Exploring the Limits of Verifiable Voting

Henri Devillez

Louvain School of Engineering
ICTEAM
UCLouvain
Louvain-la-Neuve
Belgium

Thesis Committee:

Dr. Josh Benaloh	*Microsoft Research*
Pr. Laurent Francis	*UCLouvain*
Dr. Benoît Libert	*ZAMA*
Pr. Olivier Pereira	*UCLouvain*
Pr. Thomas Peters	*UCLouvain*
Pr. Peter Y.A. Ryan	*University of Luxembourg*
Pr. François-Xavier Standaert	*UCLouvain*

This work was supported by the F.R.S-FNRS (FRIA grant).

Acknowledgements

This thesis marks the end of a journey that started four years ago. Along this journey, many people have accompagnied me, and this section is my way to acknowledge them. Writing a thesis is not an easy task, and I am absolutely certain that their support has been crucial in the completion of this work.

First and foremost, I would like to express my immense gratitude to Olivier and Thomas. I am especially grateful for their availability, their patience, and their precious guidance. During these four years, I was lucky to spend countless hours in Thomas' office trying to understand his clever ideas and in Olivier's office where I could benefit from his vast knowledge and his keen eye for details. I am certain that I will vividly remember our endless discussions. I simply cannot imagine a better supervision for my thesis.

Then, I would like to thank my jury and scientific committee for our insightful discussions during my private defense. Specifically, I am thankful to Benoît and Peter for being in my scientific committee from the beginning of my thesis, and to Josh for joining the jury despite the early schedule in Seattle. Their thorough review and numerous feedbacks have been invaluable to improve the quality of this manuscript. I am also thankful to François-Xavier and Laurent for being, respectively, the secretary and president of the jury.

I have often heard that pursuing a PhD is meant to be a long and solitary process, but in this regard, I feel privileged to have been surrounded by my colleagues of the Crypto Group. I am grateful for the countless coffees, beers, jokes, conversations, and mahjong games we shared over these four years. I would also like to thank the many people I was lucky to meet in my endeavours, including – without attempting to be exhaustive – Constance, the Louvain-la-Neuve mahjong club, the midi-math team, the Deckers, Julien "et les autres", my competitive programming students, and all my friends.

Finalement, je remercie du fond du coeur ma famille pour son soutien indéfectible, l'amour inconditionnel que j'ai pu recevoir ces vingt-neuf dernières années, et surtout pour m'avoir guidé et éclairé dans mes moments de doute. Ensemble, il n'y a aucune épreuve que nous ne puissions surmonter.

Abstract

Designing secure voting protocols is a notoriously challenging task, yet it helps offering a more verifiable and more accessible electoral process. Among the difficulties faced when building such protocols, receipt-freeness, or the inability of voters to prove how they voted, is a particularly hard property to achieve.

Cryptography, the science of secure communications, has become a key ingredient in this quest of building privacy-friendly and trustworthy voting protocols. In this thesis, we develop new cryptographic algorithms in this regard. Equipped with these new tools, we then proceed to build secure voting protocols, with a particular attention to receipt-freeness.

The first part of this thesis introduces a new cryptographic primitive, Traceable Receipt-Free Encryption and a new security notion, traceable CCA security, which are particularly well-suited to generically build single-pass receipt-free protocols. We then propose efficient realizations of this primitive based on standard cryptographic assumptions.

The limits of this primitive are also explored, and we expose that it is impossible to achieve another property, cast-as-intended verifiability, without any interactive process with the voting server when the voting client is not trusted. This interaction can either have the form of a prior trusted registration or a casting process in several rounds. On the positive side, we build new protocols and we show that it is possible to attain these two properties simultaneously in each of these interactive scenarios.

The second part of the thesis focuses on more practical settings. First, we explore how a popular and well-established ballot preparation protocol can be optimized. Especially, this protocol needs to verifiably encrypt many bits representing the choices of voters, and we study how to efficiently compute these encryptions and a proof that they effectively contain bits.

Finally, we investigate postal voting protocols, an alternative to online voting systems. While being extensively used in practice in many countries, postal voting received little attention from the academic community so far. Hence, we propose a new security model and a new paper-based protocol that achieves better verifiability properties.

Contents

1. Introduction **1**
1.1. The cryptographic approach 2
1.2. Secure voting protocols 3
1.3. Outline of the thesis 8

2. Preliminaries **13**
2.1. Foundations 13
2.2. Generic building blocks 14
2.3. Verifiable voting definitions 24

I. Receipt-Freeness **47**

3. Traceable Receipt-Free Encryption **49**
3.1. Related works 50
3.2. Traceable Receipt-Free Encryption 52
3.3. Relations with more classical security definitions 61
3.4. Generic voting scheme based on a TREnc 67
3.5. Concluding remarks 78

4. TREnc constructions **81**
4.1. Generic construction 82
4.2. Trusted setup construction 91
4.3. Public coin construction for encrypting a group element . 98
4.4. Public coin construction for encrypting a sequence of bits 115
4.5. Performance and comparisons 127
4.6. Concluding remarks 128

5. Can we cast a ballot as intended and be Receipt-Free? **131**
5.1. Related works 132
5.2. New definitions for interactive voting protocols 133
5.3. Impossibility results 141
5.4. Protocols 145
5.5. Security of the protocols 157
5.6. Performance and comparisons 166
5.7. Concluding remarks 167

II. Practical Voting Scheme **171**

6. How to Verifiably Encrypt Many Bits for an Election? **173**
6.1. The Baseline 174
6.2. Fixed-base exponentiation 175
6.3. Multi-ElGamal 179
6.4. Adapting the ZK 0-1 proofs – Linear Techniques 180
6.5. Adapting the ZK 0-1 proofs – Logarithmic batching . . . 184
6.6. Benchmarking 186
6.7. Concluding remarks 190

7. Secure Postal Voting **191**
7.1. Related works 192
7.2. Postal Voting Model 193
7.3. Protocol overview 196
7.4. Protocol 200
7.5. Security 207
7.6. Concluding remarks 226

8. Conclusion **229**

List of Notations **233**

Bibliography **237**

1. Introduction

Our world is becoming more and more connected through the Internet. Nowadays, we shop, get informed, date, and entertain ourselves online. However, when it comes to participating in an election, we still need, in most cases, to come to a polling place physically. Furthermore, voting is done in many places with a paper ballot rather than with an electronic voting machine. So, it is natural to ask ourselves why we still vote in such a traditional way while most of our daily actions are now performed in a connected and digital way.

