École polytechnique de Louvain
Institut ICTEAM
UCL Groupe Crypto

Privacy-Preserving Audit Mechanisms for Multi-Party Protocols

Édouard Cuvelier

Thèse présentée en vue de l'obtention du grade de
docteur en sciences de l'ingénieur.

Composition du jury :
Pr Gildas Avoine (UCL & INSA Rennes)
Pr David Bol (UCL) - **Président**
Dr Steve Kremer (Inria Nancy - Grand Est & LORIA)
Pr Olivier Markowitch (ULB)
Pr Olivier Pereira (UCL) - **Promoteur**
Pr François-Xavier Standaert (UCL)

Louvain-la-Neuve, Belgique – 2015

Abstract

This thesis sets as goal the study and development of cryptographic multi-party protocols offering the properties of verifiability and privacy. The verifiability property guarantees the protocols participants and/or observers that the result of the execution of the protocol is exactly what is expected from a honest execution of the protocol. On the other hand, the privacy property ensures the participants that their private information is not leaked by executing the protocol. The thesis targets real-world applications as well as any multi-party function.

The first part of the work focus on cryptographic voting systems. In this case, the function to evaluate is rather simple – e.g. a sum of yes/no votes – and, we show how we conciliate the verifiability with the privacy to obtain a cryptographic voting system that offers a perfectly private audit trail of its execution. A perfectly private audit trail means that it contains no information about the voters' votes whatsoever. In addition, the trail computationally guarantees the observers that the tally of the votes is correct.

Next, we extend our study to encompass more complex functions. We work on combinatorial problems such as graph problems. In this part, following the traditional approach of secure multi-party computation, we investigate potential sources of privacy leakages that appear when turning the unsecured version of an algorithm into its secure version. We propose solutions to prevent these privacy leakages through algorithms for securely sorting shared lists and securely computing the shortest path and the maximum flow in shared graphs.

In the last part of the thesis, we follow a different approach than the traditional secure multi-party computation one. In our approach, we rely on a third party (worker) that is entrusted with the privacy of the protocol participants' inputs. We show that several important gains can be made in this setting. We propose a generic protocol that can be used to evaluate any multi-party function while offering a perfectly private audit trail of its computation. This protocol is mainly non-interactive

and offers the worker the possibility to use his own algorithms.

Finally, the solutions obtained in this thesis have been implemented. The secure multi-party protocols are available in an online prototype that can be used as such or to develop any desired new multi-party application that offers perfect privacy and computational verifiability.

Remerciements

Cette thèse a pu être menée grâce au financement de la région wallonne à travers le projet CAMUS pendant un an et pour les trois dernières années, grâce à une bourse obtenue au Fonds pour la formation à la Recherche dans l'Industrie et dans l'Agriculture (FRIA) dépendant du Fonds de la Recherche Scientifique (F.R.S. – FNRS).

Par ces lignes, je remercie toutes les personnes qui ont apporté directement ou indirectement une brique à la construction de mon mur, c'est à dire cette thèse.

En premier lieu, Olivier Pereira, mon promoteur qui m'a guidé et aidé tout au long de ces quatre années. Olivier m'a accueilli au sein du groupe crypto et ses grandes qualités humaines et professionnelles ont fait de mon cheminement de thèse une période de ma vie riche à bien des égards.

Je remercie les acteurs de premier plan de cette thèse et tout d'abord les membres de mon comité d'accompagnement, François-Xavier Standaert et Gildas Avoine, ainsi que les membres de mon jury de thèse, David Bol, Steve Kremer et Olivier Markowitch, pour les discussions, les échanges et les questions soulevées lors de la réunion de confirmation et de la défense privée. Merci à eux d'avoir accepté de se consacrer à la lecture et à l'appréciation de ce manuscrit et de mon travail.

Une place d'honneur revient également à Sylvie Baudine qui a largement contribué à améliorer l'anglais et la forme de cette rédaction ainsi que des autres écrits et présentation orales durant ma thèse. L'excellente ambiance dans le groupe est largement redevable à la bonne humeur et au sourire de Sylvie.

Notre groupe de recherche, le groupe crypto, a été un environnement excellent et chaleureux pour réaliser ma thèse. Mes remerciements vont tout d'abord à François Koeune avec qui j'ai pris contact en premier lieu pour réaliser cette thèse. François aime comme moi débattre sur tous

les sujets allant de la politique belge à la bande dessinée ; merci pour ces longues et passionnantes discussions toujours ponctuées d'humour parfois absurde que j'affectionne. Mes collègues qui ont vécu le quotidien avec moi méritent de multiples remerciements. Leur collaboration dans mon travail de recherche, leur aide et leurs réflexions ont joué une partie essentielle dans cette thèse tout comme les excellents moments partagés ensemble. Merci à tous et en particulier à ceux qui ont le plus marqué mes années de thèse, Florentin, François, François, Loïc, Lubos, Mathieu, Romain, Sophie, Stéphanie, Sylvie, Thomas et Vincent. Le groupe crypto est une équipe de recherche de ELEN qui est lui-même un pôle au sein de l'institut ICTEAM. Ces entités imbriquées m'ont permis de rencontrer et de croiser quotidiennement de nombreuses personnes que je remercie pour leur gentillesse et leur support administratif, technique, etc.

Mon pénultième remerciement va à mes amis, à ma famille et en particulier à mes parents pour leur enthousiasme et leur soutien à la réalisation de cette thèse. Merci aussi à mes oncles et tantes, ma belle famille et mes frères et sœurs pour leur curiosité et leur intérêt manifesté.

Cette thèse, je la dédie à mon épouse, Marie, mon indéfectible soutien qui m'accompagne et réalise les plus belles choses avec moi.

Contents

Glossary

Arithmetic

$\mathbb{Z}$	Set of integers
$\mathbb{N}$	Set of positive integers
$\mathbb{R}$	Set of real numbers
$\mathbb{C}$	Set of complex numbers
$\Im(c)$	Imaginary part of c, a complex number
$\mathbb{Z}_p$	Set of integers modulo p
$\mathbb{F}_q$	Finite field of order q
S^*	Set S minus the non invertible elements (respectively to an operation)
$\mathcal{P}(S)$	Power set of the set S, that is the set that contains all the subsets of S
$L : K$	Extension field L over K
$[L : K]$	Degree of the extension field

$G \simeq H$	G is homomorphic to H
$\mathsf{Dlog}_g h$	Discrete logarithm of h in base g
$E \equiv ...$	E has for equation(s) ...
E_F	Elliptic curve of equation E over a field F
$E_F[q]$	Set of q-torsion points of E_F
$\mathcal{O}_\infty$	Point at infinity of an elliptic curve
$\mathcal{P}air$	Pairing setting
φ	Euler's totient function
ϕ	Frobenius's endomorphism
η	Negligible function

$\boldsymbol{v}$	Vector
$\boldsymbol{M}$	Matrix
$x \xleftarrow{rand} S$	x is chosen uniformly at random in S
$x := ...$	x is defined as ...
$x \leftarrow ...$	x is assigned to ...
$x \stackrel{?}{=} y$	Operation checking the equality between x and y and returning an according bit
W_H	Hamming weight
$O(\cdot)$	Asymptotic complexity

Cryptographic notations and abbreviations

CDH	Computational Diffie-Hellman
DDH	Decisional Diffie-Hellman
DCR	Decisional composite residuosity
DL, DLP	Discrete logarithm and discrete logarithm problem
$\mathcal{QR}$	Quadratic residues
$\mathcal{QNR}$	Quadratic non-residues
RSA	Rivest-Shamir-Adleman
SXDH	Symmetric external Diffie-Hellman

PubK	Public-key setting
pp	Public parameters
pk	Public key
cpk	Commitment public key
sk	Secret key

K	Key space
M	Message space
C	Ciphertext space
O	Opening space
$\mathbf{C}_{\mathsf{C}}$	Commitment space
Se	Space of secrets
Sh	Space of shares
I	Input space
O	Output space

PPT	Probabilistic polynomial time
$\mathcal{A}$	Adversary
Adv	Advantage
$\mathcal{D}$	Distribution
$\mathcal{E}$	Environment
exec	Execution view
$\mathcal{F}$	Functionality
$\mathcal{G}$	Game
$x \approx y$	Distribution x is computationally indistinguishable from distribution y
$\mathcal{O}$	Oracle
Pr	Probability
$\mathcal{S}$	Simulator
$\mathcal{T}$	Trusted third party
EAV	Eavesdropper
IND	Indistinguishable
NM	Non-malleable
CPA	Chosen plaintext attack
CCA	Chosen ciphertext attack
$\mathcal{H}$	Hash function
$\mathcal{L}$	Language
NP	Non polynomial
ZK	Zero-knowledge
ZKPK	Zero-knowledge proof of knowledge
NIZKPK	Non-interactive zero-knowledge proof of knowledge
π_{DL}	Proof of discrete logarithm equality
π_{or}	Disjunctive or-proof
π_{mul}	Proof of multiplication
π_{ran}	Range proof
$\pi_{<}$	Proof of lesser than
π_{cc}	Proof of consistency
π_{ope}	Proof of knowledge of the opening
π_{cor}	Proof of correctness of the result
π_{ver}	Proof of verifiability
SMC	Secure multi-party computation

Commitment consistent encryption – Chapter 3

CC	Commitment consistent
CCE	Commitment consistent encryption
VA	Validity augmentation
CCVAE	Validity augmented CCE

PPAT	Perfectly private audit trail
PPATP	PPAT based on Pedersen and Pailler
PPATS	PPAT for simple messages
PPATC	PPAT for complex messages

M	Modular multiplication
E	Modular exponentiation
$\mathtt{E}_\mathtt{p}$	Modular exponentiation with precomputation
A	Point addition over an elliptic curve
Sm	Scalar multiplication of a point over an elliptic curve
$\mathtt{Sm}_\mathtt{p}$	Scalar multiplication of a point over an elliptic curve with precomputation
P	Pairing
U	Base unit for comparing the costs of different algorithms – U is one multiplication between two 256-bit integers
V	Memory base unit for comparing proof sizes

Cryptographic vote – Chapter 4

BB	Bulletin board
PB	Public bulletin board
SB	Secret bulletin board
Enc2Vote	Mini voting scheme
VotePriv	Vote privacy experiment

Function evaluation with PPAT – Chapter 5

$\mathcal{C}$	Clients set
$\mathcal{W}$	Worker
Π^f_{PPAT}	Protocol realizing the multi-party function evaluation with PPAT
$\mathcal{HON}$	Set of honest clients
$\mathcal{COR}$	Set of corrupted clients

Securely solving combinatorial problems – Chapter 6

$[s]$	Shared value of s
$[\boldsymbol{v}]$	Vector of shared values
$[\boldsymbol{M}]$	Matrix of shared values
$\mathtt{d}$	Decimal representation (e.g. $x^{\mathtt{d}}$)
$\mathtt{u}$	Unary representation (e.g. $x^{\mathtt{u}}$)
USM	Unary sorting algorithm with multiplicities
USNM	Unary sorting algorithm with no multiplicities
SSPBF	Secure shortest path algorithm based on Bellman-Ford
SSPD	Secure shortest path algorithm based on Dijkstra
SMFEK	Secure maximum flow algorithm based on a partial version Edmonds-Karp
SMFCEK	Secure maximum flow algorithm based on a complete version of Edmonds-Karp

Part I

Preliminaries

Chapter 1

Introduction

For the last 30 years and more especially since the advent of the global net of computers, the field of cryptography has expanded at a steady pace. The ubiquity of the technologies of information turned what was once only considered as a military weapon to an everyday consumer good through its manifold applications. Cryptography stands at a crosspoint between mathematics, computer sciences, electronics and even physics. This mixture is obviously a great opportunity for the researcher.

The directions that cryptography pursues nowadays lead to various applications ranging from classic encryption schemes to cryptographic voting systems via securely outsourcing computations or even playing poker over the telephone (without being able to cheat). The most spectacular progress in the next decades would probably involve the advent (?) of quantum computers and the use of efficient and practical fully homomorphic encryption or the combination of both.

The power of maths. Mathematics take a prominent place in the field of cryptography since they give tools to prove the strength or the weakness of a cryptographic scheme. Indeed, when the security proofs in an accurate security model rely only on well-studied mathematical problems, they provide incontrovertible evidences. On the contrary, when the security relies on physical assumptions or obfuscation (by hiding the protocol design for example), it is difficult to obtain security guarantees without first admitting strong hypotheses such as, for example, that a sealed chip will remain sealed.

That being said, cryptography based on today's mathematical problems has its downside. One of the mathematical Millennium Prize Problems is breathtakingly holding the research community. It is the question of proving, disproving or proving the impossibility of proving

that $\mathsf{P} = \mathsf{NP}$. This interesting problem is of high concern in modern cryptography due to its intricacy in the core of the security guarantees cryptography offers. Basically, mathematical problems that are hard to solve with known techniques but that could be easily solved if given additional information, are building blocks for the security assumptions from which the cryptographic primitives such as encryption schemes are built. If $\mathsf{P} \neq \mathsf{NP}$, it means that there exists such problems for which the linear increase in the parameters (or the statement of the problem) results in supra polynomial increase in the efforts needed to solve them in the worst case. Thus, proving that $\mathsf{P} = \mathsf{NP}$ would turn the world of cryptography upside down since the computational effort thought to be supra polynomial in the parameters will unavoidably be reduced to a polynomial which is, needless to say, much lighter in terms of computations. By domino effect, the security of the current cryptographic schemes would be strongly impaired.

Cryptography, a modern tool. A branch of modern cryptography targets applications involving several parties and requiring security properties. For example, cryptographic voting gathers voters who want to perform an election with some or all the following ingredients: privacy of the votes, correctness of the tally (the tally reflects exactly the choices of the voters), coercion-free elections (a voter cannot be forced to vote against his wishes), universal verifiability (the whole election process can be audited by external observers), etc. Achieving all these properties together can be challenging for the designer of the protocol since some of them may be mutually excluding each other.

In a more general scale, coordinating several parties to perform the computation task is also challenging. Synchronizing the actions of each one in a dynamic framework, organizing the communication channels, considering the different layers embedding the protocol, all this represents possible threats that we have to tackle if we want to provide an end-to-end solution. However, this approach appears to be very demanding since its implementation requires tools coming from very different areas. Again taking cryptographic voting as an example, a complete solution of electronic voting necessitates the installation of polling booth and voting machines with authorities to manage them. But then, we would have to make sure that these machines or the software installed are not corrupted. This raises the question of who can verify this and how, etc. We see that cutting down the problem into smaller pieces is thus a necessity. In this thesis we focus on the pieces for which cryptography has much to offer by proposing elegant and efficient solutions.

1.1 Research project

This thesis has been elaborated around one main property, that is **verifiability**. Schemes, protocols or applications can feature this property allowing designated parties, the auditors, to assert the smooth progress of the process. The goal is two-sided. First the auditors ensure themselves that everything went in accordance with what was planned. Second, the participants to the protocol are enforced to provide convincing outcomes showing their correct participation. The fear of being caught ensuring their honest behaviour. This objective can be achieved by providing the auditors with an irrefutable proof at the end of the process. We call it the **audit trail**.

The verifiability property is sought in many applications. For example, in cloud computing, a client outsources its computations to a server. However, the client has to make sure that the result sent back from the server is correct. In electronic auctions like those performed by the stock exchange market auctioneer, verifiability would allow the seller and the buyer of stocks to check that their offer or demand was met under the rules of the stock exchange without having to blindly trust the auctioneer.

One major concern of this thesis is to conciliate the verifiability property with the **privacy** property. Privacy ensures that the secrets of the participants are not leaked to other parties or eavesdroppers. Recalling the above examples, in cloud computing, if the client is a company, it might be critical that the cloud only access and compute on secret data. This is because most private companies are very sensitive about their private data due to industrial espionage. In some electronic auctions, a privacy leakage could ruin the whole process since learning the bids of other participants allows one to bid accordingly.

There is a difference between privacy and confidentiality (or secrecy) which is also evoked in this thesis. Both are properties aiming at protecting the secret nature of a piece of information. However, while confidentiality is absolute in the sense that every single piece of information must remain hidden, privacy leaves some room that depends on the context. To be more precise, privacy hides everything but what could be deduced by the context. In the voting example, a ballot is only private and not confidential since given the result of the elections, we can retrieve some information about the vote contained in the ballot. In the more general case of multi-party function evaluation, we say that privacy is guaranteed, which means that nothing about the private inputs of the participants can be deduced except from what can be deduced from the result of the function. In this regard, confidentiality would require that

even the result could not be divulged to the participants.

Obtaining applications that achieve at the same time both the verifiability and the privacy property is challenging. Indeed, these two properties tend to cancel each other off. This conflict naturally arises from the fact that privacy aims at hiding every piece of information while verifiability is performed by checking the validity of these exact pieces. One objective of this thesis is thus to get around this problem and propose solutions enabling both verifiability and privacy for cryptographic applications.

We set ourselves in the context of Secure Multi-Party Computation (SMC). Secure multi-party computation allows setting up multi-party functionalities in a secure way. A functionality could simply be the evaluation of a function, but it could also be a reactive process or protocol offering various possibilities of actions for the participants. SMC has been at the center of cryptography research for almost 30 years. A first series of foundational works [Yao82, BOGW88, CCD88, GMW87] demonstrated the possibility to evaluate any function in various models, the function being described as a circuit. The attention then largely focused on building solutions for the evaluation of functions of specific interest, leading to secure and efficient protocols for auctions [BDJ+06], voting [CFSY96], benchmarking [BFK+09], face recognition [SSW09] or AES evaluation [PSSW09] to only mention a few.

The main goal of this thesis is to study and develop cryptographic protocols with audit mechanisms as well as to provide a generic way to obtain new ones. We also focus on finding solutions to concrete problems such as cryptographic voting, electronic auctions or the outsourcing of multi-party computation on private data to a third party that will produce a proof of the correctness of its computations.

1.2 Main contributions

As mentioned above, our research takes place in a general framework which is secure multi-party computation. Different approaches to enable security in multi-party protocols exist and we mainly consider three of them in this thesis.

First, an SMC scenario where we exclude that any third party performs computations. In this case, all the participants are set on equal footing in the sense that none of them is designated to perform the computations alone while the others are only sending their inputs and waiting for the result. In this scenario, every party collaborates to the computation of the functionality in a distributed way. We will sometimes

refer to this scenario as "classic" SMC.

In a second scenario, we split the set of parties into two subsets. The first one regroups the clients and the second one regroups the workers. Here we have a different assignment of tasks. The clients will provide the inputs while the workers compute and give the result back to the clients.

A third scenario similar to the second one considers only one worker in place of several. The clients still provide the inputs to the worker which compute the result and give it back to the clients. As we will see, this scenario offers less privacy to the clients than second one. However, it has the advantage to reduce the computational efforts needed for the clients to verify the correctness of the computation.

While the first scenario is often unrelenting in terms of trust and makes sense when the parties do not want to trust anyone but themselves, the second and third scenarios leave some room for cases where part of the trust can be given away to other parties. As a result, those schemes are more flexible. For example, the first scenario can be used to solve problems where a little number of parties (2 or 3) are involved since then, the number of communication channels (one per pair of parties) is manageable. However, thousands of voters organizing a vote without relying on election authorities to perform the tally of the votes can be challenging. In this case, two last scenarios are more suitable.

The work presented in this thesis proposes secure solutions to multi-party protocols in each of the three scenarios. The cryptographic voting schemes and the multi-party function evaluation proposal take respectively place in the second and third scenario while the proposal on simple combinatorial problems is set up in the first scenario.

Cryptographic voting. The first contribution of this thesis concerns cryptographic voting. The results were presented at the ESORICS conference in 2013 in [CPP13].

In this work, we introduce the first efficient cryptographic voting system that features a perfectly-private audit trail guaranteeing at the same time, the perfect privacy of the ballots and the computational verifiability of the elections. Perfect privacy has to be taken in the sense of the theory of information, meaning that privacy is ensured even against an unbounded adversary. In addition, the voting scheme ensures that the verifiability of the elections, that is the possibility for the observers and the voters to check that the election process and outcome, took place properly.

We propose several implementations of our voting scheme based on a new cryptographic primitive called Commitment Consistent Encryption.

This primitive is not confined to cryptographic vote and we find it other uses in unrelated applications that are also presented in this thesis.

The implementations based on our primitives are efficient compared to naïve implementations and we developed a prototype to test their efficiency. The prototype in itself is also an achievement of this thesis and its intrinsic value is demonstrated through the applications generically developed to securely evaluate multi-party functions.

Multi-party function evaluation. A second contribution of this thesis is the generic solution proposed to securely evaluate any multi-party function in a clients-worker setting where the clients receive a perfectly private audit trail of the function evaluation.

A preliminary version of this work was presented at the conference SDTA in 2014 in [CP14]. The proposal extends the results obtained for the cryptographic voting where the function to evaluate is rather simple (in a 0/1 vote, the function is a sum). Here we consider any function and we specifically target those needing algorithms to be evaluated. These algorithms use arithmetic operations as additions and multiplications as well as branchings such as if-then-else sections.

We present three test applications: solving a linear system of equations, auctions, and finding the shortest path in a graph as well as a generic method to evaluate any circuit-based function and achieve the perfectly private audit trail. We point out that our method has an interesting advantage, that is, in some cases, it allows reducing the complexity of the verification of the solution for the clients. Indeed, when the classical solution to obtain verifiability often goes through (re-) computing the entire algorithm, our solution proposes a short cut where the verification process only depends on the solution and on the inputs of the clients. For example, a sorting algorithm like Quicksort runs in $\mathcal{O}(n \log n)$ which is the computational cost for the clients if they run together the algorithm on their private inputs. However, if they are given the sorted list of their inputs, the clients only need to check that the list is sorted, which can be done in $\mathcal{O}(n)$ steps.

Finally, as mentioned above, we developed a prototype implementation of our applications and used it to test our results. This prototype is meant to be reused to design other secure multi-party function evaluations that automatically provide a perfectly private audit trail.

Simple combinatorial problems. In this third contribution, we endeavour to solve simple combinatorial problems and in particular graph problems. Part of this work was presented this work at the Financial Cryptography and Data Security conference in 2013 [ACM+13].

The objective of this research was to analyse and build SMC protocols to securely solve simple combinatorial problems. Our secure solutions tackle the sorting problem and two graph problems, namely the shortest path and the maximum flow. Our protocols offer privacy and verifiability in a threshold model. This means that the security properties are guaranteed as long as a fixed number of the parties remains honest.

Our analysis uncovered interesting complexity gaps between the unsecured and the secure versions of the classic algorithms we used as it is, for example, the case for the Dijkstra algorithm. We designed our solutions to alleviate the difference in the efficiency metrics that appears in SMC. Indeed, the cost of the atomic operations does not transpose exactly in SMC. This forces us to rethink the algorithms to avoid as much as possible the costly operations such as comparisons. Finally, we also discuss the different privacy leakages that might appear with naïve implementations and the precautions we have to take to avoid them.

Our secure algorithms were implemented in a Python SMC framework called VIFF [VIF] which gave us qualitative timing results and recommendations regarding the practicability of our solutions.

1.3 Organization of the thesis

The thesis is organized in three parts. In the first one, after this introduction, we provide the cryptographic background. The second part contains the various contributions of the thesis while the last part draws our conclusions.

More specifically, in Chapter 2, we detail the cryptographic primitives such as encryption, commitment schemes as well as zero-knowledge proofs of knowledge. We build up the security of these schemes that can be achieved through the cryptographic assumptions. We also present secret sharing schemes that lead to classic secure multi-party computation.

In Chapter 3, we designed a new cryptographic primitive, the Commitment Consistent Encryption (CCE) that is used in the next chapters. We give the definition and the validity augmentation that enforces the consistency of the primitive. Then we present two concrete instantiations and a generic construction of the new encryption scheme. We also provide the details of the prototype implementation performed in Python and used to test the efficiency of our instantiations.

Then we get down to business with our first secure multi-party application, cryptographic voting. Chapter 4 is dedicated to the new voting schemes that we propose. It covers the security notions and properties a

voting scheme must possess and how we obtain those properties through the use of CCE schemes. We prove the security of our concrete instantiation and generic construction of voting schemes and we analyse their efficiency theoretically and through our prototype implementation.

Chapter 5 extends the use of the CCE primitive by proposing a generic construction to evaluate any multi-party function and provide a perfectly private audit trail. We first state the functionality we wish to achieve and then we detail the real-world protocols and the tools that realize the functionality in the presence of adversaries described in our threat model. We prove the security of our protocol in this model. Next, we show how to use our protocol in three test applications and we analyse their complexity as well as their efficiency thanks to our prototype implementation.

Finally, in Chapter 6, we tackle various simple combinatorial problems within the classic SMC approach. We present different graph problems and the interest to solve them securely. We identify differences in the efficiency metrics and complexity gaps between the secure and unsecure versions of the classic algorithms used (Bellman-Ford, Dijkstra, Edmonds-Karp). We explain how our secure implementations are designed to avoid as much as possible the efficiency overheads and we give qualitative time measurements for each of them.

We conclude and draw future perspectives in Chapter 7.

Chapter 2
Building Blocks

This chapter describes the different cryptographic tools that are used throughout this thesis. As a starting point, we recall the cryptographic assumptions and primitives such as encryption schemes, sigma-protocols and zero-knowledge proofs of knowledge. We also overview secret sharing scheme and secure multi-party computation.

For smooth reading of this chapter, we recommend the reader to be familiar with the number theory notions of group, rings and order. In case of need, we invite the reader to look at the memento in Appendix A.

2.1 Public-key cryptography

The historic purpose of cryptography is to convey messages secretly through the use of an encryption algorithm and a secret key. In modern cryptography, we distinguish **symmetric** cryptography from **asymmetric** cryptography also called **public-key** cryptography. In symmetric cryptography, the same key is used for encryption and decryption whereas in public-key cryptography, a pair of keys, the public-key pk and the secret key sk, are used for encryption and decryption respectively. The possibility to render the encryption key public opens a range of new applications unreachable by symmetric cryptography. For example, the exchange between two persons of their personal public keys via an insecure network allows them to communicate securely. Thanks to that, electronic (e-) signatures, e-voting and e-commerce are today common services for users connected to a global network. Public-key cryptography finds its roots in the seminal paper of W. Diffie and M. Hellman in 1976 entitled "New directions in cryptography" [DH76]. This work sets the basis of the security in public-key cryptography which relies on the computational difficulty to solve some families of mathematical problems.

In 1978, Rivest, Shamir and Adleman [RSA78] introduced the first public-key cryptosystem based on a hard mathematical problem: the difficulty to factorize large integers. This system is widely used nowadays and better known by the name RSA. Since then, other mathematical problems have been used in public cryptography and the second most famous is certainly the **discrete logarithm problem** (DLP). In the following, we present one factorization problem related to RSA, the DLP as well as convenient variants of the DLP used in this thesis.

2.1.1 Integer factorization based assumptions

Even though it is not used in this thesis, we mention RSA as it is one of the most widely known encryption system. Its security relies on the following factorization problem, informally that it is hard to decompose the product of two large primes.

Related to factoring but not known to be identical, the **decisional composite residuosity** DCR assumption states that it is hard to distinguish random elements of $\mathbb{Z}_{n^2}^*$ from random elements of n-th residues in $\mathbb{Z}_{n^2}^*$ defined as $\mathsf{Res}(n^2) := \{y \in \mathbb{Z}_{n^2}^* | \exists x \in \mathbb{Z}_{n^2} : y = x^n \mod n^2\}$.

Assumption 2.1 (DCR)**.** *Given n, a_0, a_1 where $n = pq$ the product of two large primes, a random element $a_b \in \mathbb{Z}_{n^2}^*$ and a random element*

$a_{1-b} \in \mathsf{Res}(n^2)$ *where* b *is a random bit, it is hard to decide which of* a_0 *or* a_1 *belongs to* $\mathsf{Res}(n^2)$.

This assumption studied by P. Pailier in 1999 [Pai99] leads to a useful encryption scheme presented in Section 2.1.3.

In the same vein, the **quadratic residuosity assumption** states that it is hard to distinguish random elements of a $\mathbb{Z}_n^*$ that are chosen among the quadratic residues (in $\mathcal{QR}_n$) from elements that are randomly chosen among the non-quadratic residues (in $\mathcal{QNR}_n$) where n is the product of two large primes. This problem goes back to Gauss [Gau86] and was exploited by Goldwasser and Micali [GM84] for the so called Goldwasser-Micali encryption scheme.

2.1.2 Discrete logarithm based assumptions

In 1985, T. ElGamal proposed a new public-key cryptosystem based on discrete logarithm problems [ElG85]. This system enjoys a broad range of cryptographic applications nowadays.

This family of problems is usually based on a cyclic group $G = (S, .)$ of large prime order $q > 2^k$ where k is a security parameter. Relatively to a generator g, any random element h can be expressed as a power of g: $h = g^x$ for a unique x modulo the group order. We say that the **discrete logarithm** DL of h in base g is x.

Assumption 2.2 (DL)**.** *Given a cyclic group* G *of a large prime order* $q > 2^k$ *and a generator* g *of* G*, for any random element* $h \xleftarrow{rand} G$*, it is hard to compute the discrete logarithm of* h *in base* g.

This assumption does not hold in any groups but the best known algorithms take sub-exponential time in k to find the discrete logarithm in prime order q subgroups of multiplicative groups $\mathbb{Z}_p^*$. However, on subgroups of some elliptic curves, the best known algorithms are exponential in k which allows shorter key size for the same level of security [MVOV96, CFA+05].

Related to the DL, we now look at two variants of a problem called **Diffie-Hellman problem** and introduced by Diffie and Hellman in [DH76]. The first is the **computational Diffie-Hellman** (CDH) and the second is the **decisional Diffie-Hellman** (DDH).

Assumption 2.3 (CDH)**.** *Given* (G, g, g_a, g_b) *where* G *is a large prime order* $q > 2^k$ *multiplicative group,* g *is a generator of* G*, and* g_a, g_b *are random elements of* G*, it is hard to compute* g^c *where* $c := \mathsf{Dlog}_g g_a \cdot \mathsf{Dlog}_g g_b$.

Assumption 2.4 (DDH)**.** *Given* (G, g, g_a, g_b, g_c) *where* G *is a large prime order* $q > 2^k$ *multiplicative group,* g *is a generator of* G*, and* g_a, g_b *are random elements of* G *and* $g_c \in G$*, it is hard to decide whether* $DLog_g g_c = DLog_g g_a \cdot DLog_g g_b$ *or if* g_c *is a random group element independently chosen from* g_a *and* g_b*.*

Being able to break CDH obviously means being able to break DDH. However the converse is not true and some groups in which DDH is easy remain strong toward the CDH assumption. Such groups can be highlighted when a symmetric bilinear structure such as a pairing exists. In short, a pairing $\mathcal{P}air$ is composed of three groups G_1, G_2, G_3 and a bilinear mapping $e : G_1 \times G_2 \to G_3$ such that $\forall (g_1, g_2) \in G_1 \times G_2$, $\forall a, b \in \mathbb{Z}$ we have that $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$. We say that the pairing is **symmetric** when $G_1 = G_2$ and asymmetric otherwise. Given the bilinear property of a symmetric pairing, DDH becomes easy in G_1. Indeed, $e(g^a, g^b) = e(g, g)^{ab}$ is easily compared to $e(g^c, g) = e(g, g)^c$. The groups where DDH is easy while CDH is still hard are called *Gap-Diffie Hellman* groups. We provide a more detailed look on pairings is provided in Appendix A.3.

When working with pairings, we have to state the security assumptions related to the hardness of the CDH problem. The two basics are the **Symmetric External Diffie-Hellman** (SXDH) for asymmetric pairings [ACHdM05] and the *Decisional Linear Assumption* (DLIN) for symmetric pairings [BBS04].

Assumption 2.5 (SXDH)**.** *Given an asymmetric pairing* $\mathcal{P}air_{asym} := (G_1, G_2, G_3, e)$*, the* DDH *problem is hard in both* G_1 *and* G_2*.*

Thanks to their nice bilinear property, pairing-based cryptosystems are part of the constructions presented in Section 3.4 and Section 4.4.2. These constructions will only rely on asymmetric pairings.

2.1.3 Public-key encryption scheme

We describe the cryptographic primitives that are relevant to this thesis and we begin with the public-key encryption scheme. Loosely speaking, an encryption scheme allows messages to be encrypted and decrypted through the use of a public key and a secret key respectively.

Definition 2.1 (Public-key encryption scheme)**.** *A* ***public-key encryption scheme*** Π_E *is a triple of algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *such that :*

$\mathsf{Gen}(1^\lambda)$**:** *on input* 1^λ *a string of length* λ *where* λ *is a security parameter, outputs a key triple* $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \in \mathbf{K}$ *composed of*

- *public parameters* pp *describing the encryption setting (such as public generators), the specification of a message space* $\mathbf{M}$*, a ciphertext space* $\mathbf{C}$ *and a finite key space* $\mathbf{K}$*,*
- *a public key* pk*, and*
- *a secret key* sk*.*

$\mathsf{Enc}(\mathsf{pk}, m)$**:** *on input the public key* pk *and a message* $m \in \mathbf{M}$*, outputs a ciphertext* $c \in \mathbf{C}$*.*

$\mathsf{Dec}(\mathsf{sk}, c)$**:** *on input the secret key* sk *and a ciphertext* $c \in \mathbf{C}$*, outputs a decryption of the ciphertext* $\bar{m} \in \mathbf{M}$*.*

We assume that Gen and Enc are probabilistic algorithms while Dec is deterministic. We also assume that pp is available to all algorithms apart from Gen. The scheme Π_E possesses the following property:

Correctness: $\forall(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$, $\forall m \in \mathbf{M}$, we have for $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$ that $\mathsf{Dec}(\mathsf{sk}, c) = m$ with probability 1.

Saying that an encryption scheme is secure raises several questions about what we mean by security or what kind of adversary the encryption scheme will encounter. Loosely speaking, we say that a scheme is secure if no Probabilistic Polynomial Time (PPT) adversary is able to "break" it, meaning that it is able to retrieve meaningful information from the ciphertext. Since it is not feasible to enumerate all kind of adversaries, we proceed by reduction which essentially works as follow :

1. let $\mathcal{A}$ be a PPT adversary able to "break" the scheme Π.
2. from $\mathcal{A}$, we build a PPT adversary $\mathcal{A}'$ against some cryptographic assumption. We explain how $\mathcal{A}'$ uses $\mathcal{A}$ to break the assumption.
3. the success probability $\Pr_{\mathcal{A}'}$ of $\mathcal{A}'$ is the success probability $\Pr_{\mathcal{A}}$ of $\mathcal{A}$ eventually divided by a polynomial function. On the other hand, since $\Pr_{\mathcal{A}'}$ is bounded by the probability to break the assumption, so is $\Pr_{\mathcal{A}}$. Thus, if the probability to break the assumption is negligible, so must be the success probability $\Pr_{\mathcal{A}}$ of $\mathcal{A}$.

Above, we say that a function $\eta : \mathbb{N} \to \mathbb{R}^{\geq 0}$ is **negligible** if for any $c \in \mathbb{N}$ we have that $\lim_{\lambda \to \infty} \eta(\lambda).\lambda^c = 0$. An **overwhelming** function is close to 1 up to a negligible function.

In this regards, the security of some cryptographic scheme, given a security parameter λ, rests on the fact that the probability for every adversary that runs in polynomial time in λ to break any useful cryptographic assumption must be negligible.

Proceeding in this way, we do not have to care too much about the abilities of an adversary as long as we know that he is limited in his computational power and we define precisely what kind of information he is given access to perform his attack. For example, we allow the adversary to eavesdrop communication, to "play" with the encryption scheme or even to access side-channel information issued by the algorithm's execution. Doing so, we differentiate classes of adversaries or encryption schemes depending on the point of view.

In the following we present experiments defining the goals of the adversary. There are mainly two different goals : **indistinguishability of encryption** (IND) where we want an adversary to be unable to learn any information about a plaintext m, given a cipertext c. This was introduced by Goldwasser and Micali [GM84]. The second goal is **non-malleability** (NM) and is due to Dolev et al. [DDN98]. In this case, we want an adversary to be unable, given a ciphertext c, to output another ciphertext c' such that the respective plaintexts m and m' are in a meaningful relation (e.g. $m' = 2m$).

The first security definition targets the weakest adversary, the eavesdropper. To encounter him, we denote the eavesdropping indistinguishability experiment $\mathsf{PubK}^{\mathsf{ind\text{-}eav}}_{\mathcal{A},\Pi}(\lambda)$ where Π_E is an encryption scheme (Gen, $\mathsf{Enc}, \mathsf{Dec}$), $\mathcal{A}$ is a PPT adversary and λ is the security parameter. In this experiment the adversary is given the public key of the encryption scheme which turns out to be the same as giving the adversary an access to an encryption oracle. For this reason, the security against eavesdroppers is equivalent to the security against **chosen plaintext attack** (CPA) described in the experiment $\mathsf{PubK}^{\mathsf{ind\text{-}cpa}}_{\mathcal{A},\Pi}(\lambda)$:

Experiment 2.1 ($\mathsf{PubK}^{\mathsf{ind\text{-}cpa}}_{\mathcal{A},\Pi}(\lambda)$)**.**

1. *run* $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$.
2. *give* $(\mathsf{pp}, \mathsf{pk})$ *to* $\mathcal{A}$. *Then,* $\mathcal{A}$ *has access to the encryption oracle* $\mathsf{Enc}(\mathsf{pk}, \cdot)$. *Finally,* $\mathcal{A}$ *outputs* $m_0, m_1 \in \mathbf{M}$.
3. *select* $b \xleftarrow{rand} \{0,1\}$ *and compute* $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$. *Hand* c *to* $\mathcal{A}$.
4. *give* $\mathcal{A}$ *access to the encryption oracle* $\mathsf{Enc}(\mathsf{pk}, \cdot)$. *Then* $\mathcal{A}$ *outputs a bit* b'.
5. *the output of the experiment is* 1 *if* $b = b'$ *and* 0 *otherwise.*

Definition 2.2. *A public-key encryption scheme* $\Pi := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has* ***indistinguishable encryption under chosen plaintext attack***

if for any PPT *adversary* $\mathcal{A}$*, there exists a negligible function* η *such that*

$$Pr[\mathsf{PubK}^{\mathsf{ind\text{-}cpa}}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \frac{1}{2} + \eta(\lambda)$$

We say that the scheme is IND-CPA*-secure.*

In the above experiment, the security requirement states that no adversary is able to distinguish between two random looking ciphertexts. A stronger requirement of security called **non-malleability** targets an adversary pursing the following goal. This adversary is able to extract information from a given ciphertext on a message m in order to produce another one on a message m' such that there is a meaningful relationship between m and m'. Of course the notion of "meaningful relationship" must be clarified. To illustrate it, imagine an adversary seeing an encryption $\mathsf{Enc}(m)$, would be able to output $\mathsf{Enc}(m+1)$ without decrypting m. This adversary would be able to win every time a two-bidders auction where the bids are encrypted and sent to the auctioneer if he is able to eavesdrop the bid of the other candidate before submitting his.

The NM-CPA security is defined through Experiment 2.2 [DDN98]. In this experiment, the adversary is split into two parts $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. Adversary $\mathcal{A}_1$ runs first and describes a valid message space $\mathbf{M}_s$ from which messages are pulled out. Then, $\mathcal{A}_2$ is presented with the encryption of a message from $\mathbf{M}_s$ and produces a vector of ciphertexts possibly related to this message. The experiment measures if this vector is indeed related to the message encrypted or if it is related to any random message.

Experiment 2.2 ($\mathsf{PubK}^{\mathsf{nm\text{-}cpa}}_{\mathcal{A},\Pi,b}(\lambda)$)**.** *Let the adversary* $\mathcal{A}$ *be split into* $(\mathcal{A}_1, \mathcal{A}_2)$*,* $\Pi := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *an encryption scheme and* b *a bit.*

1. *run* $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$.

2. *give* $(\mathsf{pp}, \mathsf{pk})$ *to* $\mathcal{A}_1$*. Then,* $\mathcal{A}_1$ *has access to the encryption oracle* $\mathsf{Enc}(\mathsf{pk}, \cdot)$*. Finally,* $\mathcal{A}_1$ *outputs* $(s, \mathbf{M}_s)$ *where* s *is the description of a message space* $\mathbf{M}_s \subset \mathbf{M}$.

3. *select* $m_0, m_1 \xleftarrow{rand} \mathbf{M}_s$ *such that* $|m_0| = |m_1|$ *and compute* $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_0)$*. Hand* $c, s, \mathbf{M}_s$ *to* $\mathcal{A}_2$.

4. *give* $\mathcal{A}_2$ *access to the encryption oracle* $\mathsf{Enc}(\mathsf{pk}, \cdot)$*. Then* $\mathcal{A}_2$ *outputs the description of a relation* R *and a vector of ciphertexts* $\boldsymbol{c'}$.

5. *decrypt the vector* $\boldsymbol{c'}$ *to obtain,* $\boldsymbol{m'} \leftarrow \mathsf{Dec}(\mathsf{sk}, \boldsymbol{c'})$. *Then, define the output of the experiment as follows:*

$$\forall c' \in \boldsymbol{c'} : c' \in \mathbf{C} \tag{2.1}$$

$$\wedge\, c \notin \boldsymbol{c'} \tag{2.2}$$

$$\wedge \begin{cases} R(m_0, \boldsymbol{m'}) & \text{if } b = 0 \\ R(m_1, \boldsymbol{m'}) & \text{if } b = 1 \end{cases} \tag{2.3}$$

In Experiment 2.2, the output condition 2.1 ensures that there is no invalid ciphertext and condition 2.2 ensures that the adversary $\mathcal{A}_2$ cannot copy the ciphertext c. The condition 2.3 is at the core of the non-malleability definition. Indeed, the goal of the adversary is to find a vector of ciphertexts $\boldsymbol{c'}$ whose decryption $\boldsymbol{m'}$ is meaningful with respect to the description of R, that is $R(m_b, \boldsymbol{m'})$ holds. We differentiate two cases with the bit b to measure if the adversary produces with a non-negligible probability ciphertexts that are meaningfully related to the encrypted message ($b = 0$) from the case of the adversary that produces ciphertexts that are meaningfully related to any random message ($b = 1$).

Definition 2.3. *A public-key encryption scheme* $\Pi := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has* ***non-malleable encryption under chosen plaintext attack*** *if for any* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* η *such that*

$$|Pr[\mathsf{PubK}^{\mathsf{nm\text{-}cpa}}_{\mathcal{A},\Pi,0}(\lambda) = 1] - Pr[\mathsf{PubK}^{\mathsf{nm\text{-}cpa}}_{\mathcal{A},\Pi,1}(\lambda) = 1]| \leq \eta(\lambda)$$

We say that the scheme is NM-CPA*-secure.*

NM-CPA secure encryption scheme are used in the different chapters of this thesis. In multi-party settings, non malleability is a crucial property to repel simple attacks from corrupted parties. For example, in our voting schemes of Chapter 4, we must prevent voters from cleverly copying the votes of other voters. As another example, in the shortest path applications of Chapters 5 and 6, it would be harmful if a party would be able to decide the weight of one of his edges according to the weight of someone else's edges. Otherwise, it would allow a corrupted party to lever off the output of the protocol.

For completeness we evoke a stronger kind of security called security against **chosen ciphertext attacks** (CCA) where an adversary is also given access to a decryption oracle before and after the challenge ciphertext in step 3 of Experiment 2.2.

There is a slight difference between the types of CCA security leading to a mid-point between CPA and CCA. Indeed, after the challenge

ciphertext in step 3, by giving $\mathcal{A}_2$ access to the decryption oracle or not, we define a variant of CCA. We note CCA1 when no access is given, and CCA2 (or CCA) when access is given. A CCA1-secure scheme is also called **non adaptive** against **chosen ciphertext attack**.

So far we can combine the different security notions by mixing the goals of the adversary $\{\mathsf{IND}, \mathsf{NM}\}$ with the different types of attacks $\{\mathsf{CPA}, \mathsf{CCA1}, \mathsf{CCA2}\}$ (we omit EAV) to obtain six security notions :

$$\{\mathsf{IND\text{-}CPA}, \mathsf{IND\text{-}CCA1}, \mathsf{IND\text{-}CCA2}, \mathsf{NM\text{-}CPA}, \mathsf{NM\text{-}CCA1}, \mathsf{NM\text{-}CCA2}\}$$

The work of Bellare et al. [BDPR98] presents and proves the relations and implications between these security notions. This can be summarized in a graph that we reproduce in Figure 2.1. In this graph, we observe four hierarchical levels between the different security notions. A security definition in a given level implies the security notions in the levels below but does not imply those of the above or equal levels.

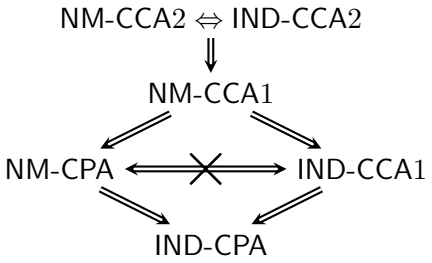


Figure 2.1: **Relations between security notions :** $A \Longrightarrow B$ means that there is a proof that security notion A implies security notion B. $A \not\Longleftrightarrow B$ means that there is a proof that security notion A does not imply security notion B and the other way around.

As a first practical public-key encryption scheme, we present the **ElGamal encryption** introduced in [ElG85]. This scheme can be made IND-CPA secure under the hardness of the decisional Diffie-Hellman problem (Assumption 2.4).

Definition 2.4 (ElGamal encryption)**.** *A public-key encryption scheme* $\Pi := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is an* ***ElGamal encryption scheme*** *if it is such that*

$\mathsf{Gen}(1^\lambda)$**:** *on input* 1^λ *a string of length* λ *where* λ *is a security parameter, produce* G *a multiplicative cyclic group of prime order* q *of length* λ*-bit as well as a generator* g *of* G*. Then choose* $x \xleftarrow{rand} \mathbb{Z}_q$ *and compute* $h := g^x$*. Set* $\mathsf{pp} := (G, q, g, \mathbf{M}, \mathbf{C})$ *where* $\mathbf{M} = G$ *and* $\mathbf{C} = G \times G$*. Set* $\mathsf{pk} := h$ *and* $\mathsf{sk} := x$ *then output* $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk})$*.*

$\mathsf{Enc}(\mathsf{pk}, m)$**:** *on input the public key* pk *and a message* $m \in \mathbf{M}$*, choose* $r \xleftarrow{rand} \mathbb{Z}_q$ *and compute* $c_1 := g^r$ *and* $c_2 := mh^r$*. Output* $c := (c_1, c_2)$*.*

$\mathsf{Dec}(\mathsf{sk}, c)$**:** *on input the secret key* sk *and a ciphertext* $c \in \mathbf{C}$*, parse* c *as* (c_1, c_2) *and compute* $\bar{m} := c_2/c_1^x$*. Output* $\bar{m}$*.*

Under the hardness of Assumption 2.4, this encryption scheme is IND-CPA-secure but not NM-CPA-secure.

Proposition 2.1. *An ElGamal encryption scheme* $\Pi := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* IND-CPA*-secure if the* DDH *assumption holds in* G *where* G *is the group generated in* Gen*.*

However, it is easy to see that the scheme is not NM-CPA-secure. Imagine an adversary receiving $c := (c_1, c_2) = (g^r, mh^r)$ an encryption of m in step 3 of Experiment 2.2. Then in step 4, the adversary outputs the ciphertext $c' := (c_1 g^s, c_2 h^s)$ for a random s. Then c' is an encryption of m as well which breaks the NM-CPA security for the identity relation R.

Homomorphic encryption

Performing operations over encrypted messages opens a panel of cryptographic applications. This is possible when we use an **homomorphic encryption scheme**, that is an encryption scheme where the encryption algorithm acts as an homomorphism between the message space $\mathbf{M}$ and the ciphertext space $\mathbf{C}$. For example some electronic voting schemes like those studied in Chapter 4 rely on homomorphic encryption to perform the tallying operations or to enable verifiability.

Definition 2.5. *An* ***homomorphic encryption scheme*** *is a public-key encryption scheme* $\Pi := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *such that* $\forall m_1, m_2 \in \mathbf{M}$*,*

$$\mathsf{Enc}(\mathsf{pk}, m_1 \star m_2) = \mathsf{Enc}(\mathsf{pk}, m_1) \diamond \mathsf{Enc}(\mathsf{pk}, m_2)$$

where $\star, \diamond$ *are the group operations in* $\mathbf{M}$ *and* $\mathbf{C}$ *respectively. We say that the encryption algorithm is homomorphic for* $\star$ *(or* $\star$ *homomorphic).*

We observe that the ElGamal encryption scheme (Definition 2.4) is homomorphic since we have for $c := (g^r, mh^r) \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$ and $c' := (g^{r'}, m'h^{r'}) \leftarrow \mathsf{Enc}(\mathsf{pk}, m')$ that $c \cdot c'$, the component-wise product, equals $(g^{r+r'}, mm'h^{r+r'})$ which is an encryption of mm'. The Enc algorithm plays the role of a multiplicative homomorphism. However, the homomorphic property of the ElGamal encryption scheme is not very

helpful for a direct practical use. Indeed, remember that m must be a group element of G. Actually, this message space is not very practical, for example, if we want to encrypt strings or numbers. In this case, we have to use an encoding operation, $\kappa : \mathbf{M}_{real} \to \mathbf{M}$ where $\mathbf{M}_{real}$ is the "real" message space targeted. The encryption process is thus $\mathsf{Enc} \circ \kappa : \mathbf{M}_{real} \to \mathbf{C}$ and the decryption process is $\kappa^{-1} \circ \mathsf{Dec} : \mathbf{C} \to \mathbf{M}_{real}$. This works nicely but if we want to keep the homomorphic property of the scheme, we must have that κ is also homomorphic. There lies our problem. Indeed, if we could find such efficient homomorphic κ^{-1} between $\mathbf{M}$ (G) and an organized space $\mathbf{M}_{real}$ which is equivalent to a space with an order relation ($\mathbf{M}_{real} \subset \mathbb{Z}$), then the discrete logarithm problem assumption does not hold in G and our encryption scheme cannot be made IND-CPA secure.

To get around this issue we introduce a new encryption scheme based on ElGamal and called **exponential ElGamal**. In this scheme, we see that the above encoding function κ is the modular exponentiation and thus κ^{-1} is the discrete logarithm function. For this reason, the message space must remain small.

Definition 2.6 (Exponential ElGamal encryption)**.** *A public-key encryption scheme* $\Pi := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is an* ***exponential ElGamal encryption scheme*** *if it is such that*

$\mathsf{Gen}(1^\lambda)$**:** *on input* 1^λ *a string of length* λ *where* λ *is a security parameter, produce* G *a multiplicative cyclic group of prime order* q *of length* λ*-bit as well as random generators* $g_1, g_2 \xleftarrow{rand} G$*. Then choose* $x \xleftarrow{rand} \mathbb{Z}_q$ *and compute* $h_1 := g_1^x$*. Set* $\mathsf{pp} := (G, q, g_1, g_2, \mathbf{M}, \mathbf{C})$ *where* $\mathbf{M} = \mathbb{Z}_q$ *and* $\mathbf{C} = G \times G$*. Set* $\mathsf{pk} := h_1$ *and* $\mathsf{sk} := x$ *then output* $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk})$*.*

$\mathsf{Enc}(\mathsf{pk}, m)$**:** *on input the public key* pk *and a message* $m \in \mathbf{M}$*, choose* $r \xleftarrow{rand} \mathbb{Z}_q$ *and compute* $c_1 := g_1^r$ *and* $c_2 := g_2^m h_1^r$*. Output* $c := (c_1, c_2)$*.*

$\mathsf{Dec}(\mathsf{sk}, c)$**:** *on input the secret key* sk *and a ciphertext* $c \in \mathbf{C}$*, parse* c *as* (c_1, c_2) *and compute* $\bar{m} := c_2 / c_1^x$*. Output* $\mathsf{Dlog}_{g_2} \bar{m}$*.*

We see that in order for the decryption algorithm to work, the message space must be limited to a small part of $\mathbb{Z}_q$. The homomorphic property of the scheme is preserved. However, the encryption is now an additive homomorphism, $\mathsf{Enc} : (\mathbb{Z}_q, +) \to (G, .)$.

To avoid the decryption limitation problem of exponential ElGamal, we present another homomorphic encryption scheme, that is the Paillier

encryption scheme proposed in [Pai99] that can be made IND-CPA secure under the hardness of the decisional composite residuosity (DCR) problem (see Assumption 2.1).

Definition 2.7 (Paillier encryption)**.** *A **Paillier encryption scheme** is a public-key encryption scheme* $\Pi := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *such that*

$\mathsf{Gen}(1^\lambda)$**:** *on input* 1^λ *a string of length* λ *where* λ *is a security parameter, produce* $n = pq$ *where* p *and* q *are primes of length* λ*-bit. Set* $\mathsf{pp} = \mathsf{pk} := n$ *and* $\mathsf{sk} := \varphi(n) = (p-1)(q-1)$ *then output* $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk})$*. We have that* $\mathbf{M} = \mathbb{Z}_n$ *and* $\mathbf{C} = \mathbb{Z}^*_{n^2}$*.*

$\mathsf{Enc}(\mathsf{pk}, m)$**:** *on input the public key* pk *and a message* $m \in \mathbf{M}$*, choose* $r \xleftarrow{rand} \mathbb{Z}^*_n$ *and compute* $c := r^n(1+n)^m \mod n^2$*. Output* c*.*

$\mathsf{Dec}(\mathsf{sk}, c)$**:** *on input the secret key* sk *and a ciphertext* $c \in \mathbf{C}$*, compute* $\bar{m}$ *as*

$$\bar{m} := \frac{c^{\varphi(n)} \mod n^2 - 1}{n} \cdot \varphi(n)^{-1} \mod n$$

Output $\bar{m}$*.*

We see that the correctness of the decryption holds since,

$$\begin{aligned}
\bar{m} &= \frac{r^{n\varphi(n)}(1 + [m\varphi(n) \mod n]n) \mod n^2 - 1}{n} \cdot \varphi(n)^{-1} \mod n && (2.4)\\
&= \frac{(1 + [m\varphi(n) \mod n]n) \mod n^2 - 1}{n} \cdot \varphi(n)^{-1} \mod n && (2.5)\\
&= \frac{[m\varphi(n) \mod n]n}{n} \cdot \varphi(n)^{-1} \mod n && (2.6)\\
&= [m\varphi(n) \mod n] \cdot \varphi(n)^{-1} \mod n\\
&= m
\end{aligned}$$

where in equation 2.4, we use the fact that $(1+n)^a = 1 + an \mod n^2$ for any $a \in \mathbb{Z}_n$ by computing the binomial coefficients. In equation 2.5, $r^{n\varphi(n)} = 1$ since $r^n \in \mathsf{Res}(n^2)$ is of order $\varphi(n)$. Finally in equation 2.6, we have that $1 + [m\varphi(n) \mod n]n$ is always smaller than n^2, so we can drop the $\mod n^2$ part.

