronic voting and the possibilities offered by the more advanced voting systems. We then design a cryptographic protocol that guarantees confidentiality and integrity to implement Majority Judgment. Along with the theoretical construction, we implement a proof of concept and provide benchmarks that show reasonable running times. In the end, it appears that Blockchain does not adapt well to these use cases, but that currency cryptographic techniques can do more that usually assumed by non-specialists.

Keywords

cryptography blockchain distributed vote privacy integrity