The reason is that voting *securely* is hard. First, we desire to offer *vote secrecy* to the voters to protect them and ensure they can freely express their choice without coercion. In a traditional election, the voting booth is a simple yet efficient way to provide this property,[1] which is not satisfied as easily in a remote election. Second, we desire the election process to be *trustworthy*. The voters should have confidence that their ballot has not been tampered with and that their vote has been correctly counted. Using an electronic device brings an inevitable uncertainty: what if the machine operates maliciously and computes the result of the election out of thin air?

Combining at the same time these two antagonistic objectives, privacy and verifiability, is notoriously difficult and traditional paper-based voting works well. Before asking ourselves whether we could design a secure electronic voting scheme, we should stop and think whether we should. And in fact, voting devices can bring advantages that are worth considering. As a first example, electronic devices can help make voting more accessible to persons with disabilities. Furthermore, it might be difficult for some voters, such as expatriates or people living in a low-density area, to access a voting polling station and Internet voting can definitely help them partake in an election.

On top of that, technology can be used in conjunction with paper-based voting. In a purely paper-based election, a voter does not know in practice if the ballot they put in the ballot box is indeed correctly counted or not. Even if the tallying process was fully transparent, it would require them to watch the ballot box until the tally and watch

[1] Or at least used to be before the advent of smartphones and other small recording devices [Ben13].

carefully the tally process to be completely convinced that their vote is counted. With an electronic voting protocol, we can for example scan the paper ballots and encrypt them to have publicly verifiable electronic records, but we still have the original ballots to audit the result obtained electronically [BST19]. With this in mind, there are convincing reasons to try and design electronic elections. And to achieve this, we need other tools beyond a pen, paper, and a voting booth.

1.1. The cryptographic approach

Cryptography is the science of secure communications, which appears as a perfect tool for building secure voting protocols. The first task of a cryptographer is to build algorithms. For example, a symmetric encryption scheme is a set of algorithms that can be used to hide a message using a key and then recover the message using the same key. On the other hand, there are two keys in an asymmetric (or public key) encryption scheme: a public key, used to encrypt the messages, and a secret key, used to decrypt them. We often use a branch of mathematics called number theory to build these algorithms. But building alone is not sufficient if we do not know what our algorithm is doing or if looking back at our example, we do not define what "hide" means.

Hence, the second task of the cryptographer is to formally model the properties our constructions achieve.[2] Especially, we should ask ourselves against *whom* we want our algorithm to be secure. We call this entity the *adversary*, and depending on the assumptions we made on the power of this adversary, we can formally define what we expect from our algorithm.

For example, an important property of encryption schemes is the ciphertext indistinguishability, which is defined as a game. In this game, we ask the adversary to give us two messages and encrypt one of them. The adversary then tries to guess which of those messages was encrypted, and if it cannot guess it with a probability substantially better than one over two, then this means that our encryption hides the message in some way. We can control the power of the adversary by giving it, for example, the public key used to encrypt the messages or access to a decryption oracle that decrypts ciphertexts of its choice.[3] These games can tell how well the encryption scheme hides the messages, and depending on

[2] Timewise, this second task usually comes before the design of the protocol: we set what we want before achieving what we want.

[3] There are usually restrictions on which ciphertexts can be decrypted, since the game could be trivialized if the adversary can ask for a decryption of the encryption it received.

the level of security we want to obtain, we can decide how to build our algorithm to obtain it.

The third task of the cryptographer, but not the least, is to *prove* that an algorithm satisfies the notion that was defined.[4] Crucially, this is the foundation of *provable security*, and with these formal proofs, we can rigorously assess some insights about the security of the protocol. To prove the security of an algorithm, it does not suffice to show that a given attack does not work against our algorithm since it is impossible to enumerate and consider all the possible attacks. We ideally desire a more global approach, where we directly prove that any adversary, whatever it is doing, cannot break our algorithm (or, in other words, win the game that we defined).

A common approach is the proof by *computational reductions* and relies on the presumed hardness of a well-studied problem. For example, in some mathematical cyclic groups, it is believed to be hard to compute the discrete logarithm of a group element in a given base. That is, there is no known efficient algorithm to compute it despite many efforts. Hence, the proof strategy is to show that if there exists an efficient adversary that can attack our algorithm, then we can efficiently transform this adversary into an algorithm that can solve this well-studied problem believed to be hard. This is the approach that we will typically use in this thesis.

Of course, provable security does not capture the whole picture. Among other things, we need this computational assumption to hold, the security definitions to faithfully model the real-world behavior of an adversary, the implemented computer program to correctly reflect the formal algorithm and the security proofs to be free of mathematical fallacies. Nevertheless, provable security provides at least a better understanding of the security of the cryptographic protocol.

1.2. Secure voting protocols

Naturally, it is preferable to follow the same approach when building secure voting protocols, otherwise unexpected attacks can compromise these protocols. An excellent example is the Swiss Internet voting system [Swi23], whose designers are publishing publicly their protocol and security proofs of its security, and are running public audit and intrusion tests to evaluate and improve its security.

[4] Similarly, this third task is usually intertwined with the design of the protocol and the modeling of the security defition

A blueprint for single-pass voting protocols There is a common blueprint based on simple cryptographic primitives that can be used to build a secure voting protocol, and even today, many voting protocols are still heavily influenced by these ideas proposed as far as 40 years ago [CF85; CGS97]. We can cite, for example and without the intention to be exhaustive, the following impactful protocols: Helios 2.0 [Adi+09], STAR-vote [Ben+13], Belenios [CGG19] and ElectionGuard [Ele24].