Proposition 2.2. *A Paillier encryption scheme* $\Pi := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* IND-CPA*-secure if the* DCR *assumption holds for* n *generated in* Gen.

Moreover, for a Paillier encryption scheme, we have $\forall m_1, m_2 \in \mathbb{Z}_n$:

$$\begin{aligned}\mathsf{Enc}(\mathsf{pk}, m_1) \cdot \mathsf{Enc}(\mathsf{pk}, m_2) &= r_1^n (1+n)^{m_1} \cdot r_2^n (1+n)^{m_2} \\ &= (r_1 r_2)^n (1+n)^{m_1+m_2} \\ &= \mathsf{Enc}(\mathsf{pk}, m_1 + m_2).\end{aligned}$$

The Paillier encryption scheme is thus homomorphic for the addition with the encryption algorithm acting as an homomorphism between $\mathbb{Z}_n \times \mathbb{Z}_n^*$ and $\mathbb{Z}_{n^2}^*$. The Paillier encryption scheme is used in our generic construction in Chapter 3 that we use in Section 4.3 for our generic voting system.

By their very nature, homomorphic encryption schemes are malleable. Achieving non-malleability for those schemes while keeping their homomorphic property is possible but will require additional tools like the proofs of knowledge introduced in Section 2.1.5. This particular construction will be used to perform NM-CPA secure encryption scheme with homomorphic property in our voting schemes of Chapter 4.

Note that when $\mathbf{M}$ and $\mathbf{C}$ are rings and that the encryption algorithm acts as a ring homomorphism between them, we say that the encryption scheme is **fully homomorphic**. Such encryption schemes exist [Gen09] but remain to be improved regarding efficiency in order to be used for concrete applications.

2.1.4 Commitment scheme

Loosely speaking, commitment scheme allows us to commit on a message through a **commitment** which does not leak information about the message until an **opening** of the commitment is revealed. A commitment can be thought as a sealed box in which someone (say Alice) hides a message. The box is then handed to someone else (Bob) who makes sure that the content remains unchanged until Alice gives Bob the key to open the box. The commitment scheme is said to be **hiding**, when, given a commitment, it is hard to retrieve information about the message – the box hides its contents – and, the commitment scheme is said to be **binding**, when, given a commitment on some message m and its respective opening, it is hard to forge another opening that opens to a different message m' – it is hard to modify the content of the box –.

Definition 2.8 (Commitment scheme)**.** *A* ***commitment scheme*** Π_{C} *is a triple of algorithms* $(\mathsf{Gen}_{\mathsf{C}}, \mathsf{Com}, \mathsf{Verify})$ *such that:*

$\mathsf{Gen}_{\mathsf{C}}(1^\lambda)$**:** *on input* 1^λ *a string of length* λ *where* λ *is a security parameter, defines a finite key space* $\mathbf{K}_{\mathsf{C}}$ *and a commitment key* $\mathsf{cpk} \in \mathbf{K}_{\mathsf{C}}$

that contains the specification of a message space $\mathbf{M}_{\mathsf{C}}$*, a commitment space* $\mathbf{C}_{\mathsf{C}}$ *and an opening space* $\mathbf{O}$*.*

$\mathsf{Com}(\mathsf{cpk}, m)$**:** *on input the public key* cpk *and a message* $m \in \mathbf{M}_{\mathsf{C}}$*, outputs a commitment* $d \in \mathbf{C}_{\mathsf{C}}$ *and an opening* $o \in \mathbf{O}$*.*

$\mathsf{Verify}(\mathsf{cpk}, d, o, m)$**:** *on input the public key* cpk*, a message* $m \in \mathbf{M}_{\mathsf{C}}$ *and opening* $o \in \mathbf{O}$*, returns either* 1 *or* 0*.*

We assume that $\mathsf{Gen}_{\mathsf{C}}$ and Com are probabilistic algorithms while Verify is deterministic. The scheme Π_{C} possesses the following property:

Correctness: $\forall \mathsf{cpk} \leftarrow \mathsf{Gen}_{\mathsf{C}}(1^{\lambda})$, $\forall m \in \mathbf{M}_{\mathsf{C}}$, $\forall d, o \leftarrow \mathsf{Com}(\mathsf{cpk}, m)$, we have that $\mathsf{Verify}(\mathsf{cpk}, d, o, m) = 1$ with probability 1.

We specify the hiding and binding properties through the following experiments and definitions. During Experiments 2.3 and 2.4, once the adversary receives cpk, he is able to compute $\mathsf{Com}(\mathsf{cpk}, \cdot)$ at will.

Experiment 2.3 ($\mathsf{PubK}^{\mathsf{hiding}}_{\mathcal{A},\Pi_{\mathsf{C}}}(\lambda)$)**.**

1. *run* $\mathsf{cpk} \leftarrow \mathsf{Gen}(1^{\lambda})$*.*
2. *give* cpk *to* $\mathcal{A}$*. Then* $\mathcal{A}$ *outputs* $m_0, m_1 \in \mathbf{M}_{\mathsf{C}}$*.*
3. *flip a coin* $b \xleftarrow{rand} \{0,1\}$*, then compute* $d, o \leftarrow \mathsf{Com}(\mathsf{cpk}, m_b)$ *and hand* d *to* $\mathcal{A}$*.*
4. $\mathcal{A}$ *outputs a bit* b'*.*
5. *the output of the experiment is* 1 *if* $b = b'$ *and* 0 *otherwise.*

Definition 2.9 (Hiding commitment scheme)**.** *A commitment scheme* Π_{C} *defined as above is* ***computationally hiding****, if for all* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* η *such that*

$$Pr[\mathsf{PubK}^{\mathsf{hiding}}_{\mathcal{A},\Pi_{\mathsf{C}}}(\lambda) = 1] \leq \frac{1}{2} + \eta(\lambda).$$

Moreover, the commitment scheme Π_{C} *is* ***perfectly hiding****, if for every adversary* $\mathcal{A}$*, we have*

$$Pr[\mathsf{PubK}^{\mathsf{hiding}}_{\mathcal{A},\Pi_{\mathsf{C}}}(\lambda) = 1] = \frac{1}{2}.$$

Experiment 2.4 ($\mathsf{PubK}^{\mathsf{binding}}_{\mathcal{A},\Pi_{\mathsf{C}}}(\lambda)$)**.**

1. *run* $\mathsf{cpk} \leftarrow \mathsf{Gen}(1^{\lambda})$*.*

2. *give* cpk *to* $\mathcal{A}$. *Then* $\mathcal{A}$ *outputs* d, m, o, m', o' *where* $m' \neq m$.

3. *the output of the experiment equals* 1 *if*

$$\mathsf{Verify}(\mathsf{cpk}, d, m, o) = 1 \wedge \mathsf{Verify}(\mathsf{cpk}, d, m', o') = 1$$

and 0 *otherwise.*

Definition 2.10 (Binding commitment scheme)**.** *A commitment scheme* Π_{C} *defined as above is* ***computationally binding****, if for any* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* η *such that*

$$Pr[\mathsf{PubK}^{\mathsf{binding}}_{\mathcal{A},\Pi_{\mathsf{C}}}(\lambda) = 1] \leq \eta(\lambda).$$

Moreover, the commitment scheme Π_{C} *is* ***perfectly binding****, if for every adversary* $\mathcal{A}$*, we have*

$$Pr[\mathsf{PubK}^{\mathsf{binding}}_{\mathcal{A},\Pi_{\mathsf{C}}}(\lambda) = 1] = 0.$$

Observe that it is not possible to achieve a commitment scheme that would be perfectly hiding and perfectly binding at the same time. Indeed, if we suppose that such a scheme exists, it would mean that it is not possible for any adversary $\mathcal{A}_1$ with unbounded computational power to find that a commitment d on a message m with opening o an a second message $m' \neq m$ with opening o' such that both $\mathsf{Verify}(\mathsf{cpk}, d, o, m)$ and $\mathsf{Verify}(\mathsf{cpk}, d, o', m')$ accept because it would break the perfectly binding property. Nevertheless, a second unbounded adversary $\mathcal{A}_2$ on input d can compute every possible commitment which will, eventually but certainly, lead him to find m and o so that $\mathsf{Verify}(\mathsf{cpk}, d, o, m)$ accepts. However this is a violation of the perfectly hiding property.

A rather essential commitment scheme that is used throughout this thesis, is the so called **Pedersen commitment scheme** proposed by Pedersen in [Ped92]. The binding property of the scheme relies on the DL problem (Assumption 2.2).

Definition 2.11 (Pedersen commitment)**.** *A* ***Pedersen commitment scheme*** *is a commitment scheme* $\Pi_{\mathsf{C}} := (\mathsf{Gen}_{\mathsf{C}}, \mathsf{Com}, \mathsf{Verify})$ *such that,*

$\mathsf{Gen}_{\mathsf{C}}(1^\lambda)$**:** *on input* 1^λ *a string of length* λ *where* λ *is a security parameter, produce* G *a multiplicative cyclic group of prime order* q *of length* λ*-bit as well as generators* $g, h \xleftarrow{rand} G$*. Output* $\mathsf{cpk} := (g, h)$*. We have that* $\mathbf{M}_{\mathsf{C}} := \mathbb{Z}_q$*,* $\mathbf{C}_{\mathsf{C}} := G$ *and* $\mathbf{O} := \mathbb{Z}_q$*.*

$\mathsf{Com}(\mathsf{cpk}, m)$**:** *on input the public key* cpk *and a message* $m \in \mathbf{M}_{\mathsf{C}}$*, choose* $o \xleftarrow{rand} \mathbb{Z}_q$ *and compute* $d := g^m h^o$*. Output* d, o*.*

$\mathsf{Verify}(\mathsf{cpk}, d, o, m)$**:** *on input the public key* cpk*, a commitment* $d \in \mathbf{C}_{\mathsf{C}}$*, an opening* $o \in \mathbf{O}$ *and a message* $m \in \mathbf{M}_{\mathsf{C}}$*, return* $d \stackrel{?}{=} g^m h^o$*.*

Proposition 2.3. *The Pedersen commitment scheme of Definition 2.11 is perfectly hiding and computationally binding if one chooses* h *at random in* G *and that thus* $\mathsf{Dlog}_g h$ *is unknown.*

Proof.

perfectly hiding: given a commitment $d \in G$, $\forall m \in \mathbf{M}_{\mathsf{C}} = \mathbb{Z}_q$, we show that it is possible to find $o \in \mathbf{O} = \mathbb{Z}_q$ such that $d = g^m h^o$. Indeed, since g is a generator of G, there exists $x, y \in \mathbb{Z}_q$ such that $h = g^x$ and $d = g^y$. It suffices then to set $o = (y - m)/x$ to obtain our equality. This shows that a commitment can be opened to any message. Moreover, we see that the distribution of the commitments in G is independent of the message and uniformly distributed in G thanks to the random choice of o.

computationally binding: imagine, on the contrary, a computationally bounded adversary $\mathcal{A}$ succeeding to output, with non negligible probability, a commitment d on a message m with opening o as well as another message m' with opening o such that $d = g^m h^o$ and $d = g^{m'} h^{o'}$ at the same time. Then we can use $\mathcal{A}$ to break the DLP. Indeed, since $g^m h^o = g^{m'} h^{o'}$, we have that $g^{m-m'} = h^{o-o'}$ and thus that $\mathsf{Dlog}_g h = (o - o')/(m - m')$. Since h was chosen at random, $\mathcal{A}$ can be used to break the DLP with non negligible probability.

□

Trapdoor commitments. We define another kind of commitment schemes which are called **trapdoor** commitments. The trapdoor commitment scheme features one additional property: given a special value called the trapdoor, it is possible to open a commitment to any message. For example, the Pedersen commitment scheme can be turned into a trapdoor commitment if instead of choosing randomly the generator h in $\mathsf{Gen}_{\mathsf{C}}$, we choose randomly $x \xleftarrow{rand} \mathbb{Z}_q$ and set $h = g^x$. The trapdoor is then the value x. It is now clear that given x, we can compute an opening to any desired message (by following the steps in the proof of the perfectly hiding property of Proposition 2.3).

2.1.5 Proof of knowledge and sigma-protocols

In verifiable protocols, a party (person, server, user) often needs to prove to another party that he has access to some private information (secret

key, credential, encrypted message, etc.). This proof takes the form of an exchange between the first party usually called the Prover and the second party called the Verifier. At the end of the exchange, the Verifier agrees or disagrees that the Prover has access to the private information or to some knowledge. We call this exchange a **proof of knowledge**. To be more specific about what is meant by "knowledge", we must first clarify the way problems can be formulated. For this purpose, we define a formal NP-language $\mathcal{L}_{\mathsf{NP}}$ composed of a set of words and grammatical rules and used to express **statements** s. For these statements we say that the set $W(s)$ contains the solutions for which we can build acceptable proofs for s. We call $w \in W(s)$ one of the solutions or **witnesses** of s. This language $\mathcal{L}_{\mathsf{NP}}$ allows us to consider any binary relation $R \subset \mathcal{L}_{\mathsf{NP}} \times W(s)$ for some computational problem in NP. Loosely speaking, we say that a protocol between the Prover and the Verifier is a proof of knowledge for a relation R on a language $\mathcal{L}_{\mathsf{NP}}$, if the following holds. Given the common input s, and the private input w for the Prover such that $(s, w) \in R$, at the end of the protocol, the Verifier is convinced that the Prover knows a witness w. The classical definition of the proof of knowledge is given by Bellare and Goldreich [BG93]. We refer to [Dam99, Gol05, HL10] for additional details on this section.

Sigma-protocols

Sigma (or Σ) -protocols turn out to be an elegant way to obtain proofs of knowledge [Cra97]. It is a three-move exchange between the Prover and the Verifier (hence the Σ form).

Definition 2.12 (Σ-Protocol)**.** *A* Σ***-protocol*** π *for a relation* R *on an* NP*-language* $\mathcal{L}_{\mathsf{NP}}$*, is a pair of interactive algorithms* (Prover, Verifier) *such that, on inputs* $(s, w) \in R$ *for* Prover *and,* s *for* Verifier*, a three-move interaction takes place:*

1. Prover *outputs a "message"* $\mathfrak{a}$ *to* Verifier*.*
2. Verifier *selects a "challenge"* $\mathfrak{e}$ *of length* t*-bit uniformly at random from a challenge space and sends it to* Prover*.*
3. Prover *sends a "response"* $\mathfrak{z}$ *and halts.*

Eventually, Verifier *evaluates a predicate* Check *on the statement* s *and the transcript* $\mathfrak{t} := (\mathfrak{a}, \mathfrak{e}, \mathfrak{z})$ *and returns* 0 *or* 1*, then halts.*

Moreover, the Σ*-protocol satisfies the following properties:*

Completeness *An honest exchange between* Prover(s, w) *and* Verifier(s) *always accepts if* $(s, w) \in R$*.*

Special soundness *There exists a polynomial time extractor E that, on input of two valid transcripts $\mathfrak{t} := (\mathfrak{a}, \mathfrak{e}, \mathfrak{z})$ and $\mathfrak{t}' := (\mathfrak{a}, \mathfrak{e}', \mathfrak{z}')$ with respect to the same s where $\mathfrak{e} \neq \mathfrak{e}'$, returns a correct witness w. As a result, it is hard to produce invalid proofs.*

Special honest verifier zero-knowledge *There exists a probabilistic polynomial time simulator M which on inputs $s \in \mathcal{L}_{\mathsf{NP}}$ and a random challenge $\mathfrak{e}$, outputs a valid transcript $\mathfrak{t} := (\mathfrak{a}, \mathfrak{e}, \mathfrak{z})$ in the sense that this transcript is perfectly indistinguishable from a transcript issued from a real interaction between* Prover *and* Verifier*.*

Let us take a closer look at the special soundness and the special honest verifier zero-knowledge properties. The idea behind the soundness property is that it gives the Verifier the guarantee that the Prover really knows a witness w. And by knowing, we mean here that, somehow, the Prover is able to compute at least one valid w. Indeed, if the soundness holds, an honest Prover being able to answer correctly two different challenges must therefore be able to compute the witness. On the contrary, a cheating Prover who does not know w can only be able to answer at most one challenge (otherwise he knows w after all) with a probability of 2^{-t} which corresponds to the probability of answering correctly by randomly guessing.

Regarding the special honest verifier zero-knowledge property, it allows producing transcripts that look exactly like real conversations between Prover and Verifier but without the knowledge of a witness. This can be done without violating the soundness property. The reason is that the components of the transcript do not need to be produced in the same order as those of a real conversation. The goal of this property is to make sure that transcripts issued from real conversations do not leak any information about the witnesses since they are not distinguishable from any other transcripts.

The key property that we want to achieve in proofs of knowledge is the guarantee that the Verifier or any observer of the transcript of the proof cannot gain any information about the witness whatsoever except for the one bit of information that tells if the Prover knows the witness for a given statement or not. This property is called **zero-knowledge** and was formalized in the seminal paper of Goldwasser, Micali and Rackoff [GMR85]. It can be shown (see e.g. [HL10]) that given a perfectly hiding trapdoor commitment and a Σ-protocol for a relation R we can achieve a proof of knowledge with the zero-knowledge property for R. This proof is called a **zero-knowledge proof of knowledge** or ZKPK.

We make the distinction between **perfect**, **statistical** and **computational** zero-knowledge which indicates the distance between the two

distribution in the honest verifier zero-knowledge property of Definition 2.12. For any Verifier, the first distribution comes from the simulated transcripts issued by simulator M and the second distribution comes from the transcripts issued from the real interaction between Prover and Verifier. Perfect zero-knowledge means that the two distributions are equal and statistical zero-knowledge means that the statistical distance between them is negligible. We have that perfect zero-knowledge (as stated in Definition 2.12) implies statistical zero-knowledge which in turns implies computational zero-knowledge.

ZKPK plays a significant role in this thesis. As an illustration but also for later use, we describe here the sigma-protocol that can be used to obtain a proof that the discrete logarithm of some group element relative to a given generator is known. Let p be a prime, q be a prime divisor of $p-1$ and $g \in \mathbb{Z}_p^*$ be a generator of order q of the group G. For a random $w \xleftarrow{rand} \mathbb{Z}_q$ chosen by Prover, the relation we target is

$$R_{\mathsf{DL}} := \{(s, w) | s = (G, p, q, g, h) \wedge h = g^w\}$$

where

Protocol 2.1 (Σ-protocol for DL knowledge)**.**

Prover **and** Verifier **input:** (G, p, q, g, h).

Prover **private input:** w *such that* $h = g^w$.

Process:

1. Prover *chooses* $x \xleftarrow{rand} \mathbb{Z}_q$ *and computes* $\mathfrak{a} = g^x$. *Then it sends* $\mathfrak{a}$ *to* Verifier.
2. Verifier *selects a challenge* $\mathfrak{e} \xleftarrow{rand} \{0,1\}^t$ *where* $2^t < q$ *and sends it to* Prover.
3. Prover *computes* $\mathfrak{z} := x + \mathfrak{e}w \mod q$ *and sends* $\mathfrak{z}$ *to* Verifier. *Eventually,* Verifier *checks that* $\mathfrak{a}h^{\mathfrak{e}} \stackrel{?}{=} g^{\mathfrak{z}}$.

Proposition 2.4. *Protocol 2.1 is a* Σ*-protocol.*

Proof. Completeness is straightforward. Regarding soundness, given two valid transcripts $\mathfrak{t} := (\mathfrak{a}, \mathfrak{e}, \mathfrak{z})$ and $\mathfrak{t}' := (\mathfrak{a}, \mathfrak{e}', \mathfrak{z}')$ if we divide $\mathfrak{a}h^{\mathfrak{e}'} = g^{\mathfrak{z}'}$ by $\mathfrak{a}h^{\mathfrak{e}} = g^{\mathfrak{z}}$ side by side, we get $h^{\mathfrak{e}-\mathfrak{e}'} = g^{\mathfrak{z}-\mathfrak{z}'}$. We can extract w as $(\mathfrak{z} - \mathfrak{z}')/(\mathfrak{e} - \mathfrak{e}')$ which is well defined since $\mathfrak{e} \neq \mathfrak{e}'$. Considering the special honest verifier zero-knowledge, given $(h, \mathfrak{e})$ as inputs, the simulator M proceeds in this way: it selects a random $\mathfrak{z} \xleftarrow{rand} \mathbb{Z}_q$ and computes $\mathfrak{a} = g^{\mathfrak{z}}h^{-\mathfrak{e}}$. It is clear that $(\mathfrak{a}, \mathfrak{e}, \mathfrak{z})$ has the same probability distribution as a transcript issued from a real exchange between Prover and Verifier. □

2.1.6 The Fiat-Shamir transformation

It is possible to render sigma-protocols non-interactive which is of great interest when the number of exchanges between the different parties involved in a protocol has to be minimized. The Prover prepares a non-interactive proof that is later verified by the Verifier. The result is a Non-Interactive Proof of Knowledge (NIZKPK).

Definition 2.13 (Non-interactive proof of knowledge)**.** *A couple of efficient algorithms* (Prove, Check) *is called a* ***non-interactive proof of knowledge*** π_{NI} *on an* NP*-language* $\mathcal{L}_{\mathsf{NP}}$ *for a relation* R *if it is such that:*

$\mathsf{Prove}(s, w)$**:** *on inputs a statement* s *and a witness* w *with* $(s, w) \in R$*, produce a transcript* $\mathfrak{t}$*.*

$\mathsf{Check}(s, \mathfrak{t})$**:** *on inputs a statement* s *and a transcript* t*, outputs* 0 *or* 1*.*

where Prove *is a probabilistic algorithm run by the* Prover *and* Check *is a deterministic algorithm run by the* Verifier*. Moreover, we have the following properties:*

Completeness $\forall (s, w) \in R$, $\forall \mathfrak{t} \leftarrow \mathsf{Prove}(s, w)$ *we have, with overwhelming probability, that* $\mathsf{Check}(s, \mathfrak{t}) = 1$.

Zero-knowledge *(informal) there exists a simulator that can produce simulated transcripts such that no adversary can distinguish a real proof from a simulated one with a probability non-negligibly better than* $1/2$*. We refer to [BR93, BPW12] for details.*

In order for this setting to work, imagine that the Prover and the Verifier have access to a **random oracle**. This entity sets up a random function $r : \{0,1\}^k \to \{0,1\}^l$, for some k, l and answers bit-string queries a of length k by returning $r(a)$, a uniformly distributed, independent of a, random bit-string of length l. In addition, the random oracle remembers former queries and replies consistently.

If such a random oracle exists, the Prover can convince the Verifier without interacting with him and thus it allows to remove the second step in a Σ-protocol. To do that, the Prover sends $\mathfrak{a}$ to the random oracle and uses the oracle answer $r(\mathfrak{a})$ as the challenge $\mathfrak{e}$ to compute $\mathfrak{z}$. Then the Prover sends $(\mathfrak{a}, \mathfrak{z})$ to the Verifier which, in turn, obtains $\mathfrak{e}$ from the oracle queried on $\mathfrak{a}$ and checks $\mathfrak{z}$. This course of actions is not different for the Prover since, as before, he has no influence on the choice of the challenge. Moreover, the Verifier is now forced to be honest since he can no longer choose the challenge. The only real difference is that the parties are free to call the oracle at will.

With all these considerations, in order to build the simulator of the zero-knowledge property, we must emulate the random oracle. The simulator will answer queries on behalf of the random oracle by maintaining a list of oracle query/response pairs. Then, the simulated transcripts are produced via the emulated random oracle of the simulator. The question that remains is the following. Is it possible to produce an imitation random oracle that is indistinguishable from an ideal one?

One way to implement the random oracle is to use cryptographic hash functions as done in the Fiat-Shamir/Blum transformation [FS87] (attributed to Blum by [BR93]) described below. This leads to efficient protocols. However, the counterpart of using hash functions is that the security model, in which security proofs are formalized, shifts from the **standard model** where no unrealistic assumptions are admitted to a security model called the **random oracle model**. This model introduced by Bellare and Rogaway [BR93] makes the assumption that (good) cryptographic hash functions exist and are indistinguishable from perfectly random functions (or random oracles). Although hash functions provide a fairly good and unpredictable (with respect to today's knowledge) source of randomness, it is clear that they will never be equivalent to random oracles. Indeed, hash functions are deterministic and not random, which means that it is possible to know them entirely.

Nevertheless, security achieved in the random oracle model does not translate to the standard model (see [CGH04, Dam07]). It seems that security proofs made in the random oracle model are a healthy start to ensure security at minima against design flaws and well-studied attacks but it is not a sufficient condition if we seek security guarantee in the real-world.

While keeping this in mind, we present the Fiat-Shamir/Blum transformation that turns Σ-protocols into non-interactive proofs using cryptographic hash functions. For this purpose, we define algorithm Recomb used to recombine the $\mathfrak{a}$ message from the statement s, a challenge $\mathfrak{e}$ and a response $\mathfrak{z}$ produced in the Σ-protocol.

Theorem 2.1 (Fiat-Shamir/Blum transformation [FS87, BPW12])**.** *Given a Σ-protocol $\pi := (\mathsf{Prover}, \mathsf{Verifier})$ on an NP-language $\mathcal{L}_{\mathsf{NP}}$ for a relation R, an efficient cryptographic hash function $\mathcal{H}$ and a deterministic algorithm Recomb, a **Fiat-Shamir/Blum transformation** of π is a non-interactive proof π_{NI} defined as follows:*

$\mathsf{Prove}(s, w)$**:** *on inputs a statement s and a witness w with $(s, w) \in R$, run the Prover on step 1 with (s, w) as inputs to obtain the message $\mathfrak{a}$ and compute $\mathfrak{e} := \mathcal{H}(s, \mathfrak{a})$. Then run the Prover on step 3 with*

$\mathfrak{e}$ *as input to obtain the response* $\mathfrak{z}$*. Output the transcript as* $\mathfrak{t} := (\mathfrak{e}, \mathfrak{z})$*.*

$\mathsf{Check}(s, \mathfrak{t})$**:** *on inputs a statement* s *and a transcript* $\mathfrak{t}$*, parse* $\mathfrak{t}$ *as* $(\mathfrak{e}, \mathfrak{z})$ *and compute* $\mathfrak{a}' := \mathsf{Recomb}(s, \mathfrak{e}, \mathfrak{z})$*. Then output* $\mathfrak{e} \stackrel{?}{=} \mathcal{H}(s, \mathfrak{a}')$*.*

Moreover, $\forall (s, w) \in R$*,* $\forall \mathfrak{a}$ *obtained from the* Prover *in step* 1 *of* π*,* $\mathfrak{t} \leftarrow \mathsf{Prove}(s, w)$*, we have that* $\mathsf{Recomb}(s, \mathfrak{e}, \mathfrak{z}) = \mathfrak{a}$ *holds with overwhelming probability.*

As an illustration of the Fiat-Shamir/Blum heuristic, we apply it to the Σ-protocol for DL knowledge (Protocol 2.1) to obtain a non-interactive proof of DL knowledge:

Protocol 2.2 (Non-interactive proof for DL knowledge)**.**
Given $(s, w) \in R_{\mathsf{DL}}$ *(*$s := (G, q, g, h)$ *and* w *such that* $h = g^w$*). We define* $\pi_{\mathsf{NI,DL}} := (\mathsf{Prove}, \mathsf{Check})$ *as follows:*

$\mathsf{Prove}(s, w)$**:** *choose* $x \xleftarrow{rand} \mathbb{Z}_q$ *and computes* $\mathfrak{a} = g^x$ *and* $\mathfrak{e} := \mathcal{H}(s, \mathfrak{a})$*. Then compute* $\mathfrak{z} := x + \mathfrak{e}w \mod q$ *and output* $t := (\mathfrak{e}, \mathfrak{z})$*.*

$\mathsf{Check}(s, t)$**:** *get* $\mathfrak{a}' \leftarrow \mathsf{Recomb}(s, \mathfrak{e}, \mathfrak{z})$ *where* $\mathsf{Recomb}(s, \mathfrak{e}, \mathfrak{z})$ *outputs* $g^{\mathfrak{z}} h^{-\mathfrak{e}}$*. Then, output* $\mathfrak{e} \stackrel{?}{=} \mathcal{H}(s, \mathfrak{a}')$*.*

Despite the inconvenient of falling in a weaker security paradigm, the use of non-interactive proofs of knowledge achieved through hash functions opens the door to very practical and efficient applications. The ones targeted in this thesis strongly rely on NIZKPK. One of the main reasons is that we remove the constraints due to communications between the parties. Communications, especially when they need to be synchronized, are curbing large scale multi-party applications. For example, it is not conceivable to force a voter to stay behind his computer with a stable internet connexion to provide challenges to election authorities or other voters during the whole election process.

2.2 Secret sharing

Historically, secret sharing was introduced to split a secret between different parties in a way that only one subset if not all of them can reconstruct the secret. Secret sharing can be used for secret data storage where the user splits his private data between different data centers with the guarantee for the user that none of the servers can retrieve his personal information. In the same fashion, secret sharing comes in handy for threshold encryption (Section 2.2.1) to split the private decryption

key between the key holders. Finally secret sharing is convenient in protocols involving multiple parties where there is often a need to spread the private information of the participants among themselves to allow computations while preserving the privacy.

Secret sharing schemes were proposed independently by Shamir [Sha79] and Blakey [Bla79]. A whole literature of secret sharing schemes has been proposed to achieve diverse functionalities such as specific hierarchical rules between the parties or to allow computations over the shares (a short survey on secret sharing can be found in [Bei11]). We present here the main ideas of secret sharing as well as Shamir's secret sharing scheme, which is used in this thesis.

In the following, we adopt the conventional notation where for a secret value s, $[s]$ is a vector of *shares* of the secret s (in particular, $[s]_i$ is one of the share of s usually given to participant i), and we commonly say that $[s]$ is the *shared value* of s. We also use the notation $[\boldsymbol{v}]$ to designate a vector of shared values for example $[\boldsymbol{v}] := ([v_1], \ldots, [v_n])$ on the list of secret values $(v_1, \ldots, v_n)$. In the same way, we use the notation $[\boldsymbol{M}]$ to designate a matrix of shared values.

Definition 2.14 (Secret sharing scheme)**.** *A* ***secret sharing scheme*** Υ *between n parties denoted $P_1, ..., P_n$, is a pair of efficient algorithms* $(\mathsf{Share}, \mathsf{Open})$*, the specification of a space* $\mathbf{Se}$ *for the secrets, the specification of a space* $\mathbf{Sh}$ *for the shares as well as a recovery family of sets $R \subseteq \mathcal{P}(\{1, ..., n\})$ (where $\mathcal{P}$ designates the power set and R is the family of sets of the parties that can jointly open a secret from the shares) such that:*

$\mathsf{Share}(s)$**:** *on input a secret $s \in \mathbf{Se}$, output a vector of shares $[s] := ([s]_1, ..., [s]_n) \in \mathbf{Sh}^n$. Note that the shares are supposed to be distributed among the parties, P_i receiving share $[s]_i$.*

$\mathsf{Open}([s])$**:** *on input a vector of shares $[s] := ([s]_{l_1}, ..., [s]_{l_t}) \in \mathbf{Sh}^t$ with $t \geq 1$ and where $\{l_1, ..., l_t\} \in R$, output a secret $s' \in \mathbf{Se}$. Note that the vector $[s]$ is supposed to be aggregated by the parties.*

Algorithm Share *is probabilistic while algorithm* Open *is deterministic. Moreover, we have the following properties:*

Correctness *$\forall s \in \mathbf{Se}$, $\forall([s]_1, ..., [s]_n) \leftarrow \mathsf{Share}(s)$, $\forall [s] := ([s]_{l_1}, ..., [s]_{l_t})$ such that $\{l_1, ..., l_t\} \in R$ we have that $\mathsf{Open}([s]) = s$ with overwhelming probability.*

Perfect Privacy *(Unauthorized set of parties cannot recover anything about the secret) $\forall [s] := ([s]_{l_1}, ..., [s]_{l_t})$ such that $\{l_1, ..., l_t\} \notin R$, we*

have $\forall s', s'' \in \mathbf{Se}$ *with* $s' \neq s''$ *that:*

$$Pr[[s] \subset ([s']_1, ..., [s']_n) | ([s']_1, ..., [s']_n) \leftarrow \mathsf{Share}(s')] \\ = Pr[[s] \subset ([s'']_1, ..., [s'']_n) | ([s'']_1, ..., [s'']_n) \leftarrow \mathsf{Share}(s'')].$$

We can achieve complex recovery sets R corresponding to hierarchical structure between the parties (e.g. P_1 can recover the secret alone but P_2 cannot recover it without the help of P_1, etc.) but one of the simplest is the threshold structure where $R := \{A \subseteq \{1, ..., n\} | |A| \geq t\}$ with $1 \leq t \leq n$ meaning that if at least t parties joint together, they can open a secret from their shares. Secret sharing schemes of this kind are called *t-out-of-n threshold secret sharing schemes.* One example of such a scheme is given by Shamir in [Sha79] and is based on polynomial interpolation.

Definition 2.15 (Shamir's threshold secret sharing scheme)**.** *Given a finite field of order* q, $\mathbb{F}_q$ *and* $\alpha_1, ..., \alpha_n \xleftarrow{rand} \mathbb{F}_q^*$ *where the* α_i*-s are distinct, we define a* ***t-out-of-n Shamir's threshold secret sharing scheme*** $\Upsilon := (\mathsf{Share}, \mathsf{Open})$ *between n parties denoted* $P_1, ..., P_n$*, as a secret sharing scheme where* $\mathbf{Se} = \mathbf{Sh} := \mathbb{F}_q$*,* $R := \{A \subseteq \{1, ..., n\} | |A| = t_A \geq t\}$ *and such that:*

$\mathsf{Share}(s)$**:** *choose* $a_1, ..., a_t \xleftarrow{rand} \mathbb{F}_q$ *and compute* $[s]_i := s + \sum_{j=1}^{t} a_j \alpha_i^j$ *for* $i = 1, ..., n$*. Note that* $[s]_i$ *is the evaluation of the t-degree polynomial* $f(x) := s + \sum_{j=1}^{t} a_j x^j$ *at* α_i*. Output* $([s]_1, ..., [s]_n)$*.*

$\mathsf{Open}([s])$**:** *if* $[s]$ *is not of the form* $([s]_{l_1}, ..., [s]_{l_{t_A}})$ *where the* l_i*-s are the indexes of existing parties, halt. Otherwise compute and output* s' *as*

$$\sum_{i=1}^{t_A} [s]_{l_i} \prod_{\substack{j=1 \\ j \neq i}}^{t_A} \frac{\alpha_{l_j}}{\alpha_{l_j} - \alpha_{l_i}}$$

Note that s' *is the evaluation of Lagrange's interpolation polynomial* $g(x) := \sum_{i=1}^{t_A} [s]_{l_i} \prod_{\substack{j=1 \\ j \neq i}}^{t_A} \frac{\alpha_{l_j} - x}{\alpha_{l_j} - \alpha_{l_i}}$ *in* 0*.*

It is easy to see that the above definition holds correctness. Regarding perfect privacy, it suffices to notice that every set of at most $t-1$ parties holding $t-1$ points can recover exactly one polynomial of degree $t-1$ and q distinct polynomials of degree t corresponding to the q

possible choices of an extra point in $\mathbb{F}_q$. This shows that a group of less than t parties might open its vector of shares to any possible secret in **Se** without any preference.

Secret sharing is at the root of numerous multi-party computation schemes as it offers privacy of the inputs and in some schemes, the possibility for the parties to compute over the shares. This is the case for Shamir's threshold secret sharing scheme that features addition and multiplication operations over the shared secrets.

Consider the two shared values $[s_1], [s_2]$ issued from $\mathsf{Share}(s_1)$ and $\mathsf{Share}(s_2)$ respectively. To obtain a shared value $[s_3]$ opening on $s_1 + s_2$, each party P_i computes $[s_3]_i := [s_1]_i + [s_2]_i$. Since the sum of the two corresponding polynomials $f_1(x), f_2(x)$ is a polynomial of the same degree $f_3(x)$ and that $f_3(0) = f_1(0) + f_2(0)$ the operation is consistent. For the multiplication, the product of the shares $[s_3]_i := [s_1]_i[s_2]_i$ leads to polynomial $f_3(x) := f_1(x)f_2(x)$ for which $f_3(0) = s_1 s_2$. However, this polynomial is now of degree $2n$. To reduce this degree to n, the parties need to re-share the vector $[s_3]$. In practice, each party P_i computes $[u_i] \leftarrow \mathsf{Share}([s_3]_i)$ and sends $[u_i]_j$ to party P_j. Then, each party P_i computes $[v]_i \leftarrow \mathsf{Open}(([u_1]_i, ..., [u_n]_i))$. We see that the shared value $[v] := ([v]_1, ..., [v]_n)$ distributed among the parties opens on the secret $s_1 s_2$ through a polynomial of degree t. It is worth noticing that the addition operation is almost free for the parties since it can be done locally while the multiplication operation requires a re-sharing phase which implies a round of communication between the parties. This simple difference will have a huge impact on the design of the protocols and is a continuous concern throughout this thesis and in particular in Chapter 6.

2.2.1 Threshold encryption scheme

From secret sharing, there is one step to cross to achieve threshold encryption scheme. Threshold encryption is very useful when it comes to distributing the risk of privacy leakage due to the fact that the secret key of an encryption scheme may fall into the wrong hands. Indeed, in some scenarios, we do not want to entrust only one party with the responsibility to decrypt any encrypted message. To achieve this, the secret key must somehow be split between a set of key holders. Then, the key holders must all agree if they want to decrypt a ciphertext. This gives some guarantees that only desired ciphertext are decrypted. For example, in voting schemes, after that the tally is computed over the ciphertext thanks to the homomorphic property of the encryption scheme used, the election authorities gather and decrypt the resulting ciphertext

(and only this one).

Loosely speaking, a **threshold encryption scheme** allows us to spread the decryption key among a set of key holders. Then, the decryption of a given message requires the collaboration of a subset of key holders whose number is fixed as a security parameter. For scheme based on the DLP assumption, one way to achieve this goal was proposed by Pedersen in [Ped91] to turn the ElGamal encryption scheme into a threshold ElGamal encryption scheme. The idea is quite simple. Given a public generator g of a prime order q group G, each of the key holders (or authority) P_i selects randomly $x_i \xleftarrow{rand} \mathbb{Z}_q$, computes $h_i = g^{x_i}$ and sends h_i to every other P_j. Then, each P_i computes the public key $\prod h_i = g^{\sum x_i}$ while the secret key $X := \sum x_i \mod q-1$ remains secret to everyone unlesss every party join together to recover X. The key distribution algorithm can also be based on polynomial interpolation as what is done in the previous section, and can be used to achieve t-out-of-n threshold encryption schemes where a subset t of the n parties can gather in order to decrypt a ciphertext. As an illustration, the ElGamal threshold encryption is in use in the current version of the Helios voting system [ADMPQ09, AdMP12].

There is, obviously, threshold encryption schemes based on different cryptographic assumptions. Damgård and Jurik propose a threshold encryption version of the Paillier encryption scheme in [DJ01] which could be used in the generic instantiation of our voting scheme in Section 4.3. For the RSA assumption, we mention the schemes presented in [BF97, FMY98, FS01]. However, note that in this case, the key generation algorithm is trickier to set up compared to the one presented above. It may require a trusted third party or an additional secure multi-party protocol emulating this third party for jointly generating an RSA modulus with unknown factors.

2.2.2 From secret sharing to secure multi-party computation

Multi-party computation comes down to this: a set of n parties $(P_1, ..., P_n)$ wishes to evaluate a function f over their respective private inputs $(x_1, ..., x_n)$ to obtain $f(x_1, ..., x_n)$. We talk about **secure multi-party computation** (SMC) when in addition, the parties receive guarantees about the *privacy* of their inputs and about the *correctness* of the result. This problem was introduced by Yao in his 1982 paper [Yao82] through the millionaires' problem in which two millionaires want to know which of them is the richest without revealing what is on their bank account. This section follows notations and definitions

of [HL10, CDN10].

It should be stressed clearly that the privacy property ensures the privacy of the parties inputs except everything that could be inferred from the result. For example, an auctioneer not winning an auction is allowed to deduce that the winning bid is higher than his own bid.

In order to provide security guarantees for SMC protocols, we ought to define the kind of adversary we encounter. Literature considers two main categories: first the *honest-but-curious* adversary also called the **passive adversary**. This adversary is able to corrupt some of the parties *statically* or *adaptively* and his goal is to learn the private information of the (uncorrupted) parties. We say the adversary is honest since he does not make the corrupted parties deviate from the protocol. On the contrary, the second kind of adversary called *malicious* adversary or **active adversary** is able to corrupt the parties **and** force them to arbitrarily deviate from the protocol. Doing so, the active adversary seeks one or both of two goals: learn about the private information of the parties, and modify the result of the evaluation of the function.

We present a multi-party protocol in terms of ideal functionality, that is, we define our expectations in an ideal world and then, we confront the real world execution of the protocol to a simulation of the protocol in the ideal world. We expect these two executions to be equal or at least indistinguishable. Doing so, we do not have to explicit every possible behaviour of an adversary. Instead, we show that any real adversary can be efficiently simulated which means that his capabilities are limited within the ideal functionality definition. For multi-party protocol, the ideal functionality is played by a trusted third party that collects the private inputs of the parties, computes the target function and then discloses the result to each party. It is quite easy to see that in this scenario, both privacy and correctness hold perfectly.

Functionality 2.1 (Secure multi-party computation: $\mathcal{F}_{\mathsf{SMC},f}$)**.** *Let $(P_1, ..., P_n)$ be a set of parties holding the private inputs $(x_1, ..., x_n) \in \mathbf{I}^n$ respectively where $\mathbf{I}$ is the input space and let $f : \mathbf{I}^n \to \mathbf{O}$ be the function that the parties wish to evaluate over their private inputs where $\mathbf{O}$ is the output space. We define the* SMC ***functionality*** *$\mathcal{F}_{\mathsf{SMC},f}$ that is realized by an uncorrupted trusted third party $\mathcal{T}$ as follows:*

1. *each party P_i sends his input x_i to $\mathcal{T}$. If $x_i \notin \mathbf{I}$, $\mathcal{T}$ aborts otherwise $\mathcal{T}$ stores x_i.*

2. *when $\mathcal{T}$ has received all the inputs, $\mathcal{T}$ computes $y := f(x_1, ..., x_n)$ and sends y to each P_i then $\mathcal{T}$ halts.*

In Definition 2.16 below, we use the notation $\mathsf{exec}_{\pi}^{\mathcal{E},\mathcal{A}}(\boldsymbol{z})$ to denote the environment $\mathcal{E}$'s view of the execution of protocol π on input $\boldsymbol{z}$ in the presence of adversary $\mathcal{A}$.

Definition 2.16 (Secure multi-party computation protocol)**.** *Protocol $\pi_{\mathsf{SMC},f}$ is a **passively (or actively) secure multi-party computation protocol** if it securely computes a function f for a set of parties $(P_1, ..., P_n)$ holding the private inputs $(x_1, ..., x_n) \in \mathbf{I}^n$ in the presence of passive (or active) adversaries if for any environment $\mathcal{E}$ and for any* PPT *adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ that simulates the behaviour of $\mathcal{A}$ such that:*

$$\mathsf{exec}_{\mathcal{F}_{\mathsf{SMC},f}}^{\mathcal{E},\mathcal{S}}(\boldsymbol{z}) \cong \mathsf{exec}_{\pi_{\mathsf{SMC},f}}^{\mathcal{E},\mathcal{A}}(\boldsymbol{z})$$

where $\boldsymbol{z}$ states the initial input of every participant of the protocol and where the symbol $\cong$ means that the two distributions are computationally indistinguishable.

A direct way to build SMC protocols relies on Shamir's secret sharing scheme. We can see that this provides passive security (e.g. [CCD88, GRR98]) as long as a threshold of corrupted parties is not crossed. Relying on various techniques such as *oblivious transfer*, many other SMC protocols exist that provide either passive or active security in a more and more efficient way (non exhaustively, [LPS08, PSSW09, BDOZ11, BNTW12, NNOB12, DKL+13]). Some of these protocols have been implemented and laid down frameworks for SMC prototyping or applications (most notably, Sharemind [BLW08], Fairplay [MNPS04], VIFF [DGKN09], TASTY [HKS+10], SEPIA [BSMD10], SCAPI [EFLL12], Wysteria [RHH14]). The framework VIFF [VIF] was used during the thesis for prototyping (see Chapter 6). It is based on Shamir's secret sharing scheme and offers passive security as well as active security however for this last case, the authors do not give any strong guarantee.

In the protocols we address using SMC, we analyse the complexity as if the operations involved (additions, multiplications, comparisons) are all atomic. In this regards, we treat these operations as calls to black boxes and focus on the complexity of the algorithm itself. Although this approach allows us to obtain results independent of the SMC scheme used, we have to keep in mind that a complete complexity analysis would take into account the real cost of the operations and in particular the communication cost. For example, in VIFF (as likely in most SMC frameworks), one multiplication involves a round of communication between the parties whereas one comparison costs about 165 multiplications. Moreover, note that the number of communication

channels grows quadratically in the number of parties and linearly for each party.

Part II

Auditable Multi-Party Function Evaluation

Chapter 3

Commitment Consistent Encryption: A New Cryptographic Primitive

The commitment consistent encryption primitive was introduced in our paper [CPP13] presented at the ESORICS conference in 2013 within the framework of cryptographic voting. Since the range of application of this primitive was enlarged during this thesis, we dedicate this chapter to its presentation.

3.1 Introduction

One of the cornerstone of this thesis is the introduction of a new primitive called **commitment consistent encryption** (CCE). This primitive offers a commitment scheme within an encryption scheme in a way that both functionalities and advantages are combined together. Indeed, our audit mechanisms require a trail allowing observers to verify the correctness of the result (the tally of the votes in Chapter 4 and the output of the function in Chapter 5). As it is available to the public, this trail could lead to the impairment of the privacy property. Imagine however that the public audit trail contains only perfectly hiding commitments. Then, through the trail, no information can be leaked whatsoever in the sense of the theory of information. This provides us a perfectly private audit trail (PPAT).

On the other hand, in a multi-party setting, the private inputs of the parties (clients, voters, etc.) must be conveyed to the set of people (workers, voting authorities, servers, etc.) that actually perform the computations. This is naturally achieved throughout an encryption scheme. The trick is now to combine commitment and encryption in a consistent way for one main reason. Indeed, since the audit trail is used to assert the correctness of the result, it must be bound to the private inputs on which the result is computed. This gives guarantee to the clients and to the workers. First, to the clients, for they only see the audit trail and this must be enough to assure them that the result is correct and consistent with the inputs committed by the other clients. Then, to the workers since they ought to be sure that they will be able to compute a result that is consistent with the audit trail, otherwise the proof of correctness would be violated.

3.1.1 Our contributions

This chapter introduces the commitment consistent encryption primitive that is used throughout this thesis. This primitive is de facto an encryption scheme that can be made homomorphic if needed. We explain the mechanism that allows us to enforce the consistency of the commitment part and the encryption part of the CCE. This mechanism that we call validity augmentation does not go against the homomorphic property of the scheme. Moreover it allows us to build NM-CPA secure CCE scheme.

We provide a generic and two concrete implementations of the CCE scheme in this chapter. The generic construction is made from a commitment scheme and an encryption scheme such that the opening and the message space of the commitment scheme correspond to the mes-

sage space of the encryption scheme. The first concrete implementation is the combination of a Pedersen commitment scheme and a Paillier encryption scheme. While this construction based on well-known tools is easy to set up, it also has some drawbacks. Its threshold version, possibly required for a multi-party setting, needs a tedious key generation algorithm and its efficiency is far behind of what we achieve in the second concrete construction. This is due to the RSA sized parameters of security used. The second construction, however, relies on ElGamal encryption in prime order groups on elliptic curves which at the same level of security are 16 times smaller.

Finally, we implemented a prototype in Python to test and compare our CCE instances. The code has been written in a friendly-user way to allow the reproduction of our results as well as the use of our libraries that implement all the arithmetic needed to perform operations on elliptic curves and on extension fields. The prototype also allowed us to implement three test applications of secure multi-party function evaluation that are described in Chapter 5. In this regards, the code available online at [Cuv15] is meant to be used to create other secure multi-party function evaluation that provide a perfectly private audit trail.

Organisation of the chapter. In Section 3.2, we introduce the CCE primitive. In Section 3.3, we provide generic instantiation of CCE while Section 3.4 present an optimal construction called PPATS that is used in both Chapter 4 and 5. Finally, in Section 3.5, we detail the parameter choice and the implementation performed in Python for the PPATS.

3.2 Commitment consistent encryption

A CCE primitive is a traditional public key encryption scheme that offers an extra feature: from any CCE ciphertext, it is possible to derive a commitment on the encrypted message, and the private key can also be used to obtain an opening on that commitment. We expect the clients to CC encrypt their private inputs, which will allow the worker to compute the output of the result of the function over the private inputs (e.g., by decrypting the homomorphic sum of the ciphertexts if the function is a sum like it may be the case in electronic voting). Furthermore, when receiving a CC ciphertext, the workers can use a DerivCom algorithm to derive commitments from CC ciphertexts and post that commitment on a public bulletin board **PB**. This provides a PPAT if the commitments are perfectly hiding. In order to offer universal verifiability, the workers can also make use of an Open algorithm that makes it possible to derive

openings of commitments on the output of the function.

For simplicity, we make our whole treatment in the single-key setting. The extension to the full threshold setting can be made using traditional techniques as sketched in Section 2.2.

Definition 3.1 (CC Encryption)**.** *A* ***commitment consistent encryption scheme*** Π *is a tuple of efficient algorithms* (Gen, Enc, Dec, DerivCom, Open, Verify) *and the description of a key space* $\mathbf{K}$ *(as* $\mathbf{K}_{\mathsf{pp}} \times \mathbf{K}_{\mathsf{pk}} \times \mathbf{K}_{\mathsf{sk}}$*) such that:*

$\mathsf{Gen}(1^\lambda)$**:** *Given a security parameter* λ*, output a triple* $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \in \mathbf{K}$*, respectively the public parameters, the public key and the secret key. The public parameters are implicitly given to every other algorithms and contain the description of the message space* $\mathbf{M}$*, the ciphertext space* $\mathbf{C}$*, the commitment space* $\mathbf{C}_{\mathsf{C}}$ *and the opening space* $\mathbf{O}$*.*

$\mathsf{Enc}(\mathsf{pk}, m)$**:** *Output a ciphertext* $c \in \mathbf{C}$ *which is an encryption using the public key* pk *of a message* $m \in \mathbf{M}$*.*

$\mathsf{Dec}(\mathsf{sk}, c)$**:** *From a ciphertext* $c \in \mathbf{C}$*, output a message* $\bar{m} \in \mathbf{M}$ *using the secret key* sk*.*

$\mathsf{DerivCom}(\mathsf{pk}, c)$**:** *From a ciphertext* $c \in \mathbf{C}$*, output a commitment* $d \in \mathbf{C}_{\mathsf{C}}$ *using the public key* pk*.*

$\mathsf{Open}(\mathsf{sk}, c)$**:** *From a ciphertext* $c \in \mathbf{C}$*, output an opening value* $\bar{o} \in \mathbf{O}$ *using the secret key* sk*.*

$\mathsf{Verify}(\mathsf{pk}, d, o, m)$**:** *From a message* $m \in \mathbf{M}$*, a commitment* $d \in \mathbf{C}_{\mathsf{C}}$ *with respect to key* pk *and an opening value* $o \in \mathbf{O}$*, output a bit. This algorithm checks the validity of the opening* (m, o) *with respect to* d *and* pk*.*

Dec, DerivCom, Open *and* Verify *are deterministic while* Gen *and* Enc *are probabilistic.*

We expect CCE schemes to satisfy the following correctness properties.

Correctness: $\forall(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$, $\forall m \in \mathbf{M}$, $\forall c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$, it holds with overwhelming probability in λ that $\mathsf{Dec}(\mathsf{sk}, c) = m$ and that

$$\mathsf{Verify}(\mathsf{pk}, \mathsf{DerivCom}(\mathsf{pk}, c), \mathsf{Open}(\mathsf{sk}, c), \mathsf{Dec}(\mathsf{sk}, c)) = 1.$$

The security properties that we can expect from a CCE scheme and for the derived commitments are the traditional ones and we will discuss later the ones that are appropriate for our applications.

As a first example of CCE motivating our further developments, consider the following ElGamal based CC encryption scheme:

$\mathsf{Gen}(1^\lambda)$**:** Output a triple $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \in \mathbf{K}$ where pp is the description of a group G of prime order q such that $\mathbf{M} := \mathbb{Z}_q$, $\mathbf{C} = \mathbf{O} := G^2 \times \mathbb{Z}_q^2$, and $\mathbf{C}_\mathsf{C} := G$. The public key pk consists of two generators g and h of G as well as an efficient hash function $\mathcal{H} : \mathbf{K}_\mathsf{pp} \times G^4 \to \mathbb{Z}_q$ while the secret key sk is α where $h := g^\alpha$.