This blueprint proceeds as follows: First, a set of trustees generate the election's key. This key can be seen as the key of an encryption scheme, which is a combination of a set of secret keys, generated distributively and independently by the trustees, and a public key, that can be publicly derived from this key generation process. Then, the voters can use the public key to compute their ballots consisting of encryptions of their votes. These ballots are then sent to the election officials, who publish them on a public digital bulletin board, possibly after some processing. Finally, the trustees compute the result of the election together. There are two main ways to compute this result in a privacy-friendly manner. The first way is to use an encryption scheme that makes it possible to compute an encryption of the tally of the election from the ballots on the bulletin board. The trustees can then compute the decryption of this aggregated result, which does not leak the content of individual ballots. This approach is called *homomorphic* tallying. The other approach uses so-called *mixing networks*. This tool verifiably shuffles the ballots, such that the link between voters and their ballots are broken. Finally, the trustees decrypt each individual ballot and publish the result on the bulletin board.

One last ingredient is needed to complete our description of this blueprint protocol. We want the voters to be convinced that the result is computed correctly, even if the trustees are malicious. Alone, those official entities should not be able to alter the election's winner. Hence, we use an additional cryptographic primitive, *zero-knowledge proofs*. These proofs are used to convince the voters that the trustees correctly executed the tallying process without revealing anything about the secret key of the election or anything that could be used to break the voter's privacy. There is another use of these zero-knowledge proofs. When casting a ballot, the voter can also prove that their ballot contains a valid vote. Indeed, since the ballot contains an encryption of the vote, this is a convenient way to ensure that this encryption does not contain adversarial material.

Importantly, no interaction between the voters and the bulletin board is needed to cast a ballot. Such voting protocols are said to be *single-pass*, and this absence of interaction offers a simple and efficient way to vote,

which is crucial in practice.

Security model But what are we expecting from a secure voting protocol? Earlier, we talked about the vote secrecy and the trust that the voter's ballot correctly encode their voting preference and that the result of the election is correctly computed.

Regarding the first point, one way to formulate the vote secrecy is that an adversary should not be able to learn anything about the voting patterns of the voter, except what can be learned from the result of the election. For single-pass voting system, the ballot privacy (or BPRIV) notion [Ber+15] is an excellent definition that captures what we desire. Even more, if the encryption scheme satisfies a confidentiality notion of NM-CPA security, then we can generically build a voting protocol following the blueprint discussed previously that provably has ballot privacy [BPW12].

Related to the vote secrecy, *receipt-freeness* is a stronger notion that one could expect from a secure voting protocol for high-stakes elections. Indeed, no one should be able to get any information about a voter's vote, even if this voter is willing to break their own privacy. While this looks counter-intuitive at first (why would a voter do that?), this makes a lot of sense in the context of vote-buying. Indeed, if someone can prove how they voted, then they could sell their vote to a vote-buyer, which is, at best, anti-democratic. For single-pass voting protocols, there is a convenient definition inspired by BPRIV [Cha+16].

For the second point, we usually require different flavors of verifiability: *individual* and *universal verifiability*. Individual verifiability guarantees that a voter can be convinced that their ballot is correctly encoding their vote and that this ballot is correctly recorded on the bulletin board. On the other hand, universal verifiability guarantees that anyone can be convinced that the result of the election is correctly computed from the bulletin board. There are good computational definitions encompassing these two notions [Cor+14] if we consider that only the bulletin board and the trustees can behave maliciously. Moreover, we can attain these security notions thanks to the public bulletin board (voters can simply check that their ballots are on the board) and the zero-knowledge proof that the election result is correctly computed.

However, providing protection against a malicious voting device and guaranteeing that this voting device did not encrypt another vote during the crafting of the ballot is notoriously more difficult, and there are fewer computational definitions formalizing this kind of verifiability, also known as *cast-as-intended* verifiability. For the sake of completeness, there is

a last kind of verifiability, eligibility verifiability, which is important as well. This last notion ensures that only eligible voters can cast a vote.

Receipt-freeness Unfortunately, most protocols following the blueprint solution are not receipt-free. This is because the voter can provide their vote and all the randomness used in the encryption algorithm to an adversary, who can then re-encrypt the vote using the same randomness and check if the resulting ciphertext appears on the bulletin board.

There are several approaches to protect a voter from a receipt-freeness adversary, which include revoting [SHD10; LQT20], fake credentials [JCJ05; CCM08], or ballot rerandomization [BT94; HS00; Bla+11; Cha+16]. In a protocol with revoting, voters can cast several ballots, and for example, only the last one will be counted. If the receipt-freeness adversary cannot detect new ballots on the bulletin board, then they can never be sure whether a voter has revoted or not. In protocols based on fake credentials, the voter receives from a prior registration a valid voting credential and fake voting credentials, which are indistinguishable. To compute a valid ballot, the voter needs to use the valid voting credential, while to deceive the receipt-freeness adversary, the voter can use the fake voting credential. At the end of the election, the trustees will only count the ballots that use the real credential.

Contrary to the previous approaches, the protocols based on ballot rerandomization may not need interaction to cast a ballot. In this approach, the voter sends a ballot containing an encryption of their vote as usual, and the casting server rerandomizes it before publishing it on the public bulletin board. As the randomness of the ballot has been refreshed, the voter does not have a receipt of their vote anymore.

Benaloh and Tuinstra were the first to define the receipt-freeness property [BT94] and to propose a protocol in the same vein as the ballot rerandomization approach. However instead of refreshing the randomness of a voter's ballot, the idea of this protocol was to let the voting server compute a ballot for every option, and then give these ballots to the voter, with a non-transferable cut-and-choose proof of the correspondance between the ballots and the choices. Since the voter did not compute the ballot, they cannot return the randomness used to compute the ballot as a receipt, similarly to the ballot rerandomization approach. This idea was then improved by Hirt and Sako using designated verifier proofs [HS00], even though their protocol required many interactions between the voter and the casting server and was thus impractical, and then refined by Hirt in [Hir01] using a constant number of interactions. More than 10 years later, Blazy et al. introduced the concept of signatures

on randomizable ciphertexts [Bla+11], which opened the way to non-interactive receipt-free voting. Non-interactive receipt-free voting was then formalized, and a complete construction was proposed by Chaidos et al. in BeleniosRF [Cha+16]. In this thesis, we will systematize and extend this approach.