$\mathsf{Enc}(\mathsf{pk}, m)$**:** For a message $m \in \mathbf{M}$, select randomly $r, r' \in \mathbb{Z}_q$, compute $c_0 := g^r$, $d = g^m h^r$, $\mathfrak{a}_\mathsf{cc} := (g^{r'}, h^{r'})$, $\mathfrak{e}_\mathsf{cc} := \mathcal{H}(\mathsf{pp}, c_0, h^r, \mathfrak{a}_\mathsf{cc})$ and $\mathfrak{z}_\mathsf{cc} := r' + \mathfrak{e}_\mathsf{cc} r \mod q$ and output $c = (c_0, d, \mathfrak{e}_\mathsf{cc}, \mathfrak{z}_\mathsf{cc}) \in \mathbf{C}$.

$\mathsf{Dec}(\mathsf{sk}, c)$**:** Parse c as $(c_0, d, \mathfrak{e}_\mathsf{cc}, \mathfrak{z}_\mathsf{cc})$ and output $\bar{m} \in \mathbf{M}$ as $\mathsf{Dlog}_g d/c_0^\alpha$.

$\mathsf{DerivCom}(\mathsf{pk}, c)$**:** Parse c as $(c_0, d, \mathfrak{e}_\mathsf{cc}, \mathfrak{z}_\mathsf{cc})$ and output $d \in \mathbf{C}_\mathsf{C}$.

$\mathsf{Open}(\mathsf{sk}, c)$**:** Parse c as $(c_0, d, \mathfrak{e}_\mathsf{cc}, \mathfrak{z}_\mathsf{cc})$ and output $\bar{o} = (c_0, o_0, \mathfrak{e}_\mathsf{cc}, \mathfrak{z}_\mathsf{cc}) \in \mathbf{O}$ where $o_0 = c_0^\alpha$.

$\mathsf{Verify}(\mathsf{pk}, d, o, m)$**:** Parse o as $(c_0, o_0, \mathfrak{e}_\mathsf{cc}, \mathfrak{z}_\mathsf{cc})$ above and test if

$$\mathfrak{e}_\mathsf{cc} \stackrel{?}{=} \mathcal{H}(\mathsf{pp}, c_0, o_0, g^{\mathfrak{z}_\mathsf{cc}} c_0^{-\mathfrak{e}_\mathsf{cc}}, h^{\mathfrak{z}_\mathsf{cc}} o_0^{-\mathfrak{e}_\mathsf{cc}}).$$

This encryption scheme produces traditional ElGamal ciphertexts together with a Zero-Knowledge Proof of Knowledge (ZKPK) of the randomness and the plaintext. This proof is called a **proof of consistency** or a **validity proof** and denoted π_cc. The second element of the ElGamal ciphertext can be seen as a Pedersen commitment, which is made binding thanks to the ZKPK. While offering a particularly simple verifiable decryption procedure (only o_0 needs to be computed), this scheme has two major limitations for the applications we have in mind: it is not homomorphic, and the validity of ciphertexts cannot be determined without decryption. For example, in the context of cryptographic voting, the first limitation prevents us from using traditional tallying procedures, while the second could make a tallying procedure fail if encrypted votes happen to be invalid.

We address these two concerns by introducing the notion of Validity Augmentation (VA) for CCE schemes. A VA guarantees the validity of CCE ciphertext. This makes sure that the workers are able to produce an output from the ciphertexts they receive.

Contrary to CCE ciphertexts, our VA ciphertexts do not need to be homomorphic: as soon as the workers are convinced of the validity of a VA ciphertext, they can strip it and recover an homomorphic CCE ciphertext for further operations.

From an operational point of view, a validity augmentation of a CCE scheme adds three algorithms: $\mathsf{Expand}, \mathsf{Strip}$ and Valid. Expand augments the public key for the needs of the other algorithms. Valid takes an augmented CCE ciphertext c^{VA} that contains a CCE ciphertext along with some proofs of validity, and runs a verification procedure on those proofs to make sure that it is possible to extract from the ciphertext a commitment and an encryption of an opening for that commitment. This is the crucial part to convince the workers that they will indeed be able to produce an output. Eventually, Strip removes those proofs to provide some homomorphic properties such as additivity on the encrypted messages.

Definition 3.2 (Validity augmentation of CCE)**.** *Given a* CCE *scheme* $\Pi := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{DerivCom}, \mathsf{Open}, \mathsf{Verify})$*, we say that scheme* $\Pi^{\mathsf{VA}} := (\mathsf{Gen}^{\mathsf{VA}}, \mathsf{Enc}^{\mathsf{VA}}, \mathsf{Dec}^{\mathsf{VA}}, \mathsf{DerivCom}^{\mathsf{VA}}, \mathsf{Open}^{\mathsf{VA}}, \mathsf{Verify}^{\mathsf{VA}}, \mathsf{Expand}, \mathsf{Strip}, \mathsf{Valid})$ *is a* ***validity augmentation of the*** CCE ***scheme*** Π *if* Π^{VA} *is a* CCE *scheme equipped with three additional efficient algorithms* Expand*,* Strip *and* Valid *that satisfy the following conditions.*

Augmentation: $\mathsf{Gen}^{\mathsf{VA}}$ *runs* Gen *to get* $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk})$ *and outputs an updated triple* $(\mathsf{pp}^{\mathsf{VA}}, \mathsf{pk}^{\mathsf{VA}}, \mathsf{sk}^{\mathsf{VA}}) := (\mathsf{pp}, \mathsf{Expand}(\mathsf{pk}), \mathsf{sk})$*.*

Validity: *we have* $\forall (\mathsf{pp}^{\mathsf{VA}}, \mathsf{pk}^{\mathsf{VA}}, \mathsf{sk}^{\mathsf{VA}}) \leftarrow \mathsf{Gen}^{\mathsf{VA}}(1^\lambda)$*,* $\forall m \in \mathbf{M}$*,* $\forall c^{\mathsf{VA}} \leftarrow \mathsf{Enc}^{\mathsf{VA}}(\mathsf{pk}^{\mathsf{VA}}, m)$*, that* $\mathsf{Valid}(\mathsf{pk}^{\mathsf{VA}}, c^{\mathsf{VA}}) = 1$*. Moreover, for every honestly generated ciphertext and keys and, for any* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* η *such that the probability for* $\mathcal{A}$ *to win the following game is bounded by* η*:*

1. *run* $\mathsf{Gen}^{\mathsf{VA}}(1^\lambda)$ *to get* $(\mathsf{pp}^{\mathsf{VA}}, \mathsf{pk}^{\mathsf{VA}}, \mathsf{sk}^{\mathsf{VA}})$ *and send* $(\mathsf{pp}^{\mathsf{VA}}, \mathsf{pk}^{\mathsf{VA}})$ *to* $\mathcal{A}$*.*
2. $\mathcal{A}$ *generates* c^{VA}*.*
3. *compute* $d \leftarrow \mathsf{DerivCom}^{\mathsf{VA}}(\mathsf{pk}^{\mathsf{VA}}, c^{\mathsf{VA}})$*,* $o \leftarrow \mathsf{Open}^{\mathsf{VA}}(\mathsf{sk}^{\mathsf{VA}}, c^{\mathsf{VA}})$ *and* $m \leftarrow \mathsf{Dec}^{\mathsf{VA}}(\mathsf{sk}^{\mathsf{VA}}, c^{\mathsf{VA}})$ *as well as* $\beta \leftarrow \mathsf{Valid}(\mathsf{pk}^{\mathsf{VA}}, c^{\mathsf{VA}})$ *and* $\beta' \leftarrow \mathsf{Verify}(\mathsf{pk}, d, o, m)$*. Output* β *and* β'*.*

We have that $\Pr[\beta = 1 \wedge \neg\beta' = 1] \leq \eta(\lambda)$*. This condition guarantees that decryption and opening succeed.*

Consistency. $\forall m \in \mathbf{M}$*, the distributions of* $\mathsf{Strip}(\mathsf{pk}^{\mathsf{VA}}, \mathsf{Enc}^{\mathsf{VA}}(\mathsf{pk}^{\mathsf{VA}}, m))$ *and* $\mathsf{Enc}(\mathsf{pk}, m)$ *are the same, that is, we can strip a* VA *ciphertext into a normal one. Furthermore, the decryption, opening and*

commitment derivation of Π^{VA} *are consistent with those of* Π*: for every ciphertext and generated keys, it must hold that*

$$\begin{aligned}\mathsf{Dec}^{\mathsf{VA}}(\mathsf{sk}^{\mathsf{VA}}, c^{\mathsf{VA}}) &= \mathsf{Dec}(\mathsf{sk}, \mathsf{Strip}(\mathsf{pk}^{\mathsf{VA}}, c^{\mathsf{VA}})),\\ \mathsf{Open}^{\mathsf{VA}}(\mathsf{sk}^{\mathsf{VA}}, c^{\mathsf{VA}}) &= \mathsf{Open}(\mathsf{sk}, \mathsf{Strip}(\mathsf{pk}^{\mathsf{VA}}, c^{\mathsf{VA}})),\\ \mathsf{DerivCom}^{\mathsf{VA}}(\mathsf{pk}^{\mathsf{VA}}, c^{\mathsf{VA}}) &= \mathsf{DerivCom}(\mathsf{pk}, \mathsf{Strip}(\mathsf{pk}^{\mathsf{VA}}, c^{\mathsf{VA}}))).\end{aligned}$$

The algorithm Expand *is probabilistic while* Strip *and* Valid *are deterministic.*

We refer to the result of the augmentation of a CCE scheme as a CCVA encryption scheme or simply a CCVAE scheme.

According to this definition, validity augmented schemes are validity augmented for themselves. While this is not a problem, it will not be useful since the purpose of the augmentation is to provide a validity verification mechanism while not being required to preserve the homomorphic properties of the basic scheme.

3.3 Generic commitment consistent encryption scheme

In this section, we propose a generic construction of a CCVAE scheme as well as a simple instance of this construction based on a combination of the Paillier cryptosystem (Definition 2.7 [Pai99]) and a Pedersen commitment (Definition 2.11 [Ped92]). This natural instantiation leads to a fairly inefficient and complex scheme, but provides the main ingredients that we will use next. Note that the generic construction is suitable for voting with PPAT (see Section 4.3).

3.3.1 Construction from standard building blocks

Our generic construction of a CCE scheme is based on a traditional homomorphic commitment scheme and on two homomorphic encryption schemes for encrypting the committed message and the opening value.

We do not discuss here the way the function evaluation is performed and we delay this aspect to the specific applications of Chapter 4 and 5.

The $\mathsf{Com} + \mathsf{Enc}_1 + \mathsf{Enc}_2$ construction

Let $\Pi_{\mathsf{C}} = (\mathsf{Gen}_{\mathsf{C}}, \mathsf{Com}, \mathsf{Verify})$ be a commitment scheme and $\Pi_{\mathsf{E}_i} = (\mathsf{Gen}_i, \mathsf{Enc}_i, \mathsf{Dec}_i)$ be an encryption scheme for $i = 1, 2$. We assume the following mild condition for the generation algorithms: $\mathsf{Gen}_{\mathsf{C}}, \mathsf{Gen}_1$

and Gen_2 can be run on a common input returned by a probabilistic algorithm Setup such that we will get $\mathbf{M}_1 \subset \mathbf{M}_\mathsf{C}$ and $\mathbf{O} \subset \mathbf{M}_2$ where $\mathbf{M}_i$ is the message space of Π_{E_i}. Then we define the following generic CCE scheme $\Pi_\mathsf{G} = (\mathsf{Gen}_\mathsf{G}, \mathsf{Enc}_\mathsf{G}, \mathsf{Dec}_\mathsf{G}, \mathsf{DerivCom}_\mathsf{G}, \mathsf{Open}_\mathsf{G}, \mathsf{Verify}_\mathsf{G})$:

$\mathsf{Gen}_\mathsf{G}(1^\lambda)$**:** Run $\mathsf{Setup}(1^\lambda)$ to get a common public parameter pp and compute

$$\begin{aligned} \mathsf{cpk} &\leftarrow \mathsf{Gen}_\mathsf{C}(\mathsf{pp}), \\ (\mathsf{pk}_1, \mathsf{sk}_1) &\leftarrow \mathsf{Gen}_1(\mathsf{pp}), \\ (\mathsf{pk}_2, \mathsf{sk}_2) &\leftarrow \mathsf{Gen}_2(\mathsf{pp}) \end{aligned}$$

Output $\mathsf{pk} = (\mathsf{cpk}, \mathsf{pk}_1, \mathsf{pk}_2)$ and $\mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2)$ with the above specification.

$\mathsf{Enc}_\mathsf{G}(\mathsf{pk}, m)$**:** Parse pk as $(\mathsf{cpk}, \mathsf{pk}_1, \mathsf{pk}_2)$. For $m \in \mathbf{M} \subset \mathbf{M}_1$ compute $(d, o) \leftarrow \mathsf{Com}(\mathsf{cpk}, m)$, $c_1 \leftarrow \mathsf{Enc}_1(\mathsf{pk}_1, m)$ and $c_2 \leftarrow \mathsf{Enc}_2(\mathsf{pk}_2, o)$. Output the ciphertext $c = (d, c_1, c_2)$.

$\mathsf{Dec}_\mathsf{G}(\mathsf{sk}, c)$**:** Parse sk as $(\mathsf{sk}_1, \mathsf{sk}_2)$ and c as (d, c_1, c_2) and return $\perp$ if it has not the right form. Otherwise return the plaintext $\bar{m} := \mathsf{Dec}_1(\mathsf{sk}_1, c_1)$.

$\mathsf{DerivCom}_\mathsf{G}(\mathsf{pk}, c)$**:** Parse pk as $(\mathsf{cpk}, \mathsf{pk}_1, \mathsf{pk}_2)$ and c as (d, c_1, c_2) and return $\perp$ if it has not the right form. Otherwise return d.

$\mathsf{Open}_\mathsf{G}(\mathsf{sk}, c)$**:** Parse sk as $(\mathsf{sk}_1, \mathsf{sk}_2)$ and c as (d, c_1, c_2) and return $\perp$ if it has not the right form. Otherwise return the value $\bar{o} := \mathsf{Dec}_2(\mathsf{sk}_2, c_2)$.

$\mathsf{Verify}_\mathsf{G}(\mathsf{pk}, d, o, m)$**:** Parse pk as $(\mathsf{cpk}, \mathsf{pk}_1, \mathsf{pk}_2)$ and output $\mathsf{Verify}(\mathsf{cpk}, d, o, m)$.

The following theorem shows that schemes built according to this approach, using secure components, keep the same level of security.

Theorem 3.1. *Let* Π_C *be a computationally hiding commitment scheme and* Π_{E_1}*,* Π_{E_2} *be two* IND-CPA *secure encryption schemes which together support the* $\mathsf{Com} + \mathsf{Enc_1} + \mathsf{Enc_2}$ *construction. Then, the resulting scheme* Π_G *consists in an* IND-CPA *secure* CCE *scheme.*

Proof. First, the CCE correctness follows immediately from the correctness of the underlying components. Now, let us focus on the security property. Let $\mathcal{A}$ be an adversary against Π_G. From $\mathcal{A}$ we build an adversary $\mathcal{B}$ against at least one of the underlying schemes Π_C, Π_{E_1} or

Π_{E_2}, which succeeds with a closely related success probability. For convenience we denote Π_{C} by Π_0, Π_{E_1} by Π_1 and Π_{E_2} by Π_2. Given any message $m \in \mathbf{M}$, we consider the following distributions.

$\mathcal{D}_3$ is the distribution $(d, c_1, c_2) \leftarrow \mathsf{Enc}_{\mathsf{G}}(m)$ of the encryptions of m in Π_{G}.

$\mathcal{D}_2$ Defined as $\mathcal{D}_3$ except for the generation of c_2. Instead it computes $c_2^* \leftarrow \mathsf{Enc}_2(o^*)$ for a random opening value o^*.

$\mathcal{D}_1$ Same as $\mathcal{D}_2$ except that c_1 is replaced by $c_1^* \leftarrow \mathsf{Enc}_1(m^*)$ for a random m^*.

$\mathcal{D}_0$ Same as $\mathcal{D}_1$ except that d is replaced by d^* computed with an independent random value.

Remark that $\mathcal{D}_0$ is the uniform random distribution on $\mathbf{C}_{\mathsf{C}} \times \mathbf{C}_1 \times \mathbf{C}_2$ which is not the uniform random distribution on $\mathbf{C}_{\mathsf{G}}$.

In the CPA experiment, $\mathcal{A}$ outputs (m_0, m_1) and must distinguish $\mathcal{D}_3(m_0)$ from $\mathcal{D}_3(m_1)$. We define

$$\mathsf{Adv}_{\mathcal{A}}^{\Pi_{\mathsf{G}}}(\lambda) := |\Pr[1 \leftarrow \mathcal{A}(\mathcal{D}_3(m_0))] - \Pr[1 \leftarrow \mathcal{A}(\mathcal{D}_3(m_1))]|$$

and we set $\Pr_{b,k} = \Pr[1 \leftarrow \mathcal{A}(e) | e \leftarrow \mathcal{D}_k(m_b)]$.

We bound $\mathsf{Adv}_{\mathcal{A}}^{\Pi_{\mathsf{G}}}(\lambda)$ by a linear combination of $\mathsf{Adv}_{\mathcal{B}}^{\Pi_k}(\lambda)$. Note that the CPA experiment for $\mathcal{B}$ is defined for a slightly different but equivalent experiment: $\mathcal{B}$ has to distinguish whether an output in $\mathbf{C}_{\mathsf{C}}, \mathbf{C}_1$ or $\mathbf{C}_2$ is computed from a chosen message or from a uniformly distributed message in $\mathbf{M}_{\mathsf{C}}, \mathbf{M}_1$ or $\mathbf{M}_2$.

Let us see how $\mathcal{B}$ works. $\mathcal{B}$ runs $\mathsf{Gen}(1^\lambda)$ generating instances for Π_0, Π_1 and Π_2. On these inputs, $\mathcal{B}$ runs $\mathcal{A}$ and receives (m_0, m_1). $\mathcal{B}$ flips two coins $b \leftarrow 0, 1$ and $k \leftarrow 0, 1, 2$. Depending on b and k, $\mathcal{B}$ follows one of the next patterns.

$k = 0$**:** $\mathcal{B}$ computes c_1^* and c_2^* identically to $\mathcal{D}_0(m_b)$ and $\mathcal{D}_1(m_b)$. Then $\mathcal{B}$ queries the Π_0 oracle $\mathcal{O}_0$ on m_b and receives $\tilde{d}$. $\mathcal{O}_0$ computes $\tilde{d}$ as $\mathsf{Com}(m_b)$ or as $\mathsf{Com}(m^*)$ by tossing a coin. Finally $\mathcal{B}$ sets $c = (\tilde{d}, c_1^*, c_2^*)$.

$k = 1$**:** $\mathcal{B}$ computes d and c_2^* identically to $\mathcal{D}_1(m_b)$ and $\mathcal{D}_2(m_b)$. Then $\mathcal{B}$ queries the Π_1 oracle $\mathcal{O}_1$ on m_b and receives $\tilde{c}_1$. $\mathcal{O}_1$ computes $\tilde{c}_1$ as $\mathsf{Enc}_1(m_b)$ or as $\mathsf{Enc}_1(m^*)$ by tossing a coin. Finally $\mathcal{B}$ sets $c = (d, \tilde{c}_1, c_2^*)$.

$k = 2$**:** $\mathcal{B}$ computes d and c_1 identically to $\mathcal{D}_2(m_b)$ and $\mathcal{D}_3(m_b)$. Then $\mathcal{B}$ queries the Π_2 oracle $\mathcal{O}_2$ on the opening value o given by the computation of the commitment and receives $\tilde{c}_2$. $\mathcal{O}_2$ computes $\tilde{c}_2$ as $\mathsf{Enc}_2(o)$ or as $\mathsf{Enc}_2(o^*)$ by tossing a coin and sends $\tilde{c}_2$ to $\mathcal{B}$. Finally $\mathcal{B}$ sets $c = (d, c_1, \tilde{c}_2)$.

In all these cases, $\mathcal{B}$ sends c to $\mathcal{A}$. $\mathcal{B}$ outputs the guess of $\mathcal{A}$. It is clear that $\mathcal{B}$ runs in polynomial time if $\mathcal{A}$ does. Using the triangular inequality, from $\mathsf{Adv}^{\Pi_\mathsf{G}}_{\mathcal{A}}(\lambda) = |\mathrm{Pr}_{0,3}(\lambda) - \mathrm{Pr}_{1,3}(\lambda)|$ and $\mathrm{Pr}_{0,0}(\lambda) = \mathrm{Pr}_{1,0}(\lambda)$, we have:

$$\begin{aligned}
\mathsf{Adv}^{\Pi_\mathsf{G}}_{\mathcal{A}}(\lambda) &\leq \sum_{k=0}^{2}\sum_{b=0}^{1} |\mathrm{Pr}_{b,k}(\lambda) - \mathrm{Pr}_{b,k+1}(\lambda)| \\
&\leq \sum_{k=0}^{2} |\mathrm{Pr}_{0,k+1}(\lambda) - \mathrm{Pr}_{0,k}(\lambda)| + |\mathrm{Pr}_{0,0}(\lambda) - \mathrm{Pr}_{1,0}(\lambda)| \\
&\quad + \sum_{k=0}^{2} |\mathrm{Pr}_{1,k}(\lambda) - \mathrm{Pr}_{1,k+1}(\lambda)| \\
&\leq \frac{1}{6}\sum_{k=0}^{2}\sum_{b=0}^{1} \mathsf{Adv}^{\Pi_k}_{\mathcal{B}}(\lambda) \\
&\leq \eta(\lambda)
\end{aligned}$$

where η is a negligible function. □

Validity augmentation

If the commitment and encryption schemes used in the construction above are compatible with Σ-protocols (see Section 2.1.5), then it is easy to augment this IND-CPA secure CCE scheme into an NM-CPA secure CCVAE scheme.

Theorem 3.2 ([BPW12], informal)**.** *Let* Π_E *be an* IND-CPA *secure encryption scheme and* π *be a* Σ*-protocol in the* NP*-language* $\mathcal{L}_\mathsf{NP}$ *for the relation* $R := \{((\mathsf{pp}, \mathsf{pk}, c), m) | (\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda), c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)\}$ *with special soundness, special honest verifier zero-knowledge, unique responses and a challenge space* $\{0,1\}^t$ *where the inverse of* t *is negligible in* λ*. Let* $\mathcal{H}$ *be a random oracle* $\{0,1\}^* \to \{0,1\}^t$*. Then the following scheme is* NM-CPA *secure:*

- *The key generation algorithm is the one of* Π_E *augmented with* $\mathcal{H}$*.*
- *In order to encrypt* m*, run the encryption algorithm of* Π_E*, then compute the corresponding proof of knowledge made non-interactive using the Fiat-Shamir transform (Theorem 2.1).*

- *In order to decrypt a ciphertext, check the proof and in case of validity run the decryption algorithm of* Π_{E} *on the ciphertext part, or return* $\perp$ *otherwise.*

This Σ-protocol immediately provides a validity augmentation. More precisely, in the case of our Π_{G} construction, we obtain a proof in $\mathcal{L}_{\mathsf{NP}}$ for the relation $R := \{((\mathsf{pp}, \mathsf{pk}, c), (m, o)) | c = (d, c_1, c_2), (d, o) \leftarrow \mathsf{Com}(\mathsf{pk}, m), c_1 \leftarrow \mathsf{Enc}_1(\mathsf{pk}, m), c_2 \leftarrow \mathsf{Enc}_2(o)$: such a proof not only guarantees the knowledge of the plaintext and opening, but also that the ciphertext is valid.

We can then define a validity augmentation in a straightforward way: Expand adds the oracle $\mathcal{H}$ to the public key, Strip removes the sigma proof from the ciphertext, and Valid returns "1" only if the proof is valid. The validity condition holds thanks to the completeness and the soundness of the proof.

The consistency of the augmentation is straightforward by inspection of Definition 3.2.

In the case of threshold CCE scheme, we observe that this upgrading makes the extension of threshold IND-CPA schemes to threshold NM-CPA schemes immediate, and prevents common difficulties from happening in threshold versions of non-malleable encryption schemes for which the ciphertext validity tests require the knowledge of the private key (see, e.g.,[CG99]).

3.3.2 Instance based on Pedersen and Paillier

The PPATP scheme.

The Π_{G} construction can be instantiated using a combination of Paillier and Pedersen commitments, inspired from a proposal by Moran and Naor [MN10]. This leads to a scheme that we call PPATP: the idea is to select a traditional Paillier modulus N^2, and then to perform the Pedersen commitments in a subgroup of the quadratic residues modulo a prime $P = 2kN + 1$, whose order equals N. The small public odd k-value is not fixed to facilitate the generation of P.

Protocol 3.1 (PPATP CCE scheme)**.**

$\mathsf{Gen}_{\mathsf{P}}(1^{\lambda})$**:** *Choose two* λ*-bit safe primes,* $p = 2p' + 1$ *and* $q = 2q' + 1$*, and compute the public modulus* $N = pq$ *for the Paillier encryption. The corresponding secret key* sk_{P} *is the number* $l := \mathrm{lcm}(\varphi(p), \varphi(q)) = 2p'q' = \varphi(N)/2$*. Then pick a prime* $P = 2kN+1$ *and two public generators* g *and* h *of a* N*-order subgroup of the quadratic residues* $\mathcal{QR}_P$ *for the Pedersen commitment. Output*

$\mathsf{pp_P} = (P, N)$, $\mathsf{pk_P} = (g, h)$ *and* $\mathsf{sk_P} = l$. *Consequently, we have that* $\mathbf{M} = \mathbf{O} := \mathbb{Z}_N$, $\mathbf{C_C} := \langle g \rangle$ *and* $\mathbf{C} := \mathbf{C_C} \times (\mathbb{Z}^*_{N^2})^2$.

$\mathsf{Enc_P}(\mathsf{pk_P}, m)$**:** *For* $m \in \mathbb{Z}_N$, *choose* $r \xleftarrow{rand} \mathbb{Z}_N$ *and* $s, t \xleftarrow{rand} \mathbb{Z}^*_N$, *compute* $d = g^m h^r$ *in* $\mathbb{Z}^*_P$ *and,* $c_1 = (N+1)^m s^N$ *and* $c_2 = (N+1)^r t^N$ *in* $\mathbb{Z}^*_{N^2}$. *Output the ciphertext* $c = (d, c_1, c_2)$.

$\mathsf{Dec_P}(\mathsf{sk_P}, c)$**:** *Parse* c *as* (d, c_1, c_2) *and compute* $m_0 := ((c_1^l \mod N^2) - 1)/N$. *Output* $\bar{m} := m_0(l^{-1} \mod N) \in \mathbb{Z}_N$ *as in the Paillier decryption (Definition 2.7).*

$\mathsf{DerivCom_P}(\mathsf{pk_P}, c)$**:** *Parse* c *as* (d, c_1, c_2) *and output* d.

$\mathsf{Open_P}(\mathsf{sk_P}, c)$**:** *Parse* c *as* (d, c_1, c_2) *and compute* $o_0 := ((c_2^l \mod N^2) - 1)/N$. *Output* $\bar{o} := o_0(l^{-1} \mod N) \in \mathbb{Z}_N$ *as in the Paillier decryption.*

$\mathsf{Verify_P}(\mathsf{pk}, d, o, m)$**:** *Output* 1 *if* $d \stackrel{?}{=} g^m h^o \mod P$. *Otherwise output* 0.

Thanks to the careful choice of the parameters (the messages and opening values lie in $\mathbb{Z}_N$), the scheme Π_{P} is additively homomorphic, which makes it convenient to apply the generic transformation described above by using a simple non interactive proof of knowledge that enjoys all the properties we need for the validity augmentation.

Protocol 3.2 (The validity augmentation of PPATP)**.**

$\mathsf{Expand_P}(\mathsf{pp_P}, \mathsf{pk_P}, \mathsf{sk_P})$**:** *Augment the public key* $\mathsf{pk_P}$ *with the description of an efficient hash function* $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_N$ *and return* $(\mathsf{pp_P^{VA}}, \mathsf{pk_P^{VA}}, \mathsf{sk_P^{VA}})$.

$\mathsf{Enc_P^{VA}}(\mathsf{pk_P^{VA}}, m)$**:** *Compute* $c := \mathsf{Enc_P}(\mathsf{pk_P}, m)$ *using random values* $r \in \mathbb{Z}_N$, $s, t \in \mathbb{Z}^*_N$ *as above. Then select random* m', r' *in* $\mathbb{Z}_N$ *and* s', t' *in* $\mathbb{Z}^*_N$, *and compute* $\mathfrak{a}_{\mathsf{cc}} := \mathsf{Enc_P}(\mathsf{pk_P}, m')$ *using* r', s', t' *as random values. Compute* $\mathfrak{e}_{\mathsf{cc}} := \mathcal{H}(\mathsf{pp_P^{VA}}, \mathsf{pk_P^{VA}}, c, \mathfrak{a}_{\mathsf{cc}})$ *and then* $\mathfrak{z}_{\mathsf{cc}} := (\mathfrak{z}_m, \mathfrak{z}_r, \mathfrak{z}_s, \mathfrak{z}_t)$ *as* $(m' + \mathfrak{e}_{\mathsf{cc}} m, r' + \mathfrak{e}_{\mathsf{cc}} r, s' s^{\mathfrak{e}_{\mathsf{cc}}}, t' t^{\mathfrak{e}_{\mathsf{cc}}})$. *The ciphertext is made of* $c^{\mathsf{VA}} := (c, \mathfrak{e}_{\mathsf{cc}}, \mathfrak{z}_{\mathsf{cc}})$.

$\mathsf{Valid_P}(\mathsf{pk_P^{VA}}, c^{\mathsf{VA}})$**:** *Parse* c^{VA} *as* $(c, \mathfrak{e}_{\mathsf{cc}}, (\mathfrak{z}_m, \mathfrak{z}_r, \mathfrak{z}_s, \mathfrak{z}_t))$ *and check whether all the elements of the ciphertext are properly encoded. Compute* $d' := g^{\mathfrak{z}_m} h^{\mathfrak{z}_r} d^{-\mathfrak{e}_{\mathsf{cc}}}$, $c_1' := (N+1)^{\mathfrak{z}_m} \mathfrak{z}_s^N c_1^{-\mathfrak{e}_{\mathsf{cc}}}$ *and* $c_2' := (N+1)^{\mathfrak{z}_r} \mathfrak{z}_t^N c_2^{-\mathfrak{e}_{\mathsf{cc}}}$, *and test whether the following equality holds:* $\mathfrak{e}_{\mathsf{cc}} \stackrel{?}{=} \mathcal{H}(\mathsf{pp_P^{VA}}, \mathsf{pk_P^{VA}}, c, \mathfrak{a}_{\mathsf{cc}}')$ *for* $\mathfrak{a}_{\mathsf{cc}}' := (d', c_1', c_2')$. *If the verifications fail, output* $\perp$ *else output 1.*

$\mathsf{Strip}_{\mathsf{P}}(\mathsf{pk}_{\mathsf{P}}^{\mathsf{VA}}, c^{\mathsf{VA}})$**:** *Parse* c^{VA} *as* $(c, \mathfrak{e}_{\mathsf{cc}}, \mathfrak{z}_{\mathsf{cc}})$ *following the description of* $\mathsf{pk}_{\mathsf{P}}^{\mathsf{VA}}$*. If that fails, output* $\perp$*, and* c *otherwise.*

The other CCVA *algorithms are entirely determined by Definition 3.2.*

This instance of our generic construction is very simple. Unfortunately, it exhibits at least two important limitations for a practical use. First, being based on a Paillier cryptosystem, for a practical threshold version of the PPATP, the threshold key generation algorithm either requires the existence of a single trusted party that produces an RSA modulus N and forgets its factorization, or the use of fairly sophisticated multi-party computation protocol to generate this modulus in a distributed way [DJ01]. The first option is often challenging to setup in practice, while the second can seldom be feasible as it requires substantial expertise from the key holders.

As we will see in Section 3.5, the computational cost of this scheme stands in contrast with encryption schemes based on prime order groups: they enjoy considerably simpler key generation procedures [GJKR07], and typically enable much more efficient computation in the underlying groups, which can be of 256-bit instead of 2048-bit order for similar security levels.

As mentioned above, the PPATP scheme is a slight variant of a scheme suggested by Moran and Naor [MN10] and also used by Demirel et al. [DvdGA12]. Their version however relies on cut-and-choose techniques to prove the validity of ciphertexts and is therefore considerably less efficient.

3.4 Commitment consistent encryption for simple messages

The CCE instantiation presented here is a construction based on elliptic curves. It is efficient compared to the PPATP scheme based on the factorization problem for the same level of security. Accordingly, our implementation shows better timing results (see Table 3.3).

This instantiation is based on the TC2 perfectly hiding commitment scheme proposed by Abe et al. [AHO10] as well as on ElGamal encryption. It is scaled to encrypt small size message since its decryption algorithm rests on retrieving the discrete logarithm of a group element.

The mathematical materials involved in the following construction and the arithmetic deployed around them regards extension fields, elliptic curves and pairings. Since these notions might seem abstruse to the unfamiliar reader, we cover them in the memento of Appendix A.

Computational setting. The security of the PPATS relies on the hardness of the SXDH problem (Assumption 2.5) for the pairing-based constructions. This setting is noted $\mathcal{P}air_{\mathsf{SXDH}} := (q, G_1, G_2, G_3, e, P, Q)$ where $(G_1, +)$, $(G_2, +)$ and $(G_3, \cdot)$ are groups of order q, a λ-bit prime, e is an efficient and non-degenerating bilinear map $e : G_1 \times G_2 \to G_3$ and P, Q are generators of G_1 and G_2 respectively. These groups are such that there is no known efficient mapping between G_1 and G_2 in either direction.

Protocol 3.3 (The PPATS scheme)**.**

$\mathsf{Gen}_{\mathsf{S}}(1^\lambda)$**:** *The public parameters* pp *contain the description groups* G_1, G_2 *and* G_3*, the pairing* e *and generators* P, P_1 *for* G_1 *and* Q *for* G_2*. Note that* $\mathsf{Dlog}_P P_1$ *is unknown. Select* $x \xleftarrow{rand} \mathbb{Z}_q$ *and compute* $Q_1 = xQ$*. The public key is* $\mathsf{pk} := Q_1$ *and the private key* $\mathsf{sk} := x$*. Return* $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk})$*.*

$\mathsf{Enc}_{\mathsf{S}}(\mathsf{pk}, m)$**:** *Pick* $r, s \xleftarrow{rand} \mathbb{Z}_q$ *and compute* $d := rP + mP_1$*,* $c_1 := sQ$*, and* $c_2 := rQ + sQ_1$*. Return* $c := (c_1, c_2, d)$*.*

$\mathsf{Dec}_{\mathsf{S}}(\mathsf{sk}, c)$**:** *Parse* c *as* (c_1, c_2, d) *and compute* $\tilde{m} := e(P, xc_1 - c_2)e(d, Q)$*. Return* $\mathsf{Dlog}_{e(P_1,Q)}\tilde{m}$*.*

$\mathsf{DerivCom}_{\mathsf{S}}(\mathsf{pk}, c)$**:** *Parse* c *as* (c_1, c_2, d) *and return* d*.*

$\mathsf{Open}_{\mathsf{S}}(\mathsf{sk}, c)$**:** *Parse* c *as* (c_1, c_2, d)*, compute and return* $o := c_2 - xc_1$*.*

$\mathsf{Verify}_{\mathsf{S}}(\mathsf{pk}, d, o, m)$**:** *Verify the equality* $e(P, o) \stackrel{?}{=} e(d - mP_1, Q)$*. If it holds return* 1*, otherwise return* 0*.*

We delay to Section 4.4.1 the VA augmentation of the PPATS and the proof that the augmented scheme offers NM-CPA security.

3.5 Implementing commitment consistent encryption

A prototype implementation of the PPATS (but also the PPATC of Section 4.4.2 and the test applications of Chapter 5) was performed in Python. This allowed us to test our CCE schemes "in the real world" but also to build a generic protocol that generates a perfectly private audit trail in the general case of multi-party function evaluation of Chapter 5. We put a lot of efforts to select fast algorithms and used precomputations when available to reduce the cost of the basic arithmetic, group and field operations. The choice of Python is motivated by the fact that

the code and the applications are meant to be as transposable as possible. Thus, prototyping new cryptographic applications in the vein of what is done in Chapter 5 is facilitated. Moreover, the PPAT technique developed for voting systems in Chapter 4 could easily be integrated to the Helios voting system [AdMP12] for the server side even though it would be necessary to implement the PPATS in JavaScript for the voters. Finally, the VIFF framework that we used to test our SMC applications in Chapter 6 is based on Python. Moreover since this work was achieved before the prototype implementation discussed here, it seemed natural to go on with Python. Nevertheless, we do not ignore that using an optimized language with dedicated hardware operations should bring a nice improvement to our timing results. We made all our code available online [Cuv15] for the reproduction of our results.

The implementation was made from scratch and thus provides all the materials needed for the arithmetic on finite field, on extension field and on elliptic curves (EC) as well as the algorithms for the pairings. This last part is the trickiest one since it is the heaviest operation required in the different schemes for decryption and verification. For that reason, lots of efforts and optimizations targeted the pairing operation resulting in $\sim 20ms$ per pairing, which is reasonable considering the high-level language used.

We refer to the notations and results given in Appendix A for the constructions below. We recall that ϕ is the Frobenius endomorphism, $\phi(x, y) \mapsto (x^p, y^p)$.

In order to implement the PPATS scheme we work with an optimized set of Barreto-Naerhig (BN-)curves [BN06, PSJNB11] which enables extension fields with nice properties (for more details on the motivations of this choice, we refer to Appendix A.3). We denote $\mathbb{F}_p$ the prime field where p is a λ-bit prime. $E_{\mathbb{F}_p}$ is the BN elliptic curve of equation $y^2 = x^3 + b$ where $b = c^4 + d^6$ is a parameter in $\mathbb{F}_p$.

In $E_{\mathbb{F}_p}$, we pick P a generator of prime order q where $q = |E_{\mathbb{F}_p}|$. The primes p and q are given by $p = p(u) = 36u^4 + 36u^3 + 24u^2 + 6u + 1$ and $q = q(u) = 36u^4 + 36u^3 + 18u^2 + 6u + 1$ for $u \in \mathbb{Z}$. The EC point P generates a group $G_1 := E_{\mathbb{F}_p}[q] = E_{\mathbb{F}_p}$ that will be used to compute the perfectly hiding commitments of the CCE scheme.

We also consider $E'_{\mathbb{F}_{p^2}}$ the twisted elliptic curve over the extension field $\mathbb{F}_{p^2}$ where $E' \equiv y^2 = x^3 + \bar{\xi}$ with $\xi := c^2 + d^3 i$. Indeed, the BN elliptic curve $E_{\mathbb{F}_{p^{12}}}$ admits a sextic twist $E'_{\mathbb{F}_{p^2}}$ which means that there exists a group homomorphism $\psi : E'_{\mathbb{F}_{p^2}} \leftarrow E_{\mathbb{F}_{p^{12}}}$. In other words, the twist allows us to represent points of $G_2 := E_{\mathbb{F}_{p^{12}}}[q] \cap \ker(\phi - p)$ originally in $E_{\mathbb{F}_{p^{12}}}$ in a group $G'_2 := E'_{\mathbb{F}_{p^2}}[q]$ over a smaller field, namely $\mathbb{F}_{p^2}$.

In particular, the homomorphism ψ is given by $\psi : (x, y) \mapsto (\mu^2 x, \mu^3 y)$ where μ is a root of $(Y^2 - \xi)$ in $\mathbb{F}_{p^{12}}$. In $E_{\mathbb{F}_{p^{12}}}$, we find the EC point Q, a generator of $G_2 \simeq G'_2 \subset E'_{\mathbb{F}_{p^2}}$, a prime order q group. For now on, we consider indistinctly a point in G_2 from its image point in G'_2 through the homomorphism ψ^{-1}. Obviously, we prefer computations in $E'_{\mathbb{F}_{p^2}}$ whenever possible. The group G_2 will be used to compute the encryption part of the CCE scheme.

The exponential ElGamal encryption part of the scheme in G_2 allows decryption for a small range of messages m (for example $m < 2^{16}$). We use the "comb2" algorithm of [HDNP11] for scalar multiplication of EC points when precomputation is available. When it is not however, we rely on optimized field and elliptic curve arithmetic operations [Joy08, FVV09]. We perform elliptic curve operations in the Jacobian coordinates and the algorithms used for the field operations are mentioned below. We also rely on several optimizations to compute pairings when necessary [FVV09, DSD07, BGDM+10, AKL+11, SBC+09] as explained below.

The extension field $\mathbb{F}_{p^2}$ is naturally constructed as $\mathbb{F}_p/(i^2+1)$. Then we build $E_{\mathbb{F}_{p^{12}}}$ in two steps. First, we tower $\mathbb{F}_{p^6}$ above $\mathbb{F}_{p^2}$ as $\mathbb{F}_{p^6} := \mathbb{F}_{p^2}/(X^3 - \xi)$. Second, we tower $\mathbb{F}_{p^{12}}$ above $\mathbb{F}_{p^6}$ as $\mathbb{F}_{p^{12}} := \mathbb{F}_{p^6}/(Y^2 - \xi)$. Note that ξ is neither a cubic root nor a square root in $\mathbb{F}_{p^2}$. To sum up, we have $\mathbb{F}_{p^{12}} : \mathbb{F}_{p^6} : \mathbb{F}_{p^2} : \mathbb{F}_p$. Note that the multiplication and squaring operations in each field are handled differently. For $\mathbb{F}_{p^2}$ and $\mathbb{F}_{p^{12}}$, it is the complex multiplication and squaring algorithm while for $\mathbb{F}_{p^6}$, we use the Karatsuba algorithm for the multiplication and the Chung-Hasan algorithm for the squaring.

This setting offers an optimal ate pairing if one considers $G_3 \subset \mathbb{F}^*_{p^{12}}$ as the subgroup of q-th roots of unity. This pairing was introduced in [HSV06] and optimized for BN-curves [DSD07, BGDM+10, Ver10, AKL+11]. We define the non degenerate bilinear ate pairing $e : G_1 \times G_2 \to G_3$. This pairing is used in the decryption algorithm and it is also used to provide consistency zero-knowledge proofs, π_{cc}. Refraining here from going into deeper details, in the next section, we describe Algorithm 3.1 that is used to compute the optimal ate pairing. Its complexity is linear in the length of parameter u.

3.5.1 Computational workload

In order to compare the different PPAT schemes, at equivalent security level, we estimate the cost of each operation in the different fields: $\mathbb{Z}^*_P$ (or equivalently $\mathbb{Z}_N$) and $\mathbb{Z}^*_{N^2}$ for the PPATP schemes (Section 3.3.2).

G_1 and G_2 for the PPATS and PPATC schemes (the PPATC construction is introduced in Section 4.4.2, for complex ballot voting). We associate a unit cost denoted `U` to the multiplication of two 256-bit integers and as a preliminary assumption, we assume that this cost grows quadratically with the length of the operands. We target a security level equivalent to 2048-bit RSA modulus N. Table 3.1 details the costs of each operation in terms of `U` in the underlying fields.

The elliptic curve points are represented in Jacobian coordinates. Thus, the addition (A) of two different EC points requires 12 multiplications (M) and 4 squarings (S) while the duplication (D) of one EC point requires $4M + 6S$. Regarding the scalar multiplication of an EC point, when precomputation is not available or when the point is not a fixed base, we use the "double and add" algorithm which requires $128A + 256D$ operations in average. However, when precomputation is possible, we use the comb2 algorithm with a window size of 10. In this case the complexity of the scalar multiplication algorithm for a multiplier of length 256-bit is $26A + 12D$. This method requires the computation and storage of 2^{10} points per fixed base which represents 98.304 kB for a point of G_1 and 196.608 kB for a point of G_2.

In the integer field, the algorithm used for modular exponentiation is the classic "square and multiply" which requires $1.5l$ multiplications in average where l is the length of the exponent. When precomputation is available and when the base is fixed, we use the comb2 algorithm as before with a window size of 10. The complexity of the algorithm is $205M + 102S$. In the case of an exponentiation modulus N, storing 2^{10} group elements per fixed base takes 262.144 kB while it rises to 524.288 kB when the modulus is N^2.

Regarding the complexity of the pairings, we report all the details in Section 3.5.1.

In Table 3.2, we compare the cost of the different algorithms of the CCE schemes devised in this thesis. They are put on equal footing in terms of security levels. We consider two cases, the encryption of a 0/1 message with the PPATP and the PPATS and the encryption of a 256-bit message with the PPATP and the PPATC.

Table 3.3 gives the different timing measured with our prototype implementation for the algorithms and proofs of the PPATP, PPATS and PPATC. The tests were performed on a standard laptop: Intel® Core i5-3320M CPU @ 2.60GHz×4 with 7.7 GB of RAM. We differentiate the cost for the client from the cost for the server in anticipation with the applications of Chapter 4 and 5 where the clients are respectively voters and participants to a multi-party function evaluation and where the server(s) is(are) respectively the election authority(ies) and the worker

Table 3.1: Costs of operations in the different PPAT schemes.
▷ `U` is the number of multiplications between two 256-bit integers.
▷ `M` in $\mathbb{Z}_P^*$ is the modular multiplication between two 2048-bit elements.
▷ $\texttt{E}_\texttt{p}$ and `E` in $\mathbb{Z}_P^*$ are the modular exponentiations where the base, the exponent and the modulus are 2048-bit long. $\texttt{E}_\texttt{p}$ designates the use of precomputed table for a fixed base.
▷ `M` in $\mathbb{Z}_{N^2}^*$ is the modular multiplication between two 4096-bit elements.
▷ $\texttt{E}_\texttt{p}$ and `E` in $\mathbb{Z}_{N^2}^*$ are the modular exponentiations where the exponent and the base are 2048-bit long while the modulus is 4096-bit long.
▷ `A` is the addition of two EC point in the underlying group.
▷ $\texttt{Sm}_\texttt{p}$ and `Sm` are the scalar multiplications of an EC point by a 256-bit integer in the underlying group. $\texttt{Sm}_\texttt{p}$ designates the use of precomputed table for a fixed base.
▷ `P` is the pairing from $G_1 \times G_2 \to G_3$ involving operations in these three groups.

		Operation	`U`	**Algorithm used**
PPATP	$\mathbb{Z}_P^*$	`M`	64	
		$\texttt{E}_\texttt{p}$	19,648	comb2
		`E`	196,608	square and multiply
	$\mathbb{Z}_{N^2}^*$	`M`	256	
		$\texttt{E}_\texttt{p}$	78,592	comb2
		`E`	786,432	square and multiply
PPATS & PPATC	G_1	`A`	16	
		$\texttt{Sm}_\texttt{p}$	592	comb2
		`Sm`	4,608	double and add
	G_2	`A`	64	
		$\texttt{Sm}_\texttt{p}$	2,366	comb2
		`Sm`	18,432	double and add
	$G_{1,2,3}$	`P`	47,522	Ate pairing

that evaluates the function. Tables 3.2 and 3.3 show important differences between PPATP, PPATS and PPATC: computing a PPATP ciphertext is theoretically 227 times more expensive than computing a PPATC ciphertext. However this difference is less evident in the timing measurements. One reason might be that the quadratic growth of the multiplication cost stated as a preliminary assumption is not accurate enough. A second reason might be that the way Python handles and generate objects comes with extra hidden features that slow the execution. Nevertheless, it is worth noticing that the extra work required to compute the pairings does not outweigh the benefits of the elliptic curve setting.

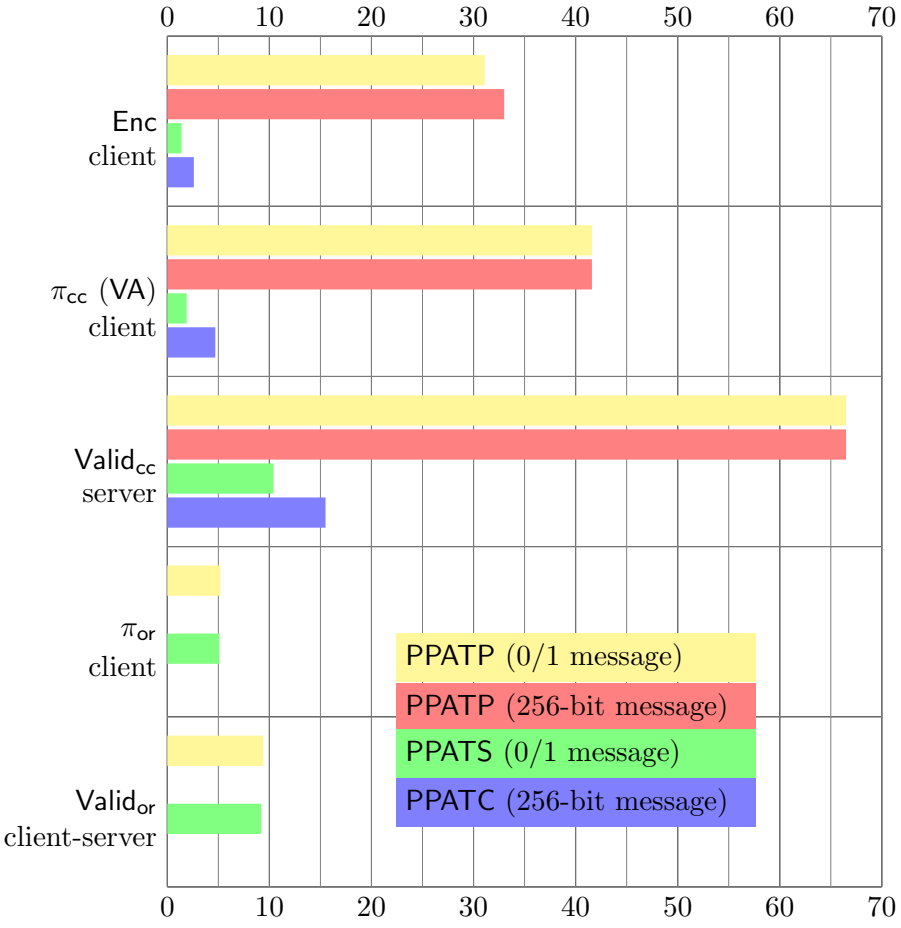
Table 3.2: Costs of the algorithms of two versions of PPATP and of the PPATS and PPATC.

		$\mathbb{Z}_P^*$			$\mathbb{Z}_{N^2}^*$			Total cost
		M	$\mathrm{E_p}$	E	M	$\mathrm{E_p}$	E	U
PPATP (0/1 vote)	Enc	1	1	0	2	1	2	1,671,680
	Verify	1	1	0	0	0	0	19,712
	π_{cc} (VA)	5	2	2	2	2	2	2,163,392
	π_{or}	2.5	2	1	0	0	0	236,064
	Dec	2	0	0	0	0	1	786,560
	Open	2	0	0	0	0	1	786,560
	$\mathsf{Valid}_{\mathsf{cc}}$	2	2	1	4	2	4	3,539,968
	$\mathsf{Valid}_{\mathsf{or}}$	3	2	2	0	0	0	432,704
PPATP (256-bit vote)	Enc	1	2	0	2	2	2	1,769,920
	Verify	1	2	0	0	0	0	39,360
	π_{cc} (VA)	5	2	2	2	2	2	2,163,392
	Dec	2	0	0	0	0	1	786,560
	Open	2	0	0	0	0	1	786,560
	$\mathsf{Valid}_{\mathsf{cc}}$	2	2	1	4	2	4	3,539,968

		G_1			G_2				Total cost
		A	$\mathrm{Sm_p}$	Sm	A	$\mathrm{Sm_p}$	Sm	P	U
PPATS (0/1 vote)	Enc	1	3	0	1	1	0	0	4,222
	Verify	0	0	0	1	0	0	2	95,108
	π_{cc} (VA)	1	3	0	1	2	0	0	6,588
	π_{or}	0	0	0	2	2	1	0	23,292
	Dec	1	0	1	0	0	0	2	99,668
	Open	1	0	1	0	0	0	0	4,624
	$\mathsf{Valid}_{\mathsf{cc}}$	3	3	2	2	2	1	0	17,743
	$\mathsf{Valid}_{\mathsf{or}}$	0	0	0	4	2	2	0	41,852
PPATC (256-bit vote)	Enc	2	5	0	1	2	0	0	7,788
	Verify	1	0	0	0	0	0	3	142,582
	π_{cc} (VA)	3	5	1	1	2	0	0	12,412
	Dec	1	0	1	0	0	0	0	4,624
	Open	1	0	1	0	0	0	0	4,624
	$\mathsf{Valid}_{\mathsf{cc}}$	6	5	4	2	2	1	0	44,780

Caption of Table 3.2 (previous page):
`M` is the number of modular multiplications.
$\texttt{E}_\texttt{p}$ is the number of modular exponentiations with precomputation available for a fixed base.
`E` is the number of modular exponentiations.
`U` is the number of multiplications between two 256-bit integers.
`A` is the number of point additions over the Elliptic Curve (EC).
$\texttt{Sm}_\texttt{p}$ is the number of scalar multiplications over the EC with precomputation available for a fixed base.
`Sm` is the number of scalar multiplications over the EC.
`P` is the number of pairings.

Table 3.3: Time measurements in ms.



Complexity of the pairing

The pairing operations present in our PPAT algorithms is one of the most expensive since it is performed in $\mathbb{F}_{p^{12}}$. For this reason, we deploy many optimizations to fasten its computation. Its computation is performed thanks to Algorithm 3.1. We recall that an overview on pairing can be found in Appendix A.3.

Algorithm 3.1: Optimal ate pairing

Input: $P \in G_1$ and $Q \in G_2$.
Output: $x := e(P, Q)$

$T \leftarrow Q$
$x \leftarrow 1_{\mathbb{F}_{p^{12}}}$
write $r \leftarrow abs(6u + 2)$ as $r = \sum r_i 2^i$
for $i \leftarrow \log_2 r - 1$ **to** 0 **do**
  $x \leftarrow x^2.\mathbf{Tang}_T(P)$
  $T \leftarrow 2T$
  if $r_i = 1$ **then**
    $x \leftarrow x.\mathbf{Line}_{T,Q}(P)$
    $T \leftarrow T + Q$
  end
end
if $u < 0$ **then**
  $T \leftarrow -T$
  $x \leftarrow x^{-1}$
end
$Q_1 \leftarrow \phi(Q)$
$Q_2 \leftarrow \phi^2(Q)$
$x \leftarrow x.\mathbf{Line}_{T,Q_1}(P)$
$T \leftarrow T + Q_1$
$x \leftarrow x.\mathbf{Line}_{T,Q_2}(P)$
$x \leftarrow x^{-1}.\phi^6(x)$
$x \leftarrow x.\phi^2(x)$
$x \leftarrow x^{\frac{p^4-p^2+1}{q}}$

Comments on Algorithm 3.1.

- Operations on T and Q are performed in $E'_{\mathbb{F}_{p^2}}$ while operations on x are performed in $\mathbb{F}_{p^{12}}$.

- $\mathbf{Tang}_T(P)$ is a function that, first, computes the tangent line to the curve E' passing by T and then evaluates this tangent in P. The result stands in $\mathbb{F}_{p^{12}}$. This function requires $13M + 2S$ in $\mathbb{F}_p$.

- Similarly, $\mathbf{Line}_{T,Q}(P)$ is a function that, first, computes the line passing by T and Q and then evaluates it to the point P. The result stands in $\mathbb{F}_{p^{12}}$. This function requires $12M$ in $\mathbb{F}_p$.

- The multiplication of two elements in $\mathbb{F}_{p^{12}}$ requires $128M$ in $\mathbb{F}_p$. The squaring of one element in $\mathbb{F}_{p^{12}}$ requires $12S + 98M$ in $\mathbb{F}_p$. The inversion of one element in $\mathbb{F}_{p^{12}}$ requires $145M + 30S + 3I$ in $\mathbb{F}_p$ where I is the number of modular inversions.

- ϕ is the Frobenius endomorphism. Over points of $E'_{\mathbb{F}_{p^2}}$, it requires $8M$ in $\mathbb{F}_p$. Indeed, in this case, $\phi(Q) = \phi(x_Q, y_Q) = (x_Q^p, y_Q^p) = (x_Q.\Im(\xi^{\frac{p-1}{3}}), y_Q.\Im(\xi^{\frac{2(p-1)}{3}}))$ where $\xi^{\frac{p-1}{3}}$ and $\xi^{\frac{2(p-1)}{3}}$ can be precomputed. Over points of $E_{\mathbb{F}_{p^{12}}}$, observe that for $x \in \mathbb{F}_{p^{12}}$, we can write x as $x_0 + x_1\sqrt[3]{\xi} + x_2\sqrt[3]{\xi}^2 + (x_3 + x_4\sqrt[3]{\xi} + x_5\sqrt[3]{\xi}^2)\sqrt[2]{\xi}$ where $x_i \in \mathbb{F}_{p^2}$ and then, compute $\phi(x) = x^p$ as

$$\begin{aligned}\phi(x) = \overline{x_0} \quad & +\overline{x_1}.\xi^{\frac{p-1}{3}} \quad && \sqrt[3]{\xi} + \quad && \overline{x_2}.\xi^{\frac{2(p-1)}{3}}\sqrt[3]{\xi}^2 \\ + (\overline{x_3}.\xi^{\frac{p-1}{2}} \quad & +\overline{x_4}.\xi^{\frac{p-1}{2}}.\overline{\xi^{\frac{p-1}{3}}} && \sqrt[3]{\xi} + && \overline{x_5}.\xi^{\frac{p-1}{2}}.\overline{\xi^{\frac{2(p-1)}{3}}}\sqrt[3]{\xi}^2)\sqrt[2]{\xi}\end{aligned}$$

  resulting in $7M$ in $\mathbb{F}_{p^2}$ or $28M$ in $\mathbb{F}_p$. As before, we rely on the fact that $\xi^{\frac{p-1}{3}}, \xi^{\frac{2(p-1)}{3}}$ and $\xi^{\frac{p-1}{2}}$ can be precomputed. For more details, see [BGDM+10].

- The exponentiation at the last line of the algorithm is obviously called the final exponentiation. Its role is to normalize $x \in \mathbb{F}_{p^{12}}$ so that the pairing result is unique and gives an element of order q. In the original algorithm, the exponent is $(p^{12} - 1)/q$ which can be written as $(p^6 - 1)(p^2 + 1)(p^4 - p^2 + 1)/q$. The first two multipliers are handled thanks to the Frobenius in lines 16 to 22 and it remains the last one. The final exponentiation weights a lot in the total computational effort, and for that reason we rely on addition chain technique as well as, again, the Frobenius to optimize its computation. This method targeted to BN-curves is presented in [SBC+09] and its cost is $(640W_H(u) + 2425)M + (640|u| + 78)S$.

- As a result, the exact complexity of the algorithm is $(181|6u + 2| + 204W_H(6u + 2) + 640W_H(u) + 640|u| + 3318)M + (2|6u + 2| + 136)S + 9I$ in $\mathbb{F}_p$.

With careful choice of parameters, the optimal ate pairing main loop is short and the number of operations can be reduced when $6u + 2$ has low Hamming weight. Nevertheless, the main loop remains the main computational contributor of the algorithm. For our prototype tests, we choose a curve proposed in [PSJNB11] whose parameters are given below:

$$\begin{aligned}
c, d &= 1 \\
b &= 2 \\
\xi &= 1 + i \\
E &\equiv y^2 = x^3 + b \\
E' &\equiv y^2 = x^3 + 1 - i \\
u &= -2^{62} + 2^{55} + 1 \\
p := p(u) &= 167981087310158322849408041422317339098891871\cdots \\
&\quad 21439069848933715426072753864723 \\
q := q(u) &= 167981087310158322849408041422317339097595791\cdots \\
&\quad 60340475274902837886416557021594 9 \\
|p| = |q| &= 254 \\
|u| &= 64 \\
|6u + 2| &= 66 \\
W_H(u) &= 3 \\
W_H(6u + 2) &= 5
\end{aligned}$$

In our setting, the number of operations to compute one pairing rises to $47,218M + 268S + 9I$ which is roughly $47,522U$ if we assimilate squarings to multiplications and if we approximate one inversion to five multiplications.

Chapter 4

Cryptographic Vote with Perfectly Private Audit Trail

Cryptographic vote is one of the most broadly used cases of multi-party computation. It is a sensitive area where the security requirements are strong and where cryptography has much to offer to a large public. It is, therefore, a well-studied branch of cryptography and, for us, a natural gateway to verifiable multi-party function evaluation. Indeed, in voting schemes the tally function to evaluate can be very simple in some cases (e.g. the sum of the votes) and starting from the results of this Chapter, we will extend the function evaluation to any arithmetic function in Chapter 5.

In the current chapter we describe the results presented at the conference ESORICS 2013 in the paper "Election Verifiability or Ballot Privacy: Do we Need to Choose?" [CPP13]. In this work, the goal is to conciliate the confidentiality property and the verifiability property that a voting scheme must possess. The confidentiality guarantees a voter that his vote remains secret and verifiability ensures that the tally of the election is in accordance with the votes of the legitimate voters. Prior to this work, it seems that these two properties tended to cancel each other out in the sense that one cannot be completely fulfilled without jeopardizing the other. On the other hand, our solution proposes a verifiable voting scheme that is also perfectly private. To reach this objective, we use the **commitment consistent encryption** (CCE) introduced in Chapter 3. This primitive enables us to build the first universally verifiable voting scheme with a perfectly private audit trail (PPAT) and practical complexity. That is:

- the audit trail that is published for verifying elections guarantees everlasting privacy, and
- the computational load required from the participants is only increased by a small constant factor compared to traditional voting schemes, and is optimal in the sense of Cramer, Gennaro and Schoenmakers [CGS97].

These properties make it possible to introduce election verifiability in large scale elections as a pure benefit, that is, without loss of privacy compared to a non-verifiable scheme and at a similar level of efficiency.

We propose different approaches for constructing voting schemes with PPAT from CCE, as well as an additional CCE constructions. A first voting scheme is based on the PPATS of Section 3.4 and is tailored for elections with a small number of candidates. The second, based on another CCE scheme, is suitable for elections with complex ballots.

Finally, we present the concrete implementation and timing results obtained for our PPAT constructions.

4.1 Introduction

Elections enable a set of voters to express their opinion regarding one or more questions, and build an aggregate outcome from these personal opinions. While very simple election mechanisms, like hand raising, can be very convenient to organize, various properties are usually required from voting schemes nowadays, and those are not guaranteed by a hand raising process.

Vote privacy is probably the most important property that has been added on top of correctness/verifiability (guaranteed by the hand raising process). It even became mandatory for public elections in most countries during the 19^{th} century, as a way to prevent coercion and bribery [Sal06].

Elections guaranteeing the privacy of the votes while preserving the correctness of the outcome are unfortunately much harder to organize in a trustworthy way: as usual, correctness and privacy guarantees tend to conflict.

As a result, most voting schemes used today enforce privacy at the expense of the correctness properties: in the traditional paper-based scheme, it is most of the time impossible for a voter to convince himself that his vote is included in the ballot boxes that are tallied (he has to trust election officers on that), and the same happens with the commonly deployed non-verifiable electronic voting schemes which also makes it impossible for the voters to verify what is counted by the computers, if there is anything counted at all.

As a way to solve this problem, universally verifiable voting systems were proposed in the seminal works of Benaloh et al. [Ben87, CF85]. These works have been followed by a considerable body of research during the last 25 years (see [CEC+08, CFSY96, CGS97, DJ01, FOO93, HS00, MN06, RBH+09, SK95] for instance).

Universally verifiable elections are carried out by including in the voting process the production of an audit trail (which can be electronic, made of paper, or both). This enables voters to check that their vote was recorded properly and that the election outcome is consistent with all the votes submitted by legitimate voters (formal definitions appear in [KTV10, KRS10] for instance.)

The adoption of universally verifiable technologies is however complicated if the audit trail, provided in order to guarantee the correctness of an election, in turn weakens the privacy of the votes: this raises questions about the relative importance of the correctness improvement resulting from the audit trail versus the potential decrease of privacy that results from that same audit trail, as well as about the consequences of any (even partial) failure with respect to one of these properties. These are sensitive problems, and the balance between these requirements will typically depend on the specifics of each election (stakes, voter population, culture, ...).

This compromise between correctness and privacy needs to be made in the vast majority of the verifiable voting schemes that were proposed [Ben87, CGS97, DJ01, FOO93, HS00, RBH+09, SK95] (we discuss the few exceptions in Section 4.1.2) including those that were used in

real-world elections. The public audit trail of all those voting systems indeed includes information that could reveal individual votes if a computationally secure cryptosystem is broken. This will eventually happen in a hard to predict future, either because of the increase of power of computing devices, or because of a cryptanalytic breakthrough that may happen at any time.

For instance, the voting system Helios [ADMPQ09] publishes the encrypted votes, which may eventually be revealed if the encryption scheme used is broken. This in part motivated the decision of the IACR to only display aliases instead of voter names on their election bulletin board: in case of broken encryption, the election bulletin board would then only reveal the content of encrypted votes but not their author (the voting server is still aware of the link between aliases and voters, though, and these aliases circulate in cleartext emails). Such a procedure however impairs eligibility verifiability, as it becomes infeasible for the voters to verify whether the ballots present on the bulletin board were submitted by legitimate voters or are the result of ballot stuffing by the organizers [KRS10, ACKR13].

In a similar way, Scantegrity II [CCC+10] publishes a Q table containing the confirmation codes that were unveiled during the voting phase, and, as soon as there are a few dozens of voters, the content of this table will determine uniquely the value of the seed used to build the original P table. This in turn reveals the votes corresponding to all the voter's receipts. This may be enough to defeat the purpose of the introduction of privacy in voting systems, since voters may be coerced just by fear of a future loss of privacy.

4.1.1 Our contributions

We address this problem using the commitment consistent encryption (CCE) introduced in Chapter 3, that can be plugged in voting schemes as a replacement for traditional encryption. The use of this primitive makes it possible to obtain verifiable elections with a perfectly private audit trail (PPAT), that is, an audit trail that preserves the privacy of the votes even when facing a computationally unbounded adversary. As a result, adding a PPAT on top of a traditional voting scheme provides the benefits of universally verifiable voting technologies without interfering with the privacy properties of the original system.

As an important example of application, we investigate the use of CCE for building single-pass [BCP+11] voting schemes with PPAT. Those voting schemes support a voting process that executes asynchronously and in a single step, which makes them well-suited for large scale elec-

tions: voters just produce their ballot and send it to the authorities. The reception of the ballots and the tally are then orchestrated by a set of authorities, who are also in charge of publishing the election audit trail. The correctness of this audit trail ensures the correctness of the election outcome even if **all** the authorities are corrupted. Still, the privacy of the votes relies on the number of corrupted authorities to be lower than a certain threshold.

With this application in mind, we design two efficient CCE encryption schemes. The first of our schemes is additively homomorphic and is particularly suitable for elections based on homomorphic tallying. It is however limited to elections that have a small election outcome space (e.g., elections in which the outcome is simply the sum of votes received by the candidates). Our second scheme is suitable for elections with mixnet-based tallying, in which all ballots are decrypted after shuffling, which allows supporting arbitrary ballot formats. We eventually propose a third scheme that is flexible enough to be used in both contexts but is complicated and much less efficient to use.

Our first two schemes admit simple distributed and threshold key generation procedures: all the computations happen in prime order groups and the standard threshold key generation techniques available in such groups apply [GJKR07]. This is particularly important, especially in terms of round complexity, as the trustees of an election will often not be able to set up specific software for running key generation: for instance, the Helios voting system used by IACR relies on n-out-of-n distributed key generation just to keep the key generation ceremony simple (traditional threshold key generation would require more than one single round).

These two CCE schemes are also very efficient, making them usable in JavaScript applications like Helios for instance: based on the performance on the JSBN cryptographic library, the preparation of any vote that can be encoded on 256 bits requires less than a second. Our prototype implementation realized in Python allowed us to perform timing measurements confirming the efficiency of our schemes.

Based on these schemes, we obtain the first universally verifiable voting protocols with PPAT and optimal efficiency (in the sense of [CGS97]):

- the ballot size and the voter computational load do not depend on the number of voters nor on the number of authorities and
- the workload of the tallying authorities grows linearly with the number of voters and candidates.

Furthermore, our schemes do not rely on expensive cut-and-choose techniques: the number of exponentiations to be performed is independent

of the security parameter.

4.1.2 Related works

Very few voting protocols offer a perfectly private audit trail, and they all require either an amount of work by the voters that grows linearly with the number of trustees, or the use of specific communication channels, or are inefficient.

A first class of voting schemes that can offer a PPAT is based on blind signatures [FOO93]. Here, ballots are blindly signed by an authority, then unblinded by the voters who eventually publish their authority signed ballot through an anonymous channel. The vote privacy issue is here taken care of by the anonymous channel and the audit trail only contains anonymous information. Setting up a perfectly anonymous channel can however be very challenging in a large scale election.

A second approach was proposed by Cramer, Franklin, Schoenmakers and Yung [CFSY96]. Here, a verifiable secret sharing scheme is used by the voters to distribute the information needed to tally their vote. The shares are then distributed to the authorities either through private channels or protected by encryption. The computational load of the voters then grows linearly with the number of authorities. This motivated the consecutive proposal by Cramer, Gennaro and Schoenmakers of a scheme that offers a computationally private audit trail but a work load for the voters that is independent of the number of authorities [CGS97].

In the same spirit as the work of Cramer et al. [CFSY96], Moran and Naor proposed a voting scheme with everlasting privacy [MN10]. Here again, the privacy of the votes is protected through secret sharing and the complexity of the ballot preparation task grows linearly with the number of authorities.

As far as we know, our solutions are the first to offer a PPAT while being based on the third approach of e-voting, that is, the tallying of threshold encrypted ballots [Ben87, CF85, CGS97, HS00]. In a contemporary work, Demirel, van de Graaf and Araújo [DvdGA12, DvdG12] explore a similar problem and propose a solution based on the combination of Pedersen's commitments and Paillier's encryption proposed by Moran and Naor [MN10]. As acknowledged by these authors, this solution is not practical: it relies on cut-and-choose zero-knowledge proofs (ZKPK), which makes it slower than ours by approximately 4 orders of magnitude for comparable security levels and requires the execution of sophisticated SMC protocols for distributed key generation by the trustees.

In terms of modelling, symbolic techniques have also been recently

proposed to model everlasting privacy [ACKR13].

Two flavours of verifiability. Just like privacy comes in computational and information theoretic flavours, election verifiability can be computational or information theoretic. Again, just as in the case of ballot privacy, the verifiability property is computational in most of the currently known efficient voting schemes. A common place where this computational aspects appear is in the zero-knowledge proofs that are used in these schemes, which are usually only computationally sound (typically relying on the Fiat-Shamir heuristic). If computational assumptions are broken, this could allow voters to fake a vote validity proof, or trustees to fake a decryption proof. The lack of soundness of these proofs might become apparent in an unpredictable future, when people will be able to break the encryption scheme used. This is however expected to happen way too late to provide an effective way of correcting the election outcome.

We point out important practical differences between the effects of computational privacy and computational verifiability.

1. While breaking the privacy of the votes can be harmful at any time, an adversary who would like to fake the outcome and the audit trail of an election needs to break the system during the election (that is, provide audit information that would pass all verification procedures even though the properties that the verification is supposed to guarantee are violated), since this is the time when the audit trail must be published. Breaking a scheme in real time seems much more demanding than breaking it in some future.

2. Universal verifiability is a correctness property that people adopt by comparing verifiable systems with the non-verifiable systems that they used in the past. We believe that this adoption can be simplified if it does not impact the other properties of the system and, in particular, if the audit trail that is produced does not decrease the privacy properties of the previously used systems. (Similar considerations motivated the design of Scantegrity: its practical adoption is expected to have been facilitated by the absence of the need to decrease the usability of the paper ballots [CCC+10].)

We believe that those reasons support the development of voting schemes with PPAT.

Coercion resistance. The historical motivation for introducing secret ballots was the prevention of bribery or coercion. The schemes we propose address the concern of a voter who fears that the audit data of an election could reveal his vote. This concern is certainly the most ubiquitous and hard to prevent through law enforcement or by voter education: it does not require any visible step by a coercer who just needs to look at available data. We do not focus on specific coercion resistance procedures in our simple application examples, as coercion prevention is a much broader problem than what can be addressed at a protocol level, especially when vote-by-mail is authorized or when nothing prevents bringing camera phones in a voting booth. Our schemes are however compatible with most existing approaches, e.g., revoting as first used in Estonia or coercion detection [GRBR13].

Organisation of the Chapter. The rest of this Chapter is organized as follows. Section 4.2 states the computational assumptions we use. In Section 4.3 discusses security properties that these encryption schemes need to satisfy for use in voting applications. Then, Section 3.3 describes a generic construction of CC and CCVA encryption schemes. Section 4.4 defines two efficient CCVA encryption schemes and explains how they can be plugged in classical voting schemes. We finally analyze the efficiency of our solutions in Section 4.6.

4.2 Computational setting

We rely on the DCR problem (Assumption 2.1) for the security of our Paillier based scheme in Section 3.3. The security of our two efficient schemes described in Section 4.4 relies on the hardness of the DDH problem (Assumption 2.4) as well as the SXDH problem (Assumption 2.5) for the pairing-based constructions. In this setting, we assume the existence of a bilinear group generator that, on input 1^λ, produces a description of bilinear groups $\mathcal{P}air_{\mathsf{SXDH}} = (q, G_1, G_2, G_3, e, g_1, g_2)$ where G_1, G_2 and G_3 are groups of order q, a λ-bit prime, e is an efficient and non-degenerating bilinear map $e : G_1 \times G_2 \to G_3$ and g_1, g_2 are generators of G_1 and G_2 respectively. We expect that these groups are chosen in such a way that there is no known efficient mapping between G_1 and G_2 in either direction. Common concrete choices include the use of BLS and BN curves [BLS03, BN06].

Note that all our schemes could be easily adapted to the symmetric pairing settings, typically by relying on the hardness of the DLIN problem instead of DDH [BBS04]. The choice we made provides more efficient

protocols and makes it also possible to compute in smaller fields for equivalent security levels.

4.3 Voting scheme with perfectly private audit trail

In the spirit of [BCP+11], we now propose a "minivoting" scheme, that we use to describe how a validity augmented CCE scheme (Definition 3.2) can be used to submit ballots in an election. We then describe the security guarantees that CCVAE schemes need to provide for their application in voting with PPAT.

The minivoting scheme we consider follows a classic workflow. First, a setup phase takes place, during which two clean bulletin boards **PB** and **SB** are created and elections keys are generated and appropriately published. The board **PB** contains the public audit trail, while **SB** is kept secret by the authorities and used to compute the tally. Voters then produce their ballots by encrypting their votes and send these ballots to the election authorities. The ballots are processed by these authorities, and the bulletin boards are updated accordingly. At the end of the voting phase, a tallying protocol is executed and the election outcome is published.

Definition 4.1 (Minivoting scheme)**.** *Let* Π *be a* CCVAE *scheme, and let* ρ *be a result function that takes a set of valid votes and produces the corresponding election outcome. From these, we build a minivoting scheme* $\mathsf{Enc2Vote}(\Pi, \rho)$ *as follows:*

$\mathsf{Setup}(1^n)$**:** *run the key generation algorithm* Gen *of* Π *on the same input to obtain a triple* $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk})$*. Initialize a public and a secret bulletin board,* $\boldsymbol{PB}$ *and* $\boldsymbol{SB}$*, to* $\perp$*.*

$\mathsf{Vote}(\mathsf{pk}, v)$**:** *encrypt a vote* v *with* pk *using* Π *to obtain a ballot* b*. This is executed by voters to prepare their ballot.*

$\mathsf{ProcessBallot}(\mathsf{pk}, b, \mathbf{PB}, \mathbf{SB})$**:** *reject* b *if it is already present in* $\boldsymbol{SB}$*. Otherwise, run* $\mathsf{Valid}(\mathsf{pk}, b)$ *and reject* b *if it fails. If all these steps succeed, append* b *on* $\boldsymbol{SB}$ *and* $\mathsf{DerivCom}(\mathsf{pk}, b)$ *on* $\boldsymbol{PB}$*. This is executed by the authorities every time a ballot is received.*

$\mathsf{Tally}(\mathsf{sk}, \mathbf{PB}, \mathbf{SB})$**:** *decrypt all ballots on* $\boldsymbol{SB}$ *to obtain a vector of votes* $\boldsymbol{v}$*, and publish* $\rho(\boldsymbol{v})$ *on* $\boldsymbol{PB}$*. This is executed by the authorities.*

A minivoting scheme does not require any proof of the validity of the ballots (for example that they would encrypt 0 or 1 in an approval

voting system), nor publish any specific information regarding a proof of correctness of the tally, which will be needed for universal verifiability. For modularity, we address these concerns separately: the structure of these proofs of correctness will indeed depend on the result function ρ.

We now focus on the privacy of the votes that is offered in such a minivoting scheme, which we capture through the vote privacy experiment 4.1 $\mathsf{VotePriv}\boldsymbol{BB}(\lambda)$, strongly inspired from the ballot privacy game of Bernhard et al. [BCP+11] but with perfect instead of computational security. This experiment is then used to define the notions of perfectly private audit trail (PPAT) and ballot privacy.

Experiment 4.1 (The Vote Privacy experiment $\mathsf{VotePriv}\boldsymbol{BB}(\lambda)$)**.**

1. *The challenger randomly picks a bit* $\beta \xleftarrow{rand} \{0,1\}$*. He also runs the* Setup *algorithm of the voting scheme on input* 1^λ *and obtains the triple* $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk})$ *and empty bulletin boards* $\boldsymbol{PB}_\beta$ *and* $\boldsymbol{SB}_\beta$*. He then sends* pp, pk *to* $\mathcal{A}$ *and creates two other empty bulletin boards* $\boldsymbol{PB}_{1-\beta}$ *and* $\boldsymbol{SB}_{1-\beta}$*.* $\mathcal{A}$ *is allowed to see the board* $\boldsymbol{BB}_\beta$*, where* $\boldsymbol{BB}$ *is a parameter of the experiment.*

2. $\mathcal{A}$ *may address two types of queries to the challenger:*

   $\mathsf{Vote}(v_0, v_1)$**:** *the challenger executes* $\mathsf{Vote}(\mathsf{pk}, v_i)$*, obtaining a ballot* b_i*, and then runs* $\mathsf{ProcessBallot}(\mathsf{pk}, b_i, \boldsymbol{PB}_i, \boldsymbol{SB}_i)$*, for* $i \in \{0,1\}$*.*

   $\mathsf{Ballot}(b)$**:** *the challenger executes* $\mathsf{ProcessBallot}(\mathsf{pk}, b, \boldsymbol{PB}_\beta, \boldsymbol{SB}_\beta)$*. If it succeeds, the challenger runs* $\mathsf{ProcessBallot}(\mathsf{pk}, b, \boldsymbol{PB}_{1-\beta}, \boldsymbol{SB}_{1-\beta})$*.*

3. *The challenger computes the tally* $t_0 := \mathsf{Tally}(\mathsf{sk}, \boldsymbol{SB}_0)$ *and appends* t_0 *on* $\boldsymbol{PB}_\beta$ *and* $\boldsymbol{SB}_\beta$*.*

4. $\mathcal{A}$ *outputs a bit* β'*. If* $\beta = \beta'$ *then the output of the experiment is 1 and we say that* $\mathcal{A}$ *wins.*

This experiment is the basis of our notion of perfectly private audit trail, as well as our notion of ballot privacy that matches the one of Bernhard [BCP+11]. The meaning of "perfectly" is the one of the security in the *information theoretic* sense.

Definition 4.2 (Perfectly Private Audit Trail (PPAT))**.** *A minivoting scheme* $\mathsf{Enc2Vote}(\Pi, \rho)$ *has a* ***perfectly private audit trail*** *(*PPAT*) if, for every adversary* $\mathcal{A}$*,* $\Pr[\mathsf{VotePriv}\boldsymbol{PB}(\lambda) = 1] = \frac{1}{2}$*.*

Since this definition does not place any bound on the computational power of the adversary, the everlasting privacy of the votes is guaranteed against people who only see (and record) the **PB** board. In some contexts (e.g., when using groups of unknown order), it is useful to relax the above definition by accepting statistical indistinguishability and tolerating a negligible advantage over $\frac{1}{2}$.

Independently of this, the private bulletin board, only seen by the authorities, should provide computational ballot privacy.

Definition 4.3 (Ballot Privacy [BCP+11])**.** *We say that a minivoting scheme* Enc2Vote(Π, ρ) *has* ***ballot privacy*** *if, for every* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* η *such that,*

$$\Pr[\mathsf{VotePriv}\boldsymbol{SB}(\lambda) = 1] = \frac{1}{2} + \eta(\lambda).$$

Security. The following two theorems define security properties of a CCVAE scheme that guarantees the PPAT and ballot privacy of the corresponding minivoting scheme. This will be most useful for the constructions that we describe next.

Theorem 4.1. *Let* Π *be a* CCVAE *scheme, and let* ρ *be a result function. If the output of* DerivCom *is perfectly hiding, then the minivoting scheme* Enc2Vote(Π, ρ) *has a perfectly private audit trail.*

Proof. The view of the adversary is the VotePriv$\mathbf{PB}(\lambda)$ experiment and this view is independent of β: **PB** only contains perfectly hiding commitments and then a tally that is always computed from $\mathbf{SB}_0$, which is independent of β. □

Theorem 4.2 ([BPW12])**.** *Let* Π *be an* NM-CPA CCVAE *scheme, and let* ρ *be a result function. Then the minivoting scheme* Enc2Vote(Π, ρ) *has ballot privacy.*

Proof. The proof of this theorem is very similar to the one that appears in [BPW12]. It is based on the characterization of NM-CPA security by Bellare and Sahai [BS99], which is the IND-CPA game in which the adversary is also allowed to make one single parallel decryption query after his test query.

Suppose that a PPT adversary $\mathcal{A}$ is able to break the ballot privacy property for Enc2Vote(Π, ρ). We then build an adversary $\mathcal{B}$ who can break the NM-CPA security of Π as follows. Let us consider a (polynomial) upper-bound l on the number of Vote and Ballot queries that $\mathcal{A}$ does, and the distributions $H_0, \ldots, H_l$, where H_i is produced as follows:

1. The key generation algorithm of Π is executed, and the public part of the resulting key is submitted to $\mathcal{A}$. Empty boards $\mathbf{SB}$, $\mathbf{SB}_0$ and $\mathbf{SB}_1$ are initialized as well, but only $\mathbf{SB}$ is part of the view of $\mathcal{A}$. (The public board can be derived from the private one.)

2. $\mathcal{A}$ then performs Vote and Ballot queries sequentially which we number from 1 to (at most) l. On the j-th query:

   - If it is a $\mathsf{Ballot}(b)$ query and b does not appear on $\mathbf{SB}$, then it is appended on $\mathbf{SB}$, $\mathbf{SB}_0$ and $\mathbf{SB}_1$. Else it is rejected.
   - If it is a $\mathsf{Vote}(v_0, v_1)$ query, then v_0 and v_1 are encrypted and posted on $\mathbf{SB}_0$ and $\mathbf{SB}_1$ respectively. Furthermore, if $j \leq i$ then the encryption of v_0 is posted on $\mathbf{SB}$, else the encryption of v_1 is posted there.

3. When $\mathcal{A}$ performs a Tally query, all ballots on $\mathbf{SB}_0$ and $\mathbf{SB}_1$ are decrypted, and the corresponding votes are tallied. If the results are identical, then they are appended on $\mathbf{SB}$, else $\perp$ is posted there.

It is easy to observe that H_0 and H_l produce the view of $\mathcal{A}$ when $\beta = 0$ and $\beta = 1$ respectively. Now, since $\mathcal{A}$ can distinguish H_0 from H_l with non negligible probability ϵ, it must also be able to distinguish H_i from H_{i+1} with non negligible probability at least ϵ/l, for at least one value of i. We use this to build an adversary $\mathcal{B}$ against the $\mathsf{NM\text{-}CPA}$ security of Π as follows.

1. $\mathcal{B}$ receives a public key from Π and forwards it to $\mathcal{A}$.

2. $\mathcal{B}$ then interacts with $\mathcal{A}$ on the Vote and Ballot queries, in the natural way, producing a view identical to the one $\mathcal{A}$ would see in both H_i and H_{i+1}.

3. If the $i+1$-th query is a $\mathsf{Vote}(v_0, v_1)$ query, then $\mathcal{B}$ forwards the two votes v_0 and v_1 as challenge messages in the $\mathsf{NM\text{-}CPA}$ experiment.

4. $\mathcal{B}$ interacts with $\mathcal{A}$ in the natural way for the further Ballot and Vote queries.

5. When receiving a Tally query, $\mathcal{B}$ submits all unique ballots that were sent as Ballot queries to the $\mathsf{NM\text{-}CPA}$ decryption oracle and recovers the corresponding plaintexts. Note that since duplicate ballots are rejected, this will be a valid query with overwhelming probability.

6. $\mathcal{B}$ then uses those decrypted ballots and the content of the Vote queries to compute the tallies corresponding to $\mathbf{SB}_0$ and $\mathbf{SB}_1$, and answers $\mathcal{A}$ accordingly.

7. $\mathcal{B}$ outputs the bit it receives from $\mathcal{A}$.

We can observe that the view of $\mathcal{A}$ in these interactions is the one of H_i and H_{i+1} depending on whether v_0 or v_1 is encrypted as part of the NM-CPA challenge query. So, the success probability of $\mathcal{A}$ in distinguishing H_i from H_{i+1} is equal to the one that $\mathcal{B}$ wins the NM-CPA game. □

Minivoting based on the generic construction of CCE

The generic construction Π_{G} of Section 3.3.1 is a good candidate for these theorems. Indeed, if the commitment scheme in Π_{G} is perfectly-hiding, then all the conditions of Theorem 4.1 and Theorem 4.2 hold: the resulting minivoting scheme enjoys the PPAT as well as the ballot-privacy.

Adding extra proofs. The minivoting scheme resulting from Π_{G} allows any voter to check whether her/his vote is posted on the public bulletin board. Depending on the election specifics, extra proofs can be added in order to prove the validity of the votes, or to prove that the tally is consistent with the posted commitments. These proofs will not influence the PPAT property as long as they are perfect (or at least statistical) zero-knowledge.

Paillier-Pedersen instance. We may now build a voting scheme based on the PPATP scheme presented in Section 3.3.2 and instantiating Π_{G}. For simplicity, we present this voting scheme in terms of a single authority. The extension to distributed or threshold mechanisms is immediate and follows from standard techniques in the case of active malicious authorities in our Paillier-based scheme [DJ01]. We assume some unique and efficiently verifiable encoding for the elements of our various groups. This is needed in order to avoid some trivial malleability relations.

By construction this scheme satisfies all the conditions of validity and security of Theorem 3.2 (in the random oracle model, assuming the hardness of the DCR problem). As a result, for any tallying function ρ, the corresponding minivoting scheme Enc2Vote(PPATP, ρ) has a PPAT and guarantees ballot privacy.

Unfortunately, this instantiation based on Paillier is not satisfactory for the reasons raised in Section 3.3.2. First, the threshold key generation algorithm is tedious and challenging as the key is the factorization of an RSA modulus. Second, the computational cost of this scheme can be fairly expensive, especially if we desire to use an homomorphic tallying approach that will usually require us to compute a number of modular exponentiations that will be 5 to 10 times higher than the number of candidates [CGS97]. These two reasons motivate the efficient constructions presented in the next sections and offering the same level of security while relying on smaller groups.

4.4 Efficient commitment consistent encryption schemes with validity augmentation

This section describes two much more efficient and usable constructions of CCVAE schemes. These schemes do not follow the generic approach presented in the previous section but combine encryptions and commitments in a more efficient way. They also make it possible to perform the whole computation in prime order groups.

The first scheme, PPATS introduced in Section 3.4, allows using traditional ballot validity proof techniques and completing the tally through the homomorphic addition of encrypted votes. The decryption process however involves a stage of exhaustive search of the plaintext (just as the exponential ElGamal scheme used in many applications). This restricts the use of this scheme to elections where this kind of exhaustive search can be done, e.g., when the outcome is simply a count of the number of votes that each candidate received. The second scheme, PPATC, is tailored for mixnet based tallying procedures: the ciphertexts are not additively homomorphic but the decryption procedure is efficient regardless of the message. In both tally procedures we explicitly show how the process does not affect the PPAT as well as the ballot privacy of voting schemes provided by our CCVAE schemes.

Again we present our voting schemes in terms of a single authority. The extension to distributed or threshold mechanisms is immediate and follows from standard techniques in the case of active malicious authorities in prime order group based schemes [GJKR07].

4.4.1 Elections with simple ballots

In the daily life, there are situations where voting schemes are only concerned with small domain result functions e.g., with a **yes-or-no** vote. Since the result of the election lies in a very small range, it is

efficient enough for the talliers to make an exhaustive search to find this result.

We consider here the most simple election case where the voters have to approve or reject a proposal by encrypting either a 0 value or a 1 value. This case can be generalized to many others (general approval voting, ...) using standard techniques, e.g., [CGS97, HS00].

As part of the tallying procedure the talliers will be required to perform an exhaustive search to find $M = \sum_{i \leq L} m_i$ where the m_i's are the L voter's choices. This is not a real issue as long as M is bounded by L.

The PPATS scheme makes use of two compatible homomorphic ingredients: exponential ElGamal encryption and the TC2 perfectly hiding commitment scheme proposed by Abe et al. [AHO10], which is binding in the $\mathcal{P}air_{\mathsf{SXDH}}$ setting (see Section 4.2). The resulting CCE scheme is compatible with Σ-protocols, and the definition of a validity augmentation is then simple. In Protocol 4.1, we denote the NIZKPK for commitment consistency by π_{cc}.

Unlike what is done in Section 3.4, we prefer to use here the multiplicative notations for the groups G_1 and G_2. This allows more compact expressions but does not affect the scheme.

Protocol 4.1 (The PPATS CCVAE scheme)**.**

$\mathsf{Gen}_{\mathsf{S}}^{\mathsf{VA}}(1^\lambda)$**:** *Generate setting* $\mathcal{P}air_{\mathsf{SXDH}} = (q, G_1, G_2, G_3, e, g_1, g_2)$ *where* $|q| = \lambda$ *together with the following additional public random generators* $h_1 = g_1^{x_1} \xleftarrow{rand} G_1$ *and* $h_2 \xleftarrow{rand} G_2$*. The triple* $(\mathsf{pp}_{\mathsf{S}}, \mathsf{pk}_{\mathsf{S}}, \mathsf{sk}_{\mathsf{S}})$ *is defined as* $((\mathcal{P}air_{\mathsf{SXDH}}, h_2), h_1, x_1)$*. The augmented key* $\mathsf{pk}_{\mathsf{S}}^{\mathsf{VA}} = \mathsf{Expand}(\mathsf{pk}_{\mathsf{S}})$ *is obtained by adding the description of an efficient hash function* $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_q$*, resulting in the triple* $(\mathsf{pp}_{\mathsf{S}}^{\mathsf{VA}} = \mathsf{pp}_{\mathsf{S}}, \mathsf{pk}_{\mathsf{S}}^{\mathsf{VA}}, \mathsf{sk}_{\mathsf{S}}^{\mathsf{VA}} = \mathsf{sk}_{\mathsf{S}})$*. Consequently, we have that* $\mathbf{M} = \mathbf{O} := \mathbb{Z}_q$*,* $\mathbf{C}_{\mathsf{C}} := G_2$ *and* $\mathbf{C} := G_2 \times G_1^2 \times \mathbb{Z}_q^3$*.*

$\mathsf{Enc}_{\mathsf{S}}^{\mathsf{VA}}(\mathsf{pk}_{\mathsf{S}}^{\mathsf{VA}}, m)$**:** *For* $m \in \mathbb{Z}_q$*, choose random values* $r, s \xleftarrow{rand} \mathbb{Z}_q$ *and compute* $c := \mathsf{Enc}_{\mathsf{S}}(\mathsf{pk}_{\mathsf{S}}, m)$ *as* $(d, c_1, c_2) := (g_2^m h_2^r, g_1^s, g_1^r h_1^s)$*. Then, add the following consistency proof. Compute* $\mathfrak{a}_{\mathsf{cc}} := (g_2^{m'} h_2^{r'}, g_1^{s'}, g_1^{r'} h_1^{s'})$ *for random* $m', r', s' \xleftarrow{rand} \mathbb{Z}_q$*. Compute* $\mathfrak{t}_{\mathsf{cc}} := (\mathfrak{e}_{\mathsf{cc}}, \mathfrak{z}_{\mathsf{cc}})$ *where* $\mathfrak{e}_{\mathsf{cc}} = \mathcal{H}(\mathsf{pp}_{\mathsf{S}}^{\mathsf{VA}}, \mathsf{pk}_{\mathsf{S}}^{\mathsf{VA}}, c, \mathfrak{a}_{\mathsf{cc}})$ *and* $\mathfrak{z}_{\mathsf{cc}} := (\mathfrak{z}_m, \mathfrak{z}_r, \mathfrak{z}_s) = (m' + \mathfrak{e}_{\mathsf{cc}} m, r' + \mathfrak{e}_{\mathsf{cc}} r, s' + \mathfrak{e}_{\mathsf{cc}} s)$*. Output the ciphertext* $c^{\mathsf{VA}} = (c, \mathfrak{t}_{\mathsf{cc}})$*.*

$\mathsf{Dec}_{\mathsf{S}}^{\mathsf{VA}}(\mathsf{sk}_{\mathsf{S}}^{\mathsf{VA}}, c^{\mathsf{VA}})$**:** *Parse* c^{VA} *as* $(d, c_1, c_2, \mathfrak{t}_{\mathsf{cc}})$ *and return the discrete logarithm* $\bar{m} := \mathsf{Dlog}_{e(g_1,g_2)} e(c_1^{x_1}/c_2, h_2) e(g_1, d)$*.*

$\mathsf{DerivCom}_{\mathsf{S}}^{\mathsf{VA}}(\mathsf{pk}_{\mathsf{S}}^{\mathsf{VA}}, c^{\mathsf{VA}})$**:** *Parse* c^{VA} *as* $(d, c_1, c_2, \mathfrak{t}_{\mathsf{cc}})$ *and return* d*.*

$\mathsf{Open}_{\mathsf{S}}^{\mathsf{VA}}(\mathsf{sk}_{\mathsf{S}}^{\mathsf{VA}}, c^{\mathsf{VA}})$**:** *Parse* c^{VA} *as* $(d, c_1, c_2, \mathfrak{t}_{\mathsf{cc}})$*, then compute and output the ElGamal decryption* $\bar{o} := c_2/c_1^{x_1}$ *(i.e.,* g_1^r*, consisting of the TC2 opening value with respect to* d*).*

$\mathsf{Verify}_{\mathsf{S}}^{\mathsf{VA}}(\mathsf{pk}_{\mathsf{S}}^{\mathsf{VA}}, d, o, m)$**:** *Return* 1 *only if* $e(o, h_2) = e(g_1, d/g_2^m)$*.*

$\mathsf{Valid}_{\mathsf{S}}(\mathsf{pk}_{\mathsf{S}}^{\mathsf{VA}}, c^{\mathsf{VA}})$**:** *Parse* c^{VA} *as* $(c, \mathfrak{t}_{\mathsf{cc}}) = (d, c_1, c_2, \mathfrak{e}_{\mathsf{cc}}, \mathfrak{z}_{\mathsf{cc}})$ *and parse* $\mathfrak{z}_{\mathsf{cc}}$ *as* $(\mathfrak{z}_m, \mathfrak{z}_r, \mathfrak{z}_s)$*. Output 1 only if* $\mathfrak{e}_{\mathsf{cc}} \stackrel{?}{=} \mathcal{H}(\mathsf{pp}_{\mathsf{S}}^{\mathsf{VA}}, \mathsf{pk}_{\mathsf{S}}^{\mathsf{VA}}, c, \mathfrak{a}'_{\mathsf{cc}})$ *where* $\mathfrak{a}'_{\mathsf{cc}} := (g_2^{\mathfrak{z}_m} h_2^{\mathfrak{z}_r} d^{-\mathfrak{e}_{\mathsf{cc}}}, g_1^{\mathfrak{z}_s} c_1^{-\mathfrak{e}_{\mathsf{cc}}}, g_1^{\mathfrak{z}_r} h_1^{\mathfrak{z}_s} c_2^{-\mathfrak{e}_{\mathsf{cc}}})$.

The algorithm $\mathsf{Strip}_{\mathsf{S}}$ *returns* c *from* c^{VA} *in the obvious way. Applying* $\mathsf{Strip}_{\mathsf{S}}$ *to* PPATS *ciphertexts leads to an homomorphic* CCE *scheme.*

Theorem 4.3. *The above* PPATS *scheme is an* NM-CPA *secure* CCVAE *scheme in the random oracle model within the* $\mathcal{P}air_{\mathsf{SXDH}}$ *setting.*

Proof. We first observe that PPATS is a CCVAE scheme. Indeed, the formal aspects of the definition are satisfied, and $\mathsf{Valid}_{\mathsf{S}}$ algorithm is sound thanks to the soundness of the π_{cc} proof and from the binding property of the TC2 scheme. Both these properties hold in the $\mathcal{P}air_{\mathsf{SXDH}}$ setting.

To verify that PPATS offers NM-CPA security, we show that a PPATS ciphertext is made of an IND-CPA element c. The NM-CPA security then follows from the fact π_{cc} is a NIZKPK of the plaintext and randomness used to compute c.

To show the IND-CPA security of c, we show that it is indistinguishable of a random tuple of (independent) group elements, using the following intermediary distributions. Let m be an element of $\mathbb{Z}_q$.

$\mathcal{D}_3$ is the regular distribution $(g_2^m h_2^r, g_1^s, g_1^r h_1^s)$ produced by encrypting m using random values $r, s \xleftarrow{rand} \mathbb{Z}_q$.

$\mathcal{D}_2$ is the distribution $(g_2^m h_2^r, g_1^s, g_1^r h_1^{s^*})$ identical to $\mathcal{D}_3$ except that the third element is now computed using a fresh random $s^* \xleftarrow{rand} \mathbb{Z}_q$. Any distinguisher between $\mathcal{D}_3$ and $\mathcal{D}_2$ can solve the DDH problem in G_1 with the same success probability.

$\mathcal{D}_1$ is the distribution $(g_2^m h_2^r, g_1^s, g_1^{s^*})$, identical to $\mathcal{D}_2$.

$\mathcal{D}_0$ is the distribution $(h_2^r, g_1^s, g_1^{s^*})$, which is identical to $\mathcal{D}_1$.

We observe that the 3 elements of $\mathcal{D}_0$ are all random and independent. So, the advantage of any adversary against this encryption scheme has its success probability bounded by the probability of breaking DDH in G_1 by any adversary using the same computational effort (up to the computation of a constant number of modular exponentiations).

The validity augmentation part of PPATS simply adds a secure sigma proof. The NM-CPA security of the scheme is obtained by applying Theorem 3.2. □

Proving vote validity.

Considering the minivoting scheme based on PPATS, it is easy to provide additional verifiability mechanism to ensure the validity of a vote. Consider for instance the case of an approval election. In this case, we need to prove that a voter submitted an encryption of $m = 0$ or $m = 1$. This can be done requiring each voter to append to the encryption of her choice a non-interactive 0-1 or-proof π_{or} on the public commitment part [CDS94]. More precisely, when computing $c^{\mathsf{VA}} = (c, \mathfrak{t}_{\mathsf{cc}})$ where $c = (d, c_1, c_2)$ the voter conducts the following extra steps: she computes $\mathfrak{a} := (\mathfrak{a}_0, \mathfrak{a}_1)$ where $\mathfrak{a}_m := h_2^{r'}$, $\mathfrak{a}_{1-m} := h_2^{\mathfrak{z}_{1-m}}(d/g_2^{1-m})^{-\mathfrak{e}_{1-m}}$ for random $r', \mathfrak{e}_{1-m}, \mathfrak{z}_{1-m} \xleftarrow{rand} \mathbb{Z}_q$, then defines $\mathfrak{t}_{\mathsf{or}} := (\mathfrak{e}_0, \mathfrak{e}_1, \mathfrak{z}_0, \mathfrak{z}_1)$ with $\mathfrak{e}_m := \mathcal{H}(\mathsf{pp}_{\mathsf{S}}^{\mathsf{VA}}, \mathsf{pk}_{\mathsf{S}}^{\mathsf{VA}}, d, \mathfrak{a}) - \mathfrak{e}_{1-m}$ and with $\mathfrak{z}_m := r' + \mathfrak{e}_m r$.

Given the commitment d and the proof $\mathfrak{t}_{\mathsf{or}} = (\mathfrak{e}_0, \mathfrak{e}_1, \mathfrak{z}_0, \mathfrak{z}_1)$, anybody can verify that d can only be opened to a vote for 0 or 1: compute $\mathfrak{a}' = (\mathfrak{a}'_0, \mathfrak{a}'_1)$ as $\mathfrak{a}'_i = h_2^{\mathfrak{z}_i}(d/g_2^i)^{-\mathfrak{e}_i}$, for $i = 0, 1$, and check whether $\mathfrak{e}_0 + \mathfrak{e}_1 \stackrel{?}{=} \mathcal{H}(\mathsf{pp}_{\mathsf{S}}^{\mathsf{VA}}, \mathsf{pk}_{\mathsf{S}}^{\mathsf{VA}}, d, \mathfrak{a}')$. Furthermore, since the π_{or} proof is perfect zero knowledge, it can indeed safely appear on the public bulletin board **PB** without affecting the privacy goals in any way.

Note that this π_{or} proof can be easily adapted for the PPATP scheme encrypting only 0/1 votes. To avoid redundancy, we do not detail this proof, but we analyse in Section 3.5 the PPATP scheme for 0/1 messages and the corresponding π_{or} proof as a point of comparison with the *PPATo* scheme.

Elections with homomorphic tallying from PPATS**.**

We can now use this scheme to build a voting scheme PPATSVote based on $\mathsf{Enc2Vote}(\mathsf{PPATS}, \rho_{\mathsf{S}})$ but from which we modify the Tally algorithm as follows.

1. **Stripping**: Once the polls are closed, the authorities run $\mathsf{Valid}_{\mathsf{S}}$ and $\mathsf{Strip}_{\mathsf{S}}$ on the CCVAE ciphertexts stored on **SB**, obtaining CCE homomorphic ciphertexts.

2. **Aggregation**: The authorities multiply those ciphertexts, obtaining one resulting CCE ciphertext c_{tot}.

3. **Decryption**: The authorities compute $v = \mathsf{Dec}_{\mathsf{S}}(\mathsf{sk}_{\mathsf{S}}, c_{\mathsf{tot}})$ the result of the election. To prove the correctness of the decryption,

they also run Open_S on c_tot, obtaining an opening value o_tot. Finally the authorities append (v, o_tot) to **PB**.

Theorem 4.4. *The* PPATSVote *scheme offers a* PPAT *and ballot privacy in the* $\mathcal{P}air_{\mathsf{SXDH}}$ *setting in the random oracle model.*

Proof. The PPATSVote scheme is equivalent to the $\mathsf{Enc2Vote}(\mathsf{PPATS}, \rho_\mathsf{S})$ scheme except that it also discloses the opening value o_tot on **PB**. This value is fully determined by the commitment on the outcome and by the outcome itself, which implies that it does not provide any extra information to an unbounded adversary, and the PPAT property offered by $\mathsf{Enc2Vote}(\mathsf{PPATS}, \rho_\mathsf{S})$ is then preserved. The trustees having access to **SB** also see the decryption factors produced by Dec. They are however indistinguishable from random group elements under DDH, as for standard ElGamal decryption, and therefore do not help breaking ballot privacy. □

Audit procedure. The audit procedure is performed through the following steps:

1. Run all the verification procedures on the commitments displayed on **PB**. If the verification procedure fails for any commitment, abort.

2. Multiply all the commitments, obtaining a commitment d_tot on the election outcome v_tot.

3. Verify that the announced outcome v_tot and the opening value o_tot passes the $\mathsf{Verify}_\mathsf{S}^\mathsf{VA}(\mathsf{pk}_\mathsf{S}^\mathsf{VA}, d_\mathsf{tot}, o_\mathsf{tot}, v_\mathsf{tot})$ algorithm. Abort if it is not the case.

The first step guarantees the validity of the votes posted, while the second and last ones guarantee that the tally matches the posted votes. The binding property of the commitment scheme guarantees that the only opening that the authorities will ever be able to provide comes from a honest tallying process.

We emphasize that this last verification is very efficient: it only requires the verification of an opening of one constant-size commitment—no ZKPK is needed here, contrary to traditional approaches.

As far as eligibility may be concerned, the bulletin board can also associate a name with each commitment recorded on **PB** without affecting the PPAT. This offers any observers the possibility to verify that the posted votes have been submitted by valid voters (e.g., by interrogating those voters in case of doubt).

Verifiability/Accountability. Verifiability allows us to check that the votes have been properly recorded and tallied. In order to decide what action must be taken if a verification fails, it may be useful to have a stronger property: accountability. This property was highlighted by Küsters et al. [KTV10] and applied to the Bingo voting scheme and then to several variants of the Helios voting system [KTV12].

While plugging the PPATS scheme into Helios would not have any noticeable impact on the verifiability analysis of Helios proposed by Kremer et al. [KRS10], the distinction between the private and public board and between perfect and computational privacy has more impact on the accountability analyses of Küsters et al. [KTV12]. In particular, while the ballot validity test is fully public in Helios, replacing ElGamal encryption with the PPATS scheme adds a step during which authorities could decide to reject a ciphertext because the π_{cc} proof would be invalid, which could not be verified from the content of **PB** since neither π_{cc} nor the corresponding statement appear on that board. As a result, it will not be possible to determine whether the authorities or the voter are cheating without disclosing to a judge information that only offer conditional privacy. Different strategies for improving the accountability in the case of Helios have been explored in [ADMPQ09, KTV12]. A rigorous cryptographic analysis of verifiability/accountability of a fully-fledged voting system is an open problem (note that all current works on Helios [KRS10, KTV12] abstracted the cryptographic aspects and, as a result, overlooked the recently found attacks on the verifiability of Helios [BPW12]), and is out of our scope.

Variations on PPATS. In the $\mathcal{P}air_{\mathsf{SXDH}}$ setting, the group operations in G_2 are typically much more expensive than those in G_1. As a result, it might be more efficient to use a slightly modified version of PPATS for more complex ballot validity proofs (e.g., the vote is an integer in a larger range). Indeed the ciphertext could be extended with an extra element $c_3 = g_1^m f_1^s$ in G_1 (for a third generator f_1 of G_1) which, together with c_1 would consist of an ElGamal (re-)encryption of m. The validity proofs could then be on c_3, and the decryption algorithm could become faster as well. However, a proof that (d, c_3) is well-formed would also have to be published, which makes that construction more expensive than the PPATS scheme that we consider since the computations of elements in G_2 for the simple 0/1 validity proof have a lower cost.

4.4.2 Elections with complex ballots

The PPATS scheme is appropriate for elections with simple ballots. In some elections, it is however useful to be able to encode complex votes in a single ciphertext. This happens for instance in elections with a very large number of candidates or with complex tallying rules that make the homomorphic aggregation approach impractical, or in elections where arbitrary write-ins need to be supported. For those elections with complex ballots, a tallying approach based on verifiable mixnets is usually adopted. This motivates our definition of the PPATC scheme below. This scheme has an efficiency comparable to the previous one but offers efficient decryption procedures for arbitrary plaintext. The corresponding CCE scheme is however not additively homomorphic anymore, but this is not a problem in a mixnet setting since ballots are individually decrypted. ElGamal encryption is a core ingredient of this scheme, together with the $\mathcal{P}air_{\mathsf{SXDH}}$-secure and perfectly hiding commitment scheme of Abe et al [AHO12].