Postal voting Despite the many efforts of the secure voting community, building a secure internet voting protocol remains challenging. For the voters who cannot vote in person, there is however an alternative to Internet voting, which is the more traditional postal voting. In fact, postal voting is widely adopted in several countries, such as Germany, Switzerland, or in some states of the United States.

One of the main advantages of postal voting is that the cast-as-intended property is trivially satisfied since the vote is directly hand-marked on paper. Moreover, a postal voting election is typically simpler and easier to implement as it does not need a complex infrastructure and complex code involving cryptographic algorithms run on many different devices.

Still, postal voting typically offers very few guarantees to voters that their ballots have been received untampered at the election office and that their vote secrecy has not been compromised. In most traditional postal voting processes, voters send an envelope containing their signature and an additional smaller envelope inside with their hand-marked ballot to separate them from the signature. In this process, the signature is crucial for eligibility reasons but makes it trivial for a malicious agent to open the envelopes and link the vote inside it to the corresponding asserting voter. Regarding verifiability, voters sometimes receive a confirmation that a ballot with their signature has been delivered, but generally do not receive further guarantees regarding the integrity of their ballot.

In the last decades, there have been several proposals of postal voting protocols aiming at better security guarantees. In 2013, Benaloh, Ryan and Teague suggested a verifiable protocol using randomized partial checking [BRT13]. In this protocol, voters fill out their ballot on a computer and obtain a printed ballot containing a human-readable description of their choices as well as a signed encryption of these choices. The human-readable description and the encryption can then be published on a public bulletin board so that anyone can check that their ballot is there and compute the tally. However, a colluding malicious voting device and postal channel can deceive a voter and tamper their vote without being detected and the signature on the ballot still enables the same identification attacks. An improvement of this protocol was proposed recently by McMurtry et al. [McM+21]. This protocol offers better

receipt-freeness and detects vote tampering with a greater probability.

In 2021, Benaloh proposed another direction to build more secure postal voting protocols with the STROBE system [Ben21]. In this approach, voters receive ballots that can be hand-marked directly and an additional code for each choice corresponding to a one-hot vector of the encryption of this choice. During the tally, the selected codes are published on a bulletin board and the result of the election can be tied to the corresponding vectors of encryptions of the selected choices which are homomorphically aggregated to ensure verifiability. Crucially, voters in this protocol receive two ballots: one is cast normally and the other one is marked to be audited. In this second case, the election authorities provide an opening of the encryptions corresponding to the codes of this audited ballot to make sure that they are consistent with the choices on the ballot so that malicious ballots are detected with probability one over two. More recently, Crimmins et al. proposed the RemoteVote and SafeVote protocols [CRH22] which are adapting this methodology to other kinds of auditing processes which avoid the need to send two ballots to the voter.

However in all of these novel approaches, there is no systemic security analysis of the protocols. Especially, there is no formal definition of verifiability or privacy and thus no proof that these protocols attain the claimed properties, and we will fill this gap in this thesis.

1.3. Outline of the thesis

We will describe in Chapter 2 the definitions related to voting protocols and the cryptographic building blocks that we will use to build the contributions of this thesis.

Then comes the heart of the thesis, which is divided into two parts. These two parts reflect two different approaches to building better voting systems, both relying on modern cryptography. In the first part, we focus on new cryptographic primitives, protocols, and models for receipt-freeness. Then, in the second part, we look at already existing protocols and try to improve them with classic and well-proven cryptographic tools.

In Chapter 3, we propose a new cryptographic primitive, the traceable receipt-free encryption (TREnc) scheme and a companion security notion, traceable CCA (TCCA) security. As its name suggests, this primitive is particularly useful for generically building receipt-free voting protocols with the ballot rerandomization approach. Like NM-CPA security, which is enough to imply ballot privacy in a protocol following the standard

blueprint, TCCA security is enough to imply receipt-freeness. Then in Chapter 4, we build several efficient TREncs that are provably secure in different assumptions and use cases. The security of these constructions only relies on a standard computational assumption on asymmetric bilinear groups, the SXDH assumption. These results were presented at the Asiacrypt 2022 conference [DPP22b] and at the SCN 2024 conference [DPP24a].

Then, in Chapter 5, we look at the limits of single-pass voting protocols. Besides receipt-freeness, we try in this chapter to obtain the cast-as-intended verifiability as well, but we show that this is impossible for a single-pass voting system. However, if we allow an interactive voting process, then this becomes possible, and we build three protocols showing this result under different settings. Since there was no satisfying computational definition for receipt-freeness with an interactive casting process, we propose a new definition of independent interest. The materials of this chapter were presented at the IEEE Symposium on Security and Privacy 2024 [Dev+24].

We have a more practical point of view in the second part of the thesis. While the technologies used in the first part of the thesis have potential, they are relatively complex and need more time to mature and be used in real-life protocols. In contrast, other protocols are already used in practice, and it is thus worth thinking about how we can improve them without completely changing the techniques.

In Chapter 6, we look at the blueprint for single-pass voting described above and try to improve it. The bottleneck of this protocol is usually the computation of the ballot, which requires the computation of an encryption and zero-knowledge proof that the ballot is correctly computed. We will explore how we can optimize the computation of this ballot using standard tools. These findings were presented at the ESORICS 2022 conference [DPP22a].

Finally, Chapter 7 considers the design of more secure postal voting. We will propose a new model in a variant of the universally composable framework and propose a provably secure protocol in this model. This protocol has several advantages, including an easy and intuitive ballot-casting process based on hand-marked ballots, an optional mechanism for the voter to verify that their ballot has been correctly tallied based on return codes, and very simple cryptographic primitives overall. This work is not yet published but already accessible on the IACR eprint [DPP24b].

Related Publications

[Dev+24] Henri Devillez, Olivier Pereira, Thomas Peters, and Quentin Yang. "Can we cast a ballot as intended and be receipt free?" In: *IEEE SP 2024*. IEEE, 2024, pp. 3440–3457.

[DPP22a] Henri Devillez, Olivier Pereira, and Thomas Peters. "How to Verifiably Encrypt Many Bits for an Election?" In: *ESORICS 2022 Proceedings, Part II*. Vol. 13555. LNCS. Springer, 2022, pp. 653–671.

[DPP22b] Henri Devillez, Olivier Pereira, and Thomas Peters. "Traceable Receipt-Free Encryption". In: *Advances in Cryptology - ASIACRYPT 2022 Proceedings, Part III*. Vol. 13793. LNCS. Springer, 2022, pp. 273–303.

[DPP24a] Henri Devillez, Olivier Pereira, and Thomas Peters. "Practical Traceable Receipt-Free Encryption". In: *SCN 2024 Proceedings, Part I*. Vol. 14973. LNCS. Springer, 2024, pp. 367–387.

[DPP24b] Henri Devillez, Olivier Pereira, and Thomas Peters. "Verifiable and Private Vote-by-Mail". In: *IACR Cryptol. ePrint Arch.* (2024).

2. Preliminaries

2.1. Foundations

2.1.1. Computational Setting

Throughout this thesis, we are going to manipulate *algorithms* that can be seen as a program that follows a sequences of steps to transform an input x into an output y. These steps are not necessarily deterministic, and it is actually crucial in our cryptographic setting that some of these steps have a random outcome. In other words, we manipulate probabilistic algorithms, that can be modeled as Turing machines.

But also, we are interested in *efficient* algorithms. A traditional way to define efficency is to require that the number of steps of the algorithm (representing the time) is polynomially bounded by the size of the input. We abridge probabilistic polynomial time algorithms as PPT algorithms.

More formaly, we write $y = P(x; r)$ an algorithm P that takes an input x and random coins r to produce an output y. We call the set of possible output obtained through an execution of P as the *range* of P. Moreover, we can *rewind* an algorithm: at any step, we can reverse the steps done by the algorithm and restart an execution from a given step with new random coins.

We also consider interactive algorithms (in that case, we call them *protocols*) between two or more machines which can send messages to each other. In the case of two parties, we define an interactive protocol $\mathcal{T}$ as a pair of two algorithms $(\mathcal{T}_1, \mathcal{T}_2)$ and write an execution of this protocol as $y_1, y_2 \leftarrow [\mathcal{T}_1(x_1) \leftrightarrow \mathcal{T}_2(x_2)]$ where x_i and y_i are respectively the input and output of $\mathcal{T}_i$.

Each of these algorithms $\mathcal{T}_i$ takes as input a state τ_i (which is initially set to the input x_i of the algorithm) and outputs a message m_i and an update of the state. The message is then concatenated to the other machine's states. Also, the machines proceeds sequentially. That is, the first computes its message and outputs it to the second machine, then the second machine computes its own computation with its stated augmented of the first message. The two parties repeat this procedure until one message is equal to $\perp$. In that case, no more messages are output and the parties return their output.

2.1.2. Cryptographic group and assumptions

Our algorithms will often work with a cyclic group, noted $(\mathbb{G}, g, q)$ where g is a generator and q the order of the group. That is, any element of $\mathbb{G}$ can be written as g^i for $0 \leq i < q$, and q is typically a prime number.

We will also work with bilinear groups noted $(\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, q, e, g, \hat{g})$. Each of $\mathbb{G}$, $\hat{\mathbb{G}}$ and $\mathbb{G}_T$ is also a cyclic group of order q, with respective generators $g, \hat{g}$ and g_T. Also, e is a bilinear map: for any $g, \hat{g} \in \mathbb{G} \times \hat{\mathbb{G}}$ and $x, y \in \mathbb{Z}_q^2$, $e(g^x, \hat{g}^y) = e(g, \hat{g})^{xy}$ and $e(g, \hat{g}) = g_T$.

In both cases, we assume that there is an algorithm $\mathsf{Setup}(\lambda)$ that generates the public parameters containing a description of the cryptographic group. λ is the security parameter and controls the security level of our algorithms, or in other words the amount of computation needed to break them. Typically, the order of the group is set to be at least equal to $\mathcal{O}(2^\lambda)$. These public parameters can then directly be used by the cryptographic algorithms described in Section 2.2 to ensure they operate on the same group and can compose with each others.

Before describing what we expect from these groups, we define a crucial notion. We say that a function μ is *negligible* if it decreases faster than the inverse of any polynomial. In other words, for any polynomial p, there exists $N > 0$ such that for any $x \geq N$, $\mu(x) \leq \frac{1}{|p(x)|}$. Equipped with this definition, we can now define the following *hardness assumption.*

The security of most of our protocols will rely on a computational assumption, called the *Decisional Diffie-Hellman* (DDH) assumption. This assumption states that given uniformly random elements g^a and g^b, it is difficult to distinguish a random element g^c from g^{ab}.

Given this security parameter, we now define the following game. We pick a random bit b and generate random $(x, y, z) \in \mathbb{Z}_q$. Then, we either give $(1^\lambda, g^x, g^y, g^{xy})$ as input to an algorithm $\mathcal{A}$ if $b = 0$ or $(1^\lambda, g^x, g^y, g^z)$ if $b = 1$. We say that $\mathcal{A}$ solves the DDH instance (or that $\mathcal{A}$ wins the game) if it outputs the bit that we picked. Now we can say that the DDH assumption holds if any PPT algorithm $\mathcal{A}$ wins the game with a probability bounded by $\frac{1}{2} + \mu(\lambda)$, where μ is a negligible function. In a bilinear group, we call the assumption that DDH holds in both $\mathbb{G}$ and $\hat{\mathbb{G}}$ the SXDH assumption .

2.2. Generic building blocks

For all of our building blocks, we assume that we run a $\mathsf{Setup}(1^\lambda)$ algorithm which outputs some public common parameters pp used in any key generation algorithm of the following building blocks.