Protocol 4.2 (The PPATC CCVAE scheme)**.**

$\mathsf{Gen}_{\mathsf{C}}^{\mathsf{VA}}(1^{\lambda})$**:** *Generate* $\mathcal{P}air_{\mathsf{SXDH}} = (q, G_1, G_2, G_3, e, g_1, g_2)$ *for* $|q| = \lambda$ *together with the following additional public random generators* $h_1 := g_1^{x_1}$, $f_1 := g_1^{x_2}$ *in* G_1 *and* $h_2 \xleftarrow{rand} G_2$. *The triple* $(\mathsf{pp}_{\mathsf{C}}, \mathsf{pk}_{\mathsf{C}}, \mathsf{sk}_{\mathsf{C}})$ *is defined as* $((\mathcal{P}air_{\mathsf{SXDH}}, h_2), (h_1, f_1), (x_1, x_2))$. *The augmented key* $\mathsf{pk}_{\mathsf{C}}^{\mathsf{VA}} := \mathsf{Expand}(\mathsf{pk}_{\mathsf{C}})$ *is computed by adding to* pk_{C} *the description of an efficient hash function* $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_q$, *resulting in the triple* $(\mathsf{pp}_{\mathsf{C}}^{\mathsf{VA}} = \mathsf{pp}_{\mathsf{C}}, \mathsf{pk}_{\mathsf{C}}^{\mathsf{VA}}, \mathsf{sk}_{\mathsf{C}}^{\mathsf{VA}} = \mathsf{sk}_{\mathsf{C}})$. *We have that* $\mathbf{M} = \mathbf{O} := G_1$, $\mathbf{C}_{\mathsf{C}} := G_1 \times G_2$ *and* $\mathbf{C} := G_1^4 \times G_2 \times \mathbb{Z}_q^4$.

$\mathsf{Enc}_{\mathsf{C}}^{\mathsf{VA}}(\mathsf{pk}_{\mathsf{C}}^{\mathsf{VA}}, m)$**:** *For* $m \in G_1$, *choose random values* $r, s, t \xleftarrow{rand} \mathbb{Z}_q$ *and compute the ciphertext* $c := \mathsf{Enc}_{\mathsf{C}}(\mathsf{pk}_{\mathsf{C}}, m)$ *as* $(c_1, c_2, c_3, d_1, d_2) := (g_1^r, g_1^s, h_1^t f_1^s, m h_1^r, g_2^t h_2^r)$. *Add the following consistency proof* π_{cc}. *Select random values* $r', s', t' \xleftarrow{rand} \mathbb{Z}_q$ *and* $m' \xleftarrow{rand} G_1$ *and compute* $\mathfrak{a} := (g_1^{r'}, g_1^{s'}, h_1^{t'} f_1^{s'}, m' h_1^{r'} g_2^{t'} h_2^{r'})$ *and* $\mathfrak{e}_{\mathsf{cc}} := \mathcal{H}(\mathsf{pp}_{\mathsf{C}}^{\mathsf{VA}}, \mathsf{pk}_{\mathsf{C}}^{\mathsf{VA}}, c, \mathfrak{a})$. *Then, compute* $\mathfrak{z} := (\mathfrak{z}_m, \mathfrak{z}_r, \mathfrak{z}_s, \mathfrak{z}_t)$ *where* $\mathfrak{z}_m := m' m^{\mathfrak{e}_{\mathsf{cc}}}$, $\mathfrak{z}_r := r' + \mathfrak{e}_{\mathsf{cc}} r$, $\mathfrak{z}_s := s' + \mathfrak{e}_{\mathsf{cc}} s$, $\mathfrak{z}_t := t' + \mathfrak{e}_{\mathsf{cc}} t$. *Set* $\mathfrak{t}_{\mathsf{cc}} := (\mathfrak{e}_{\mathsf{cc}}, \mathfrak{z})$. *The ciphertext* c^{VA} *is made of* $(c, \mathfrak{t}_{\mathsf{cc}})$.

$\mathsf{Dec}_{\mathsf{C}}^{\mathsf{VA}}(\mathsf{sk}_{\mathsf{C}}^{\mathsf{VA}}, c^{\mathsf{VA}})$**:** *Parse* c^{VA} *as* $(c_1, c_2, c_3, d_1, d_2, \mathfrak{t}_{\mathsf{cc}})$ *and return* $\bar{m} := d_1 / c_1^{x_1}$.

$\mathsf{DerivCom}_{\mathsf{C}}^{\mathsf{VA}}(\mathsf{pk}_{\mathsf{C}}^{\mathsf{VA}}, c^{\mathsf{VA}})$**:** *Parse* c^{VA} *as* $(c_1, c_2, c_3, d_1, d_2, \mathfrak{t}_{\mathsf{cc}})$ *and return* d *as* (d_1, d_2).

$\mathsf{Open}_{\mathsf{C}}^{\mathsf{VA}}(\mathsf{sk}_{\mathsf{C}}^{\mathsf{VA}}, c^{\mathsf{VA}})$: *Parse* c^{VA} *as* $(c_1, c_2, c_3, d_1, d_2, \mathfrak{t}_{\mathsf{cc}})$, *and return* $\bar{o} := c_3/c_2^{x_2}$.

$\mathsf{Verify}_{\mathsf{C}}^{\mathsf{VA}}(\mathsf{pk}_{\mathsf{C}}^{\mathsf{VA}}, d, o, m)$: *Parse* d *as* (d_1, d_2) *and return* 1 *if* $e(h_1, d_2) \stackrel{?}{=} e(o, g_2)e(d_1/m, h_2)$ *and* 0 *otherwise.*

$\mathsf{Valid}_{\mathsf{C}}(\mathsf{pk}_{\mathsf{C}}^{\mathsf{VA}}, c^{\mathsf{VA}})$: *Parse* c^{VA} *as* $(c_1, c_2, c_3, d_1, d_2, \mathfrak{e}_{\mathsf{cc}}, \mathfrak{z}_m, \mathfrak{z}_r, \mathfrak{z}_s, \mathfrak{z}_t)$ *and test whether all elements of the ciphertext are properly encoded. Return* 1 *only if* $\mathfrak{e}_{\mathsf{cc}} \stackrel{?}{=} \mathcal{H}(\mathsf{pp}_{\mathsf{C}}^{\mathsf{VA}}, \mathsf{pk}_{\mathsf{C}}^{\mathsf{VA}}, c, \mathfrak{a}')$ *where*

$$\mathfrak{a}' := (g_1^{\mathfrak{z}_r} c_1^{-\mathfrak{e}_{\mathsf{cc}}}, g_1^{\mathfrak{z}_s} c_2^{-\mathfrak{e}_{\mathsf{cc}}}, h_1^{\mathfrak{z}_t} f_1^{\mathfrak{z}_s} c_3^{-\mathfrak{e}_{\mathsf{cc}}}, \mathfrak{z}_m h_1^{\mathfrak{z}_r} d_1^{-\mathfrak{e}_{\mathsf{cc}}}, g_2^{\mathfrak{z}_t} h_2^{\mathfrak{z}_r} d_2^{-\mathfrak{e}_{\mathsf{cc}}}).$$

The algorithm $\mathsf{Strip}_{\mathsf{C}}$ *returns* c *from* c^{VA} *in the obvious way.*

Applying $\mathsf{Strip}_{\mathsf{C}}$ to a PPATC ciphertext leads to a CCE ciphertext that is homomorphic with respect to the curve group law in G_1, which is sufficient to obtain the randomization properties needed for mixing. The use of the PPATC scheme also requires the existence of an efficient mapping between the votes and G_1. This can be achieved easily in most cases. For instance, most pairing friendly curves of the form $y^2 = x^3 + b$ on $\mathbb{F}_q$ have q chosen in such a way that any message y in $\mathbb{Z}_q$ can be mapped on a point $((y^2 - b)^{\frac{1}{3}}, y)$ [BN06].

As far as efficiency is concerned, note that a voter will never have to compute elements in G_3 and especially pairings when creating a ballot.

Theorem 4.5. *The* PPATC *scheme described above is an* NM-CPA *secure* CCVAE *scheme in the random oracle model in the* $\mathcal{P}air_{\mathsf{SXDH}}$ *setting.*

Proof. This proof follows the scheme of the proof of Theorem 4.3. We first observe that PPATC is a CCVAE scheme. Indeed, the formal aspects of the definition are satisfied, and $\mathsf{Valid}_{\mathsf{C}}$ algorithm is sound thanks to the soundness of the π_{cc} proof and from the binding property of the scheme of Abe et al.[AHO12]. Both these properties hold in the $\mathcal{P}air_{\mathsf{SXDH}}$ setting.

To verify that PPATC offers NM-CPA security, we show that a PPATC ciphertext is made of an IND-CPA element c. The NM-CPA security then follows from the fact that π_{cc} is a NIZKPK of the plaintext and randomness used to compute c.

To show the IND-CPA security of c, we show that c is indistinguishable from a random tuple of (independent) group elements, using the following intermediary distributions. Let m be an element of G_1.

$\mathcal{D}_5$ is the regular distribution $(g_1^r, g_1^s, h_1^t f_1^s, mh_1^r, g_2^t h_2^r)$ produced by encrypting m using random values r, s, t.

$\mathcal{D}_4$ is the distribution $(g_1^r, g_1^s, h_1^t f_1^{s^*}, mh_1^r, g_2^t h_2^r)$ identical to $\mathcal{D}_5$ except that the third element is now computed using a fresh random s^*. Any distinguisher between $\mathcal{D}_5$ and D_4 can solve the DDH problem in G_1 with the same success probability.

$\mathcal{D}_3$ is the distribution $(g_1^r, g_1^s, f_1^{s^*}, mh_1^r, g_2^t h_2^r)$, which is identical to $\mathcal{D}_4$.

$\mathcal{D}_2$ is the distribution $(g_1^r, g_1^s, f_1^{s^*}, mh_1^r, g_2^t)$, which is identical to $\mathcal{D}_3$.

$\mathcal{D}_1$ is the distribution $(g_1^r, g_1^s, f_1^{s^*}, mh_1^{r^*}, g_2^t)$, obtained by replacing r with a random fresh value r^* in the fourth element. Again, any distinguisher between $\mathcal{D}_2$ and D_1 can solve the DDH problem in G_1 with the same success probability.

$\mathcal{D}_0$ is the distribution $(g_1^r, g_1^s, f_1^{s^*}, h_1^{r^*}, g_2^t)$, which is identical to $\mathcal{D}_1$.

We observe that the five elements of $\mathcal{D}_0$ are all random and independent. So, the advantage of any adversary against this encryption scheme has its success probability bounded by twice the probability of breaking DDH in G_1 by any adversary using the same computational effort (up to the computation of a constant number of modular exponentiations).

Finally, observe that the NIZKPK π_{cc} added to obtain the PPATC scheme is perfectly zero knowledge and thus does not break the privacy property. We conclude by applying Theorem 3.2. □

A verifiable shuffle for voting systems with PPAT

Now we would like to shuffle the PPATC ciphertexts and publish openings of the corresponding anonymized commitments. Since our scheme is randomizable, this does not raise any specific concern.

We also need to make the shuffle verifiable, that is, to provide a proof of shuffle, which needs to preserve the information theoretic privacy of **PB**. Various perfect (or statistical) ZK proofs of shuffles can be used for that purpose [Gro10, JJR02, TW10]: these guarantee that a simulator can produce a proof of shuffle just from the inputs and outputs of that shuffle that is indistinguishable from a real proof, even by an unbounded adversary.

In our context, we need to shuffle both the CCE ciphertexts and the extracted commitments in a verifiable way, with a single permutation, to keep track of their concordance. The commitment consistent shuffle approach proposed by Terelius and Wikström [Wik09, TW10] seems particularly natural for that purpose. This approach splits the proof of shuffle into two stages. First a perfectly hiding commitment on the permutation matrix used in the shuffle is computed and made public.

This is the most computationally intensive part of the protocol and, interestingly, it is independent of the actual values that we need to shuffle and of the randomization factors that will be applied on the ciphertexts. Then, a much cheaper proof is produced that shows that the shuffle performed on the ciphertexts is consistent with the commitment on that permutation matrix. In our case, that proof can be computed both for the PPATC ciphertexts on **SB** and for the corresponding commitments on **PB**.

We sketch the resulting tallying protocol below.

1. **Stripping**: The authorities run $\mathsf{Valid}_\mathsf{C}$ and $\mathsf{Strip}_\mathsf{C}$ on the ciphertexts stored on **SB**, obtaining a vector $\boldsymbol{c}$ of L ciphertexts and a vector $\boldsymbol{d}$ of commitments.

2. **Permutation commitment**: The authorities select a random permutation σ and compute a commitment u on that permutation, together with a validity proof π_σ.

3. **Shuffle**: The authorities select random vectors $\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{t} \xleftarrow{rand} \mathbb{Z}_q^l$ and compute a vector of ciphertexts $\boldsymbol{c'}$ where $c'_i := c_{\sigma^{-1}(i)} a_i$ (with component-wise operations) where $a_i := \mathsf{Enc}_\mathsf{C}(\mathsf{pk}_\mathsf{C}, 1_{G_1})$ using respectively $r_{\sigma^{-1}(i)}, s_{\sigma^{-1}(i)}, t_{\sigma^{-1}(i)}$ as random values. The last two components of $\boldsymbol{c'}$ are posted on **PB** and denoted $\boldsymbol{d'}$.

4. **Proof of shuffle**: The authorities compute two commitment consistent proofs of shuffle with respect to the committed permutation σ: $\pi_{\sigma(\boldsymbol{c})}$ that shows that $\boldsymbol{c'}$ is indeed a shuffle of $\boldsymbol{c}$ and $\pi_{\sigma(\boldsymbol{d})}$ shows that $\boldsymbol{d'}$ is a shuffle on $\boldsymbol{d}$. Then, $\pi_{\sigma(\boldsymbol{c})}$ is posted on **SB** while $\pi_{\sigma(\boldsymbol{d})}$ is posted on **PB**.

5. **Decryption of openings**: The authorities verify the proofs, then decrypt all the ciphertexts in $\boldsymbol{c'}$ and run Open_C on these ciphertexts in order to obtain the opening values for the corresponding commitments. The plaintexts and opening values are published on **PB**.

Of course, the three middle stages of this procedure, corresponding to the verifiable shuffling, should be repeated by several independent authorities.

The tally audit procedure for an observer consists in the following stages.

- Public observers verify the proof of permutation commitment π_σ and abort if it fails.

- Public observers verify the proof of shuffle $\pi_{\sigma(\boldsymbol{d})}$ and abort if it fails.

- Public observers verify that the authorities published valid openings for the shuffled commitments $\boldsymbol{d'}$ and abort otherwise.

The fact that this whole procedure preserves the PPAT follows from the fact that all the commitments are perfectly hiding and that all the proofs can be made perfect zero-knowledge.

4.5 Implementation and efficiency measures

A prototype implementation of the different PPAT CCVAE schemes was performed in Python. It includes the different commitment consistent encryption schemes as well as the consistency proofs and other extra proofs that are needed for the verifiability. The libraries and the code developed are made available online at [Cuv15]. The details of this implementation, the choice of the parameters as well as the computational workload are provided in Section 3.5.

In Table 3.1 (page 60) we gathered the costs of each operations where the U cost represents the multiplication between two 256-bit integers. Table 3.2 (page 61) details the cost of each algorithm of the PPAT schemes at comparable security levels which is equivalent to 2048-bit RSA modulus N. As already mentioned in Section 3.5, computing a PPATP ciphertext is roughly 227 times more expensive than computing a PPATC ciphertext. This simple observation puts the PPATS and PPATC far ahead in terms of efficiency.

The cost of the PPATS and PPATC schemes is low enough to make it possible to use these schemes even on fairly slow platforms. For instance, considering the computation of a ciphertext in JavaScript in a browser using the JSBN library, which allows computing a point multiplication in a 256-bit prime order group in less than 30ms in the Chrome web browser, the computation of a PPATC ciphertext that can encode a 256-bit vote would take less than a second [HDNP11].

Several timing measurements have been realized on the Python implementation. The code was run on a standard laptop: Intel® Core i5-3320M CPU @ 2.60GHz×4 with 7.7 GB of RAM. Table 3.3 (page 62) summarizes the costs for the CCE of the PPAT schemes as well as the costs for the VA of the ciphertexts. We distinguish the cost for the voters (computation) from the cost for the talliers (verification).

4.6 Conclusion

In this Chapter, we proposed the systematic design of voting schemes with a perfectly private audit trail. We showed how our CCVAE schemes are suitable for the organization of large-scale elections.

The PPAT scheme mentioned in Section 3.3.1 is fully generic and can be used with all classical tallying techniques. Its key generation algorithm is fairly sophisticated, though, and this scheme is also quite inefficient compared to our other schemes. We address then two other voting schemes based on CCVAE schemes, PPATS already introduced in Section 3.4 and a new CCVAE scheme, PPATC, that are much more efficient and simple to use though less flexible. Nevertheless, they are highly relevant for the two most widely used vote tallying techniques: homomorphic aggregation and mixnets.

The costs of computing a PPATS and a PPATC ciphertexts are similar. The associated tallying techniques are very different though, being much more complex for PPATC. A mixnet based technique also reveals much more information than a technique based on the homomorphic aggregation of ballots. As a result, we would recommend using the PPATS scheme as long as the ballot format allows it, even if the resulting ballot preparation cost is higher than the one that would be obtained by using PPATC.

The ideas exposed in this chapter and beforehand in [CPP13] as the PPATS scheme will probably be part of the STAR-Vote system [BBB+13]. This new electronic voting system is meant to be used for elections in Travis County (Texas, USA).

Finally, the techniques and the use of the CCE primitive introduced in the current chapter are the starting point of Chapter 5 where we present verification mechanisms applicable to any function we wish to evaluate in multi-party. Indeed, in the present case of a voting scheme, the function is basically a sum function which, combined with the homomorphic property of the encryption scheme, makes it easy to verify. However, in Chapter 5, we extend our approach to any function. In this case, as the function is more complex, the verification of the result might become tedious if we rely only on traditional verification techniques.

Chapter 5

Function Evaluation with Perfectly Private Audit Trail

This chapter covers results presented at the *1st Symposium on Data Security in Auvergne, 2014* in a paper entitled "Multi-Party Function Evaluation with Perfectly Private Audit Trail" [CP14].

In this chapter, we extend the results obtained in Chapter 4. Indeed, in the electronic voting scenarios we presented, the function to evaluate is quite simple. In the best case, the function is the sum of the 1/0 votes. The audit procedure is then performed easily since there is an homomorphism between the message space containing the votes and the commitment space containing the perfectly hiding commitments on the votes that are posted on the public bulletin board. However, if we want to evaluate more complex functions, we need additional audit mechanisms to encompass the different operations in the function. For example, we naturally wish to include multiplications and comparisons but also branchings in the function operands.

With this objective in mind, this chapter proposes an efficient and simple protocol for the evaluation of functions getting their inputs from multiple parties in a way that guarantees the correctness of the computation to everyone. Our protocol finds applications in a clients-worker environment where we assume that the clients have a strong incentive to collaborate if they receive a high level guarantee about the result correctness. Note that this setting differs from the setting presented in Chapter 4. Here, the worker is trusted with the privacy of the inputs, and given this assumption, our protocol provides perfect privacy for the clients. By putting the emphasis on the verifiability property rather than on the privacy one as it is the case in most approaches, the goal was to observe if there was a interesting trade-off for the clients' computational effort. As a matter of fact, we point out that in some applications where the

verification of the solution is cheaper than its computation, the clients' gain is not negligible.

Compared to the traditional Secure Multi-Party Computation (SMC) setting, where workers do not learn any information about the inputs, but which is usually quite challenging to deploy in practice, our solution considerably decreases the amount of work for the worker, and often enables a considerably faster verification process by the clients.

Our construction relies on homomorphic *commitment consistent encryption* presented in Chapter 3 and used in Chapter 4. We rely on non-interactive zero-knowledge arguments of knowledge to provide proof that the result is correct.

We present three unrelated applications of our technique: solving a system of linear equations, an auction scheme and the search of the shortest path in a shared graph. These examples illustrate the ease of use and the advantage in terms of complexity of our approach. We made a prototype implementation, which provides encouraging timing result.

5.1 Introduction

In a multi-party function evaluation, a set of clients wish to evaluate a function of their inputs. While keeping the privacy of their inputs, the clients want a strong guarantee that the output of the evaluation is correct and consistent with the inputs. With the growth of outsourcing computation, the needs for such a functionality increase. Numerous techniques involving highly refined cryptography exist and offer solutions to this problem. In fact, the whole branch of SMC studies protocols and mechanisms and offers solutions. We use classic SMC techniques in Chapter 6 to solve problems similar to those tackled here. It will provide us with an interesting comparison point for our algorithms.

The work in the present chapter takes place in this history but favours settings where a third party is trusted for the privacy of the inputs but not trusted for the correctness of the result. This assumption frees us from difficulties that arise when turning SMC into real-life applications. To point out some of these challenges, one could mention the need to run (synchronized) servers, the communication complexity that is quadratic in the number of clients and a need of large bandwidth. Moreover, in SMC settings where computations are outsourced to several independent trustees, it is sometimes challenging to find those independent trustees with the adequate technical knowledge. As a result of these constraints, deployment of SMC protocols is often slowed down, and real-world applications of SMC are typically played between a small number of players representative of larger groups, trusted for privacy, and sometimes for honestly playing the protocol as well. For example, the sugar beet auction in Denmark [BCD+09] was performed between three parties representing farmers, buyers and the SMC project promoters. This is also the case in some cryptographic voting systems such as Helios [ADMPQ09] where, despite the simplicity of the function that is evaluated (a sum), the tallying is performed by a small set of trustees sharing the private key of a distributed encryption scheme, and all voters need to trust them for privacy.

In the setting we consider, by relaxing the constraint on privacy, we observe an appealing gain in complexity in several problems that otherwise would not be considered by classical SMC algorithms due to a large computational and communication overhead. In particular, this work offers an easy-to-set-up solution for securely solving several NP-hard problems that would be prohibitive to solve using SMC techniques.

5.1.1 Our contributions.

This chapter presents a technique of multi-party function evaluation that fits very well in a scenario where the clients want absolute confidence in the correctness of the result. While the clients do not trust other clients over the privacy of their inputs, they are not reluctant to rely on a trustworthy third party to preserve the secrecy of the inputs. Moreover, the clients do not wish to install servers to run a computationally and bandwidth intensive task but would rather gladly rely on a worker to do the job.

The level of interaction in our protocol is minimal: the clients submit their inputs to the worker as a single message, and a single public proof is made available at the end of the computation. This makes our solution practical even for applications based on a web interface that clients could use to submit their input, and later retrieve the outcome of the computation. (A similar setting was considered by Halevi et al. [HLP11] for multi-party computation, with impossibility results for large classes of functions due to the lack of interactions.)

We define the security properties of our scheme through ideal functionalities for secure function evaluation. Our protocol guarantees the correctness of the output, even if the worker is corrupted. Furthermore, our protocol guarantees information theoretic privacy if the worker is honest.

Moreover, we show that the possible complexity gain for the clients is non negligible compared with classical SMC technique. This is most visible when solving problems in NP: indeed, while SMC require secure solving of the problems, bringing the cryptography-related overhead on the computation phase, we compute in the clear and the complexity of the audit phase is only related to the computation verification task, which can be much more efficient. We illustrate our technique via three test applications: solving a system of linear equations, electronic auctions and finding the shortest path in a graph. Finally, we give some insight on the performances obtained for these applications through our prototype implementation realized in Python.

5.1.2 Related works.

Recent works suggest and prove a sharp improvement in multi-party protocols practicability. For instance, Damgård et al. [DKL+13] present the "SPDZ" protocol which offers security against active adversary and achieves high performance for real world applications. "SPDZ" follows an offline/online model where most heavy precomputations are performed offline which renders the online phase very efficient. Recently, an

improvement made on "SPDZ" and proposed by Baum et al. [BDO14] has offered the possibility to audit the SMC protocol. In their proposal, a set of servers compute the function and provide a transcript of the computations that will be used later by the clients to verify the correctness. The techniques used there lose most of their appeal when considering our setting: our worker evaluates the function on cleartext data, as fast as it can be, and the output is delivered immediately. Then the audit data can be computed, possibly by performing a completely different computation (e.g., if verifying the computation can be performed differently than recomputing). The verification of the audit data is then performed using public key cryptography techniques in both cases.

In the area of verifiable computation, the "Pinocchio" protocol proposed by Parno et al. [PHGR13], and its refinement "Gepetto" [CFH+15] are highly efficient solutions that offer public verifiability in a single client-worker setting. However, the protocol does not aim at providing privacy of the inputs. Even more efficient than "Pinocchio", Backes et al. [BBFR14] have recently developed a three-party protocol where a worker is requested to prove computations to a client over authenticated data received from a single trusted source. The proof of computation is privacy-preserving. This solution focuses on computation performed with inputs from a single source, while we focus on problems involving numerous parties. Along this line, by using the construction of Parno, Zhang et al. [ZPK14] propose "Alitheia" a single-client verifiable computation system for graph problems such as the shortest path and the maximum flow studied in this thesis.

Our security model is similar to the one of Choi et al. [CKKC13] who achieve non-interactive multi-client verifiable computation by relying on garbled circuits, oblivious transfer and fully homomorphic encryption. In the follow-up works of Goldwaser et al. [GGG+14] and Gordon et al. [GKL+15], the solution uses functional encryption, a primitive that allows to compute a specific function over encrypted data. In these works, the function is chosen in advance through consensus by the clients which is a stronger requirement than our proposal. Moreover, relying on fully homomorphic encryption has not allowed them yet to provide an efficiency analysis.

Much closer to our technique, Rabin et al. [RST08, PRST08] present a secrecy-preserving proof of correctness scheme for the evaluation of any function with straight line computation through an agreed public circuit. Indeed, similar to what is done in this paper, they propose to perform the proof of correctness on the commitments on the inputs of the function. A parallel circuit is evaluated by a worker on the commitments and every operation is validated by a zero-knowledge proof of knowledge. While

the proposed schemes rely on symmetric cryptography and a split-value representation to perform cut-and-choose proofs, we show in this work better timing results as well as more compact proofs using homomorphic cryptography based on elliptic curves.

The structure of the chapter is as follows: in Section 5.2 we describe the functionality and the security proofs. In Section 5.3, we present the building blocks needed for the generic implementation of Section 5.4. Section 5.5 details the three test applications while Section 5.6 provides the technical information, the complexity study, and the timing measurements obtained from our prototype implementation.

5.2 Secure multi-party function evaluation

5.2.1 The ideal protocol

In this section, we formalize the protocol expectations in terms of an ideal functionality, following the notations and definitions of [Can01]. In this regard, let us consider a set of clients $\mathcal{C} = \{C_1, \cdots, C_n\}$. Each C_i gets a private input $x_i \in I$, the input space. We define the *ideal functionality* that we denote $\mathcal{F}^f$ as a process that receives inputs from the clients and then computes the function $f : I^n \to O$ where O is the output space. In all cases, when the functionality provides an output, this output is correct. In the case of a passive adversary $\mathcal{A}_p$, that is, an adversary who learns the internal state of the corrupted parties but lets them follow the protocol, the functionality $\mathcal{F}^f_{\mathcal{A}_p}$ also guarantees that the clients do not learn anything about each other's inputs (apart from what might be derived from the output of the function). When the adversary is active $\mathcal{A}_a$ (which is equivalent to considering a corrupted worker,) the client's inputs are leaked, but the correctness of the outcome is still guaranteed (functionality $\mathcal{F}^f_{\mathcal{A}_a}$). Functionalities $\mathcal{F}^f_{\mathcal{A}_p}$ and $\mathcal{F}^f_{\mathcal{A}_a}$ are presented in Protocol 5.1.

We also want to consider a slightly different functionality in the presence of active adversary, $\mathcal{F}^f_{\mathcal{A}_{a^*}}$ that, instead of sending each private input to the adversary at the moment she receives one, will send all the private inputs in one block once they are all collected. This small difference prevents the active adversary $\mathcal{A}_{a^*}$ from performing one kind of attack on $\mathcal{F}^f_{\mathcal{A}_a}$, that is from dynamically changing the inputs of the corrupted clients when receiving the inputs of other clients. As we will see, implementing $\mathcal{F}^f_{\mathcal{A}_{a^*}}$ in the real-world comes with some more constraints. This functionality is presented in Protocol 5.2.

Protocol 5.1: The ideal functionalities $\mathcal{F}^f_{\mathcal{A}_p}$ and $\mathcal{F}^f_{\mathcal{A}_a}$

1. Upon receiving $(\mathsf{Send}, C_i, x_i)$ from a client C_i or adversary $\mathcal{S}$, if $x_i \in I$, store x_i, otherwise abort. Then, in the case of
   - a **passive adversary**, send C_i to adversary $\mathcal{S}$ and halt.
   - an **active adversary**, send (C_i, x_i) to adversary $\mathcal{S}$ and halt.
2. Upon receiving $\mathsf{Compute}$ from $\mathcal{S}$, evaluate $y := f(x_1, \cdots, x_n)$. Send y to every client C_i and $\mathcal{S}$, then halt.

Protocol 5.2: The ideal functionality $\mathcal{F}^f_{\mathcal{A}_{a^*}}$

1. Upon receiving $(\mathsf{Send}, C_i, x_i)$ from a client C_i or adversary $\mathcal{S}$,
   if $x_i \in I$, store x_i, otherwise abort. Then, send C_i to adversary $\mathcal{S}$ and halt.
2. When all the inputs are received, send $x_1, \cdots, x_n$ to $\mathcal{S}$.
3. Upon receiving $\mathsf{Compute}$ from $\mathcal{S}$, evaluate the function to get $y := f(x_1, \cdots, x_n)$. Finally, send y to every client C_i and $\mathcal{S}$, then halt.

5.2.2 The real protocol

We now turn to the design of our real-world protocol that realize the ideal functionalities.

We require our protocol to produce a perfectly private audit trail (PPAT) of its computation, that is, the privacy guarantees offered by our protocol will be perfect in the sense of information theory. For simplicity, we focus on the case of static corruption: corruption of parties happen before the beginning of the protocol, and not dynamically as the protocol is executing.

We build the protocol Π^f_{PPAT} which realizes these functionalities in the real world in the presence of passive and active adversaries. In this protocol, most of the work of the functionality is performed by an entity called the Worker $\mathcal{W}$. First, we require that each client C_i sends his private input to $\mathcal{W}$ through a secure channel. For example, the

secure channel could be achieved through an encryption/decryption of the inputs between C_i and $\mathcal{W}$. Given a public-key conventional CPA-secure encryption algorithm $((\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, we demand that in protocol Π^f_{PPAT}, C_i sends $e_i := \mathsf{Enc}(\mathsf{pk}, x_i)$ to $\mathcal{W}$, and $\mathcal{W}$ computes $x_i = \mathsf{Dec}(\mathsf{sk}, e_i)$, which gives him the private input of C_i to compute f. Up to this point, our protocol offers secure function evaluation in the presence of a passive adversary.

To ensure that every client receives the same result at the end of the protocol in the presence of an adversary corrupting the worker, we need to ask the worker to prove the correct evaluation of the function. This proof will be posted on a Public Bulletin Board **PB**, which maintains publicly available every input sent to him by any parties.

So, as a first step in our protocol, every client sends a commitment $d_i \leftarrow \mathsf{Com}(\mathsf{cpk}, x_i)$ of his private input to **PB**. We assume that we have at hand a perfectly hiding and computationally binding commitment scheme $\Pi_{\mathsf{C}} := (\mathsf{Gen}_{\mathsf{C}}, \mathsf{Com}, \mathsf{Verify})$ (see Section 2.1.4 for details about commitment schemes).

Along with commitment d_i, we require the client to produce a proof denoted $\pi_{\mathsf{ver}}(d_i)$ that ensures some property of his private input (such as being in a range of values). This proof must at least convince that d_i was obtained through the Com algorithm with a value known by the client, and make the commitment non-malleable [DDN98] in order to prevent a client from choosing his input as a function of the input of another client [BPW12].

Finally, we require $\mathcal{W}$ to post the evaluation of f on **PB**. As $\mathcal{W}$ might be corrupted by $\mathcal{A}_a$, we require that $\mathcal{W}$ also publishes a proof denoted π_{cor} of the correctness of the evaluation of f. The key point is that the verification of π_{cor} relies on the commitments $d_1, ..., d_n$ posted by the clients on **PB**. In the general case, this verification involves the computation of a commitment d_y that must be a commitment on y computed from $d_1, ..., d_n$. With this requirement, an active adversary who would be willing to cheat during the function evaluation process would need to be able to break the binding property of the commitment scheme or the soundness of the proof π_{cor}.

Protocol Π^f_{PPAT} and Formal Security

The proofs we are referring to here are built from the notion of sigma (or Σ)-protocols [Dam04] covered in Section 2.1.5. In the following, we define relations R in a formal NP-language such as $R \subset \mathcal{L}_{\mathsf{NP}} \times W(s)$ where s is called the statement and $w \in W(s)$ a witness of s.

A Σ-protocol allows us to produce zero-knowledge proof which is, in

essence, the couple of algorithms (Prove, Check) with the special honest verifier zero-knowledge property where Prove is the algorithm executed by Prover. When it comes to implementation, one key point is the production of "good" challenges which forces us to choose between the different security models in cryptography. However it is possible to achieve the security of our scheme in the standard model. For efficiency reasons, we implemented the scheme in the random oracle model and thus proved the security in this model.

We rely on the Fiat-Shamir/Blum transformation (Theorem 2.1) to turn Σ-protocols into non-interactive zero-knowledge proofs of knowledges (NIZKPK). A complete overview of NIZKPK and Σ-protocols can be found in Section 2.1.5.

The relations for the NIZKPK π_{cor} and $\pi_{\mathsf{ver}}(d_i)$ mentioned in Π^f_{PPAT} are defined as follows:

$$\begin{aligned} R^{\mathsf{cor}} :=& \{((y, d_1, ..., d_n), (x_1, o_1, ..., x_n, o_n)) | y = f(x_1, ..., x_n) \\ & \wedge_i \mathsf{Verify}(\mathsf{cpk}, d_i, o_i, x_i) = 1\} \\ R^{\mathsf{ver}} :=& \{(d, (x, o)) | \mathsf{Verify}(\mathsf{cpk}, d, o, x) = 1 \wedge x \in I\} \end{aligned}$$

where the algorithm $\mathsf{Verify}(\mathsf{cpk}, d, o, x)$ of the commitment scheme returns 1 only if d is a commitment on x with opening o.

These proofs are published on **PB** and π_{cor} will be checked by each C_i at the end of the protocol to convince itself of the correctness of the function's computation. We are now ready to define protocol Π^f_{PPAT} (Protocol 5.3) which realizes functionalities $\mathcal{F}^f_{\mathcal{A}_p}$ and $\mathcal{F}^f_{\mathcal{A}_a}$ in the presence of $\mathcal{A}_p$ and $\mathcal{A}_a$ respectively. The protocol that realizes $\mathcal{F}^f_{\mathcal{A}_{a^*}}$ in the presence of $\mathcal{A}_{a^*}$ is an adaptation of Π^f_{PPAT} and we describe it in the comments. We show in Section 5.4 how to build protocol Π^f_{PPAT} for any function f and how $\mathcal{W}$ can prepare the proof π_{cor}.

Remarks on Protocol 5.3 Π^f_{PPAT}:

- To realize functionality $\mathcal{F}^f_{\mathcal{A}_{a^*}}$, we demand that the clients do not send directly their private inputs and openings x_i, o_i to $\mathcal{W}$ in step 1. Instead the clients wait until every other client C_i has published a commitment d_i and a proof $\pi_{\mathsf{ver}}(d_i)$ on **PB**. At this point only, they send their x_i, o_i to $\mathcal{W}$ through the secure channel. This additional step prevents the adversary from modifying dynamically the inputs of the corrupted clients when they receive the inputs of the honest clients. While this change looks quite simple to stop these kinds of attacks, its major drawback is that it adds an extra step for the clients during which they have to check that the commitments have all been published on **PB** to proceed with the rest

Protocol 5.3: Π^f_{PPAT}

Input: Each C_i has his private input $x_i \in I$ for $i = 1, \cdots, n$.
Output: Each C_i receives $y := f(x_1, \cdots, x_n)$.

1. Each C_i computes a perfectly hiding commitment on x_i : $d_i, o_i \leftarrow \mathsf{Com}(\mathsf{cpk}, x_i)$ as well as a proof $\pi_{\mathsf{ver}}(d_i)$ on some property that x_i must meet. C_i publishes d_i and $\pi_{\mathsf{ver}}(d_i)$ on **PB**. Then, C_i sends x_i, o_i to $\mathcal{W}$ through the *secure channel*.

2. $\mathcal{W}$ runs $\mathsf{Check}_{\mathsf{ver}}$ of $\pi_{\mathsf{ver}}(d_i)$ on each d_i and aborts if one of the checks is false. $\mathcal{W}$ runs the Verify algorithm on each triple (d_i, o_i, x_i) and aborts if one verification fails. Otherwise, on inputs $x_1, ..., x_n, o_1, ..., o_n$, $\mathcal{W}$ computes $y := f(x_1, ..., x_n)$ and a proof of correctness π_{cor} of the result. Then, $\mathcal{W}$ publishes y and π_{cor} on **PB**.

3. Each C_i runs $\mathsf{Check}_{\mathsf{ver}}$ of $\pi_{\mathsf{ver}}(d_i)$ on each d_i and $\mathsf{Check}_{\mathsf{cor}}$ of π_{cor} on $(y, d_1, ..., d_n)$. If each verification accepts, then C_i accepts output y, otherwise C_i aborts.

of the protocol. This reduces the benefits of the non-interactivity of the original protocol one may seek. Since considering the realization of $\mathcal{F}^f_{\mathcal{A}_{a^*}}$ in addition of $\mathcal{F}^f_{\mathcal{A}_a}$ would not bring tremendous changes to discuss, we now focus only on the realization of $\mathcal{F}^f_{\mathcal{A}_a}$ by protocol Π^f_{PPAT}.

- We might also change this protocol slightly to provide another functionality where the evaluation y of the function is not disclosed to the clients in step 3. However, the clients receive a commitment d_y on the output y with the guarantee that $y = f(x_1, ..., x_n)$. The worker proves that he performed the computations honestly but without revealing the result. In this way, we might use the protocol as a subroutine to securely compute larger functions.

We can see that the security of the protocol in the presence of a passive adversary $\mathcal{A}_p$ rests on the fact that every piece of information present on **PB** is either perfectly hiding or zero-knowledge. However, in the presence of an active adversary $\mathcal{A}_a$ the privacy of the scheme is not guaranteed: a corrupted worker could disclose all the clients' inputs. Nevertheless, in this scenario, we assert that the verifiability property still stands. In other words, $\mathcal{A}_a$ could leak the private inputs but is not able to tamper with the correctness of the function evaluation. Thus,

protocol Π^f_{PPAT} realizes both functionalities in the presence of passive and active adversaries respectively.

Our first result shows that Protocol Π^f_{PPAT}, executed with an ideal bulletin board, realizes the functionality $\mathcal{F}^f_{\mathcal{A}_p}$ in the presence of passive adversary $\mathcal{A}_p$, and has a perfectly private audit trail.

Theorem 5.1. *Let Π_{C} be a perfectly hiding commitment scheme and π_{cor} and π_{ver} be perfect zero-knowledge proofs. Then, for any set of corrupted clients, there is a simulator such that, for any environment, the views resulting from the following two situations are identical:*

- *interacting with the bulletin board, the clients and the worker playing the Π^f_{PPAT} protocol.*
- *interacting with the ideal functionality $\mathcal{F}^f_{\mathcal{A}_p}$ and the simulator.*

The view of the environment includes its accesses to the bulletin board (controlled by the simulator in the second case), submitting the input x_i to the clients (or to $\mathcal{F}^f_{\mathcal{A}_p}$), and obtaining the outcome y in return.

Informal proof. We proceed by a set of game hops to show that the view of the environment and the adversary is indistinguishable between the real execution of the protocol and the ideal execution simulating the functionality $\mathcal{F}^f_{\mathcal{A}_p}$. The key points are that

1. the commitments published on **PB** reveal no information whatsoever about the committed values.
2. the proofs of knowledge on **PB** are perfect zero-knowledge and cannot be used by an adversary to extract information.
3. the proof π_{cor} published on **PB** computationally guarantees the soundness of the result.
4. the three first points combined form a perfectly private audit trail of the function evaluation that is computationally sound.

□

Complete proof. To prove the security of the scheme, we show that the view of any environment $\mathcal{E}$ is identical whether it is a real execution of the protocol Π^f_{PPAT} under the presence of adversary $\mathcal{A}_p$ or an ideal execution emulating the functionality $\mathcal{F}^f_{\mathcal{A}_p}$ in the presence of an adversary $\mathcal{S}$ simulating the execution of $\mathcal{A}_p$.

We create a set of games $\mathcal{G}_i$ that are indistinguishable each from the previous or the next one. The first game $\mathcal{G}_1$ representing the real

execution of the protocol and the last game $\mathcal{G}_6$ being the ideal execution emulating the functionality $\mathcal{F}^f_{\mathcal{A}_p}$.

Let $\mathsf{exec}^{\mathcal{G}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x})$ be the random variable describing the output of the environment $\mathcal{E}$ when interacting with adversary $\mathcal{A}$ and the parties in the game $\mathcal{G}$ on input $\boldsymbol{x}$. We will prove that, for any environment $\mathcal{E}$, there exists a simulator $\mathcal{S}$ such that $\mathsf{exec}^{\mathcal{G}_1}_{\mathcal{A}_p,\mathcal{E}}(\boldsymbol{x}) = \mathsf{exec}^{\mathcal{G}_6}_{\mathcal{S},\mathcal{E}}(\boldsymbol{x})$.

Without loss of generality, and for the sake of the proof, we rearrange the indexes of the clients by partitioning the set of n clients into two subsets: $\{1,...,n\} = \mathcal{HON} \cup \mathcal{COR}$ where $\mathcal{HON} = \{1,...,k\}$ indexes honest clients while $\mathcal{COR} := \{k+1,...,n\}$ indexes corrupted clients.

$\boxed{\mathcal{G}_2}$ In this game we replace the parties by dummy parties controlled by the simulator. The adversary is able to corrupt a set of clients $\mathcal{COR}$ prior to the execution.

$\mathcal{G}_1 \to \mathcal{G}_2$**:** this does not affect the view of $\mathcal{E}$.

$\boxed{\mathcal{G}_3}$ Starting from $\mathcal{G}_2$ we define a set of hybrid games denoted $\mathcal{G}_{3,i}$ for $i = 1,...,k$. In game $\mathcal{G}_{3,1}$, the proof published on **PB** by C_1 is replaced by a simulated proof. In game $\mathcal{G}_{3,i}$, the proofs published on **PB** by the C_j for $j \leq i$ are replaced by simulated proofs. At the end, we obtain game $\mathcal{G}_3 := \mathcal{G}_{3,k}$ where all the honest clients' proofs have been replaced by simulated ones. More precisely, the game $\mathcal{G}_{3,i}$ takes place as follows:

- for $j \in \mathcal{HON}$, when client C_j prepares $d_j, \pi_{\mathsf{ver}}(d_j)$ to be published on **PB**, compute a simulated proof $\pi^*_{\mathsf{ver}}(d_j)$ generated by the NIZKPK simulator S on inputs $(d_j, \mathfrak{e})$ where $\mathfrak{e}$ is a randomly generated challenge. If $j > i$, publish $(d_j, \pi_{\mathsf{ver}}(d_j))$ on **PB**. Otherwise, if $j \leq i$, publish $(d_j, \pi^*_{\mathsf{ver}}(d_j))$ on **PB**. Meanwhile, C_j sends his private input x_j as well as the opening of d_j to $\mathcal{W}$.
- for $j \in \mathcal{COR}$, when client C_j sends his inputs to $\mathcal{W}$ and publish on **PB**, proceed as in the protocol Π^f_{PPAT}.
- when all inputs values have been submitted, and if all proofs π_{ver} are valid, the worker proceeds as in the protocol Π^f_{PPAT} by computing and publishing $y := f(x_1,...,x_n)$ and π_{cor}.

$\mathcal{G}_2 \to \mathcal{G}_3$**:** We prove that for $i = 0,...,k-1$, $\mathsf{exec}^{\mathcal{G}_{3,i}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x}) = \mathsf{exec}^{\mathcal{G}_{3,i+1}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x})$ where we define $\mathcal{G}_{3,0}$ as $\mathcal{G}_2$. We proceed by induction on i.

The first step is to prove identical views between $\mathcal{G}_{3,0}$ and $\mathcal{G}_{3,1}$. From any adversary $\mathcal{A}$ and environment $\mathcal{E}$ able to distinguish between $\mathcal{G}_{3,0}$ and $\mathcal{G}_{3,1}$, we build an adversary $\mathcal{B}$ against the **perfect**

zero-knowledge property of the NIZKPK of the scheme. We claim that the advantage of $\mathcal{A}$ is at most the advantage of $\mathcal{B}$. Adversary $\mathcal{B}$ runs $\mathcal{A}$ and $\mathcal{E}$ internally and sets up a distinguisher $\mathcal{D}$. Instead of one bulletin board, $\mathcal{B}$ sets up two bulletin boards $\mathbf{PB}_0$ and $\mathbf{PB}_1$, then $\mathcal{B}$ flips a coin β. During the game, $\mathcal{E}$ and $\mathcal{A}$ only see $\mathbf{PB}_\beta$ as $\mathbf{PB}$. Then, $\mathcal{B}$ proceeds as follows:

- when client C_1 generates $x_1, o_1, d_1, \pi_{\mathsf{ver}}(d_1)$, $\mathcal{B}$ generates a simulated proof $\pi^*_{\mathsf{ver}}(d_1)$ with the NIZKPK simulator $\mathcal{S}_{\mathsf{ver}}$. Then, $(d_1, \pi_{\mathsf{ver}}(d_1))$ is published on $\mathbf{PB}_0$ while $(d_1, \pi^*_{\mathsf{ver}}(d_1))$ is published on $\mathbf{PB}_1$.
- when client C_j for $j \neq 1$ sends $d_j, \pi_{\mathsf{ver}}(d_j)$ to the board, $\mathcal{B}$ publishes $d_j, \pi_{\mathsf{ver}}(d_j)$ on $\mathbf{PB}_0$ and $\mathbf{PB}_1$.
- once all inputs have been submitted to $\mathcal{W}$, and if all proofs π_{ver} are valid, the worker prepares y and π_{cor} that are published on $\mathbf{PB}_0$ and $\mathbf{PB}_1$.

When $\mathcal{E}$ stops, $\mathcal{B}$ runs $\mathcal{D}$ on the local output of $\mathcal{E}$ and outputs whatever $\mathcal{D}$ outputs. When $\mathcal{E}$ and $\mathcal{A}$ have access to $\mathbf{PB}_0$, they are in $\mathcal{G}_{3,0}$ and when they have access to $\mathbf{PB}_1$, they are in $\mathcal{G}_{3,1}$. It follows that whenever $\mathcal{D}$ successfully distinguishes between the outputs in the two games, then $\mathcal{B}$ has a way to distinguish between the proofs $\pi_{\mathsf{ver}}(d_1)$ and $\pi^*_{\mathsf{ver}}(d_1)$. This contradicts the perfect zero-knowledge property of the NIZKPK.

Fixing some $1 < i_1 < k$, we assume that for all $1 \leq i < i_1$, we have $\mathsf{exec}^{\mathcal{G}_{3,i}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x}) = \mathsf{exec}^{\mathcal{G}_{3,i+1}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x})$. From there, we show that $\mathsf{exec}^{\mathcal{G}_{3,i_1}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x}) = \mathsf{exec}^{\mathcal{G}_{3,i_1+1}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x})$. This last equality is verified thanks to the same kind of reduction appearing in the first step of the induction. This concludes the proof by induction.

$\boxed{\mathcal{G}_4}$ We proceed similarly as in $\mathcal{G}_3$ and create a set of intermediary games $\mathcal{G}_{4,i}$ for $i = 0, ...k$ where $\mathcal{G}_{4,0}$ is defined as $\mathcal{G}_3$ and $\mathcal{G}_4 := \mathcal{G}_{4,k}$. In game $\mathcal{G}_{4,i}$, the commitments generated by the honest clients C_j for $j \leq i$ are replaced by freshly generated random commitments. In game $\mathcal{G}_4$, every commitment from the honest clients will be replaced by randomly distributed commitments. Game $\mathcal{G}_{4,i}$ for $i = 1, ..., k$ takes place as follows:

- for $j \in \mathcal{HON}$, when client C_j prepares $d_j, \pi_{\mathsf{ver}}(d_j)$ to be published on $\mathbf{PB}$, compute a commitment d^*_j and opening o^*_j on a random value x^*_j as well as two simulated proofs $\pi^*_{\mathsf{ver}}(d_j)$ and $\pi^*_{\mathsf{ver}}(d^*_j)$ respectively generated by the NIZKPK simulator $\mathcal{S}_{\mathsf{ver}}$

on inputs $(d_j, \mathfrak{e})$ and $(d_j^*, \mathfrak{e})$ respectively where $\mathfrak{e}$ is a randomly generated challenge. If $j > i$, publish $(d_j, \pi_{\mathsf{ver}}^*(d_j))$ on **PB**. Otherwise, if $j \leq i$, publish $(d_j^*, \pi_{\mathsf{ver}}^*(d_j^*))$ on **PB**. Meanwhile, if $j > i$, send x_j, o_j to $\mathcal{W}$. Otherwise if $j \leq i$, send x_j^*, o_j^* to $\mathcal{W}$.

- for $j \in \mathcal{COR}$, when client C_j sends his inputs to $\mathcal{W}$ and publishes on **PB**, proceed as in the protocol Π_{PPAT}^f.
- when all inputs values have been submitted, and if all proofs π_{ver} are valid, $\mathcal{W}$ computes and publishes $y = f(x_1^*, ..., x_i^*, x_{i+1}, ..., x_n)$ and π_{cor} on **PB**.

$\mathcal{G}_3 \rightarrow \mathcal{G}_4$**:** As in the previous game, we proceed by induction on $i = 0, ..., k-1$ to prove that $\mathsf{exec}_{\mathcal{A},\mathcal{E}}^{\mathcal{G}_{4,0}}(\boldsymbol{x}) = \mathsf{exec}_{\mathcal{A},\mathcal{E}}^{\mathcal{G}_{4,k}}(\boldsymbol{x})$. We begin by showing that $\mathsf{exec}_{\mathcal{A},\mathcal{E}}^{\mathcal{G}_{4,0}}(\boldsymbol{x}) = \mathsf{exec}_{\mathcal{A},\mathcal{E}}^{\mathcal{G}_{4,1}}(\boldsymbol{x})$. From any adversary $\mathcal{A}$ and environment $\mathcal{E}$ able to distinguish between $\mathcal{G}_{4,0}$ and $\mathcal{G}_{4,1}$, we build an adversary $\mathcal{B}$ against the **perfectly hiding property** of the commitment scheme. We claim that the advantage of $\mathcal{A}$ is at most the advantage of $\mathcal{B}$. Adversary $\mathcal{B}$ runs $\mathcal{A}$ and $\mathcal{E}$ internally and sets up a distinguisher $\mathcal{D}$. Instead of one bulletin board, $\mathcal{B}$ sets up two bulletin boards $\mathbf{PB}_0$ and $\mathbf{PB}_1$, then $\mathcal{B}$ flips a coin β. During the game, $\mathcal{E}$ and $\mathcal{A}$ only see $\mathbf{PB}_\beta$ as **PB**. Then, $\mathcal{B}$ proceeds as follows:

- when client C_1 generates $x_1, o_1, d_1, \pi_{\mathsf{ver}}(d_1)$, $\mathcal{B}$ generates a commitment d_1^* and opening o_1^* on a random value x_1^* as well as two simulated proofs $\pi_{\mathsf{ver}}^*(d_1)$ and $\pi_{\mathsf{ver}}^*(d_1^*)$ with the NIZKPK simulator $\mathcal{S}_{\mathsf{ver}}$. Then, $(d_1, \pi_{\mathsf{ver}}^*(d_1))$ is published on $\mathbf{PB}_0$ while $(d_1^*, \pi_{\mathsf{ver}}^*(d_1^*))$ is published on $\mathbf{PB}_1$. Meanwhile, if $\beta = 0$, $\mathcal{W}$ receives x_1, o_1, else if $\beta = 1$, $\mathcal{W}$ receives x_1^*, o_1^*.
- for $j \in \mathcal{HON}, j \neq 1$, when client C_j generates x_j, o_j, d_j, $\pi_{\mathsf{ver}}(d_j)$, $\mathcal{B}$ generates a simulated proof $\pi_{\mathsf{ver}}^*(d_j)$ thanks to the NIZKPK simulator $\mathcal{S}_{\mathsf{ver}}$. Then, $(d_j, \pi_{\mathsf{ver}}^*(d_j))$ is published on $\mathbf{PB}_0$ and on $\mathbf{PB}_1$. Meanwhile, x_j, o_j are sent to $\mathcal{W}$.
- for $j \in \mathcal{COR}$, when client C_j generates $x_j, o_j, d_j, \pi_{\mathsf{ver}}(d_j)$, $\mathcal{B}$ proceeds as in game $\mathcal{G}_3$ and $d_j, \pi_{\mathsf{ver}}(d_j)$ is published on $\mathbf{PB}_0$ and $\mathbf{PB}_1$.
- once all inputs have been submitted to $\mathcal{W}$, and if all proofs π_{ver} are valid, the worker prepares y and π_{cor} to be published on $\mathbf{PB}_0$ and $\mathbf{PB}_1$.

When $\mathcal{E}$ stops, $\mathcal{B}$ runs $\mathcal{D}$ on the local output of $\mathcal{E}$ and outputs whatever $\mathcal{D}$ outputs. When $\mathcal{E}$ and $\mathcal{A}$ have access to $\mathbf{PB}_0$, they

are in $\mathcal{G}_{4,0}$ and when they have access to $\mathbf{PB}_1$, they are in $\mathcal{G}_{4,1}$. The output of the distinguisher $\mathcal{D}$ is released by $\mathcal{B}$ and it comes that the success of $\mathcal{D}$ at differentiating between $\mathcal{G}_{4,0}$ and $\mathcal{G}_{4,1}$ is used by $\mathcal{B}$ to differentiate commitment d_1 from commitment d_1^*. This contradicts the perfectly hiding property of the commitment scheme.

As before, if we assume that for a fixed $1 < i_1 < k$, we have for all $1 \leq i < i_1$ that $\mathsf{exec}_{\mathcal{A},\mathcal{E}}^{\mathcal{G}_{4,i}}(\boldsymbol{x}) = \mathsf{exec}_{\mathcal{A},\mathcal{E}}^{\mathcal{G}_{4,i+1}}(\boldsymbol{x})$, we can prove that $\mathsf{exec}_{\mathcal{A},\mathcal{E}}^{\mathcal{G}_{4,i_1}}(\boldsymbol{x}) = \mathsf{exec}_{\mathcal{A},\mathcal{E}}^{\mathcal{G}_{4,i_1+1}}(\boldsymbol{x})$ by using the same kind of reduction as in the first step. Hence, we have showed that $\mathsf{exec}_{\mathcal{A},\mathcal{E}}^{\mathcal{G}_3}(\boldsymbol{x}) = \mathsf{exec}_{\mathcal{A},\mathcal{E}}^{\mathcal{G}_4}(\boldsymbol{x})$.

$\boxed{\mathcal{G}_5}$ In this game, we replace the proof π_{cor} generated by $\mathcal{W}$ by a simulated proof π_{cor}^* that is obtained through the NIZKPK simulator $\mathcal{S}_{\mathsf{cor}}$ on inputs y (output by $\mathcal{W}$) and the commitments presents on **PB**.

$\mathcal{G}_4 \to \mathcal{G}_5$**:** From any adversary $\mathcal{A}$ and environment $\mathcal{E}$ able to distinguish between $\mathcal{G}_4$ and $\mathcal{G}_5$, we build an adversary $\mathcal{B}$ against the perfect zero-knowledge property of the NIZKPK of the scheme. The reduction works as the reductions in the proof that $\mathsf{exec}_{\mathcal{A},\mathcal{E}}^{\mathcal{G}_2}(\boldsymbol{x}) = \mathsf{exec}_{\mathcal{A},\mathcal{E}}^{\mathcal{G}_3}(\boldsymbol{x})$ yielding an adversary $\mathcal{B}$ able to break the **perfect zero-knowledge property** of the NIZKPK of the scheme.

$\boxed{\mathcal{G}_6}$ In the last game, we build the simulator $\mathcal{S}$ that proceeds as follows:

- When notified by the functionality $\mathcal{F}_{\mathcal{A}_p}^f$ of an input from an honest client C_i, $\mathcal{S}$ generates a commitment d_i^* and opening o_i^* on a random value x_i^*, and computes a simulated proof $\pi_{\mathsf{ver}}^*(d_i^*)$ with the NIZKPK simulator $\mathcal{S}_{\mathsf{ver}}$. Then $\mathcal{S}$ publishes d_i^* and $\pi_{\mathsf{ver}}^*(d_i^*)$ on **PB**.
- When a corrupted client C_j submits a commitment and a proof to the board, and the opening of the commitment to the worker, $\mathcal{S}$ submits the opened input x_j to $\mathcal{F}_{\mathcal{A}_p}^f$ on behalf of C_j.
- When all input values have been submitted, and if all π_{ver} proofs are valid, $\mathcal{S}$ submits Compute to $\mathcal{F}_{\mathcal{A}_p}^f$ and obtains y in return. $\mathcal{S}$ uses the NIZKPK simulator $\mathcal{S}_{\mathsf{cor}}$ to obtain a simulated proof π_{cor}^* based on the commitments on the board and y. Then $\mathcal{S}$ publishes y and π_{cor}^* on the board.

$\mathcal{G}_5 \to \mathcal{G}_6$**:** This does not change the view of the environment, thus we have that $\mathsf{exec}^{\mathcal{G}_6}_{\mathcal{S},\mathcal{E}}(\boldsymbol{x}) = \mathsf{exec}^{\mathcal{G}_5}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x})$.

□

Our second result shows that Protocol Π^f_{PPAT}, executed with an ideal bulletin board, realizes the functionality $\mathcal{F}^f_{\mathcal{A}_a}$ in the presence of an active adversary $\mathcal{A}_a$ who controls the worker.

Theorem 5.2. *Let Π_{C} be a binding commitment scheme and π_{cor} and π_{ver} be computationally sound proofs. Then, for a corrupted worker and any set of corrupted clients, there is a simulator such that, for any environment, the views resulting from the following two situations are indistinguishable:*

- *interacting with the bulletin board, the clients and the corrupted worker playing the Π^f_{PPAT} protocol.*
- *interacting with the ideal functionality $\mathcal{F}^f_{\mathcal{A}_a}$ and the simulator.*

The view of the environment includes its accesses to the bulletin board (controlled by the simulator in the second case), submitting the input x_i to the clients (or to $\mathcal{F}^f_{\mathcal{A}_a}$), and obtaining the outcome y in return, and every communication that the corrupted worker would make.

Informal proof. The demonstration follows the same pattern of Theorem 5.1 with the major difference that the privacy of the inputs is no longer guaranteed due to the adversary ability to corrupt the worker. However, the soundness of the proof π_{cor} remains ensuring the correctness of the result. We show that it is still essential that **PB** displays the computationally binding commitments of the clients. This condition enforces an adversary to produce a proof of knowledge on these commitments. As a result the audit trail present on **PB** is not perfectly hiding anymore but remains computationally sound. □

Complete proof. We proceed similarly to the proof of Theorem 5.1 by a set of game hops. By contrast from $\mathcal{F}^f_{\mathcal{A}_p}$ of Theorem 5.1, we observe that $\mathcal{F}^f_{\mathcal{A}_a}$ offers correctness guarantees, but no privacy at all: all inputs are immediately given to the simulator.

Formally, we prove that there exists a simulator $\mathcal{S}$ such that the view for any environment $\mathcal{E}$ of an execution of the protocol Π^f_{PPAT} under the presence of an active adversary $\mathcal{A}_a$ is indistinguishable from the view of $\mathcal{E}$ of an execution of an ideal protocol simulated by $\mathcal{S}$ and interacting with the functionality $\mathcal{F}^f_{A_a}$. In other words, we show that $\mathsf{exec}^{\mathcal{G}_1}_{\mathcal{A}_a,\mathcal{E}}(\boldsymbol{x}) \approx$

$\mathsf{exec}^{\mathcal{G}_3}_{\mathcal{S},\mathcal{E}}(\boldsymbol{x})$ where the game $\mathcal{G}_1$ represents the execution in the real world of Π^f_{PPAT} while the game $\mathcal{G}_3$ is the ideal execution and $\boldsymbol{x}$ is the input of the parties.

As previously, and without loss of generality, we partition and re-index the set $\{1, ..., n\}$ of the clients into the subsets $\mathcal{HON} \cup \mathcal{COR}$ where $\mathcal{HON} := \{1, ..., k\}$ is the set of honest clients and $\mathcal{COR} := \{k+1, ..., n\}$ is the set of corrupted clients.

$\boxed{\mathcal{G}_2}$ In this game we replace the parties by dummy parties controlled by the simulator. The adversary is able to corrupt a set of clients $\mathcal{COR}$ prior to the execution.

$\mathcal{G}_1 \to \mathcal{G}_2$**:** this does not affect the view of $\mathcal{E}$.

$\boxed{\mathcal{G}_3}$ In this game, we describe how the simulator $\mathcal{S}$ interacts with $\mathcal{F}^f_{\mathcal{A}_a}$ in order to emulate the ideal execution of the protocol. $\mathcal{S}$ internally runs the corrupted worker and clients, relays their communications with the environment, and intercepts their communications with the bulletin board which is also emulated by the simulator. The simulator proceeds as follows:

- When notified by the functionality of an input x_i from an honest client C_i, $\mathcal{S}$ generates and posts on the board a commitment d_i and a proof $\pi_{\mathsf{ver}}(d_i)$, following Π^f_{PPAT}.
- When a corrupted client C_j submits a commitment and a proof to the board, and the opening of the commitment to the worker, $\mathcal{S}$ submits the opened input x_j to $\mathcal{F}^f_{\mathcal{A}_a}$ on behalf of C_j.
- When the corrupted worker submits the result y and the associated proof π_{cor}, verify each of the proofs on behalf of the honest clients and, if they check, $\mathcal{S}$ submits $\mathsf{Compute}$ to the functionality so that it provides its output to the clients.

$\mathcal{G}_2 \to \mathcal{G}_3$**:** By following the real protocol, we provide the environment with a view that is identical to the real one. However, as the adversary now controls $\mathcal{W}$, it is possible for him to publish $(y^*, \pi_{\mathsf{cor}})$ on **PB** such that $y^* \neq y := f(x_1, ..., x_n)$ and $\mathsf{Check}_{\mathsf{cor}}(y^*, d_1, ..., d_n)$ of π_{cor} outputs 1. This can happen only in three cases:

1. the adversary breaks the **soundness property** of π_{ver}. In this case, $\mathcal{A}$ is able to prove false statements over commitments on **PB**.

2. the adversary breaks the **computationally binding property** of the commitment scheme. In this case, $\mathcal{A}_a$ is able to open commitments to any value. For example, $\mathcal{A}_a$ produces commitments d_j on both $x_j \neq x'_j$ with respective openings o_j, o'_j, such that $\mathsf{Verify}(d_j, o_j, x_j) = 1$ and $\mathsf{Verify}(d_j, o'_j, x'_j) = 1$ where $j \in J \subset \{1, ..., n\}$. This allows a corrupted worker to output $y^* \neq y$ such that $y^* = f(x_1^*, ..., x_n^*)$ where $x_j^* = x'_j$ if $j \in J$ and $x_j^* = x_j$ otherwise. In such scenario, $\mathcal{W}$ is able to output π_{cor} such that $\mathsf{Check}_{\mathsf{cor}}(y^*, d_1, ..., d_n)$ accepts.
3. the adversary breaks the **soundness property** of the π_{cor} proof. In this case, $\mathcal{A}_a$ is able to prove false statements about the correctness of the result.

Nevertheless, each situation only occurs with negligible probability.

□

5.3 Building blocks for perfectly private audit trail

The interactions between the clients and the worker involve the exchange of private inputs and the publication on a Public Bulletin Board **PB** of some trail that will be used to perform further audit of the process.

Commitment consistent encryption

We rely on the CCE primitive introduced in Chapter 3 to encrypt the private inputs of the clients and extract from the ciphertext a commitment that will be published on **PB**. This mechanism is similar to the one explained in Chapter 4. However, within the context of e-voting, an homomorphic encryption scheme that allows threshold decryption is mandatory while in other settings, the encryption scheme could be superfluous when using a physically secure channel between the clients and the workers. In this last case, we may be just fine with a commitment scheme alone. However, in most cases, we are in an intermediate situation where the inputs are sent to the worker through a not-so-secure network where encryption is not a luxury. For this reason a CCE scheme $\Pi := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{DerivCom}, \mathsf{Open}, \mathsf{Verify})$ comes in handy. For our construction, we consider a IND-CPA secure CCE scheme. Note that, given the proof π_{ver} published on **PB** along with the commitment, we

can apply Theorem 3.2 on our validity augmented CCE scheme to find our scheme to be NM-CPA secure.

We note that, when encryption is used instead of a secure channel, we must make sure that the adversary cannot submit inputs that he actually ignores by copying CCE ciphertexts. This can be prevented by using the non-malleability offered by sigma proofs to prevent any re-randomization of commitments, and by declaring duplicate commitments invalid (see the validity augmentation of Section 3.2 for details).

In the following, let the commitment obtained through the DerivCom algorithm on a CCE ciphertext be a perfectly hiding and computationally binding *homomorphic*[1] commitment as it is the case for a traditional Pedersen commitment. We suppose that from Π, we obtain a commitment algorithm $\mathsf{Com}(\mathsf{pk}, .)$ and that $\mathsf{Gen}(1^\lambda)$ outputs the public key pk which specifies a group G of order q, a λ-bit long safe prime, as well as randomly chosen g and h such as $\langle g \rangle = \langle h \rangle = G$. As a result of the random choices of g and h, $\log_h g$ is unknown. The message and the randomness spaces are equal, $\mathbf{M} = \mathbb{R} = \mathbb{Z}_q$. The commitment algorithm $\mathsf{Com}(\mathsf{pk}, m)$ outputs $c = g^m h^r$ and $a = r$ for a randomly chosen $r \in \mathbb{R}$. Finally the verification algorithm $\mathsf{Verify}_{\mathsf{cpk}}(c, a, m)$ outputs 1 if $c = g^m h^a$ and 0 otherwise.

Non interactive zero-knowledge proof of knowledge

The second tool that we need is non-interactive zero-knowledge proof of knowledge (NIZKPK) that is presented in Section 2.1.6. Below we explicit the different proofs used in our construction. We present the Σ-protocols that could easily be turned into NIZKPK by Fiat-Shamir/Blum transformation. Note that these NIZKPK are performed mostly entirely on the commitments, and for this reason we give their description considering commitments as statements. For readability, we do not differentiate the commitments obtained through the DerivCom algorithm from other commitments produced in the following proofs, and, without loss of generality, we assume they are all obtained through the Com algorithm of some commitment homomorphic scheme.

The first NIZKPK is the classical or-proof of knowledge [CDS94] denoted $\pi_{\mathsf{or}}(d)$. Another kind of well-known NIZKPK is the proof of equality of discrete logarithms between two commitments [CP93] that we refer to as $\pi_{\mathsf{DL}}(d_1, d_2)$ and presented in Protocol 2.2. Finally, the proof of the opening of the commitment is denoted $\pi_{\mathsf{ope}}(d)$. This last proof is used by default to enforce non-malleability of the commitments posted

[1] This property introduced for encryption schemes in Definition 2.5 is easily transposable for commitment schemes.

on **PB** by the clients and when no specific property is required on the private input. We give the relations for these proofs respectively:

$$R^{\mathsf{or}} := \{(d,(o,x))|\mathsf{Verify}(d,o,x) = 1 \wedge (x = 0 \vee x = 1)\}$$
$$R^{\mathsf{DL}} := \{((d_1,d_2),(d,o_1,o_2))|\mathsf{Verify}(d_1,o_1,x) = 1 \wedge \mathsf{Verify}(d_2,o_2,x) = 1\}$$
$$R^{\mathsf{ope}} := \{(d,(o,x))|\mathsf{Verify}(d,o,x) = 1\}$$

To achieve public verification of the audit trail, we may need to perform arithmetic operations on the commitments, more precisely operations on the values committed to in the commitments present on **PB**. If the commitment scheme is homomorphic - which is the case here - we only have one operation at disposal, usually addition. To allow the public verifiers to perform multiplications on the commitments, the Prover needs to add an NIZKPK each time a multiplication occurs.

NIZKPK for multiplication. We rely on the Σ-protocol presented in [DF02] to prove a multiplicative relation between commitments. More precisely, we define

$$R^{\mathsf{mul}} := \{((d_1,d_2,d_3),(o_1,x_1,o_2,x_2,o_3))|x_3 = x_1x_2 \wedge_i \mathsf{Verify}(d_i,o_i,x_i) = 1\}$$

This following protocol denoted $\pi_{\mathsf{mul}}(d_1,d_2,d_3)$ is presented in [DF02] for integer commitments on groups with hidden order but it can easily be translated to our case.

The inputs of Prover and Verifier are the commitments $d_i = g^{x_i}h^{r_i}$ for $i = 1,2,3$. In addition, Prover is given the values x_i, r_i for $i = 1,2,3$. Obviously, both parties have access to the public parameters. In the following Σ-protocol, Prover aims to convince Verifier that $x_3 = x_1x_2$. We can see that since $d_3 = d_1^{x_2}h^{r_3 - x_2r_1}$ is a commitment on the value x_2 using d_1 as base instead of g, if Prover can convince Verifier that d_2 and d_3 commit on the same value and that he can open d_1, then the case is closed.

1. Prover chooses at random $\bar{x}_1, \bar{x}_2, \bar{r}_1, \bar{r}_2, \bar{r}_3 \in \mathbb{Z}_q$ and computes $d_1^* = g^{\bar{x}_1}h^{\bar{r}_1}$, $d_2^* = g^{\bar{x}_2}h^{\bar{r}_2}$ and $d_3^* = d_1^{\bar{x}_2}h^{\bar{r}_3}$. Then Prover sends $\mathfrak{a} = (d_1^*, d_2^*, d_3^*)$ to Verifier.

2. Verifier randomly picks a challenge $\mathfrak{e}$ and sends it to Prover.

3. Prover computes and sends to Verifier: $\mathfrak{z} := (y_1, y_2, s_1, s_2, s_3)$ where $y_1 = \bar{x}_1 + ex_1$, $y_2 = \bar{x}_2 + ex_2$, $s_1 = \bar{r}_1 + er_1$, $s_2 = \bar{r}_2 + er_2$ and $s_3 = \bar{r}_3 + e(r_3 - x_2r_1)$.

4. Finally Verifier parses $\mathfrak{z}$ as $(y_1, y_2, s_1, s_2, s_3)$ and then checks that $g^{y_1}h^{s_1} = d_1^* d_1^e$, $g^{y_2}h^{s_2} = d_2^* d_2^e$ and that $d_1^{y_2}h^{s_3} = d_3^* d_3^e$.

Note that the soundness of the proof relies on the binding property of the commitment scheme since d_3 could theoretically open to any value. While keeping this in mind, we use the notation $\odot$ to denote the multiplicative operator between two commitments.

NIZKPK for interval membership (range proof). More complex algorithms could be obtained with a comparison operator at hand. This operator is achieved by using range proofs over commitments: The main idea of the proof is to decompose a commitment into several commitments respectively to a binary decomposition. Then, the Prover computes a π_{or} on each commitment. The verification of the proof is the verification of each of the π_{or} and the recombination of the commitment decomposition. However the binary decomposition is a common possibility, many refinements are available to optimize the cost of the proof when the intervals are not power of 2. The work of Canard et al. [CCJT14] provides an interesting comparison between the different methods as well as an original solution.

The comparison operator is achieved by using range proofs over commitments for the relation:

$$R^{\mathsf{ran}} := \{(d, (o, x)) | \mathsf{Verify}(d, o, x) = 1 \wedge x \in I\}$$

We denote this proof by $\pi_{\mathsf{ran}}(d, I)$. A direct way to obtain a range proof that $x \in [0, 2^{k+1}[$ is by composing k proofs $\pi_{\mathsf{or}}(x_i)$ where x_i is the binary decomposition of x.

The inputs of Prover and Verifier is the commitment $d = g^x h^r$ and a binary decomposition base $b_0, \cdots, b_k$ where $b_i = 2^i$. In addition, Prover is given the values x, r where $x \in I = [0, 2^{k+1}[$. The value x can be written as $x = \sum b_i x_i$ where $x_i \in \{0, 1\}$ for $i = 0, \cdots, k$.

1. Prover chooses at random $r_i \in \mathbb{Z}_q$ for $i = 0, \cdots, k-1$ and computes $d_i = g^{x_i} h^{r_i}$ for $i = 0, \cdots, k$ where $r_k = (r - \sum_{i=0}^{k-1} r_i b_i) b_k^{-1}$. Prover sends each d_i to Verifier and then engages a $\pi_{\mathsf{or}}(d_i)$ with Verifier for $i = 0, \cdots, k$.

2. In addition to participating in each $\pi_{\mathsf{or}}(d_i)$, Verifier checks that $d = \prod_{i=0}^{k} d_i^{b_i}$. If each proof and the recomposition of the commitment are accepted, Verifier accepts the proof.

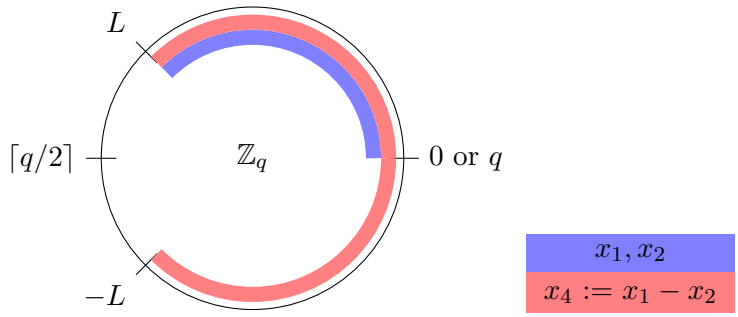


Figure 5.1: Intervals for the comparison proof.

A proof π_{ran} can be used to build a comparison operator between commitments[2] for which we define the relation:

$$R^{<} := \{((d_1, d_2, d_3), (o_1, x_1, o_2, x_2, o_3, x_3)) | x_3 = x_1 < x_2 \\ \wedge_i \mathsf{Verify}(d_i, o_i, x_i) = 1\}$$

To achieve that proof, given d_1, d_2, we compute a $\pi_{\mathsf{ran}}(d_4,]-L, L[)$ where $d_4 = \mathsf{Com}(x_4) := \mathsf{Com}(x_1) \ominus \mathsf{Com}(x_2)$. Indeed, if $x_1, x_2 \in [0, L[\subset \mathbb{Z}_q$ where $L < \lceil q/2 \rceil$ to avoid overlap, then $x_4 := x_1 - x_2 \in [-L, L[= [-L, 0[\cup [0, L[$ as shown in Figure 5.1. Given that $b_0, \cdots, b_k$ is the base for the binary decomposition of a value in $[0, L[$, we can add the extra base element $b_{k+1} = q - L$. Then x_4 admits a binary decomposition in this new base : $x_4 = \sum_{i=0}^{k+1} x_{4,i} b_i$ where $x_{4,i} \in \{0, 1\}$ for $i = 0, \cdots, k+1$. Moreover, $x_{4,k+1} = x_3$ thus $\pi_{\mathsf{ran}}(d_4, [-L, L[)$ provides the commitment d_3. As for the multiplicative operator, we use the notation for the comparison operator $\lessdot$ between commitments, we write $\mathsf{Com}(x_3) = \mathsf{Com}(x_1) \lessdot \mathsf{Com}(x_2)$.

Consistency proof. This last kind of proof denoted π_{cc} and already introduced in Section 3.2 is a part of the validity augmented CCE scheme which enforces its consistency. It allows the worker to verify that a $c := \mathsf{Enc}(x)$, the CCE of x contains a commitment that commits on the same value x that is encrypted. The protocol relation is:

$$R^{\mathsf{cc}} := \{(c, (o, x)) | c \leftarrow \mathsf{Enc}(\mathsf{pk}, x) \wedge \mathsf{Verify}(\mathsf{pk}, \mathsf{DerivCom}(\mathsf{pk}, c), o, x) = 1\}$$

[2]Since committed values belong to $\mathbb{Z}_q$, this comparison operator makes sense only on a small interval of $\mathbb{Z}_q$ where we can define a natural order. Typically an interval centred in $0 \in \mathbb{Z}_q$.

In our setting however, this relation needs not to be proven through a NIZKPK since the worker knows the message and the opening of the CCE ciphertext once he has decrypted and opened it. In any case, the worker should always check the consistency of the ciphertext to protect himself from two situations: first, not being able to compute an output consistent with the commitments present on **PB**. Second, not being able to compute a proof of correctness π_{cor} of the result.

5.4 Generic construction of the protocol

Commitment consistent encryption and non-interactive zero-knowledge proofs of knowledge are the building blocks of Protocol Π^{f}_{PPAT} which realizes the ideal functionalities in the presence of passive and active adversaries as discussed in Section 5.2. This also implies a generic construction of the correctness proof π_{cor} that allows the clients to accept or reject the result of the function evaluation. This proof is prepared in the commitment space that is defined in the Gen algorithm of the CCE scheme Π.

The commitment space is homomorphic to the message space for the addition operator. In the previous paragraphs, we allowed complex computations over commitments, given the ad hoc assistance of a Prover for multiplications and comparisons. Consequently, the set of arithmetic operators over the message space $\mathbf{M}$ is given by $S := (+, -, \cdot, <)$, and finds its counterpart set $S' := (\oplus, \ominus, \odot, \oslash)$ over the commitment space. As a result, each operator of S used to evaluate the function f can be performed over commitments by an operator of S'. In this way, we obtain a perfectly private audit trail on **PB**.

The protocol is fairly simple: each client computes a $c_i \leftarrow \mathsf{Enc}(\mathsf{pk}, x_i)$ of his private input. From c_i, C_i derives a perfectly hiding commitment $d_i \leftarrow \mathsf{DerivCom}(\mathsf{pk}, c_i)$ and, computes a $\pi_{\mathsf{ver}}(d_i)$, for example $\pi_{\mathsf{or}}(d_i)$, $\pi_{\mathsf{ran}}(d_i, I)$ or by default, $\pi_{\mathsf{ope}}(d_i)$. Then, C_i publishes d_i and $\pi_{\mathsf{ver}}(d_i)$ on **PB**, and sends c_i to $\mathcal{W}$. After having decrypted every c_i to get the private inputs of every client, $\mathcal{W}$ computes $y := f(x_1, \cdots, x_n)$. From the commitments published on **PB**, $\mathcal{W}$ computes $d_y := f(d_1, \cdots, d_n)$ as well as the NIZKPK-s for each operator of f (except for $\oplus$ and $\ominus$ which are natural operators in the commitment space): the set of the NIZKPK and all the intermediary commitments created for the needs of the proofs are executed in parallel to form π_{cor}. $\mathcal{W}$ publishes y and π_{cor} on **PB**. Finally, each C_i verifies the correctness of π_{cor} in regards to y and the reconstruction of d_y.

As we will see in the next section, this is not the most efficient way

to achieve the perfectly private audit trail since there are lots of cases where the verification of the output of the function can be done without recomputing the entire function in the commitment space.

5.5 Applications

Until now, we have seen how to generate a perfectly private audit trail of computation from the blueprint of any function. As we will see through several examples, there is a more direct way to provide the π_{cor} that guarantees the correctness of the output. The main idea is that, once given the result of the function it is much simpler to verify that it is correct. For example, once you are told that 8128 is the square root of $66,064,384$, it costs you only one squaring to agree while finding the square root by hand calculus is trickier.

In the following applications, we show how to use this trick to reduce the complexity of the proof for the client compared with the original complexity of the algorithm computing the result as it must be done in classical SMC. We selected unrelated problems to illustrate the ease of application of our technique and we conclude by pointing out in Section 5.5.4 other examples that may turn into good candidates.

5.5.1 System of linear equations

The first test application is solving a system of linear equations. It is involved as a subroutine in many algorithms as, among others, in the Lagrange polynomial interpolation or in linear programming techniques. In linear programming, the goal is to optimize a solution under a set of linear constraints. This kind of scenario fits very well in a multi-party setting. We illustrate it by an example in which a set of companies in a production line agree to cooperate in order to optimize the production of some goods but do not wish to divulge their internal work flow to each other. The gain for the companies is lower costs and the ability to reallocate resources. Nowadays, all the solutions impair the privacy in one way or another, thus preventing such benefits. We propose a solution to solve system of linear equations as a building block to, for example, find solutions of linear programming problems.

Consider the following system of linear equations L in the coefficient

space $\mathbf{M}$:

$$L \equiv \begin{cases} \alpha_{1,1}z_1 + \cdots + \alpha_{1,n}z_n & = b_1 \\ \vdots & \vdots \\ \alpha_{m,1}z_1 + \cdots + \alpha_{m,n}z_n & = b_m \end{cases}$$
$$\Leftrightarrow AZ = B$$

where $A \in \mathbf{M}^{m\times n}$, $B \in \mathbf{M}^{m\times 1}$ and Z is the matrix of variables with dimension $(n \times 1)$. The unique solution, if it exists, is $Z_s = A^{-1}B$. When the matrix is not invertible, we might produce Z_{nts} a non trivial solution of the homogeneous system $AZ = 0$.

In a multi-party setting, the constraints are given by the clients. The simplest scenario is that client C_i knows the private input α_i for $i = (1,1), \cdots, (m,n)$ and the independent coefficients b_j for $j = 1, \cdots, m$, are known to everyone.

Each C_i computes $c_i \leftarrow \mathsf{Enc}(\mathsf{pk}, \alpha_i)$ and derives from it a commitment on α_i: $d_i \leftarrow \mathsf{DerivCom}(\mathsf{pk}, c_i)$. The d_i-s are published on **PB** as well as a proof $\pi_{\mathsf{ope}}(d_i)$ computed by C_i. Meanwhile, the encryptions c_i are passed on to $\mathcal{W}$. We can combine each d_i on **PB** to form the matrix D which can be seen as a commitment on A. After having received and decrypted each c_i to get α_i, $\mathcal{W}$ computes the inverse matrix A^{-1} with his favourite method and thus the solution $Z_s = A^{-1}B$. The worker then publishes Z_s on **PB** along with π_{cor}. This proof consists of a list of m openings $o_1, \cdots, o_m$ where o_j is the opening of $b'_j = d_{j,1}z_1 \oplus \cdots \oplus d_{j,n}z_n$. Indeed, to verify the result, each client computes $B' := DZ_s$ and checks that the opening of each entry of this $(m,1)$-matrix is valid and that B' opens on the values of B. In the case of a non trivial solution Z_{nts} occurring when A is not invertible, the worker opens $B' := DZ_{nts}$ which must be a series of commitments on zero.

One might also want to include B in the private inputs of the clients. In this case, the π_{cor} is a bit different. Instead of giving the openings o_j, $\mathcal{W}$ provides a $\pi_{\mathsf{DL}}(b'_j, b_j)$ that b'_j commits on the same value as b_j for $j = 1, \cdots, m$.

π_{cor} **is Zero-Knowledge:**

Completeness It is clear that, given $o_1, ..., o_m$, any client can verify that Z_s is indeed the solution of the linear system.

Soundness This relies on the computationally binding property of the commitment scheme.

Perfect ZK As the commitment scheme is perfectly hiding, the openings of the commitments must be uniformly distributed in the space of openings.

Complexity. For the client, the complexity cost is exactly linear in the number of clients. In fact, the complexity bottleneck of the protocol is how to find of Z. Either by inverting A, by the Gauss-Jordan elimination or more efficiently by the LU decomposition method, computing Z_s requires about $O(n^3)$ operations (or $O(n^{2.373})$ with the best current algorithm). It is also noteworthy that these algorithms often require branching when for example, searching the pivot in a row. However, when performed in SMC, these operations may become costly.

5.5.2 Auctions

Another type of problems that benefits from our approach is electronic auctions. We consider a setting of simple auctions where n clients submit one bid each. The result of the auction consists of a list of the sorted bids. More precisely, each C_i computes $c_i \leftarrow \mathsf{CCE}(x_i)$ where the bid $x_i \in I = [0, L[$. From c_i, C_i derives $d_i \leftarrow \mathsf{DerivCom}(\mathsf{pk}, c_i)$ and computes $\pi_{\mathsf{ran}}(d_i, I)$. While c_i is sent to $\mathcal{W}$, each C_i publishes d_i and $\pi_{\mathsf{ran}}(d_i, I)$ on **PB**. $\mathcal{W}$ computes the sorted list $(x'_1, \cdots, x'_n)$ from $(x_1, \cdots, x_n)$ with his favourite algorithm. From the sorted list, $\mathcal{W}$ rearranges the d_i to produce a sorted list of commitments $d'_1, \cdots, d'_n$. Then, $\mathcal{W}$ computes $n-1$ commitments $e_1, \cdots, e_{n-1}$ where $e_i := d'_i \circledcirc d'_{i+1}$. As explained in Section 5.3, the $\circledcirc$ operator comes with $n-1$ proofs $\pi_{\mathsf{ran}}(d'_i - d'_{i+1},]-L, L[)$ denoted π_i. Thus, $\pi_{\mathsf{cor}} := ((e_1, o_1, \pi_1) \wedge \cdots \wedge (e_{n-1}, o_{n-1}, \pi_{n-1}))$ where o_i is an opening of e_i which must imply that e_i commits on 1. $\mathcal{W}$ publishes π_{cor} on **PB** along with $d'_1, \cdots, d'_n$. Then, each client verifies π_{cor} to validate the result of the auctions.

Note that, while there is a strong guarantee for the client that the winner(s) of the auction are legitimate winner(s), the winning bid and every other bid remain perfectly private. However, if required, $\mathcal{W}$ could also have revealed the winning bids by publishing the openings of the commitments. It is possible to transform this protocol into a sorting protocol that does not reveal which client's bid arrives in which position. This can be done, first, by randomizing the commitments on **PB** and, then, providing a proof of shuffle that the sorted list of commitments comes from the randomized list. The work of Terelius and Wikström [Wik09, TW10] proposes such an efficient technique that adapts our commitment consistent approach. This method is detailed in Section 4.4.2 for the complex ballot voting scheme with verifiable shuffling. In this way, we could use the protocol as a subroutine of an algorithm that needs sorting.

π_{cor} is Zero-Knowledge: This is clear since π_{cor} is formed by NIZKPK.

Complexity. As in the case of the linear system, the complexity for the client is linear in the number of clients. For the worker though, the highest complexity comes from the sorting algorithm (for example $O(n\log(n))$ in the case of Quicksort). However, as it is done on clear values, the cost is marginal compared with the linear number of range proofs to compute.

5.5.3 Shortest path

In this third example, we aim at showing that realizing more complex protocols can be carried out without much difficulty. In the case of the shortest path, we consider a directed graph with m vertices and n weighted edges. The goal is to find the shortest path from a source node to a target node which is the path that minimizes the sum of the weights. We denote by $v_1, \cdots, v_m$ the vertices, while $e_k^{i,j}$ is the edge from v_i to v_j, numbered k in the edges list. Similarly, we denote $w_k^{i,j}$ the positive weight of edge $e_k^{i,j}$.

This problem has been studied in SMC since it offers a potential solution to privacy-preserving GPS guidance in which the guided person does not want to reveal its location to the service provider. In a multi-party setting involving more than two players, we can imagine the following scenario. A set of competing delivering companies possessing connections with spare room available for goods might be appealed to work together to offer a joined transport solution to a client without disclosing private information. We provide more detailed examples of applications of the shortest path in Chapter 6.

The Bellman-Ford algorithm solves the shortest path problem in $O(mn)$ operations. This algorithm maintains two lists : a list of predecessors ***pred*** and a list of distances ***dist***. While the algorithm executes, $pred_i$ designates the predecessor vertex of v_i while $dist_i$ stores the distance of v_i which is the weight of the current shortest path from the source vertex to v_i.

In a multi-party setting we assume that each client C_i has w_i as private input. It is possible to turn Bellman-Ford algorithm into its secure version using classical SMC or using our technique. As previously, the derived commitments $d_i \leftarrow \mathsf{DerivCom}(\mathsf{pk}, c_i)$ are published on **PB**, while $\mathcal{W}$ gathers the $c_i \leftarrow \mathsf{Enc}(\mathsf{pk}, w_i)$ of the clients' private inputs. Then $\mathcal{W}$ decrypts and computes the shortest path. The algorithm requires a supremum value denoted $\top$ which is the maximum weight a path might have plus one. We define $\top$ as $n.L - n + 1$, where L is the bound on the weights of the edges: $w_i < L$. As a result, we also require C_i to publish with d_i, a $\pi_{\mathsf{ran}}(d_i, L)$ on **PB**. This proof must be verified later by the

other clients and $\mathcal{W}$.

Let us now focus on the computation of π_{cor} by $\mathcal{W}$. This is done by computing Algorithm 5.4 on the commitments and by providing an NIZKPK when necessary.

Algorithm 5.4: Secure shortest path based on Bellman-Ford's algorithm

Input: A graph $G = (V, E)$ where V is the list of vertices $v_1, \cdots, v_m$ and E the list of edges $e_1, \cdots, e_n$ associated to a list of committed weights $d_1, \cdots, d_n$. One of the vertex is labelled *source*.

Output: The predecessors list $\boldsymbol{pred}$ and/or the total distances list $\boldsymbol{dist}$.

1 **for** $i \leftarrow 1$ **to** m **do**
2 $\quad pred_i \leftarrow \mathsf{Com}(\mathsf{pk}, i)$
3 $\quad$ **if** $v_i = source$ **then**
4 $\quad\quad dist_i \leftarrow \mathsf{Com}(\mathsf{pk}, 0)$
5 $\quad$ **else**
6 $\quad\quad dist_i \leftarrow \mathsf{Com}(\mathsf{pk}, \top)$
7 $\quad$ **end**
8 **end**
9 **for** $k \leftarrow 1$ **to** m **do**
10 $\quad$ **for** $l \leftarrow 1$ **to** n **do**
11 $\quad\quad e_l^{i,j} = e_l$
12 $\quad\quad y \leftarrow dist_i \ominus dist_j \oplus d_l$
13 $\quad\quad x \leftarrow y \otimes \mathsf{Com}(\mathsf{pk}, 0)$
14 $\quad\quad dist_j \leftarrow dist_j \oplus x \odot y$
15 $\quad\quad pred_j \leftarrow pred_j \oplus x \odot (\mathsf{Com}(\mathsf{pk}, i) \ominus pred_j)$
16 $\quad$ **end**
17 **end**

Remarks on Algorithm 5.4:

- the predecessor of a vertex is represented by a commitment to the number of the vertex (lines 2 and 15). These commitments can be computed once and then reused when needed. The openings of these commitments must be provided in π_{cor}. We assume that the commitment algorithm Com is obtained via the CCE scheme.
- in the same way, the commitments on 0 and $\top$ on lines 4, 6, and 13 are computed once and then reused. Their openings should also be given in π_{cor}.

- the comparison on line 13 requires a π_{ran} and the two multiplications on lines 14 and 15 require each a π_{mul}. All these proofs are aggregated in π_{cor}.

π_{cor} is Zero-Knowledge: This is clear by the generic construction of π_{cor} (see Section 5.4).

Complexity. In this third application, we provide a verification algorithm that has exactly the same complexity as the algorithm itself since it is a secure version of it. As a result the complexity of the verification of the proof is $O(mn)$ for the client. However, we do not claim that a simpler verification proof in term of complexity does not exist for the client. Nevertheless, this example shows that it is always possible to obtain a complexity equal to the complexity of the algorithm.

Note that modifying the functionality could also result in an interesting decrease of the verifiability computational effort for the clients. For example, if the path p delivered by the worker is not exactly the shortest path but rather a path meeting a series of criteria. As an illustration of these criteria, let us imagine that it is mandatory for the path p to go through some fixed edges and at the same time that the weight of the path p must be under some fixed bound. Again, given the solution path p, the verification that the criteria are met could be much cheaper that the computation of the algorithm itself. Of course this example illustrate a more general way of thinking our functionalities and the requirements fixed for the solution.

5.5.4 Miscellaneous

Exploring some other problems that adapt very well with our approach, our attention was caught in particular by the wide category of NP-problems. A first interesting one is the *graph colouring problem* where we want to colour each vertex of a graph given a fixed number of colours and where two adjacent vertices cannot have the same colour. In the multi-party case we assume that the structure of the graph is secretly shared among the clients. While this problem might be hard to solve, the solution when found could be easily verified.

Another graph NP-problem is finding a *Hamiltonian cycle* in a given graph. This kind of cycles is a path that visits each vertex exactly once. No known algorithm computes the solution of this problem in polynomial time whereas verifying a given solution is linear in the number of nodes. Again, in multi-party, we assume that each client knows only a subset of the graph while the entire structure remains hidden.

Finally, other optimization problems might fit well with our technique. The *knapsack problem* is a good example with multi-party appli-

cations. For example, a set of clients share the use of a truck that has a maximum weight capacity. They all have a fixed number of items with respective secret weights. The goal is to maximize the number of items stored in the truck without exceeding its capacity. As this problem may not have a deterministic solution, the clients could require the worker to reach some minimal bound. Verifying that the bound is reached is then a matter of a simple sum.

5.6 Prototype implementation and timing results

A generic implementation of the protocol has been realized in Python. The main objective was to create a simple framework to emulate the clients-worker interaction and measure the load of work of each party in different scenarios. The implementation aims to be light and reasonably efficient by using optimized algorithms and techniques. Our implementation is a prototype. However it is already a good indicator of performance. Nevertheless, we might expect a nice efficiency gain when using optimized code in C for example. This should be worth for specifically designed applications. In this regard, we do not claim that we have the best known time results for our applications. For example, the recent work of Hamada et al. [HICT14] focussing on SMC sorting is still hundred times better than the "functionality-like" auctions of this chapter.

The details of the implementation were provided in Section 3.5 and we made the libraries and our test applications available online [Cuv15]. We use the PPATS CCE scheme introduced in Section 3.4.

Audit of the code. Our implementation is meant to be used, but our setting allows the clients and the worker to develop their own code independently. To do so, the clients and the worker have to agree on the verification procedure that is used to verify the correctness of the function evaluation. For example, in the case of auctions, after that the worker has published the list of the sorted commitments with the π_{cor} proof, the verification proceeds in two steps: first, the clients verify that the sorted list of commitments is a permutation of the original list of commitments, then, the clients verify the π_{cor} proof which is an aggregation of $\pi_<$ proofs. However, other verification procedures are possible such as verifying the sorting algorithm itself. Aside from this, the worker needs to produce and send the public key pk of the CCE scheme to the clients. If someone wants to use our implementation or

Table 5.1: **complexity and size cost for primitives and NIZKPK.**

	Computation	**Verification**	**Size**
π_{ope}	$2\mathtt{Sm}_\mathtt{p} + 1\mathtt{A}$	$\mathtt{Sm} + 2\mathtt{Sm}_\mathtt{p} + 2\mathtt{A}$	$3\mathtt{V}$
π_{or}	$4\mathtt{Sm}_\mathtt{p} + 2\mathtt{A}$	$2\mathtt{Sm} + 3\mathtt{Sm}_\mathtt{p} + 3\mathtt{A}$	$4\mathtt{V}$
π_{DL}	$4\mathtt{Sm}_\mathtt{p} + 2\mathtt{A}$	$2\mathtt{Sm} + 4\mathtt{Sm}_\mathtt{p} + 4\mathtt{A}$	$4\mathtt{V}$
π_{mul}	$6\mathtt{Sm}_\mathtt{p} + 3\mathtt{A}$	$4\mathtt{Sm} + 5\mathtt{Sm}_\mathtt{p} + 6\mathtt{A}$	$6\mathtt{V}$
$\pi_{\mathsf{ran}}(2^{k+1})$	$6k\mathtt{Sm}_\mathtt{p} + 3k\mathtt{A}$	$(3k-1)\mathtt{Sm} + 3k\mathtt{Sm}_\mathtt{p}$ $+(4k-1)\mathtt{A}$	$5k\mathtt{V} + k$

wants to rely on someone else's implementation, there are a few verifications to perform on the code. On the client's side, preventing privacy leakages means verifying that the code only CC encrypts his private information with the public key and sends only his CCE ciphertext to the worker while publishing the commitment on the public board. Moreover, to obtain verifiability, the clients must check that the code verifies the π_{cor} proof correctly. However, it is not important for the clients to audit the worker's code since the verifiability guarantee relies only on the π_{cor} proof and the privacy is, by hypothesis, entrusted to the worker.

Complexity. In the following complexity analysis, we recall the notations used in Section 3.5. We denote by $\mathtt{A}$ the EC Point addition in G_1 and by $\mathtt{Sm}$ and $\mathtt{Sm}_\mathtt{p}$, the scalar multiplication of EC Point in G_1 without and with precomputation respectively. In Table 5.1, we list the cost of different NIZKPK we use in our algorithms. We split the cost between computation and verification of the proof. In this table, we also indicate the size cost of each proof and the commitment. As a base unit, we write $\mathtt{V}$ for a λ-bit long integer which is the length of a random number in $\mathbb{F}_p$ or roughly in $\mathbb{Z}_q$. Storing one EC point could be done by storing its x-coordinate plus 1 bit of sign.

In Table 5.2, we list the complexities to compute and verify the π_{cor} of the three applications we considered in the Section 5.5. Note that in the case of a linear system, we place ourselves in the simplest scenario where the worker provides as π_{cor}, a list of openings. From the client's point of view, the complexities of the linear system and the auctions are linear in the number of clients. For the shortest path it equals the complexity of the Bellman-Ford algorithm.

Several tests were performed on a standard laptop: Intel® Core i5-3320M CPU @ 2.60GHz×4 with 7.7 GB of RAM. For these tests, the security parameter λ is 256-bit long. Even though this is the current security requirement for EC based cryptosystems, we argue that this is

Table 5.2: **complexity and proof size for π_{cor} of applications** – the linear system is performed with a coefficients matrix of size $m \times n$, each client possessing one coefficient. We consider auctions with $n+1$ bids of size $< 2^{k+1}$. The graph used for the shortest path contains m vertices and n weighted edges of weight $< 2^{k+1}$, each client possessing one weighted edge. The cost for checking the proof π_{cor} is given for one client.

	Worker	
	algorithm	preparing proof π_{cor}
Linear system solving	$O(n^3)$	$[mn]\mathtt{Sm}$ $+[2mn]\mathtt{Sm_p}$ $+[2mn]\mathtt{A}$
		proof size : $(n+2m)\mathtt{V}$
Auctions	$O(n \log n)$	$[(n+1)(3k-1)]\mathtt{Sm}$ $+[(n+1)(10k-1)$ $-6k+2]\mathtt{Sm_p}$ $+[(n+1)(7k-1)$ $-3k+1]\mathtt{A}$
		proof size : $(n-1)(5k\mathtt{V}+4\mathtt{V}+k)$
Shortest path	$O(mn)$	$[n(3k-1)]\mathtt{Sm}$ $+[mn(6k+14)$ $+2m+3kn]\mathtt{Sm_p}$ $+3k+1]\mathtt{A}$
		proof size : $mn(5k\mathtt{V}+30\mathtt{V}+k)+8m\mathtt{V}$
	Client	
	$\lvert\mathcal{C}\rvert$	checking proof π_{cor}
Linear system solving	mn	$[2mn-1]\mathtt{Sm}$ $+[2mn+2m+6]\mathtt{Sm_p}$ $+[3mn+1]\mathtt{A}$
Auctions	n	$[2n(3k-1)]\mathtt{Sm}$ $+[n(6k+2)+6k+8]\mathtt{Sm_p}$ $+[n(8k-1)+3k+3]\mathtt{A}$
Shortest path	n	$[mn(3k+8)+(n-1)(3k-1)]\mathtt{Sm}$ $+[mn(3k+12)+8m$ $+(n-1)(3k)+6k+8]\mathtt{Sm_p}$ $+[mn(4k+18)+4m$ $+(n-1)(4k-1)+3k+3]\mathtt{A}$

Table 5.3: **time results and proof size for the three applications** – timings are given in seconds. $|\mathcal{C}|$ is the number of clients. About the parameters, in the linear system, we suppose square systems and the shortest path number of vertices equals the number of edges (see Section 5.5 for details). The proof size is theoretically computed on the basis of a security parameter λ of 256-bit long.

	$\lvert\mathcal{C}\rvert$	Worker	Client	Proof Size
Linear system solving	16	$1.86e10^{-1}$	$4.14e10^{-2}$	384 B
	256	3.03	$5.62e10^{-1}$	1.54 kB
	4096	52.34	8.8	6.14 kB
Auctions	10	$3.94e10^{-1}$	$3.87e10^{-1}$	22.79 kB
	100	4	4.17	250.5 kB
	1000	40.08	42.04	2.53 MB
Shortest path	4	$2.57e10^{-1}$	$4.81e10^{-1}$	54.81 kB
	16	2.57	6.85	864.7 kB
	64	35.03	105.7	13.79 MB

too high for our protocol where a polynomial time adversary that would be able to break the correctness of the scheme needs to run an attack against the binding property of the commitment scheme (in our case, break the DLP). However, at the time scale of the protocol execution, this attack has to be performed in a short amount of time. For this reason, we suspect that a smaller security parameter would still allow a high level of security while alleviating the computational burden of the participants.

The time results including all the computations are presented in Table 5.3. Whenever possible, the computations are spread through the different cores of the processor to take advantage of the parallelization. The results show clearly the linearity in the number of clients in the first two applications while the complexity of the shortest path follows the quadratic complexity of the algorithm. The main limitations of efficiency are inherent to the way Python and the Gmpy package manage the basic modular operations of addition and multiplication. In counterpart, prototyping various applications in Python is rather facilitated while an optimized language like C might be preferred for targeted applications.

5.7 Conclusion

Current progress and real world applications in the field of secure multi-party computation are positive indicators for this branch of cryptography. Faster and reliable but also user-friendly solutions are provided to meet the needs of an emerging sector of activity.

This chapter aims at proposing a simple and efficient solution to evaluate multi-party function in a clients-worker setting where the clients want a strong guarantee over the correctness of the result. Our solution is based on perfectly hiding commitments posted on a public bulletin board for which a worker will be bound to and will provide a computationally sound proof of correctness. Combining our CCE primitive with NIZKPK results in a generic and easy-to-set up protocol that is secure against passive adversary for the privacy and secure against active adversary for the correctness of the function evaluation. Incidentally, our protocol provides a *perfectly private audit trail*.

Moreover, we show that this setting allows the clients to gain in complexity for the verification of the proof when this verification is cheaper than the algorithm used to compute the result. As a side effect, the worker is able to use his own algorithm to compute the result of the function without disclosing the intellectual property of his algorithm. This is of particular interest when the worker is a company specialized in algorithmic optimization. We illustrate the ease of use of our technique by three – rather simple but already used in real world – applications. We also provide timing results measured on our prototype implementation that indicate efficiency even though we point out that improvements could be achieved with clever optimizations.

Chapter 6

Securely Solving Combinatorial Problems with Secure Multi-Party Computation

Most of the work presented in this chapter was published in the proceedings of the "Financial Cryptography and Data Security" conference in 2013 [ACM+13].

This chapter investigates the problem of solving traditional combinatorial problems using Secure Multi-Party computation (SMC) techniques, focusing on the shortest path and the maximum flow problems. To the best of our knowledge, this is the first time these problems have been addressed in a general multi-party computation setting.

While not really focussed on audit mechanisms, this chapter offers two valuable pieces of information for this thesis: first, it presents combinatorial problems that are good candidates for multi-party real-world applications. More importantly, the analysis and measurements carried out on the prototype implementation give us a good point of comparisons for our test applications of Chapter 5. Second, this chapter highlights the complexity gaps as well as the different privacy leakages that arise in secure programming.

We point out that the results presented in this chapter do not give any verifiability guarantee outside the passive adversary model. To obtain verifiability, we must use traditional SMC schemes that are secure against active adversaries. In this case, the parties are ensured that all the participants follow the protocol.

We performed and tested secure implementations of the combinatorial algorithms by using the VIFF framework [VIF]. We used this Python framework in the passive security "mode". However, it can be switched to an active security "mode" without the need to modify the

code. Hence, the conclusions of this chapter remain valid within the active adversary model except for the timing measures which would be significantly impacted by the change of paradigm.

6.1 Introduction

Secure multi-party computation is the problem of jointly evaluating a function on a set of secret inputs without leaking anything but the output of the function. SMC has been at the center of cryptography research for almost 30 years. A first series of foundational works [Yao82, BOGW88, CCD88, GMW87] demonstrated the possibility to evaluate any function in various models, the function being described as a circuit. As already pointed out in this thesis SMC targets the evaluation of functions of specific interest, such as voting or auctions, graph problems, but also benchmarking, linear programming, secure outsourcing, etc.

One common point of all these applications is that the function evaluation process is naturally oblivious of the inputs on which the function has to be evaluated. Computing the highest one of n bids or summing n votes is carried out by performing n comparisons or sums independently of the values that are considered.

However there are large classes of problems for which the natural evaluation process depends on the input data. In that case, even if all the manipulated data are appropriately shared or encrypted, the execution flow might just be sufficient to leak undesirable information.

This is typically the case in combinatorial problems, of which graph problems are one of the most common examples. Consider, for instance, a consortium of delivery companies covering different territories through regular distribution circuits. These companies might be interested in computing the fastest way to bring a package from one place to another, but be reluctant to share between each other the precise connections they are using and the performance of their trucks. Their problem could be solved by securely evaluating traditional shortest path algorithms such as those of Bellman-Ford or Dijkstra. The immediate way of securely computing the shortest path would be to blind (encrypt or share) the weight of all the edges of the corresponding graph. However this approach could completely miss its purpose depending on the graph encoding and shortest path algorithm that are used: if the algorithm conditionally visits the graph by branching as a function of the secret weights, then the branching patterns could leak a substantial amount of secret information. In a similar way, the resolution of combinatorial problems, even on obfuscated inputs, can leak substantial information through the structure of the combinatorial object that is manipulated, as well as through its running time. We stress that this is not just a theoretical concern: numerous techniques have been developed, notably in the line of work on side-channel attacks [Koc96], that can successfully

exploit branching patterns and running times in order to recover the secrets on which computation is performed.

6.1.1 Our contributions

In this chapter, we tackle the problem of securely solving combinatorial problems in a multi-party setting through a series of examples taken from graph theory. To the best of our knowledge, this is the first time that these most classical algorithmic problems have been addressed in a general secure multi-party setting. Our solutions have applications in the numerous contexts where a graph is shared between competing entities. Natural examples include:

- privacy-preserving GPS guidance in which one party knows the map while the other knows his origin and destination,
- privacy-preserving determination of topological features in social network (the number of different ways to connect two people can be seen as a special case of the maximum flow problem, for instance, in which case each party would know his own friends but no more), or
- privacy-preserving determination of the performance of the cooperation between competing network operators (gas, electricity, logistics, ...), in which each party would know the capacity of his own infrastructure but no more.

Furthermore, our study raises several intriguing complexity gaps and suggests the exploration of various trade-offs.

Algorithm design. We focus our research on computing the shortest path and the maximum flow based on the secure arithmetic black-box functionality of Damgård and Nielsen [DN03] augmented with comparison [Tof11]. That is, our protocols assume access to a functionality that offers secure addition, multiplication and comparison. This allows us to abstract from the specific security model in which we want our protocol to be secure: depending on the implementation of the secure arithmetic black-box that is used, our protocols will be secure only in the presence of an honest majority or with up to all but one corrupted player, in the information theoretic or computational model, in front of passive or active adversaries, ... Various such implementations, in various models, are available in tools designed for multi-party computation such as FairplayMP [BDNP08], Sepia [BSMD10], Sharemind [BLW08] or VIFF [Gei10].