2.2.1. Randomizable Encryption Scheme

To build receipt-free voting protocols, a common strategy is to use a trusted party which refresh the randomness of the voters' ballot. It is thus desirable to have cryptographic algorithms that allows to rerandomize their output. We define a randomizable encryption scheme as follows:

Definition 1 (Randomizable encryption scheme)**.** *A randomizable encryption scheme is a tuple of algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand})$*:*

$\mathsf{Gen}(1^\lambda)$*: Outputs a pair of keys* $(\mathsf{pk}, \mathsf{sk})$*.* pk *is called the public key and* sk *the secret key*

$\mathsf{Enc}(\mathsf{pk}, m)$*: Given the public key* pk *and a* m*, outputs a ciphertext* c *encrypting* m*.*

$\mathsf{Dec}(\mathsf{sk}, c)$*: Given the secret key* sk *and ciphertext* c*, outputs a decrypted message* m*.*

$\mathsf{Rand}(\mathsf{pk}, c)$*: Given the public key* pk *and a ciphertext* c*, outputs a randomized ciphertext* c'*.*

Naturally, the randomization algorithm should preserve the message inside the ciphertext, and for any $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$ and any ciphertext c in the range of $\mathsf{Enc}(\mathsf{pk}, \cdot)$, we have that $\mathsf{Dec}(\mathsf{sk}, c) = \mathsf{Dec}(\mathsf{sk}, \mathsf{Rand}(\mathsf{pk}, c))$.

We also require that randomized ciphertexts have the same distribution as fresh encryptions. That is, for any ciphertext c such that $\mathsf{Dec}(\mathsf{sk}, c) = m$, $\mathsf{Enc}(\mathsf{pk}, m)$ and $\mathsf{Rand}(\mathsf{pk}, c)$ are computationally indistinguishable. This should even hold if the ciphertext c is not in the range of Enc, and we call this property *strong randomizability*. This property can be relaxed to a computational property where a PPT adversary is asked to distinguish between a fresh encryption and a rerandomization, but in this thesis we are going to only use encryption schemes with a perfect strong randomizability property.

2.2.2. Threshold encryption scheme

In a classical encryption scheme, the secret key used to decrypt the ciphertexts is owned by a unique entity. In some contexts however, it is desirable to distribute this key into several entities, such that a single party cannot decrypt a ciphertext alone. Together though, they can compute the decryption of a given ciphertext. In a voting setting, this can be motivated as the encryption scheme is used to encrypt the vote of the voters, and we thus do not want a party to be able to retrieve these private information on their own.

It is frequent to require only a fraction of these distributed keys to decrypt a ciphertext, to have better resilience. If the key is distribute in n parts and if t parts are enough to decrypt a ciphertext, we call such encryption scheme a t-out-of-n threshold encryption scheme. More formally,

Definition 2. *A threshold encryption scheme is a tuple of algorithms* $(\mathsf{TGen}, \mathsf{TEnc}, \mathsf{TPartDec}, \mathsf{TCombine})$*:*

$\mathsf{TGen}(1^\lambda, n, t)$*: Outputs a public key* pk *and* n *secret keys* $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$*.*

$\mathsf{TGen}(\mathsf{pk}, m)$*: Given the public key* pk *and a* m*, outputs a ciphertext* c *encrypting* m*.*

$\mathsf{TPartDec}(\mathsf{pk}, \mathsf{sk}_i, c)$*: Given a partial secret key* sk_i *and a ciphertext* c*, outputs a decryption factor* d_i*.*

$\mathsf{TCombine}(\mathsf{pk}, c, (d_{i_1}, \ldots, d_{i_t}))$*: Given the public key, a ciphertext and* t *decryption factors of this ciphertext, outputs a decrypted message.*

Correctness : We requires that a honest run of these algorithms for any message m and a resulting ciphertext c leads $\mathsf{TCombine}$ to output back m with overwhelming probability.

Following Damgard and Jurik [DJ01] and Fouque, Poupard and Stern [FPS00], a threshold encryption scheme is semantically secure if the underlying public-key encryption is indistinguishable against chosen-plaintext attacks (CPA-secure, for short), and if the partial decryption algorithm is simulatable (a.k.a. decryption simulatability).

IND-CPA security : For all integers $0 < t \leq n$, for any PPT adversary $\mathcal{A}$, and for any index sets $\mathcal{C} \subset [n] = \{1, \ldots, n\}$ with $|\mathcal{C}| < t$ and $\mathcal{H} = [n] \setminus \mathcal{C}$, the advantage $|\Pr[\mathsf{Exp}^{(\mathrm{t,n})\text{-}\mathrm{cpa}}_{\mathcal{A},\mathcal{H},\mathcal{C}}(\lambda)] - 1/2|$ in the experiment in Figure 2.1 is negligible.

Decryption simulatability : there must exist a PPT algorithm TSim such that, for any PPT adversary $\mathcal{A}$, for any message m, and any $t, n, \mathcal{C}, \mathcal{H}$ as above, the advantage $|\Pr[\mathsf{Exp}^{(\mathrm{t,n})\text{-}\mathrm{sim}}_{\mathcal{A},\mathcal{H},\mathcal{C}}(\lambda)] - 1/2|$ in the experiment in Figure 2.1 is negligible.