We focus on two of the most standard graph problems, chosen for their wide diversity of applications: computing shortest paths and maximum flows. For each of these problems, we discuss secure evaluation techniques inspired from classical algorithms of various complexities: Bellman-Ford and Dijkstra for the shortest path, and Edmonds-Karp for the maximum flow.

Our resulting algorithms offer quite different overheads, depending on the algorithm and the graph structure, as illustrated in Table 6.1. For those algorithms, the table shows first the traditional (non secure) complexity, then the complexity of our secure versions expressed in number of calls to the arithmetic functionality. There, we consider the case of a graph with public structure and then with private structure, meaning not only that the weight of each edge is kept secret, but also that the adjacency relation between vertices is kept private as well.

Several observations can already be made.

- The best implementations, using advanced data structures as dynamic trees [ST81] or Fibonacci heaps [FT87], are definitely nontrivial to replicate in the secure setting (see also discussion in Section 6.1.2 below). Their relevance is also unclear for the relatively small size of the problems that we are addressing, as they usually come with large constants.

- The overheads resulting from moving from the original algorithms to their secure versions largely differ between algorithms: in the case of a public structure for instance, we see either no difference, or an $|E|$ factor or a $|V|$ factor depending on the algorithm.

- The overhead resulting from hiding the graph structure largely differs depending on the algorithm and type of graph. For Bellman-Ford, the difference essentially corresponds to always handling a complete graph when the structure needs to be hidden. For Dijkstra however, the secrecy of the graph structure has no impact.

- While Bellman-Ford is traditionally less efficient than Dijkstra, this is not true anymore (asymptotically at least) for our secure variants: Bellman-Ford becomes substantially more efficient for sparse graphs (e.g., if $|E| = O(|V|)$) and the asymptotic complexities are similar for dense graphs.

The overheads in terms of number of protocol participants, round complexity, ... largely depend on the implementation of the secure arithmetic functionality, and are in line with traditional works.

Table 6.1: Asymptotic complexities: original algorithms and secure versions with public and private graph adjacency matrix.

	Optimized	Original	Public Structure	Secret Structure
Bellman-Ford	$\|V\|\|E\|$	$\|V\|\|E\|$	$\|V\|\|E\|$	$\|V\|^3$
Dijkstra	$\|E\| + \|V\| \log \|V\|$	$\|V\|^2$	$\|V\|^3$	$\|V\|^3$
Edmonds-Karp	$\|V\|^2\|E\|$	$\|V\|\|E\|^2$	$\|V\|\|E\|^2$	$\|V\|^5$

Complexity: the constants matter. In order to challenge our algorithms in practice, we implemented them all using the Virtual Ideal Functionality Framework (VIFF) of Geisler et al. [Gei10], in the honest-but-curious (or passive) model.

This allowed us to further investigate the constants hidden by the asymptotic notations discussed above. This made particularly visible the difference of cost between the different black-box primitives that we used: addition based on linear secret sharing [Sha79] (see also Section 2.2) comes for free (no communication involved), multiplication is noticeable (it involves one secret sharing), and comparison (based on Toft's protocol [Tof07]) is ≈ 165 times more expensive than a multiplication, something that strongly contrasts with the execution time of traditional algorithms.

These differences have strong practical impact and motivated some trade-offs as well.

- Our version of Dijkstra's algorithm involves only $|V|^2$ comparisons compared to $|V|^3$ (or $|V||E|$) in Bellman-Ford. As a result of this, for dense graphs or when the graph structure is secret, Dijkstra's algorithm remains considerably more efficient than Bellman-Ford's, even when the structure of the graph is public, provided that the graphs have a reasonably small size (a hundred vertices).

- For sparse public graphs that contain a small number of paths from the source to the sink, a variant of Edmonds-Karp's algorithm that relies on an exhaustive public enumeration of the source to sink paths can be considerably simpler and more efficient than a secure version of the breadth-first search for augmenting paths that is performed in the original algorithm: this allows trading expensive book-keeping and addressing operations for more but much simpler rounds.

So, besides the fact that our work offers the first solutions for the secure evaluation of various graph properties, we think that it raises

several intriguing complexity issues. Notably, we wonder whether the complexity gaps that we have are inherent to the added security or if they can be improved.

6.1.2 Related works

As mentioned above, the large majority of works on secure multi-party computation focused on functions whose evaluation execution flow is independent of the secret inputs. There are important exceptions to this, however.

Branching programs. Branching programs are decision procedures that, based on some inputs and decision parameters such as thresholds, perform a specific classification of the input. Secure versions of these programs, where a user does not learn the branching program of the server while the server does not learn the user's inputs, have been considered in various works [KJGB06], [IP07], [BPSW07], [BFK+09], [BFL+11]. While these works share our goals of hiding the data path through which the program is going, they do not aim at hiding the length of that path which, in our case at least, could leak a substantial amount of information.

Shortest path in the two-party setting. The work of Brickell and Shmatikov [BS05] addressed the problem of solving some graph problems securely and their work is, as such, the closest one to ours. Substantial differences appear, though.

First, their security model is quite different from ours. Their protocols, which are based on a privacy-preserving set union protocol, proceed by making their outputs known to the participants progressively as part of the execution (e.g., edge by edge as the protocol runs). Even though this is not revealing more than the eventual outcome, this makes their protocols unusable as sub-components of other higher-level protocols that would rely on using these outputs as part of their secret state. Revealing outputs part-by-part as the protocol runs might also be problematic in applications in which some participants could abort the protocol in the middle of its execution, based on what they have already learned. Our protocols, on the other hand, can be freely used as subroutines, and one of our secure max-flow algorithms will make use of a secure shortest-path algorithm.

Second, the graph problems they consider are different from ours as well. They do not consider the maximum flow problem at all: their work focuses on computing shortest distances, from a known source to all the

vertices or for all the vertex pairs, in a setting where all the participants assign a weight to all the edges. We further investigate the problem of computing the shortest path from a single source to a single destination, which cannot be done using their set union technique as it would reveal much more information than the specific distance we are interested in.

Eventually, their protocols are not based on generic building-blocks, like the arithmetic black-box functionality on which we rely. Specifically, their protocols are designed for the two-party computation setting in the honest-but-curious model. While these specifics allow them to develop techniques that are quite efficient in this two-party setting, it is unclear how efficient a transposition of their approach to the multi-party setting would be.

Efficient secure data structures. The problem of securely computing on data structures has been investigated by Toft [Tof11], in the case of a secure priority queue, which he implements using a variation of bucket heap. The problem studied there shares similar flavours with those we address here: to securely compute on structured data by keeping the actions independent of the inputs. The computational overhead compared to the efficiency of the original bucket heap is logarithmic, making it occupy an interestingly different spot in the list of overhead examples discussed above.

Similar effects could also be achieved through the use of oblivious memories [GO96], [DMN11]. The term "oblivious memories" denominates the set of techniques that allows hiding the access pattern of a program to memory against an eavesdropper. If the program execution is outsourced by the client to a server who might be the eavesdropper, thanks to oblivious memories, the client is ensured that the eavesdropper does not learn any information regarding the client's requests. This technique results in an overhead in the data storage as well as in the data access operations.

Organisation of the chapter. Section 6.2 describes the building blocks we will use and our main implementation choices. Section 6.4 describes our approach to the classical single source and single-pair shortest path problems, and Section 6.5 describes our approaches of the maximum flow problem.

6.2 Preliminaries

6.2.1 Black-box operations

We build our protocols on top of an ideal functionality : the arithmetic black-box functionality $\mathcal{F}_{ABB}$ of Damgård and Nielsen [DN03], whose definition captures the properties we need.

This functionality allows n parties to securely store elements of a ring $\mathbb{Z}_m$, to repeatedly perform the ring operations of addition and multiplication on these elements, and to open the result of the computation when needed. Following Toft [Tof11], we consider a slightly extended and abstracted version of this functionality that offers the possibility to perform secure comparison and consider any possible ring. So, storing, opening, addition, multiplication and comparison will be the only secure operations on which our protocols rely. Following the notations introduced in Section 2.2, we write $[x]$ to address the version of x securely stored by $\mathcal{F}_{ABB}$ and we call $[x]$ a shared value of x. We also use the notation $[\boldsymbol{v}]$ for a vector of shared values and $[\boldsymbol{M}]$ for a matrix of shared values. We denote the secure arithmetic operations on shared values in the natural way, e.g., $[z] \leftarrow [x] + [y]$ for the addition of two secrets. We use the notation $(\cdot)$ to denote the dot product between two vectors. The actual protocol implementing these operations depends on the details of the realization of this functionality. Numerous SMC schemes can be used for that purpose [GMW87, CCD88, DN03] or, for more recent approaches [BDOZ11, DKL+12, DPSZ12], depending on the security model that is appropriate.

All our protocols assume that inputs are already stored in the $\mathcal{F}_{ABB}$ functionality and give access to the stored outputs (that can be opened through opening requests to $\mathcal{F}_{ABB}$). This feature guarantees the composability of the protocols. The way inputs and outputs are shared depends on the application: they might come from a specific problem, or from the needs of a higher-level protocol using this protocol as a sub-routine, for instance.

6.2.2 Bounds

The size of the ring $\mathbb{Z}_m$ has to be chosen carefully to prevent overflows. For each protocol presented in this chapter, we provide the bounds of m and the value of $\top$ in Figure 6.1. These bounds depend on numbers such as the maximum weight $\hat{w}$ or the maximum capacity $\hat{c}$ allowed for the edges. These maxima are agreed in advance by the players. Remark that $\top$ is smaller than m. Most comparison protocols require a much

Protocol	$\top >$	$m >$
USNM & **USM**	-	-
SSPBF	$\|V\|.\hat{w}$	$f(\top)$
SSPD	$\|V\|.\hat{w}$	$\|V\|.f(\top)$
SMFEK	-	$\|V\|.f(\hat{c})$
SMFCEK	$\hat{c}$	$\max(f(\top + \hat{c}), \|V\|.f(\hat{c}))$

Figure 6.1: Minimal bounds on $\top$ and m to avoid overflows.
SSPBF and **SSPD** are the secure shortest path algorithms presented in Section 6.4.
SMFEK and **SMFCEK** are the secure maximum flow algorithms presented in Section 6.5.
$|V|$ is the number of vertices, $\hat{w}$ and $\hat{c}$ are respectively the maximum weight and the maximum capacity admissible (by convention) in a graph.

larger m than the values to compare. This dependence is taken into account via a function f.

6.2.3 Graph representation

Depending on the algorithm we are trying to compute and on the part of the graph description that is part of the secret input, different graph representation approaches will show to be useful.

In all cases, we will assume that the number of vertices in the graph is public (or at least an upper bound on it). Depending on the setting, the adjacency relationship between the vertices might be public or not. For instance, it is natural to have it public if the graph represents the connections between places on a map, but it might be desirable to keep it secret if the presence of edges reveals the existence of transactions between competing companies.

A traditional structure for storing a graph consists in storing, with every vertex, a list of its neighbours (and the weight of the corresponding edges). This structure is quite efficient in terms of memory. However, it might be quite problematic from a security point of view, as it discloses the degree of each vertex. A solution would be to tolerate the leakage of an upper bound on these degrees, but that upper bound would be close to imply the storage of a complete graph as soon as one single vertex is of high degree. Furthermore, even if the leakage of the degree of the vertices is tolerated, algorithms that perform breadth-first search on vertices and branch depending on the weight of edges could reveal a lot of information. As a result, this graph representation can show to be very effective in some cases, but completely inappropriate in others,

even when the graph structure is public.

A second traditional way of representing graphs is to store their adjacency matrix, the elements of the matrix representing the weight of the edges between vertices. This approach has the benefit of offering a storage that is independent of the graph structure.

6.2.4 Privacy leakage by execution flow

An omnipresent problem that we have to keep in mind in the design of our algorithms is the potential leakage over privacy that may occur during the execution flow. This happens mainly because the branchings that are naturally numerous in most algorithms, become a source of leakage in their secure version. Indeed, if we reveals the path that the execution takes given a fixed set of inputs, an adversary might be able to deduce all kinds of information about these inputs.

To be more concrete, imagine a sorting algorithm that requires, in the best case, exactly n comparison operations to check that a list is sorted and, in the worst case, n^2 comparisons. An adversary simply counting the number of operations or the number of executed loops, might directly deduce if the list is already or almost sorted, etc. From that, the adversary knows that the secret input belonging to some player is smaller that the one belonging to another. This is of course an intolerable leakage of information.

The most direct countermeasure is to run the algorithm as if it is always in the worst case scenario. This leads to an important overhead compared to the unsecured version of the algorithm. An "if-then-else" branching must be treated in a similar fashion, both condition must be handled with the same computations, sometimes resulting with adding a [0] to a shared value in order to update it.

As discussed in the previous sections, if the structure of the graph is private, we often have to consider an execution of the algorithm over the complete graph where the players set the "non existing" edges to an extrema value determined in advance. In this case, we did not study how a player deviant from the protocol could use this at his advantage since we suppose that we remain in passive security.

6.2.5 Unary versus decimal representation

While running our algorithms, we will often need to perform some operations on a specific vertex designated by a secret index. This will typically be performed by running that operation on all vertices, including a cancelling factor everywhere but on the vertex that needs to be

treated. An obvious way of testing whether we are working on the right vertex would be to perform a test at each step.

In some cases, we can use a more effective approach that avoids comparisons, by representing the index of the vertex i by a vector $[\boldsymbol{i}] \in \{[0],[1]\}^{1\times b}$ where each entry is $[0]$ except for the i'th which is $[1]$. We call this the **unary notation**. We can then access the weight of the edge from vertex i to vertex j by computing the matrix product $[\boldsymbol{i}].[\boldsymbol{M}].[\boldsymbol{j}]^t$ where $[\boldsymbol{M}]$ is the weighed adjacency matrix. Algorithm 6.1 provides a way to update an element in a shared list. It can be easily extended to update an element in a shared matrix. This algorithm also exemplifies a common way of emulating a branching depending on a secret value: the arithmetic operation in the loop is actually equivalent to computing **if** $[\boldsymbol{i}]_j = [1]$ **then** $[\boldsymbol{l}]_j = [x]$.

Algorithm 6.1: Update an element in a shared list and at a shared position

Input: A list $[\boldsymbol{l}]$ of length b, a shared index $[\boldsymbol{i}]$, a shared value $[x]$.
Output: The list $[\boldsymbol{l}]$ with the update $[\boldsymbol{l}]_i = [x]$.

```
for j ← 1 to b do
    [l]_j ← [l]_j + [i]_j([x] − [l]_j)
end
```

For a graph with n vertices, this protocol allows retrieving a secret position in the adjacency matrix in $O(n^2)$ multiplications instead of $O(n^2)$ comparisons, which is considerably more efficient, even if it implies a considerable overhead in storage (moving from 1 secret index to n secret bits). We note that, in all cases, this approach implies treating the graph as if it were complete, which can be a considerable waste of resources if the graph is actually sparse.

When working with secret lists or matrices we need to use Algorithm 6.2 to increment an index in unary notation which is in turn used to access an element of the list as in Algorithm 6.1. Similar to a number in unary notation, a unary index is a vector of shared $[0]$ with one element set to $[1]$ indicating a position. For example $([0],[0],[1],[0])$ indicates the third position. To increment an index, we need to push forward the shared $[1]$ in the vector. Algorithm 6.2 includes a boolean condition $[c]$ in its inputs. This condition decides whether the index is really incremented or if it remains the same. However, in both cases, the index is updated which gives no information on $[c]$ for the observers.

Going from decimal notation to unary notation and vice-versa might be useful in some scenarios. For example, if an algorithm that benefits

Algorithm 6.2: Increment an index in unary notation

Input: An index $[\boldsymbol{i}]$ in unary notation of length b and a shared boolean $[c]$.

Output: If $[c]$ is $[1]$, increment the index $[\boldsymbol{i}]$. Otherwise do nothing.

```
for j ← b − 1 to 1 do
    [i]_j ← [i]_j + [c]([i]_{j−1} − [i]_j)
end
[i]_0 ← (1 − [c])[i]_0
```

from the use of unary notation is used as a subroutine of another one running with decimal representation of the shared values. For example, sorting small integers has proved to be more efficient in the unary case than sorting algorithms that use decimal representation [Maw15, Chapter 4].

Given a value s and its unary representation vector denoted $\boldsymbol{s}^{\mathtt{u}}$ of length b, we want to translate the shared unary representation of s, given by the vector $[\boldsymbol{s}]^{\mathtt{u}}$ into a shared value of the decimal representation of s denoted $[s]^{\mathtt{d}}$. In this case, we simply need to compute the dot product between $[\boldsymbol{s}]^{\mathtt{u}}$ and $(0, ..., b-1)$ which requires b multiplications. The other way around is a bit trickier since we have to produce a vector of shared values of $[0]$ or $[1]$ where the i-th index is $[1]$ only if $s = i$. In other words, from the shared value $[s]^{\mathtt{d}}$ we have to compute the vector $[\boldsymbol{s}]^{\mathtt{u}}$ where

$$[\boldsymbol{s}]^{\mathtt{u}}_i := \begin{cases} [1] & \text{if } i = s \\ [0] & \text{otherwise.} \end{cases}$$

In this case, considering the polynomial

$$P_b(x) := \prod_{j=1}^{b-1} \frac{x^2 - j^2}{-j^2},$$

we compute $[\boldsymbol{s}]^{\mathtt{u}}_i$ as $P_b([s]^{\mathtt{d}} - i)$ which in turn gives, when $i = s$, $P_b([0]) = [1]$ and, when $i \neq s$, $P_b([s]^{\mathtt{d}} - i) = [0]$ as $s - i \in [1 - b, b - 1]$ cancels out one of the denominator factors. The change of representation from decimal to unary takes $b^3 - 2b^2 + b$ multiplications.

6.2.6 Prototyping

We implemented all our protocols over the Virtual Ideal Functionality Framework (VIFF [VIF]) to challenge their performance. We consid-

ered a 3-party execution in the information theoretic model with passive security: secret values are shared using Shamir's secret sharing (see Section 2.2), the BGW protocol is used for multiplication [BOGW88], and Toft's protocol is used for comparison [Tof07]. These choices were made for simplicity and ease of prototyping, though much more efficient protocols exist and would have led to considerably shorter running times [BDOZ11, DPSZ12].

The timing measurements were performed on a single workstation equipped with an Intel Xeon CPUs X5550 (2.67GHz) and 24GB of memory, running a standard Debian Squeeze.

6.3 Privacy-preserving sorting

Traditional sorting algorithms such as Quicksort have a complexity of $O(n \log n)$ and use the comparison of two elements as an atomic operation. However, as already mentioned, comparing two values in SMC costs much more than multiplying two values together and thus we cannot put both operations on equal footing. For example, in VIFF, the comparison operator is based on the bit decomposition protocol of Damgård [DFK+06] and, with the setting used in this chapter, one comparison costs about 165 multiplications, see [Maw15, Appendix A] for details.

Several SMC sorting solutions have been proposed in literature. Jónson et al. [JKU11] proposed a solution based on sorting networks and implemented with Sharemind which is an SMC framework enabling passive security for exactly 3 players. Their sorting protocol runs in $O(\log^2 n)$ rounds and requires $O(n \log^2 n)$ comparisons per round. In two successive Hamada et al. [HKI+13, HICT14] presented sorting solutions based on the Quicksort algorithm and the radix sort algorithm respectively. They obtain quite practical results with the following complexities: $O(\log n)$ rounds with $O(n \log n)$ comparisons for the Quicksort based algorithm and $O(1)$ rounds and $O(n \log n)$ comparisons for the radix sort based algorithm.

The two sorting algorithms we propose in this section do not rely on comparisons as atomic operations. On the other hand, they are suited only for sorting values that lay in a small range $[0, b-1]$. Indeed, as we use the unary representation, the cost per operation on an element is proportional to the length b of the interval. We distinguish the case where the n values to sort are all different from the case where there may be several identical values in the list. Note that if the values to sort are all different, we must have that $b \geq n$. The first scenario is called *unary sorting with no multiplicities* (**USNM**). The n players

$$\begin{pmatrix} [\boldsymbol{x_1}]^{\mathrm{u}} \\ [\boldsymbol{x_2}]^{\mathrm{u}} \\ [\boldsymbol{x_3}]^{\mathrm{u}} \\ [\boldsymbol{x_4}]^{\mathrm{u}} \\ [\boldsymbol{x_5}]^{\mathrm{u}} \end{pmatrix} = \begin{pmatrix} [3]^{\mathrm{u}} \\ [4]^{\mathrm{u}} \\ [1]^{\mathrm{u}} \\ [5]^{\mathrm{u}} \\ [0]^{\mathrm{u}} \end{pmatrix} = \begin{pmatrix} [0] & [0] & [0] & [1] & [0] & [0] \\ [0] & [0] & [0] & [0] & [1] & [0] \\ [0] & [1] & [0] & [0] & [0] & [0] \\ [0] & [0] & [0] & [0] & [0] & [1] \\ [1] & [0] & [0] & [0] & [0] & [0] \end{pmatrix} =: [\boldsymbol{M}]$$

$$\Downarrow$$

$$\begin{pmatrix} [1] & [1] & [0] & [1] & [1] & [1] \end{pmatrix} =: [\boldsymbol{s}]$$

$$\Downarrow$$

$$\begin{pmatrix} [0] & [1] & [3] & [4] & [5] \end{pmatrix} := [\boldsymbol{t}]$$

Figure 6.2: Unary sorting example. Five players want to sort a secret list formed by their private inputs x_i shared in unary representation.

share their private inputs x_i in unary notation, that is $[\boldsymbol{x_i}]^{\mathrm{u}}$, a vector of size b. Then, we proceed as follows:

1. the n shared values $[\boldsymbol{x_i}]^{\mathrm{u}}$ in unary representation form the rows of matrix $[\boldsymbol{M}]$ which is of size $n \times b$.
2. sum each column of $[\boldsymbol{M}]$ to obtain vector $[\boldsymbol{s}]$ of length b.
3. either open $[\boldsymbol{s}]$ to $\boldsymbol{s}$ or output the sorted list $[\boldsymbol{t}]$ as follows: for $i = 0, \ldots, b-1$, run through $[\boldsymbol{s}]$. If $[\boldsymbol{s}]_i = [1]$, append $[i]$ to $[\boldsymbol{t}]$, else if $[\boldsymbol{s}]_i = [1]$, do nothing. Appending $[i]$ to $[\boldsymbol{t}]$ is performed by updating the i-th position of $[\boldsymbol{t}]_i$ with the product $i[\boldsymbol{s}]_i$.

These three steps are illustrated in the example of Figure 6.2 and Algorithm 6.3 details the **USNM** computation.

In the second scenario where several identical values appear in the list, the vector $[\boldsymbol{s}]$ summing the columns may contain natural numbers greater than 1 corresponding to the multiplicities of the numbers in the list to sort. The sum of these multiplicities must equal n, the number of elements to sort. Similar to what is done to transform a shared value from decimal to unary representation, we use the polynomial $Q_n(x)$ defined as follows:

$$Q_n(x) := 1 - \prod_{j=1}^{n} \frac{x-j}{-j}$$

Algorithm 6.3: USNM for sorting different elements

Input: A matrix $[\boldsymbol{M}]$ where row i contains the secret shared value $[\boldsymbol{x_i}]^{\mathsf{u}}$ in unary representation of player i where $x_i \in [0, b-1]$ are all different. Matrix $[\boldsymbol{M}]$ is of dimension $n \times b$.

Output: A list $[\boldsymbol{t}]$ of length n containing the sorted shared values.

1 $[\boldsymbol{t}] \leftarrow ([0])^{1\times n}$ /* $[\boldsymbol{t}]$ stores the sorted list */
2 $\boldsymbol{d} \leftarrow (1)^{1\times n}$
3 $[\boldsymbol{s}] \leftarrow \boldsymbol{d} \cdot [\boldsymbol{M}]$ /* $[\boldsymbol{s}]$ sums the columns of $[\boldsymbol{M}]$ */
4 $[\boldsymbol{k}] \leftarrow$ sharedindex(n)
5 /* index $[\boldsymbol{k}]$ keeps track of where to insert the current value in $[\boldsymbol{t}]$ */
6 **for** $i \leftarrow 0$ **to** $b-1$ **do**
7 $\quad [c] \leftarrow [\boldsymbol{s}]_i$
8 $\quad$ /* $[c]$ is $[1]$ or $[0]$ whether the current value i is to be inserted or not in $[\boldsymbol{t}]$ */
9 $\quad [\boldsymbol{t}] \leftarrow [\boldsymbol{t}] + i[c][\boldsymbol{k}]$
10 $\quad$ /* this update of $[\boldsymbol{t}]$ is effective only if $[c]$ is $[1]$ and in this case it stores $[i]$ in $[\boldsymbol{t}]$ at the position given in $[\boldsymbol{k}]$, otherwise it just replaces $[\boldsymbol{t}]$ by itself */
11 $\quad$ incrementindex$([\boldsymbol{k}], [c])$
12 **end**

and we see that for $x \in [0, n]$,

$$Q_n(x) := \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{otherwise.} \end{cases}$$

By applying Q_n on vector $[\boldsymbol{s}]$, we obtain a condition that is used to compute the sorted list as shown in the *unary sorting with multiplicities* (**USM**) Algorithm 6.4.

Remarks on Algorithms 6.3 and 6.4:

- The function `updatevector` calls Algorithm 6.1. The functions `sharedindex`(b) returns an index in unary notion of length b set to a starting position, that is a list beginning by a [1] followed by $b-1$ [0]. The function `incrementindex`($i, cond$) calls Algorithm 6.2. It increments the index i only if *cond* is set to [1] otherwise it returns an unchanged index. Incrementing a unary index means that the [1] is pushed forward by one position (e.g. ([0], [1], [0], [0]) becomes ([0], [0], [1], [0])). This algorithm requires exactly b multiplications where b is the length of the index.

- Algorithm 6.3 requires exactly $3bn$ multiplications while Algorithms 6.4 needs exactly $3b^2 + 3n^2 + 6bn + b + n$ multiplications. No comparisons are required for neither of them.

- It is possible to keep track of the secret input owners but we have to add a tracking mechanism in parallel of the execution of the algorithm. However, in this case, the functionality is slightly different and more comparable to auctions.

We tested both algorithms on VIFF and obtained time results that are consistent with the theoretic projection. For the **USNM** Algorithm, we tested the maximum case scenario where $b = n$. Of course, in this case, the result is known in advance since it is the list of integers from 1 to n. Nevertheless it gives us a maximum bound on the execution time. A similar series of measurements was performed on the **USM** Algorithm. Both results are shown in Figure 6.3. We display the case where $b = n$ even though, in the **USM** scenario, we might accept $n \geq b$ as well.

A thorough analysis reveals that, in specific scenarios, our sorting algorithms take advantage over the secure version of the standard odd-even merge sort algorithm that uses comparisons. In particular, it appears that the **USNM** Algorithm 6.3 could be used when the number of values to sort do not exceed 1500 items. For less than 1500, the interesting range of the **USNM** depends on the bound b. As an illustration, when

Algorithm 6.4: USM for sorting with multiplicities

Input: A matrix $[\boldsymbol{M}]$ where row i contains the secret shared value $[\boldsymbol{x_i}]^{\mathrm{u}}$ in unary representation of player i where $x_i \in [0, b-1]$. Matrix $[\boldsymbol{M}]$ is of dimension $n \times b$.

Output: A list $[\boldsymbol{t}]$ of length n containing the sorted shared values.

1 $[\boldsymbol{t}] \leftarrow ([0])^{1\times n}$ /* $[\boldsymbol{t}]$ stores the sorted list */
2 $\boldsymbol{d} \leftarrow (1)^{1\times n}$
3 $[\boldsymbol{s}] \leftarrow \boldsymbol{d} \cdot [\boldsymbol{M}]$ /* $[\boldsymbol{s}]$ sums the columns of $[\boldsymbol{M}]$ */
4 $[\boldsymbol{j}] \leftarrow \mathtt{sharedindex}(b)$
5 /* index $[\boldsymbol{j}]$ keeps track of which current value to insert in $[\boldsymbol{t}]$ */
6 $[\boldsymbol{k}] \leftarrow \mathtt{sharedindex}(n)$
7 /* index $[\boldsymbol{k}]$ keeps track of where to insert the current value in $[\boldsymbol{t}]$ */
8 **for** $i \leftarrow 1$ **to** $b+n$ **do**
9 $\quad [\boldsymbol{s}_j] \leftarrow [\boldsymbol{s}] \cdot [\boldsymbol{j}]$
10 $\quad [c] \leftarrow Q_n([\boldsymbol{s}_j])$
11 $\quad$ /* $[\boldsymbol{s}]$ stores the remaining multiplicity of the current value $[\boldsymbol{j}]^{\mathrm{d}}$, $[c]$ is $[1]$ when this remaining multiplicity is > 0 */
12 $\quad [\boldsymbol{s}'_j] \leftarrow [\boldsymbol{s}_j] - [c]$
13 $\quad \mathtt{updatevector}([\boldsymbol{s}], [\boldsymbol{j}], [\boldsymbol{s}'_j])$
14 $\quad$ /* if $[c]$ is $[1]$, the remaining multiplicity of $[\boldsymbol{j}]^{\mathrm{d}}$ is decreased by one and the vector $[\boldsymbol{s}]$ is updated */
15 $\quad [\boldsymbol{t}] \leftarrow [\boldsymbol{t}] + [c][\boldsymbol{j}]^{\mathrm{d}}[\boldsymbol{k}]$
16 $\quad$ /* if $[c]$ is $[1]$, $[\boldsymbol{j}]^{\mathrm{d}}$ is appended to the sorted list $[\boldsymbol{t}]$, otherwise $[\boldsymbol{t}]$ remains unchanged */
17 $\quad \mathtt{incrementindex}([\boldsymbol{j}], 1 - [c])$
18 $\quad$ /* if $[c]$ is $[1]$, we still have to append a copy of $[\boldsymbol{j}]^{\mathrm{d}}$ to $[\boldsymbol{t}]$ thus we do not update $[\boldsymbol{j}]$ */
19 $\quad \mathtt{incrementindex}([\boldsymbol{k}], [c])$
20 $\quad$ /* if $[c]$ is $[1]$, we just have appended an element to $[\boldsymbol{t}]$ thus we update $[\boldsymbol{k}]$ that indicates where the next element needs to be inserted in $[\boldsymbol{t}]$ */
21 **end**

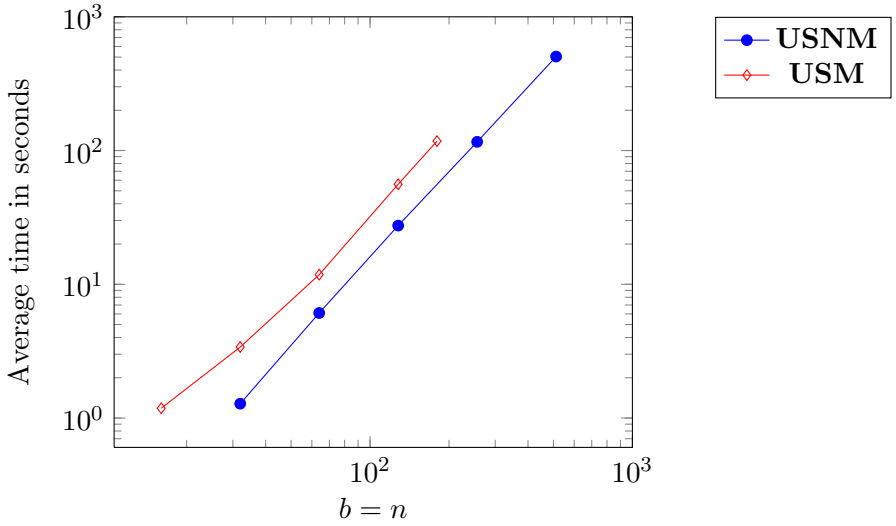


Figure 6.3: Time measurements for the implementation in VIFF of the **USNM** Algorithm 6.3 and the **USM** Algorithm 6.4. The complexity of **USNM** is $O(bn)$ and the complexity of **USM** is $O((b+n)^2)$. The slopes of the lines are ~ 2.1 for **USNM** and ~ 2.2 for **USM**.

$b = 256$, the threshold for n is $20 \leq n \leq 256$. Indeed, using **USNM** to sort less than 20 values from $[0, 256]$ is worthless since the cost induced by the size of the representation outweighs the cost of comparisons in the odd-even merge sort algorithm.

Regarding the **USM** Algorithm 6.4, its range of applicability is obviously smaller since the protocol runs $3b^2 + 3n^2 + 3bn + b + n$ more multiplications than the **USNM** algorithm. We refer to [Maw15, Chapter 4] for complementary details regarding the secure implementation of classical sorting algorithms as the odd-even merge sort. Note that the previous considerations are only valid in VIFF where one comparison counts for 165 multiplications. We might find other range of applicability for the unary sorting algorithms if we were to use another SMC framework.

This section introduced two new algorithms for sorting in unary representation that may find interest even outside the scope of secure computations. The motivation that led to conceive those algorithms was to avoid as much as possible to use comparisons since this operation is costly in traditional SMC frameworks. This very same motivation will stick in our mind through the rest of this chapter as it was already present in Chapter 5.

6.4 Privacy-preserving shortest path problem

The single-source shortest path problem is a major problem in graph theory. It has several immediate applications. The typical one is finding the shortest way to connect two cities on a road map where each city is represented by a vertex and each road between two cities by an edge. The edge weights are the road distances between cities. In this context, a user may then want to obtain driving directions without revealing neither his starting point nor his destination. Another application is the one of two entities owning each a secret location in a shared network and willing to compute the distance between them without disclosing their location. We note that such a problem is worth solving even for relatively small graphs. Consider for instance a routing network with a dozen hubs in different European countries and three competing logistic companies having each their own transportation costs for a defined set of roads. As costs typically represent sensitive information that should not be disclosed to competitors, being able to securely solve the shortest path problem for 3 parties and a graph with a dozen of vertices is quite helpful. Similar problems happen for network traffic on routers where a small number of big hubs is involved. Competing companies have to solve the shortest path to define routing scheme without revealing sensitive information about internal network configuration.

Shortest path algorithms are also used as subalgorithms for more advanced problems like the Chinese postman problem or the max-flow problem that we address below: this highlights the importance of keeping our protocols composable.

We investigate two standard algorithms for finding the single-source shortest path in a graph with weighted edges: Dijkstra's algorithm and Bellman-Ford's algorithm. The first one requires all edge weights to be positive, while the second one only assumes there is no negative-weight cycle in the input graph. As the non-secure version of all the algorithms that we treat is widely available [CLRS09], we will only briefly outline them.

6.4.1 Bellman-Ford's algorithm

The algorithm of Bellman-Ford is particularly simple, making it a natural target for building a secure version. This algorithm proceeds by repeatedly scanning all the edges, in search of adding edges that decrease the ongoing distance from the source to the various vertices. If a pass over the edges does not improve the current solution, or if the edges are scanned $|V|$ times, the algorithm halts. An interesting feature of this

algorithm is that its flow of operations only depends on the structure of the graph but not on the weight of the edges. Its drawback is its time-complexity: its classical implementation runs in $O(|V||E|)$ time.

Algorithm 6.5 (**SSPBF**) presents our *secure shortest path protocol based on Bellman-Ford*. Note that $\top$ is a number agreed in advance by the players as a higher bound for some calculations of the protocol. We refer to Section 6.2.2 for discussion of the values of $\top$ and m in all our algorithms. Note that the vertices are identified by a unique number from 0 to $|V|$. Finally, `updatevector` refers to Algorithm 6.1. The **SSPBF** protocol differs from the original algorithm only in a limited number of aspects:

1. the branching corresponding to the discovery of a shorter path is handled on Lines 9–11 through arithmetic operations as in Algorithm 6.1,

2. the early termination condition of the Bellman-Ford algorithm, which is triggered if the inner loop happens to have no effect during one pass, is removed as it may leak information. This does not invalidate the correctness of the algorithm but only increases the running time.

The structure of this algorithm makes it easy to implement with either of the two graph representations discussed above (list or matrix), making it possible to fully exploit the sparsity of graphs when it is public (we use the matrix representation if it has to be kept secret).

We see that our implementation requires $|V||E|$ secure comparisons, dominating the time required to perform $2|V||E|$ secure multiplications and $5|V||E|$ additions. These complexities grow to $O(|V|^3)$ when the graph structure is secret, as the graph is then treated as complete (i.e., augmented with edges of infinite weight).

Interestingly, this algorithm is the only one among those we treated in which our solution does not raise any asymptotic overhead (when the structure is public).

Security. The simulation of an execution of this protocol is immediate from the simulators available for the different calls that can be made by the $\mathcal{F}_{ABB}$ functionality: the simulators corresponding to each of the '$+$', '$\cdot$' and '$<$' operations can be invoked in turn, in an order defined by the protocol execution, and a number of times that only depends on public values ($|V|$ and $|E|$). The same argument will apply to the other protocols we present in this chapter, and we will therefore not come back to it.

Algorithm 6.5: SSPBF secure shortest path based on Bellman-Ford's algorithm

Input: A graph $G = (V, E)$ where $V := \{v_i\}$ is the list of vertices and $E := \{e_{j,k}\}$ the list of edges where $e_{j,k}$ is the edge from vertex v_j to vertex v_k. A set of shared weights $[\boldsymbol{w}]$ of length $|E|$, and a shared index of the position of the source vertex $[\boldsymbol{s}]$ in unary representation.

Output: The list of immediate predecessors $[\boldsymbol{p}]$ and/or total distances $[\boldsymbol{d}]$.

for $i \leftarrow 1$ **to** $|V|$ **do**
  $[\boldsymbol{p}]_i \leftarrow [0]$
  $[\boldsymbol{d}]_i \leftarrow [\top]$
end
`updatevector`$([\boldsymbol{d}], [\boldsymbol{s}], [0])$
for $i \leftarrow 1$ **to** $|V|$ **do**
  for $l \leftarrow 1$ **to** $|E|$ **do**
    $e_l = e_{j,k}$
    $[x] \leftarrow [\boldsymbol{d}]_j - [\boldsymbol{d}]_k + [\boldsymbol{w}]_l$
    $[c] \leftarrow [x] < 0$
    `/* the condition` $[c]$ `is a shared` $[1]$ `if` $[\boldsymbol{d}]_k > [\boldsymbol{d}]_j + [\boldsymbol{w}]_e$ `which means that` $[\boldsymbol{d}]_k$ `must be updated */`
    $[\boldsymbol{d}]_k \leftarrow [\boldsymbol{d}]_k + [c][x]$
    $[\boldsymbol{p}]_k \leftarrow ([1] - [c])[\boldsymbol{p}]_k + [c]j$
    `/* if the condition is met, we update` $[\boldsymbol{p}]_k$ `to vertex` v_j `(labelled to the number` j`) */`
  end
end

6.4.2 Dijkstra's algorithm

Dijkstra's algorithm computes the shortest path from the source to all vertices in the graph, that is, the shortest path tree rooted at the source. The algorithm is greedy. At each iteration one vertex (the one with the smallest distance label) is permanently updated to the *scanned* status.

Adapting Dijkstra. The fact that Dijkstra's algorithm goes through the graph in an order that depends on the weight of the edges makes it very difficult to efficiently exploit the sparsity of a graph: our best solutions have all a complexity that amounts to the one of a complete graph, and we therefore use the matrix representation in all the cases for our protocol.

Algorithm 6.6 (**SSPD**) presents our *secure shortest path algorithm based on Dijkstra*. Note that `updatevector` refers to Algorithm 6.1 and that `updaterow` is the natural extension of `updatevector` for replacing a complete row in a shared matrix. Algorithm `binarymin` was introduced by Toft in [Tof09] to obtain the minimal value out of a vector of shared values. It securely computes a shared minimal value, $[\min]$, along with a shared index $[\boldsymbol{k}]$ of its position. The protocol uses $O(n)$ comparisons and multiplications. Its overall round complexity is $O(\log(n))$ rounds. Vector $[\boldsymbol{q}]$ records the status of each vertex. An entry is set to $[0]$ if the corresponding vertex has not been scanned yet. It is updated to $[\top]$ as soon as the vertex has been scanned.

The main differences between the traditional and our secure version of Dijkstra's algorithm happen in the inner loop:

1. On Line 7, the loop goes through all vertices instead of only considering the neighbours of the current vertex. In particular, this includes an always transparent step where we consider the current vertex and gives a substantial overhead if a public sparse graph is considered.

2. On Lines 6 and 21, we need to go through all the elements of a row or a vector, even if we know that only one of them is going to be updated.

Those two modifications contribute to the same effect: they increase the original complexity of Dijkstra from $O(|V|^2)$ to $O(|V|^3)$. More precisely, the exact number of comparisons is $2|V|^2 - 3|V| + 1$ and the exact number of dot products (used for the multiplication of vectors, costing $|V|$ multiplications) is $2|V|^2 - |V|$ for $|V| \geq 4$.

As the comparison protocol we use requires 165 multiplications to compute 1 comparison, the number of multiplications to compute the

Algorithm 6.6: SSPD secure shortest path based on Dijkstra's algorithm

Input: A graph $G = (V, E)$ where V is the list of vertices and E the list of edges, a matrix of shared weights $[\boldsymbol{M}]_{i,j}$ for $i, j \in \{1, ..., |V|\}$ and a shared index of the position of the source vertex $[\boldsymbol{s}]$ in unary representation.

Output: The vector of distances $[\boldsymbol{d}]_i$ and the matrix of predecessor $[\boldsymbol{P}]_{i,j}$ for $i, j \in \{1, ..., |V|\}$.

1 **for** $i, j \leftarrow 1$ **to** $|V|$ **do**
2 $\quad [\boldsymbol{P}]_{i,j} \leftarrow [0]$
3 $\quad [\boldsymbol{d}]_i \leftarrow [\top]$
4 $\quad [\boldsymbol{q}]_i \leftarrow [0]$
5 **end**
6 `updatevector`$([\boldsymbol{d}], [\boldsymbol{s}], [0])$
7 **for** $i \leftarrow 1$ **to** $|V|$ **do**
8 $\quad [\boldsymbol{d'}] \leftarrow [\boldsymbol{d}] + [\boldsymbol{q}]$
9 $\quad [\min], [\boldsymbol{k}] \leftarrow$ `binarymin`$([\boldsymbol{d'}])$
10 $\quad$ `/* select a vertex` v_k `that has not been visited yet, its position is stored in` $[\boldsymbol{k}]$ `*/`
11 $\quad$ **for** $j \leftarrow 1$ **to** $|V|$ **do**
12 $\quad\quad$ `/* visit all the adjacent vertices of` v_k `*/`
13 $\quad\quad [x] \leftarrow ([\boldsymbol{d}] + [\boldsymbol{M}]^t_{*,j}) \cdot [\boldsymbol{k}]$
14 $\quad\quad$ `/*` $[x]$ `stores the distance of` v_j `if reached from` v_k `by using edge` $e_{k,j}$ `*/`
15 $\quad\quad [c] \leftarrow [x] < [\boldsymbol{d}]_j$
16 $\quad\quad$ `/* if the current distance of` v_j `is greater than` $[x]$`, update the matrix of predecessor and the list of distances accordingly as follows */`
17 $\quad\quad [\boldsymbol{P'}]_{j,*} \leftarrow [\boldsymbol{P}]_{j,*} + [c]([\boldsymbol{k}] - [\boldsymbol{P}]_{j,*})$
18 $\quad\quad [\boldsymbol{P}] \leftarrow$ `updaterow`$([\boldsymbol{P}], j, [\boldsymbol{P'}]_{j,*})$
19 $\quad\quad [\boldsymbol{d}]_j \leftarrow [\boldsymbol{d}]_j + [c]([x] - [\boldsymbol{d}]_j)$
20 $\quad$ **end**
21 $\quad$ `updatevector`$([\boldsymbol{q}], [\boldsymbol{k}], [\top])$
22 $\quad$ `/* vertex` v_k `is labelled to ''scanned'' by setting` $[\boldsymbol{q}]_k$ `to` $[\top]$ `*/`
23 **end**

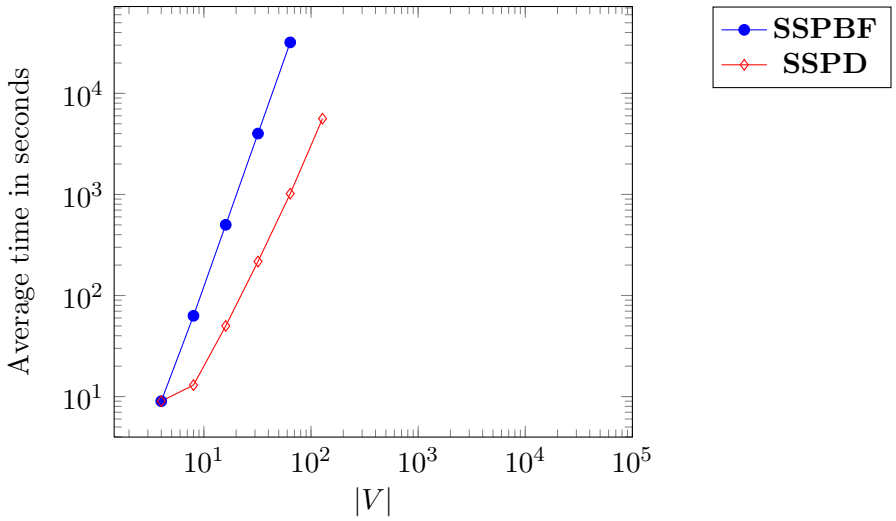


Figure 6.4: Execution times of Algorithms 6.5 **SSPBF** and 6.6 **SSPD** for a complete shortest path tree ($|E| = (|V| - 1)^2$). The slopes of the lines are ~ 3 for **SSPBF** and ~ 2.4 for **SSPD**.

shortest path in a complete tree is $2|V|^3 + 329|V|^2 - 495|V| + 165$. The switch from quadratic to cubic dominance is at 165 vertices which is precisely the number of multiplications used by a single comparison (see also [Maw15, Figure 6.1]).

Our secure version of Dijkstra comes with an overhead of a factor $|V|$ compared to the original one, even when the graph structure can be considered as public. We note that this was not the case in the work of Brickell [BS05] who considered running Dijkstra securely as well, but accepted to output the shortest paths step by step. Besides the limitation that this brings when the protocol has to be composed, we also observe that our algorithm can be used to solve problems that could not be solved by Brickell's approach, namely, computing the shortest path between two specific vertices without leaking any other information: their approach indeed leaks the shortest path to **all** vertices.

6.4.3 Implementation prototype

We ran the two shortest path protocols described above on complete graphs of various sizes. This first showed that Algorithm 6.5 can only be conveniently used for graphs where $|V||E| \approx 10^3$ (a few minutes on a standard laptop): see Figure 6.4.

The secure versions of Bellman-Ford and Dijkstra have approximately the same complexity for complete graphs. However the quadratic

number of comparisons makes it possible to run our secure version of Dijkstra on a 64-vertex complete graph in roughly twice the time taken by Bellman-Ford on a 16-vertex graph, and we were able to run it up to a 128-vertex complete graph (i.e., counting 16256 directed edges) in a bit more than an hour.

While these timings might look fairly high, they still make it possible to solve natural problems in a reasonable time. For instance, if the 3-party, 12-vertex problem outlined above could be solved in about 30 seconds.

For completeness, we point at another secure implementation of Dijkstra [Maw15, Section 6.2.6] that is based on the work of Toft [Tof11] on priority queues. Investigating various secure data structures is a good starting point to try to mitigate the cost of privacy in regular data structures where, for example, we have to consider a complete graph even when the graph is strongly sparse. Unfortunately, the approach based on priority queues does not seem to bring enough benefits compared to the solutions proposed above.

6.5 Privacy-preserving maximum flow problem

In an oriented graph where the edges have a constraint of *capacity*, the maximum flow problem consists in finding the maximum number of units that can be carried from a vertex called *source* to another vertex called *sink*. The *flow* through an edge designates the number of units passing by it. This number cannot exceed the capacity.

This problem has numerous classical applications. In the spirit of our previous examples, one of them could be competing transport companies willing to determine the capacity they could reach if they decided to achieve a joint-venture. It is natural in such a context to expect that these companies will not be willing to disclose their full network structure to each other. As in the case of the shortest path, algorithms solving the maximum flow problem are also very useful as subroutines for solving other problems. The minimum cut problem is one such traditional example, which can be solved using $O(|V|)$ invocations of the maximum flow algorithm. Solving this problem is then useful to determine where the weak points of the joint network would be.

Although we investigated many different algorithms for a transportation in SMC as the Ford-Fulkerson and the Push/Relabel algorithms, this chapter only presents two secure protocols based on the Edmonds-Karp's algorithm.

6.5.1 Edmonds-Karp's algorithm

The basic idea of the Edmonds-Karp's (EK) algorithm is to find an augmenting path in the residual graph that is the graph in which the edges are weighted by their residual capacity, i.e., the capacity minus the current flow. Each augmenting path increases the total flow so that the algorithm eventually terminates when there are no augmenting paths left. The increase is monotonic and paths are considered only once. Typically, the EK algorithm uses a breadth-first search to find the next augmenting path.

The asymptotic complexity of the traditional Edmonds-Karp's algorithm is $O(|V||E|^2)$, which we can match securely. As we have seen in the case of the shortest path problem, this complexity will be prohibitive even for very small graphs if they are complete. It therefore makes sense to focus our attention on (oriented) strongly sparse graphs, of which we consider the structure to be public. More precisely, we consider graphs in which the number of paths from the source to the sink is fairly small, e.g., bounded by a small polynomial in the number of vertices.

Algorithm 6.7 for *secure maximum flow based on Edmonds-Karp* (**SMFEK**) is given a list on input which contains all the paths sorted in a growing order of length, $\boldsymbol{p} = (p_1, ..., p_k)$ where k is the number of paths in the graph. This list is not secret as the structure is not, and can therefore be easily constructed in public. As before, the `binarymin` function refers to the function introduced by Toft in [Tof09] to compute the minimum value in a list.

The main differences between this protocol and Edmonds-Karp's approach are:

1. the public enumeration of all the paths instead of building of a breadth-first search for capacity augmenting paths, and

2. the treatment of all the paths as if they were augmenting.

Algorithm **SMFEK** is correct as the set of all the augmenting paths is contained in the set of all the paths $\boldsymbol{p}$. Moreover, it ensures the privacy of the edge capacities as no information is leaked about which path of $\boldsymbol{p}$ is augmenting and which is not.

It is easy to see that the **SMFEK** requires $O(k|V|)$ comparisons, as the length of the longest path in the graph is bounded by $|V| - 1$, and $O(k)$ multiplications. This protocol makes a crucial use of the existence of a small number of paths in the graph, something that we were not able to use in the Algorithm **SSPD** for instance. It is however highly inefficient for dense graph and would have a factorial complexity for complete graphs.

Algorithm 6.7: SMFEK secure maximum flow based on Edmonds-Karp's algorithm

Input: A graph $G = (V, E)$ where V is the list of vertices and E the list of edges, a source vertex $so \in V$, a sink vertex $si \in V$, and a list $\boldsymbol{p}$ of length k containing the paths between so and si sorted in a growing order of length. A set of capacities $[\boldsymbol{c}]_e$ and a set of flows $[\boldsymbol{f}]_e$ initially set to $[0]$ for $e \in E$. $\bar{e}$ is the edge opposite to e.

Output: The maximum flow value from so to si.

while $|\boldsymbol{p}| > 0$ **do**
  $p \leftarrow \texttt{pop}(\boldsymbol{p})$
  $[m] \leftarrow \texttt{binarymin}_{e \in p}([\boldsymbol{c}]_e - [\boldsymbol{f}]_e)$
  $[b] \leftarrow [m] > 0$
  $[x] \leftarrow [b][m]$
  `/* if` $[m]$ `is positive it is added to the current flow */`
  for $e \in p$ **do**
    $[\boldsymbol{f}]_e \leftarrow [\boldsymbol{f}]_e + [x]$
    $[\boldsymbol{f}]_{\bar{e}} \leftarrow [\boldsymbol{f}]_{\bar{e}} - [x]$
  end
end
return $\sum_{e \in S} [\boldsymbol{f}]_e$ where $S = \{e \in E | e\text{'s origin is } so\}$

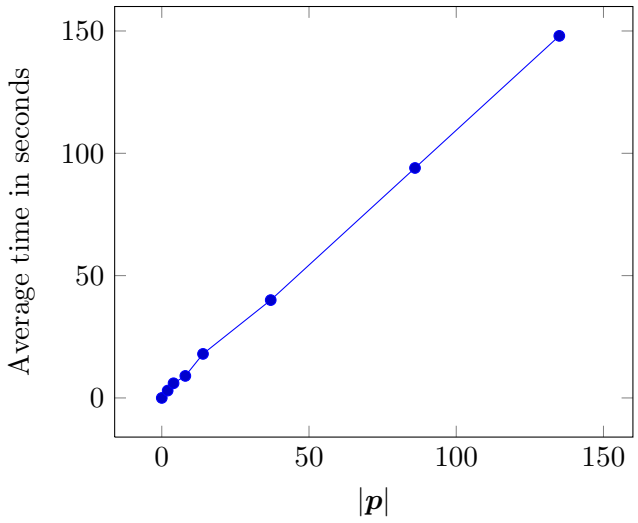


Figure 6.5: Execution times of Algorithm 6.7 for 10-vertex graphs.

This algorithm applies well to our previous example of the three competing logistic companies trying to determine the max flow in their joint networks. If we consider a case with 10 vertices and 37 different paths, the execution takes less than a minute as shown in Figure 6.5.

Algorithm 6.8 (**SMFCEK**) presents the *secure maximum flow implementation based on the complete version of Edmonds-Karp's algorithm*. The `SSPBF` function is a natural adaptation of Algorithm 6.5 that outputs the shortest path from the source to the sink in the form of a vector denoted $[\boldsymbol{p}]$ of $\{[0],[1]\}$ where a [1] at the i-th position indicates that edge i belongs to the augmenting path. Note that the first augmenting path $\boldsymbol{p}$ is public and given in input. We refer to Figure 6.1 for the value of the bound $\top$.

In line 21 of Algorithm 6.8, the update of $[\boldsymbol{w}]$ is used for `SSPBF` call so that edge j can no longer be part of a shortest path if $[\boldsymbol{c}]_j = [\boldsymbol{f}]_j$ and if no other shortest paths exist in regards of the initial graph.

The main differences between algorithm **SMFCEK** and the traditional Edmonds-Karp's algorithm are as follows :

- Each iteration goes through all the edges but only those which form the current path are updated.
- The **SSPBF** algorithm is used instead of the Breath-First-Search (BFS) algorithm to find the smallest augmenting path because there is a serious overhead in a straightforward secure implementation of the BFS. To run the **SSPBF** algorithm, **SMFCEK** main-

Algorithm 6.8: SMFCEK secure maximum flow based on complete Edmonds-Karp's algorithm

Input: A graph $G = (V, E)$ where V is the list of vertices and E the list of edges, a source vertex $so \in V$, a sink vertex $si \in V$. A list of positive capacities $[\boldsymbol{c}]_i$ for each edge. The first augmenting path $\boldsymbol{p}$.

Output: The maximum flow value from so to si.

for $i \leftarrow 1$ **to** $|E|$ **do**
  $[\boldsymbol{f}]_i \leftarrow [0]$
  $[\boldsymbol{w}]_i \leftarrow [1]$
end
for $i \leftarrow 1$ **to** $|E|$ **do**
  for $j \leftarrow 1$ **to** $|E|$ **do**
    $[\boldsymbol{c'}]_j \leftarrow (1 - [\boldsymbol{p}]_j)\top + [\boldsymbol{c}]_j$
    /* $[\boldsymbol{c'}]$ is a copy of $[\boldsymbol{c}]$ where the capacity of an edge not in $[\boldsymbol{p}]$ is set to $\geq [\top]$ */
    $[\boldsymbol{f'}]_j \leftarrow [\boldsymbol{p}]_j[\boldsymbol{f}]_j$
    /* $[\boldsymbol{f'}]$ is a copy of $[\boldsymbol{f}]$ where the flow of an edge not in $[\boldsymbol{p}]$ is set to $[0]$ */
  end
  $[m] \leftarrow \texttt{binarymin}_{j \in \{1,\ldots,|E|\}}([\boldsymbol{c'}]_j - [\boldsymbol{f'}]_j)$
  /* $[m]$ is the minimum value that path $[\boldsymbol{p}]$ can be augmented with */
  $[b_1] \leftarrow [m] > 0$
  $[x] \leftarrow [b_1][m]$
  /* if $[m]$ is positive, path $[\boldsymbol{p}]$ is augmented with $[m]$ */
  for $j \leftarrow 1$ **to** $|E|$ **do**
    $[\boldsymbol{f}]_j \leftarrow [\boldsymbol{f}]_j + [\boldsymbol{p}]_j[x]$
    $[\boldsymbol{f}]_{\bar{j}} \leftarrow [\boldsymbol{f}]_{\bar{j}} - [\boldsymbol{p}]_j[x]$
    $[b_2] \leftarrow [\boldsymbol{c}]_j - [\boldsymbol{f}]_j = 0$
    $[\boldsymbol{w}]_j \leftarrow [b_2]|V| + (1 - [b_2])[\boldsymbol{w}]_j$
  end
  $[\boldsymbol{p}] \leftarrow \texttt{SSPBF}(G, [\boldsymbol{w}], so, si)$
  /* find the next shortest augmenting path $[\boldsymbol{p}]$ in the graph */
end
return $\sum_{e \in S} [\boldsymbol{f}]_e$ where $S = \{e \in E | e\text{'s origin is } so\}$

tains a list of shared weights $[\boldsymbol{w}]$ for the edges where the weight of an edge is [1] when it remains in the residual graph and it is $[|V|]$ otherwise.

It is straightforward to see that the asymptotic complexity of the algorithm is $O(|V||E|^2)$ as the original algorithm. The number of comparisons is $|V||E|^2 + |E|^2 + |E|$ and the number of multiplications is $|V||E|^2 + 5|E|^2 + |E|$.

6.6 Conclusion

In this chapter, we proposed several algorithms for securely computing combinatorial problems and particularly graph problems. First we presented two sorting algorithms that sort elements represented in unary notation. Then, we proposed secure versions of the Bellman-Ford's and Dijkstra's algorithm to determine the shortest path in a graph. And finally, we detailed a secure version and its variant of the Edmonds-Karp's algorithm used to find the maximum flow in a graph.

Besides the interest that these protocols have in the numerous contexts in which their insecure counterparts found applications in the past (possibly relying on a trusted third party), our investigation raised interesting complexity gaps between centralized algorithms and secure protocols, ranging from a constant to something growing like the number of vertices in the graphs. It is then natural to wonder whether these gaps, when they arise, can be decreased. Various avenues appear for that purpose:

- Design efficient datastructures adapted to the investigated problems.
- Investigate whether secure comparisons, which are often a bottleneck, can be traded for other, cheaper, arithmetic operations. This raises unusual questions from a traditional algorithmic point of view, as comparisons are usually considered as basic operations.

Considering other standard combinatorial problems could also provide new insights. The protocols and results presented in this chapter are prototypes that validate the theoretical complexity evaluations. While the running times given for the protocols look unpractical for large graphs, this issue must be put in perspective. Indeed, an implementation for concrete applications should definitively be improved by relying on lower level programming languages and optimized underlying libraries. Various optimization techniques (see, e.g., [BDOZ11, DPSZ12]) would

lead to performance increases of several orders of magnitude, as has been observed in the case of the AES during the last 3 years for instance (see [DKL+12] and the references within). Promising new frameworks as SPDZ [DKL+13] are bringing SMC closer and closer to much more real-world applications.

This chapter, however, did not focus on the verifiability property that this thesis studies. This property can be obtained at will in all the protocols presented above by enabling the active security that the SMC framework provides. For example, in VIFF, this active security mode is optional while passive security is by default. We decided in our implementations to test our algorithms only in the passive mode due to two incentives: first, the active security mode of VIFF is provided with no strong guarantee on behalf of the authors. Second and more importantly, enabling active security would lead to a significant increase of the execution times of our algorithms which, in some cases, would prevent us from testing even the smallest cases. The choice we made of disabling active security is not dramatic as the conclusions of this chapter remain unchanged since they are mainly privacy related.

Part III

Conclusion

Chapter 7
Conclusion and Future Directions

Secure multi-party computation is a growing branch of cryptography. It aims at providing secure solutions for multi-party protocols that seek specific security properties. These properties are mainly – but not only – related to privacy and verifiability concerns. Among the many ways to achieve these properties in multi-party protocols, we find that most of today's solutions are based on human trust assumptions. For example, a group of voters delegate the elections process to a group of authorities which is entrusted not to leak individual votes and to compute the result honestly. As another example, a company confides its sensitive data to a consultancy service for analysis, computations and advise. In this case, the good reputation of the consultancy service serves as a guarantee for the company to ensure that the data is not sold to competitors and that some real work has been performed on the data. This situation presents some drawbacks that we do not detail here but we can say that it works to some extend. One of the strengths of the cryptographic approach to meet these problems is that it offers solutions that get rid of the human trust or the physical assumptions. This elimination of the trust assumption opens a world of new multi-party applications, in particular for applications where it was too risky for the parties to trust each other.

In this thesis, we present cryptography-based solutions to relevant multi-party protocols. We focused on the verifiability property in such protocols. This property is crucial to guarantee the correctness of the protocol execution. Still, we managed to ensure the privacy property, with some tints, for all our solutions. Our approach is against the stream of current existing cryptography-based solutions that prefer to stress the privacy before the verifiability. Among other things, this led us to propose solutions with interesting asymptotic complexities in the verification process.

Our work targets secure multi-party applications and proposes so-

lutions to real-world problems, as well as a generic way to build these solutions for other multi-party problems. We endeavour to give detailed context and proofs of security as well as a complexity analysis and concrete implementations through our prototype for every problem we tackled.

Contributions. Our first main contribution concerns the field of cryptographic voting. We presented several voting schemes that solve an open-problem in this area which is to conciliate the privacy of the voters' ballots and the verifiability of the election in its whole. Our proposal took into account efficiency and practicability concerns. This work found its way to be part of an actual voting scheme called "STAR-Vote" [BBB+13] which may be deployed for future U.S. elections in Texas.

To support this new voting systems, we developed a new cryptographic primitive called commitment consistent encryption which is of independent interest. We show how this primitive can be used to obtain a perfectly private audit trail for several multi-party applications. We detailed and proved the security mechanisms playing at the core of this perfectly private audit trail.

The concrete use of this primitive can be found in our proposal on secure multi-party function evaluation. We tackled the problem of securely evaluating a multi-party function on secret values by providing a perfectly private audit trail of its computation. The solution proposed is generic but we illustrate it with three unrelated problems, solving a linear system of equations, electronic auctions and finding the shortest path in a graph. We proposed secure algorithms to solve these problems and we provided an online prototype that implements our solutions for direct use.

Finally, we investigated other multi-party applications related to simple combinatorial problems. In this case the approach was more classic in the sense that we used already deployed secure multi-party techniques. However, the results obtained are highly relevant since they concern general problems that arise when performing secure multi-party computations. We highlighted the differences in the efficiency metrics and the complexity gaps that modify the way to design a secure algorithm. We pointed out the privacy leakages that may happen in a naive translation of classical algorithms to their secure counterpart. Note that we also proposed new sorting algorithms based on the unary notation that may prove to be useful outside the cryptographic field.

7.1 Open problems

As it frequently happens in a thesis, some ideas were found to be unfruitful, some promising tracks could not be followed by lack of time and most answers to our questions led to more interesting questions. All good things must come to an end, and thus here are the directions our research would hypothetically have followed.

During the last 4 years, we have acknowledged a steady progress in the timing computations of basic operations in secure multi-party computation. These progresses rest in part on a strong use of precomputations. For example, the "SPDZ" solution of Damgård et al. [DKL+13] relies on multiplication triplets which are precomputed and then used online to speed up multiplications. These kinds of work are currently a game-changer in the field and bring SMC closer and closer to large scale real-world applications. It would be worth applying the results and precomputations techniques to our protocols to benefit from this speed-up.

One problem that could not find a solution in this thesis was the idea of aggregating the proofs of knowledge in a generic verifiable function evaluation in order to reduce their length and their computational cost. This technique is known as batching proofs of knowledge and was introduced by Bellare et al. in [BGR98a, BGR98b] for fast verification of multiple modular exponentiations. More recently, Bayer and Groth [BG13] proposed a zero-knowledge argument for polynomial evaluation. If applied to our secure multi-party function evaluation, this would mean a drastic decrease in the complexity of the proof of correctness.

In Chapter 5, we applied our PPAT protocol to several applications. The choice of those applications was partly based on the previously investigated combinatorial problems of Chapter 6 within the VIFF framework. One of the motivations was to perform a qualitative comparison between auctions and sorting and the shortest path problem in both settings. Nevertheless, regarding the shortest path problem, we only considered the secure solution based on the Bellman-Ford algorithm. In this case, the verification of the correctness of the result (the shortest path in the form of a predecessors list and a distances list) is obtained for the clients by running the algorithm entirely. However, opposite of what is done for the auctions and the linear system, the complexity for the clients is exactly the complexity of the algorithm itself (Bellman-Ford) whereas for the two other problems, the complexity is lower. It remains an open problem to seek an algorithm that, given the shortest path in a graph, checks that it is indeed the shortest path while running with a smaller complexity than the shortest path algorithm itself. As far as we

know, this algorithm is yet to be found. On the other hand, although it was not investigated, it was suggested that verifying some properties of a modified version of the dual graph accounting the shortest path might give the expected result.

Finally, it would be interesting to adapt our setting to dynamic functionalities. Indeed, our generic construction is based on a clients-worker setting where we limit the number of interactions to its minimum. The clients send their inputs to the worker and publishes their commitments and proofs. Then, the worker performs the computations and publish the result and proof of correctness. Here we miss a group of multi-party applications where the clients can dynamically react to the outputs of the worker or even interact with each other. This category of applications is possibly full of new challenges for cryptography-based solutions.

Bibliography

[ACHdM05] Giuseppe Ateniese, Jan Camenisch, Susan Hohenberger, and Breno de Medeiros. Practical group signatures without random oracles. *IACR Cryptology ePrint Archive*, 2005:385, 2005.

[ACKR13] Myrto Arapinis, Véronique Cortier, Steve Kremer, and Mark Ryan. Practical everlasting privacy. In *POST*, volume 7796, pages 21–40. Springer, 2013.

[ACM+13] Abdelrahaman Aly, Édouard Cuvelier, Sophie Mawet, Olivier Pereira, and Mathieu Van Vyve. Securely solving simple combinatorial graph problems. In *Financial Cryptography and Data Security*, pages 239–257. Springer, 2013.

[AdMP12] Ben Adida, Olivier de Marneffe, and Olivier Pereira. Helios voting system, 2012. https://vote.heliosvoting.org/.

[ADMPQ09] Ben Adida, Olivier De Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of helios. *EVT/WOTE*, 9:10–10, 2009.

[AHO10] Masayuki Abe, Kristiyan Haralambiev, and Miyako Ohkubo. Signing on elements in bilinear groups for modular protocol design. *IACR Cryptology ePrint Archive*, 2010:133, 2010.

[AHO12] Masayuki Abe, Kristiyan Haralambiev, and Miyako Ohkubo. Group to group commitments do not shrink. In *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 301–317. Springer, 2012.

[AKL+11] Diego F. Aranha, Koray Karabina, Patrick Longa, Catherine H. Gebotys, and Julio López. Faster explicit formulas for computing pairings over ordinary curves. In *Advances in Cryptology – EUROCRYPT 2011*, pages 48–68. Springer, 2011.

[BBB+13] Susan Bell, Josh Benaloh, Michael D. Byrne, Dana DeBeauvoir, Bryce Eakin, Gail Fisher, Philip Kortum, Neal McBurnett, Julian Montoya, Michelle Parker, Olivier Pereira, Philip B. Stark, Dan S. Wallach, and Michael Winn. Star-vote: A secure, transparent, auditable, and reliable voting system. *The USENIX Journal of Election Technology Systems, 1 (1)*, pages 18–37, 2013.

[BBFR14] Michael Backes, Manuel Barbosa, Dario Fiore, and Raphael M. Reischuk. Adsnark: Nearly practical and privacy-preserving proofs on authenticated data. Cryptology ePrint Archive, Report 2014/617, 2014.

[BBJ+09] Elaine Barker, William Burr, Alicia Jones, Timothy Polk, Scott Rose, Miles Smid, and Quynh Dang. Recommendation for key management part 3: Application-specific key management guidance. *NIST special publication*, 800:57, 2009.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Advances in Cryptology – CRYPTO 2004*, pages 41–55. Springer, 2004.

[BCC+12] Steve Babbage, Dario Catalano, Carlos Cid, Benne de Weger, Orr Dunkelman, Christian Gehrmann, Louis Granboulan, Tim Güneysu, Jens Hermans, Tanja Lange, Arjen Lenstra, Chris Mitchell, Mats Näslund, Phong Nguyen, Christof Paar, Kenny Paterson, Jan Pelzl, Thomas Pornin, Bart Preneel, Christian Rechberger, Vincent Rijmen, Matt Robshaw, Andy Rupp, Martin Schläffer, Serge Vaudenay, Fré Vercauteren, and Michael Ward. Ecrypt yearly report on algorithms and keysizes. Technical report, 2012.

[BCD+09] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, et al. Secure multiparty computation goes

live. In *Financial Cryptography and Data Security*, pages 325–343. Springer, 2009.

[BCP+11] David Bernhard, Véronique Cortier, Olivier Pereira, Ben Smyth, and Bogdan Warinschi. Adapting Helios for provable ballot privacy. In *Computer Security – ESORICS 2011*, LNCS. Springer, 2011.

[BDJ+06] Peter Bogetoft, Ivan Damgård, Thomas P. Jakobsen, Kurt Nielsen, Jakob Pagter, and Tomas Toft. A practical implementation of secure auctions based on multiparty integer computation. In *Financial Cryptography*, volume 4107, pages 142–147. Springer, 2006.

[BDNP08] Assaf Ben-David, Noam Nisan, and Benny Pinkas. Fairplaymp: a system for secure multi-party computation. In *CCS*, pages 257–266. ACM, 2008.

[BDO14] Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. *IACR Cryptology ePrint Archive*, 2014:75, 2014.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *Advances in Cryptology – EUROCRYPT 2011*, pages 169–188. Springer, 2011.

[BDPR98] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology – CRYPTO'98*, pages 26–45. Springer, 1998.

[Bei11] Amos Beimel. Secret-sharing schemes: a survey. In *Coding and cryptology*, pages 11–46. Springer, 2011.

[Ben87] J Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, Yale University, 1987.

[BF97] Dan Boneh and Matthew Franklin. Efficient generation of shared RSA keys. In *Advances in Cryptology – CRYPTO'97*, pages 425–439. Springer, 1997.

[BF01] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology – CRYPTO 2001*, pages 213–229. Springer, 2001.

[BFK+09] Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Secure evaluation of private linear branching programs with medical applications. In *ESORICS 2009*, volume 5789, pages 424–439. Springer, 2009.

[BFL+11] Mauro Barni, Pierluigi Failla, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Privacy-preserving ECG classification with branching programs and neural networks. *IEEE TIFS*, 6(2):452–468, June 2011.

[BG93] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *Advances in Cryptology – CRYPTO'92*, pages 390–420. Springer, 1993.

[BG13] Stephanie Bayer and Jens Groth. Zero-knowledge argument for polynomial evaluation with application to blacklists. In *Advances in Cryptology – EUROCRYPT 2013*, pages 646–663. Springer, 2013.

[BGDM+10] Jean-Luc Beuchat, Jorge E. González-Díaz, Shigeo Mitsunari, Eiji Okamoto, Francisco Rodríguez-Henríquez, and Tadanori Teruya. High-speed software implementation of the optimal ate pairing over Barreto–Naehrig curves. In *Pairing-Based Cryptography – Pairing 2010*, pages 21–39. Springer, 2010.

[BGR98a] Mihir Bellare, Juan A. Garay, and Tal Rabin. Batch verification with applications to cryptography and checking. In *LATIN'98: Theoretical Informatics*, pages 170–191. Springer, 1998.

[BGR98b] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology – EUROCRYPT'98*, pages 236–250. Springer, 1998.

[Bla79] George R. Blakey. Safeguarding cryptographic keys. In *Proc. AFIPS 1979 National Computer Conf.*, volume 48, pages 313–317, 1979.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Advances in Cryptology – ASIACRYPT 2001*, pages 514–532. Springer, 2001.

[BLS03] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In *Security in Communication Networks*, volume 2576 of *LNCS*, pages 257–267. Springer, 2003.

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Computer Security-ESORICS 2008*, pages 192–206. Springer, 2008.

[BN06] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected areas in cryptography*, pages 319–331. Springer, 2006.

[BNTW12] Dan Bogdanov, Margus Niitsoo, Tomas Toft, and Jan Willemson. High-performance secure multi-party computation for data mining applications. *International Journal of Information Security*, 11(6):403–418, 2012.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC*, pages 1–10. ACM, 1988.

[BPSW07] Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In *ACM CCS*, CCS '07, pages 498–507. ACM, 2007.

[BPW12] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In *Advances in Cryptology – ASIACRYPT 2012*, pages 626–643. Springer, 2012.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[BS99] Mihir Bellare and Amit Sahai. Non-malleable encryption: Equivalence between two notions, and an indistinguishability-based characterization. In *Advances in cryptology – CRYPTO'99*, pages 519–536. Springer, 1999.

[BS05] Justin Brickell and Vitaly Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In *ASIACRYPT*, volume 3788, pages 236–252. Springer, 2005.

[BSMD10] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. Sepia: Privacy-preserving aggregation of multi-domain network events and statistics. *Network*, 1:101101, 2010.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 136–145. IEEE, 2001.

[CCC+10] Richard Carback, David Chaum, Jeremy Clark, John Conway, Aleksander Essex, Paul S. Herrnson, Travis Mayberry, Stefan Popoveniuc, Ronald L. Rivest, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II municipal election at Takoma Park: The first E2E binding governmental election with ballot privacy. In *USENIX Security Symposium*, pages 291–306. USENIX Association, 2010.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19. ACM, 1988.

[CCJT14] Sébastien Canard, Iwen Coisel, Amandine Jambert, and Jacques Traoré. New results for the practical use of range proofs. In *Public Key Infrastructures, Services and Applications*, pages 47–64. Springer, 2014.

[CDN10] Ronald Cramer, Ivan Damgård, and Jesper Nielsen. Secure multiparty computation. *Book draft*, 2010.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, volume 839 of *LNCS*, pages 174–187. Springer, 1994.

[CEC+08] David Chaum, Aleksander Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan Sherman, and Poorvi L. Vora. Scantegrity: End-to-End voter-verifiable

optical-scan voting. *IEEE Security and Privacy*, 6(3):40–46, 2008.

[CF85] Josh Cohen and Michael Fischer. A robust and verifiable cryptographically secure election scheme. In *Proceedings of 26th Symposium on Foundations of Computer Science*, pages 372–382, Portland, OR, 1985. IEEE.

[CFA+05] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren. *Handbook of elliptic and hyperelliptic curve cryptography*. CRC press, 2005.

[CFH+15] Craig Costello, Cedric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile verifiable computation. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, May 2015.

[CFSY96] Ronald Cramer, Matthew Franklin, Berry Schoenmakers, and Moti Yung. Multi-autority secret-ballot elections with linear work. In *Advances in Cryptology – EUROCRYPT '96*, volume 1070 of *LNCS*, pages 72–83. Springer, 1996.

[CG99] Ran Canetti and Shafi Goldwasser. An efficient *threshold* public key cryptosystem secure against adaptive chosen ciphertext attack. In *Advances in Cryptology – EUROCRYPT '99*, volume 1592 of *LNCS*, pages 90–106. Springer, 1999.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM (JACM)*, 51(4):557–594, 2004.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *LNCS*, pages 103–118. Springer, 1997.

[CKKC13] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Carlos Cid. Multi-client non-interactive verifiable computation. In *Theory of Cryptography*, pages 499–518. Springer, 2013.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[CP93] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *Advances in Cryptology – CRYPTO'92*, pages 89–105. Springer, 1993.

[CP14] Édouard Cuvelier and Olivier Pereira. Multi-party function evaluation with perfectly private audit trail. 2014. In submission.

[CPP13] Édouard Cuvelier, Olivier Pereira, and Thomas Peters. Election verifiability or ballot privacy: Do we need to choose? In *Computer Security – ESORICS 2013*, pages 481–498. Springer, 2013.

[Cra97] Ronald Cramer. *Modular Design of Secure, yet Practical Cryptographic Protocols*. PhD thesis, Univeristy of Amsterdam, 1997.

[Cuv15] Édouard Cuvelier. Implemention of the PPAT schemes and the prototype applications, 2015. `https://github.com/ecuvelier/PPAT`.

[Dam99] Ivan Damgård. Commitment schemes and zero-knowledge protocols. In *Lectures on Data Security*, pages 63–86. Springer, 1999.

[Dam04] Ivan Damgård. On Σ-protocols. `http://www.daimi.au.dk/~ivan/Sigma.ps`, 2004.

[Dam07] Ivan Damgård. A "proof-reading" of some issues in cryptography. In *Automata, Languages and Programming*, pages 2–11. Springer, 2007.

[DDN98] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In *SIAM Journal on Computing*. Citeseer, 1998.

[DF02] Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *Advances in Cryptology – ASIACRYPT 2002*, pages 125–142. Springer, 2002.

[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 *Theory of Cryptography*, pages 285–304. Springer, 2006.

[DGKN09] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. In *Public Key Cryptography – PKC 2009*, pages 160–179. Springer, 2009.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[DJ01] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Public Key Cryptography*, pages 119–136. Springer, 2001.

[DKL+12] Ivan Damgård, Marcel Keller, Enrique Larraia, Christian Miles, and Nigel Smart. Implementing AES via an actively/covertly secure dishonest-majority MPC protocol. In *Security and Cryptography for Networks*, volume 7485, pages 241–263. Springer, 2012.

[DKL+13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel Smart. Practical covertly secure MPC for dishonest majority – or: Breaking the SPDZ limits. In *Computer Security – ESORICS 2013*, pages 1–18. Springer, 2013.

[DMN11] Ivan Damgård, Sigurd Meldgaard, and Jesper Buus Nielsen. Perfectly secure oblivious ram without random oracles. In *Proceedings of the 8th TCC*, pages 144–163. Springer-Verlag, 2011.

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *CRYPTO*, volume 2729, pages 247–264. Springer, 2003.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, volume 7417, pages 643–662. Springer, 2012.

[DSD07] Augusto Jun Devegili, Michael Scott, and Ricardo Dahab. Implementing cryptographic pairings over Barreto-Naehrig curves. In *Pairing-Based Cryptography – Pairing 2007*, pages 197–207. Springer, 2007.

[DvdG12] Denise Demirel and Jeroen van de Graaf. A publicly-verifiable mixnet with everlasting privacy towards observers. Cryptology ePrint Archive, Report 2012/420, 2012.

[DvdGA12] Denise Demirel, Jeroen van de Graaf, and Roberto Araújo. Improving Helios with everlasting privacy towards the public. In A. Halderman and O. Pereira, editors, *EVT/WOTE 2012*. USENIX, 2012.

[EFLL12] Yael Ejgenberg, Moriya Farbstein, Meital Levy, and Yehuda Lindell. Scapi: The secure computation application programming interface. *IACR Cryptology ePrint Archive*, 2012:629, 2012.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology*, pages 10–18. Springer, 1985.

[FMY98] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Robust efficient distributed RSA-key generation. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 663–672. ACM, 1998.

[FOO93] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *Advances in Cryptology – AUSCRYPT '92*, volume 718 of *LNCS*, pages 244–251. Springer, 1993.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology – CRYPTO'86*, pages 186–194. Springer, 1987.

[FS01] Pierre-Alain Fouque and Jacques Stern. Fully distributed threshold RSA under standard assumptions. In *Advances in Cryptology – ASIACRYPT 2001*, pages 310–330. Springer, 2001.

[FST10] David Freeman, Michael Scott, and Edlyn Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of cryptology*, 23(2):224–280, 2010.

[FT87] Michael L. Fredman and Robert E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[FVV09] Junfeng Fan, Frederik Vercauteren, and Ingrid Verbauwhede. Faster $\mathbb{F}_p$-arithmetic for cryptographic pairings on Barreto-Naehrig curves. In *Cryptographic Hardware and Embedded Systems – CHES 2009*, pages 240–253. Springer, 2009.

[Gau86] Carl F. Gauss. *Disquisitiones Arithmeticae*. Springer, 1986.

[Gei10] Martin Geisler. *Cryptographic protocols: theory and implementation*. PhD thesis, Aarhus University Denmark, Department of Computer Science, 2010.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Symposium on theory of computing – STOC'09*, pages 169–169. ACM Press, 2009.

[GGG+14] Shafi Goldwasser, Samuel Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *Advances in Cryptology – EUROCRYPT 2014*, pages 578–602. Springer, 2014.

[Gir15] Damien Giry. Cryptographic key length recommendation, 2015. http://www.keylength.com/.

[GJKR07] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptology*, 20(1):51–83, 2007.

[GKL+15] Samuel Dov Gordon, Jonathan Katz, Feng-Hao Liu, Elaine Shi, and Hong-Sheng Zhou. Multi-client verifiable computation with stronger security guarantees. In *Theory of Cryptography*, pages 144–168. Springer, 2015.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 291–304. ACM, 1985.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, May 1996.

[Gol05] Oded Goldreich. *Foundations of cryptography: a primer*, volume 1. Now Publishers Inc, 2005.

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[GRBR13] Gurchetan S. Grewal, Mark Ryan, Sergiu Bursuc, and Perer Y. A. Ryan. Caveat coercitor: Coercion-evidence in electronic voting. In *IEEE Security and Privacy Symposium*, 2013.

[Gro10] Jens Groth. A verifiable secret shuffle of homomorphic encryptions. *Journal of Cryptology*, 23(4):546–579, 2010.

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 101–111. ACM, 1998.

[HDNP11] Laurie Haustenne, Quentin De Neyer, and Olivier Pereira. Elliptic curve cryptography in javascript. *IACR Cryptology ePrint Archive*, 2011:654, 2011.

[HICT14] Koki Hamada, Dai Ikarashi, Koji Chida, and Katsumi Takahashi. Oblivious radix sort: An efficient sorting algorithm for practical secure multi-party computation. *IACR Cryptology ePrint Archive*, 2014:121, 2014.

[HKI+13] Koki Hamada, Ryo Kikuchi, Dai Ikarashi, Koji Chida, and Katsumi Takahashi. Practically efficient multi-party sorting protocols from comparison sort algorithms. In *Information Security and Cryptology – ICISC 2012*, pages 202–216. Springer, 2013.

[HKS+10] Wilko Henecka, Stefan Kögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. Tasty: tool for automating secure two-party computations. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 451–462. ACM, 2010.

[HL10] Carmit Hazay and Yehuda Lindell. *Efficient secure two-party protocols: Techniques and constructions*. Springer Science & Business Media, 2010.

[HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *LNCS*, pages 132–150. Springer, 2011.

[HS00] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 539–556. Springer, 2000.

[HSV06] Florian Hess, Nigel P. Smart, and Frederik Vercauteren. The eta pairing revisited. *Information Theory, IEEE Transactions on*, 52(10):4595–4602, 2006.

[IP07] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In *Theory of Cryptography, TCC 2007*, volume 4392, pages 575–594. Springer, 2007.

[JJR02] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *USENIX Security Symposium*, pages 339–353. USENIX, 2002.

[JKU11] Kristján Valur Jónsson, Gunnar Kreitz, and Misbah Uddin. Secure multi-party sorting and applications. In *ACNS'11: Proceedings of the 9th international conference on Applied Cryptography and Network Security*. Citeseer, 2011.

[Jou00] Antoine Joux. A one round protocol for tripartite Diffie–Hellman. In *Algorithmic number theory*, pages 385–393. Springer, 2000.

[Joy08] Marc Joye. Fast point multiplication on elliptic curves without precomputation. In *Arithmetic of Finite Fields*, pages 36–46. Springer, 2008.

[KJGB06] Louis Kruger, Somesh Jha, Eu-Jin Goh, and Dan Boneh. Secure function evaluation with ordered binary decision diagrams. In *ACM CCS*, pages 410–420. ACM, 2006.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, pages 104–113. Springer-Verlag, 1996.

[KRS10] Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In *Computer Security – ESORICS 2010*, volume 6345 of *LNCS*, pages 389–404. Springer, 2010.

[KTV10] Ralph Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: definition and relationship to verifiability. In *CCS '10*, pages 526–535. ACM, 2010.

[KTV12] Ralph Küsters, Tomasz Truderung, and Andreas Vogt. Clash attacks on the verifiability of e-voting systems. In *IEEE Symposium on Security and Privacy (S&P 2012)*, pages 395–409. IEEE Computer Society, 2012.

[KZS+09] David Kammler, Diandian Zhang, Peter Schwabe, Hanno Scharwächter, Markus Langenberg, Dominik Auras, Gerd Ascheid, and Rudolf Mathar. Designing an asip for cryptographic pairings over Barreto-Naehrig curves. In *11th Workshop on Cryptographic Hardware and Embedded Systems (CHES '09)*, Lausanne, Switzerland, September 2009.

[LPS08] Yehuda Lindell, Benny Pinkas, and Nigel P Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *Security and Cryptography for Networks*, pages 2–20. Springer, 2008.

[Lyn07] Ben Lynn. *On the implementation of pairing-based cryptosystems*. Ph. d. thesis, Stanford University, 2007.

[Maw15] Sophie Mawet. *Negotiations using Secure Multi-Party Computation.* Ph. d. thesis, Université catholique de Louvain, 2015.

[Mil04] Victor S. Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 17(4):235–261, 2004.

[MN06] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In *26th International Cryptology Conference (CRYPTO'06)*, volume 4117 of *LNCS*, pages 373–392. Springer, 2006.

[MN10] Tal Moran and Moni Naor. Split-ballot voting: Everlasting privacy with distributed trust. *ACM Transactions on Information and System Security*, 13(2), 2010.

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay-secure two-party computation system. In *USENIX Security Symposium*, volume 4. San Diego, CA, USA, 2004.

[MOV93] Alfred J. Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *Information Theory, IEEE Transactions on*, 39(5):1639–1646, 1993.

[MVOV96] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography.* CRC press, 1996.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *Advances in Cryptology – CRYPTO 2012*, pages 681–700. Springer, 2012.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in cryptology – EUROCRYPT'99*, pages 223–238. Springer, 1999.

[Ped91] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology – EUROCRYPT'91*, pages 522–526. Springer, 1991.

[Ped92] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology – CRYPTO'91*, pages 129–140. Springer, 1992.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 238–252. IEEE, 2013.

[PRST08] David C. Parkes, Michael O. Rabin, Stuart M. Shieber, and Christopher Thorpe. Practical secrecy-preserving, verifiably correct and trustworthy auctions. *Electronic Commerce Research and Applications*, 7(3):294–312, 2008.

[PSJNB11] Geovandro C.C.F. Pereira, Marcos A. Simplício Jr, Michael Naehrig, and Paulo S.L.M. Barreto. A family of implementation-friendly BN elliptic curves. *Journal of Systems and Software*, 84(8):1319–1326, 2011.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In *Advances in Cryptology – ASIACRYPT 2009*, pages 250–267. Springer, 2009.

[RBH+09] Peter Y.A. Ryan, David Bismark, James Heather, Steve Schneider, and Zhe Xia. The Prêt à Voter verifiable election system. *IEEE Transactions on Information Forensics and Security*, 4:662–673, 2009.

[RHH14] Aseem Rastogi, Matthew A Hammer, and Michael Hicks. Wysteria: A programming language for generic, mixed-mode multiparty computations. In *IEEE Symposium on Security and Privacy 2014*, pages 655–670. IEEE, 2014.

[RSA78] Ronald L. Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[RST08] Michael O. Rabin, Rocco A. Servedio, and Christopher Thorpe. Highly efficient secrecy-preserving proofs of correctness of computation, April 18 2008. US Patent App. 12/105,508.

[Sal06] Roy G. Saltman. *The history and politics of voting technology*. Palgrave Macmillan, 2006.

[SBC+09] Michael Scott, Naomi Benger, Manuel Charlemagne, Luis J. Dominguez Perez, and Ezekiel J. Kachisa. On the final exponentiation for calculating pairings on ordinary elliptic curves. In *Pairing-Based Cryptography – Pairing 2009*, pages 78–88. Springer, 2009.

[Sch09] Peter Schwabe. BN-curves generator, 2009. http://www.ti.rwth-aachen.de/research/cryptography/bncurves.php.

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

[Sil95] Joseph H. Silverman. *The arithmetic of elliptic curves*. Springer-Verlag, Berlin, 1995.

[SK95] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme – a practical solution to the implementation of a voting booth. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology – EUROCRYPT '95*, volume 921 of *LNCS*, pages 393–403. Springer, 1995.

[SSW09] Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. Efficient privacy-preserving face recognition. In *ICISC*, volume 5984, pages 229–244. Springer, 2009.

[ST81] Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 114–122, New York, NY, USA, 1981. ACM.

[Tof07] Tomas Toft. *Primitives and Applications for Multi-party Computation*. PhD thesis, Department of Computer Science, Aarhus University, 2007.

[Tof09] Tomas Toft. Solving linear programs using multiparty computation. In *Financial Cryptography*, volume 5628, pages 90–107. Springer, 2009.

[Tof11] Tomas Toft. Secure data structures based on multi-party computation. In *PODC*, pages 291–292. ACM, 2011.

[TW10] Björn Terelius and Douglas Wikström. Proofs of restricted shuffles. In Daniel J. Bernstein and Tanja Lange, editors,

Progress in Cryptology – AFRICACRYPT 2010, volume 6055 of *LNCS*, pages 100–113. Springer, 2010.

[Ver10] Frederik Vercauteren. Optimal pairings. *Information Theory, IEEE Transactions on*, 56(1):455–461, 2010.

[VIF] VIFF. Viff: Virtual ideal functionality framework. http://viff.dk/.

[Wik09] Douglas Wikström. A commitment-consistent proof of a shuffle. In Colin Boyd and Juan Manuel González Nieto, editors, *Information Security and Privacy, 14th Australasian Conference, ACISP 2009*, volume 5594 of *LNCS*, pages 407–421. Springer, 2009.

[Yao82] Andrew C. Yao. Protocols for secure computations. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 160–164. IEEE, 1982.

[ZPK14] Yupeng Zhang, Charalampos Papamanthou, and Jonathan Katz. Alitheia: Towards practical verifiable graph processing. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 856–867. ACM, 2014.

Appendix A

Memento on number theory

A.1 Finite groups and fields

Definition A.1. *A **group** G is a set S and a composition law $\star : S \times S \to S$ such that,*

- $\forall x, y, z \in S$, $(x \star y) \star z = x \star (y \star z)$ *(associativity).*
- $\exists e \in S : \forall x \in S$, $x \star e = e \star x = x$ *(neutral element).*
- $\forall x \in S$, $\exists w \in S : x \star w = w \star x = e$ *(inverse).*

By convenience, we say that x is an element of the group $G := (S, \star)$ and we write $x \in G$ when $x \in S$. The law $\star$ is commutative when $\forall x, y \in G$, $x \star y = y \star x$. In this case the group is said *commutative* or *abelian*. One might use additive ($\oplus$) or multiplicative ($\otimes$) notation for the group law. In each case we use the convenient notations for $x \in G$ and $k \in \mathbb{Z}$:

$$kx := \underbrace{x \oplus \cdots \oplus x}_{k\ times} \text{ and } x^k := \underbrace{x \otimes \cdots \otimes x}_{k\ times}$$
$$e_\oplus = 0 \text{ and } e_\otimes = 1$$

The number of elements of S is called the order of G. When this number is finite, the group is said *finite*. An example of finite group is the set of integers modulo $n \in \mathbb{Z}_0$ with the classic addition operation ($+$). For $x, y \in \mathbb{Z}$, we say that the equivalence relation noted $x \sim_n y$ or $x \equiv y \mod n$ is true if $\exists k \in \mathbb{Z}$ such that $x = y + kn$. We note the equivalence class of x as $[x]_n := \{y \in \mathbb{Z} | x \equiv y \mod n\}$. As a result we define the group $\mathbb{Z}_n := (\{[x]_n | x \in \mathbb{Z}\}, \oplus)$ where $[x]_n \oplus [y]_n := [x + y]_n$. This group is finite and has order n.

Definition A.2. *A **subgroup** H of a group $G := (S, \star)$ is a group $(T, \star)$ where $T \subseteq S$.*

As a consequence, the neutral element e of G is the neutral element of H. For an element $x \in G$, we build the *generated subgroup* $\langle x \rangle \subseteq G$ as $\{y \in G | y = kx \text{ with } k \in \mathbb{Z}\}$. If there exists $x \in G$ such that $\langle x \rangle = G$, we say that G is *cyclic* and that x is a *generator* of G. The group $\mathbb{Z}_n$ is cyclic and $[1]_n$ is a generator of $\mathbb{Z}_n$.

Theorem A.1 (Lagrange)**.** *Let H be a subgroup of G. The order of H divides the order of G.*

Definition A.3. *A **ring** R is a set S equipped with two internal composition laws $\star$ and $\diamond$ such that,*

- *$(S, \star)$ is a commutative group. The neutral element is noted $e_\star$.*
- *$\diamond$ is associative.*
- *there exists a neutral element for $\diamond$ denoted $e_\diamond$ and different from $e_\star$.*
- *$\forall x, y, z \in S$, $x \diamond (y \star z) = x \diamond y \star x \diamond z$ and $(x \star y) \diamond z = x \diamond z \star y \diamond z$ (distributivity).*

We say that the ring is *commutative* if the law $\diamond$ is. We call the **characteristic** of the ring $R := (S, \oplus, \otimes)$, the number $k \in \mathbb{N}$ such that $k \otimes e_\otimes = e_\oplus$ $(k.1 = 0)$. If this number does not exist, we say by convention that the characteristic is 0. It is sometimes possible to identify two rings with the same structure if one can find an application between the two that respects the composition laws. This application is called a **ring homomorphism**.

Definition A.4. *A **ring homomorphism** ψ is an application from ring $R_1 := (S_1, \star, \diamond)$ to ring $R_2 := (S_2, \oplus, \otimes)$ such that $\forall x, y \in R_1$,*

- $\psi(x \star y) = \psi(x) \oplus \psi(y)$
- $\psi(x \diamond y) = \psi(x) \otimes \psi(y)$
- $\psi(e_\diamond) = e_\otimes$

We can extend the ring structure to an even richer structure called field.

Definition A.5. *A **field** F is a ring $R = (S, \star, \diamond)$ such that $\forall x \in R \setminus \{e_\star\}, \exists w \in R : x \diamond w = e_\diamond = w \diamond x$. In other words, every non neutral element is invertible.*

Definition A.6. *A **subfield** H of a field F is a field such that every operation including inversion between two elements in H produces an element of H.*

We define a **field isomorphism** in the same way as Definition A.4. The **order** and **characteristic** of a field ensue from previous definitions. One can naturally extend the group $\mathbb{Z}_p$, with the classic multiplication law (.) to form a finite field whenever p is a prime number. In this case, we use the notation $\mathbb{F}_p$ to mark out the prime field. In general, $\mathbb{F}_q$ designates a finite field of order q.

Theorem A.2. *The characteristic of a field is either* 0 *or a prime number p.*

Theorem A.3. *The order of a finite field is p^d where p is prime and $d \in \mathbb{N}$.*

For a field F and $x \in F^* := F \backslash \{e_\oplus\}$, it is convenient to define $o(x)$ as the smallest integer t such that $x^t = e_\otimes$. We call this integer the **order** of x.

Theorem A.4. *Let F be a field of order q, then F^* contains at most one element of order $q-1$ and $\forall x \in F^*$, $o(x)$ divides $q-1$.*

Following from Theorem A.4, we can see that $\forall x \in F^*$, $x^{q-1} = 1$. Moreover, F^* is a multiplicative group that is cyclic.

We can build large fields by adding elements to a base field. The result is called an **extension field**. For example, we can see that $\mathbb{C}$ is an extension field of $\mathbb{R}$ obtained by adding the imaginary number i to $\mathbb{R}$.

Definition A.7. *The field L is an **extension field** of the field K if there exists a field homomorphism from K to a subfield of L.*

We denote a field extension by $L : K$ to be read L "over" K.

Definition A.8. *The **degree** of a field extension $L : K$ is the dimension of the vector space L over K. This number is denoted $[L : K]$. If this number is finite, we say that the field extension $L : K$ is finite.*

Building extension field over each other results in a **tower** of extension fields.

Theorem A.5. *Given the tower of extension fields $M : L : K$, we have that $[M : K] = [M : L][L : K]$.*

The classic way to define extension field is first, to choose a polynomial which has no root in the base field and second, to define the field that is the quotient of the base field by the polynomial.

Definition A.9. *Given a field F, we define $F[x]$ as $\{a_0 + a_1x + a_2x^2 + ... + a_nx^n | a_0, ..., a_n \in F, n \in \mathbb{N}\}$.*

Definition A.10. *Given a field F and a polynomial $P \in F[x]$, we define $f + P.F[x]$ as $\{f + P.g | g \in F[x]\}$.*

Definition A.11. *Given a field F and a polynomial $P \in F[x]$, we define the quotient $F[x]/P.F[x]$ as $\{f + P.F[x] | f \in F[x]\}$. In this case, $f_1 + P.F[x] = f_2 + P.F[x]$ iff $f_1 - f_2$ is a multiple of P.*

Theorem A.6. *Given a field F and a polynomial $P \in F[x]$, $F[x]/P.F[x]$ is a field if and only if P is **irreducible** which means that there are no polynomials $Q_1, Q_2 \in F[x]$ with $\deg Q_1, \deg Q_2 < \deg P$ such that $P = Q_1Q_2$.*

Definition A.12. *Given a field F and an element $r \in F$, we define the **minimal polynomial** of r as the monomial polynomial $M \in F[x]$ such that r is a root of M and M is of minimal degree.*

Theorem A.7. *Given a finite field F of characteristic p such that $F \supset \mathbb{F}_p$. For $r \in F$, we note $\mathbb{F}_p[r] := \{f(r) | f \in \mathbb{F}_p[x]\}$. We have that $\mathbb{F}_p[r]$ is a subfield of F and if $P \in \mathbb{F}_p[x]$ is the minimal polynomial of r, then there exists an isomorphism from $\mathbb{F}_p[x]/P.\mathbb{F}_p[x]$ to $\mathbb{F}_p[r]$.*

It is now easy to see that $\mathbb{F}_p[x]/P.\mathbb{F}_p[x]$ is an extension field of $\mathbb{F}_p[x]$ by adding the roots of an irreducible polynomial P. The degree of this extension is exactly d, the degree of P. We can show that this extension is isomorphic to $\mathbb{F}_{p^d}$. Without entering in too many details, we present an important automorphism of field.

Definition A.13. *Given a finite field F of characteristic p. The automorphism ϕ defined by $\phi : F \to F : x \mapsto x^p$ is called the **Frobenius automorphism**. It is easy to see that $\phi(x + y) = x^p + y^p$ and that $\phi(x.y) = x^p y^p$.*

In extension fields, the Frobenius automorphism can be used to compute large exponentiation as what is done in Section 3.5.1.

A.2 Elliptic curves

Elliptic curves form a specific kind of groups that are widely used in cryptography. Their main advantage is that they offer a fast group law

and that the discrete logarithm problem is hard to solve (see Section 2.1.2) in such groups. Moreover, the level of security they offer for a reduced key size compares with cryptography based on the factorization problem (see key length recommendations [BBJ+09, BCC+12, Gir15]).

We present here a condensed view of elliptic curves.

Definition A.14. *Given a finite field F, an **elliptic curve** over F is denoted $E_F := (E, \oplus)$ where*

$$E := \{x, y \in F | y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \mathcal{O}_\infty$$

with fixed $a_1, ..., a_6 \in F$ such that

- $\nexists (x_0, y_0) \in E : 2y_0 + a_1x_0 = 0$ *and,* $3x_0^2 + 2a_2x_0 + a_4 - a_1y_0 = 0$ *simultaneously. We say that the elliptic curve is **non-singular**. This prevents us from using curves with isolated points or cusps that are problematic for computations.*

- $\mathcal{O}_\infty$ *is called the **point at infinity**. It is the neutral element for the addition law $\oplus$ which means that $\forall P \in E : P \oplus \mathcal{O}_\infty = \mathcal{O}_\infty \oplus P = P$.*

- *for $P, Q \in E \backslash \{\mathcal{O}_\infty\}$ where $P = (P_x, P_y)$ and $Q = (Q_x, Q_y)$, we define $P \oplus Q = R := (R_x, R_y)$ as follows: at first, if $P = -Q$ which means that $P_x = Q_x$ and $P_y = -Q_y$, then $R = \mathcal{O}_\infty$. In the general case,*

$$\begin{aligned} R_x :=& \lambda^2 + a_1\lambda - a_2 - P_x - Q_x \\ R_y :=& \lambda(P_x - R_x) - P_y - a_1R_x - a_3 \end{aligned}$$

where the parameter λ is given by

$$\lambda := \begin{cases} \frac{P_y - Q_y}{P_x - Q_x} & \text{if } P \neq Q \\ \\ \frac{3P_x^2 + 2a_2P_x + a_4 - a_1P_y}{2P_y + a_1P_x + a_3} & \text{if } P = Q \end{cases}$$

The equation of the curve

$$E_{we} \equiv y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

is called the *Weirstraß equation*. When the characteristic of F is > 3, it is possible to find an isomorphism over F to represent the elliptic curve in a *short Weirstraß equation*,

$$E_{swe} \equiv y^2 = x^3 + a_4x + a_6.$$

Proposition A.1. *The elliptic curve $E_F := (E, \oplus)$ of Definition A.14 is a group.*

An elliptic curve E over an extension field F of degree k might take up a non negligible part of the computational efforts when we want to perform operations on E_F. Fortunately, it is sometimes possible to find an isomorphism between the curve E_F and another curve $E'_{F'}$ on an extension field F' with a lower degree $k' < k$ (compared to a base field). The isomorphism means that every point on the curve E_F can be projected on a point on the curve E' and vice-versa. This isomorphic curve $E'_{F'}$ is called a **twist** of the curve E_F, or the twisted curve of E_F. We say that this twist is, quadratic, cubic, sextic, etc. if the degree of the field extension $[F : F']$ is 2, 3, 6, etc.

To conclude this section on elliptic curves, we point out that the Frobenius endomorphism $\phi : E_F \to E_F : (x, y) \mapsto (x^p, y^p)$ where p is the characteristic of F, is, among other things, a powerful tool to perform large scalar multiplication of points over extension fields. Indeed, we can often precompute values involved in exponentiations of multiple of p which makes them almost free. A good, but rather tedious, use of this method is given in Section 3.5 in the details of the pairing implementation (which is adapted from [BGDM+10]).

A.3 Pairings

At the scale of the history of cryptography, *pairings* are young and provoke enthusiasm among the research community as a useful tool in the hands of cryptographers. This tool is used in the applications presented in this thesis and we give here a condensed introduction.

At first, pairings were developed as a way to reduce the complexity of the discrete logarithm problem on elliptic curve by transferring it to a smaller group [MOV93]. However, when carefully choosing the elliptic curve, one can avoid this attack by rendering the discrete logarithm problem still infeasible in the smaller group. Nowadays, pairings are used in various cryptographic schemes such as tripartite Diffie-Hellman key exchange [Jou00], identity based encryption [BF01] or short signatures [BLS01].

A pairing is a bilinear mapping from two groups to a third one, all sharing the same order. We refer to the book of Cohen et al. [CFA+05] for a more detailed mathematical presentation of pairings and we focus on the essential.

Definition A.15. *A **pairing** $\mathcal{P}air$ is composed of three groups $G_1 :=$*

$(S_1, +)$, $G_2 := (S_2, +)$, $G_3 := (S_3, .)$ of the same prime order q and a map $e : G_1 \times G_2 \to G_3$ such that:

- *$\forall (P, Q) \in G_1 \times G_2$, $\forall a, b \in \mathbb{Z}$ we have that $e(aP, bQ) = e(P, Q)^{ab}$ (bilinearity)*
- *$\forall (P, Q) \in G_1 \times G_2$, we have that $e(P, Q) \neq 1_{G_3}$ (non degeneracy)*
- *$\forall (P, Q) \in G_1 \times G_2$, $e(P, Q)$ is efficiently computable (efficiency)*

When $G_1 \neq G_2$, we say that the pairing is asymmetric ($\mathcal{P}air_{asym}$) and in this case, there exists an isomorphism $\theta : G_2 \to G_1$. In the specific case where $G_1 = G_2$, we call the pairing symmetric ($\mathcal{P}air_{sym}$). Pairings offer numerous possibilities with some restrictions concerning the isomorphism θ and the possibility to hash arbitrary string into G_2. These constraints have been summarized by Galbraith et al. in [GPS08]. However, these restrictions do not concern our schemes.

Asymmetric pairings are usually instantiated on ordinary elliptic curves, G_1 is a cyclic subgroup of an elliptic curve $E_{\mathbb{F}_p}$, G_2 is a subgroup of $E_{\mathbb{F}_{p^k}}$ and G_3 is a subgroup of $\mathbb{F}^*_{p^k}$. More precisely, G_1 and G_2 are the set of q-torsion points (see Definition A.16) and G_3 is the cyclic group of the q-th roots of unity. The degree k of the extension field $\mathbb{F}_{p^k}$ is also called the **embedding** degree of the pairing.

Definition A.16 (Torsion points)**.** *Given an elliptic curve E_F over a field F, we define the set of q-**torsion points** denoted $E_F[q]$ as $\{P \in E_F | qP = \mathcal{O}_\infty\}$ which means that the order of P is either q or a factor of q. Note that $E_{\mathbb{F}_{p^k}}[q]$ contains exactly q^2 points and we have an isomorphism between $E_{\mathbb{F}_{p^k}}[q]$ and $\mathbb{Z}_q \times \mathbb{Z}_q$.*

Several families of elliptic curves are studied and well suited for pairing based cryptography. Following Ben Lynn's classification [Lyn07, Chapter 4], we point out type A, B and E for symmetric pairings and type D, F and G for asymmetric pairings. For a complete description of the families, see [FST10]. Note that type F curves (called Barreto-Naehring or BN-curves [BN06]) are used for prototyping in this thesis (see Chapters 3). These curves of equation $E \equiv y^2 = x^3 + b$ in $\mathbb{F}_p$ contain a prime order n subgroup where p and n are primes determined through the integer parameter u:

$$p = p(u) = 36u^4 + 36u^3 + 24u^2 + 6u + 1$$
$$q = q(u) = 36u^4 + 36u^3 + 18u^2 + 6u + 1$$

We chose to work with the BN-curves for several reasons. First, they are nowadays well used which makes the reproduction of our result facilitated if one wants to use an implementation of the BN-curves

different than ours. We recall that our code is available online [Cuv15]. There, we also provide a magma script that can be used to double check the computations performed with our libraries. Second, and related to the first point, many improvements of the pairings on BN-curves exist and this brought a huge upswing in the timing results (about a hundred times better than with our first naive implementation). Finally, the generation of random curves generator is pretty simple and efficient (there seems to be a good density of u such that p and q are primes although no bound on such density have been proved yet). There is an online curve generator provided on the University of Aachen website at [Sch09, KZS+09].

As previously said, the computation of the bilinear mapping $e : G_1 \times G_2 \to G_3$ must be efficient. In this view, we use bilinear mapping such as the *Weil pairing* or the *Tate pairing* whose descriptions can be found in [Sil95] for instance. Miller's algorithm allows computing Weil and Tate pairings in a straight way [Mil04]. We find many refinements of this algorithm as well as variants of the Tate pairing such as the *ate pairing* [HSV06].