2.2.3. Signatures on Randomizable Ciphertexts

When building a protocol, it is common to sign a ciphertext, such that it should be hard to forge a signature for a fresh ciphertext without

$\mathsf{Exp}^{\text{(t,n)-cpa}}_{\mathcal{A},\mathcal{H},\mathcal{C}}(\lambda)$

$\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n \leftarrow\$\ \mathsf{TGen}(1^\lambda, n, t)$
$m_0, m_1, \mathsf{st} \leftarrow\$\ \mathcal{A}(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in\mathcal{C}})$
$b \leftarrow\$\ \{0,1\}$
$b' \leftarrow\$\ \mathcal{A}(\mathsf{st}, \mathsf{TEnc}(\mathsf{pk}, m_b))$
return $b' == b$

$\mathsf{Exp}^{\text{(t,n)-sim}}_{\mathcal{A},\mathcal{H},\mathcal{C},m}(\lambda)$

$\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n \leftarrow\$\ \mathsf{TGen}(1^\lambda, n, t)$
$c \leftarrow\$\ \mathsf{TEnc}(\mathsf{pk}, m)$
$b \leftarrow\$\ \{0,1\}$
if $b = 0$:
  $\mathsf{DF} \leftarrow\$\ \{\mathsf{TPartDec}(\mathsf{sk}_i, c)\}_{i\in\mathcal{H}}$
if $b = 1$:
  $\mathsf{DF} \leftarrow\$\ \mathsf{TSim}(\mathsf{pk}, \mathcal{H}, c, m)$
$b' \leftarrow\$\ \mathcal{A}(\{\mathsf{sk}_i\}_{i\in\mathcal{C}}, \mathsf{DF})$
return $b' == b$

Figure 2.1.: IND-CPA experiment (left) and Decryption simulatability experiment (right). $\mathcal{C}$ and $\mathcal{H}$ denote the sets of corrupted and honest parties.

knowing the signing key of the signature scheme. In a classical signature scheme, this unforgeability property also holds for a rerandomization of a ciphertext for which we know the signature, which is not always desirable. Indeed, we might want to adapt a signature to the rerandomization of a ciphertext, as long as the message inside it remains unaltered.

Signatures on Randomizable Ciphertexts (SRC), introduced by Blazy et al. [Bla+11], consists in a joint randomizable public-key encryption scheme and adaptable signature scheme on the ciphertexts and public key of the encryption scheme. These features can be used as an ingredient in order to obtain receipt-freeness, as they allow a third party to rerandomize a ciphertext and the associated signature, making it infeasible for the party who produced the original ciphertext to prove the content of the randomized ciphertext, while remaining able to trace the ciphertext thanks to the valid signature.

Definition 3 (Signatures on Randomizable ciphertexts)**.** *A scheme of* signatures on randomizable ciphertexts *(SRC in short) consists in the following algorithms :* $(\mathsf{Setup}, \mathsf{SGen}, \mathsf{EGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand}, \mathsf{Sign}, \mathsf{SVer}, \mathsf{SAdapt})$*:*

- $(\mathsf{EGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand})$ *represents an encryption scheme, as defined in Definition 1. We note* $(\mathsf{ek}, \mathsf{dk})$ *the pair of public/secret keys.*
- $(\mathsf{SGen}, \mathsf{Sign}, \mathsf{SVer})$ *represents a signature scheme.* $\mathsf{SGen}(1^\lambda)$ *outputs the pair* $(\mathsf{sk}, \mathsf{vk})$ *containing the signing and verification keys of the signature scheme.* $\mathsf{Sign}(\mathsf{sk}, \mathsf{ek}, c)$ *and* $\mathsf{SVer}(\mathsf{vk}, \mathsf{ek}, c, \sigma)$ *respectively outputs a signature of a ciphertext and its encryption key and verify the validity of a signature.*

$\mathsf{Exp}^{\mathrm{src-uf}}_{\mathcal{A}}(\lambda)$

$(\mathsf{ek}, \mathsf{dk}) \leftarrow\$\, \mathsf{EGen}(1^\lambda)$
$(\mathsf{vk}, \mathsf{sk}) \leftarrow\$\, \mathsf{SGen}(1^\lambda)$
$(c, \sigma) \leftarrow\$\, \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\mathsf{ek},\cdot)}(\mathsf{vk}, (\mathsf{ek}, \mathsf{dk}))$
$m \leftarrow\$\, \mathsf{Dec}(\mathsf{dk}, \mathsf{vk}, c)$
if $m \neq \perp$ and $m \notin Q$ and $\mathsf{Ver}(\mathsf{vk}, \mathsf{ek}, c, \sigma) = 1$
then return 1

Figure 2.2.: Unforgeability experiment. $\mathcal{A}$ has access to a signing oracle $\mathsf{Sign}(\mathsf{sk}, \mathsf{ek}, \cdot)$ that takes as input a ciphertext c. It outputs a signature on c with sk if this is a valid ciphertext and add $\mathsf{Dec}(\mathsf{dk}, c)$ to the list of used messages Q.

$\mathsf{SAdapt}(\mathsf{vk}, \mathsf{ek}, c, \sigma, r')$ *outputs a valid adapted signature on a randomized ciphertext* $c' = \mathsf{Rand}(\mathsf{ek}, c; r')$.

Besides the standard correctness properties of randomizable encryption schemes and signature schemes, we also require the scheme to follow the signature-adaptation *property and plaintext unforgeable:*

Signature adaptation *For every* ek, dk *in the range of* EGen*, every* vk, sk *in the range of* SGen*, and every message* m*,* $c = \mathsf{Enc}(\mathsf{ek}, \mathsf{vk}, m)$ *and* $\sigma = \mathsf{Sign}(\mathsf{sk}, \mathsf{ek}, c)$*,*

$$\{c' = \mathsf{Enc}(\mathsf{ek}, \mathsf{vk}, m), \sigma' = \mathsf{Sign}(\mathsf{sk}, \mathsf{ek}, c')\}$$

and

$$\{c' = \mathsf{Rand}(\mathsf{ek}, \mathsf{vk}, c; r'), \sigma' = \mathsf{SAdapt}(\mathsf{vk}, \mathsf{ek}, c, \sigma, r')\}$$

are statistically indistinguishable.

Unforgeability *An SRC is* unforgeable *if for every* PPT *adversary* $\mathcal{A}$*, the experiment* $\mathsf{Exp}^{\mathrm{uf}}_{\mathcal{A}}(\lambda)$ *defined in Figure 2.2 returns* 1 *with a probability negligible in* λ*.*

2.2.4. Linearly Homomorphic Structure-Preserving Signatures

Alternatively, we can use Linearly Homomorphic Structure-Preserving (LHSP) Signatures to build more flexible signatures on randomizable ciphertexts, and thus this primitive will be at the heart of most our constructions. The structure preserving property makes it possible to sign messages that are group elements (and not just bits as in schemes based on the waters signature), while the additional linearly homomorphic

feature, introduced by Libert et al. [Lib+13], will be used to make the signatures randomizable while guaranteeing the non-malleability of the plaintext.

More precisely, this primitives allows to sign vectors of n elements of $\mathbb{G}$, the base group of a bilinear group. Moreover, given several pairs of vectors and signatures, the linearly homomorphic property of the signature scheme allows to derive without the knowledge of the signing key a signature of a linear combination of those vectors, while it is difficult to forge a signature for a message outside the span of those vectors.

Definition 4. *A linearly homomorphic structure-preserving signature scheme is a tuple of algorithms* $(\mathsf{SGen}, \mathsf{Sign}, \mathsf{SDerive}, \mathsf{SVer})$*:*

$\mathsf{SGen}(1^\lambda, n)$*: outputs the pair* $(\mathsf{sk}, \mathsf{vk})$ *containing the signing and verification keys of the signature scheme.*

$\mathsf{Sign}(\mathsf{sk}, \boldsymbol{m})$*: Given a vector* $\boldsymbol{m}$ *of* n *elements of* $\mathbb{G}$*, outputs a signature* σ*.*

$\mathsf{SDerive}(\mathsf{vk}, \{\alpha_i, \boldsymbol{m}_i\}_{i=1}^k)$*: Given a set of pair of coefficient in* $\mathbb{Z}_q$ *and vector in* $\mathbb{G}^n$*, outputs a valid signature of the message* $\sum_{i=1}^k \alpha_i \boldsymbol{m}_i$*.*

$\mathsf{SVer}(\mathsf{vk}, \boldsymbol{m}, \sigma)$*: Given a message and a signature, return* 1 *if the signature is valid or* 0 *otherwise.*

2.2.5. Non-Interactive Zero-Knowledge proof system

Throughout this thesis, we are interested in hard relations R containing pairs (x, w) where x is public and called a *statement* and w is private to some party and called a *witness*. While it is easy to determine if $(x, w) \in R$, it is supposed to be difficult to decide if $\exists w$ such that $(x, w) \in R$ knowing x alone. In other words, the set of statements L satisfying this relation is an NP language.

It is common to be willing to prove to another party the knowledge of some witness attesting that a statement is in R without revealing anything about the witness. This forges the well-established concept of Zero-Knowledge proof [GMR89], which can be either instantiated by interactive algorithms, or by non-interactive algorithms [BFM88], which we prefer for practical reasons.

In this thesis, the main use of this tool is to prove that some value y is well computed as an output of $\mathcal{T}(x)$ for some PPT algorithm $\mathcal{T}$. For example, when a voter computes a ballot $\mathsf{b} \leftarrow \mathsf{Vote}(\mathsf{v}; r)$ for a vote v using a randomness r, it is useful to have a proof that the ballot is well constructed and encodes a valid vote.

We define a non-interactive zero-knowledge proof system as follows:

Definition 5 (NIZK proof system)**.** *A non-interactive zero-knowledge (NIZK) proof system for an NP relation R is a tuple of algorithms* $(\mathsf{PGen}, \mathsf{PProve}, \mathsf{PVerify})$*:*

$\mathsf{PGen}(1^\lambda)$*: outputs the common reference string* crs *of the proof system, which helps to prove the statement.*

$\mathsf{PProve}(\mathsf{crs}, x, w)$*: Given a statement x and a witness w such that $(x, w) \in R$, outputs a proof π.*

$\mathsf{PVerify}(\mathsf{crs}, x, \pi)$*: Given a statement x and a proof π, outputs 1 if π is a convincing proof that x is in the language induced by R.*

We require that a proof system satisfies the following correctness requirements:

Completeness For any adversary PPT $\mathcal{A}$, the following probability

$$Pr\left(\begin{matrix}\mathsf{crs} \leftarrow\!\$\, \mathsf{Gen}(1^\lambda); (x, w) \leftarrow \mathcal{A}(\mathsf{crs}); \\ \pi \leftarrow\!\$\, \mathsf{PProve}(\mathsf{crs}, x, w) : (x, w) \in R \wedge \mathsf{PVerify}(\mathsf{crs}, x, \pi) = 1\end{matrix}\right)$$

is negligible in λ. Intuitively, for any $(x, w) \in R$, an honest execution of the proof algorithm convinces the verifying algorithm.

Soundess For any adversary PPT $\mathcal{A}$, the following probability

$$Pr\left(\begin{matrix}\mathsf{crs} \leftarrow\!\$\, \mathsf{Gen}(1^\lambda); (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) : \\ \forall w\, (x, w) \notin R \wedge \mathsf{PVerify}(\mathsf{crs}, x, \pi) = 1\end{matrix}\right)$$

is negligible in λ. Intuitively, a prover should not be able to convince the verifying algorithm for a false statement not in L.

Zero-Knowledge There exists two PPT algorithms SimGen and $\mathsf{SimProve}$ such that for any adversary $\mathcal{A}$:

$$\left|\begin{matrix} Pr\left(\mathsf{crs} \leftarrow\!\$\, \mathsf{Gen}(1^\lambda); \mathcal{A}^{\mathsf{P}(\cdot,\cdot)}(\mathsf{crs}) = 1\right) \\ -Pr\left(\mathsf{crs}, \mathsf{tk} \leftarrow\!\$\, \mathsf{SimGen}(1^\lambda); \mathcal{A}^{\mathsf{P}^{\mathsf{sim}}(\cdot,\cdot)}(\mathsf{crs}) = 1\right)\end{matrix}\right|$$

is negligible in λ, where $\mathsf{P}(x, w)$ and $\mathsf{P}^{\mathsf{sim}}(x, w)$ are oracles respectively returning $\mathsf{PProve}(\mathsf{crs}, x, w)$ and $\mathsf{SimProve}(\mathsf{crs}, \mathsf{tk}, x)$ in the second line. Intuitively, an adversary cannot distinguish between real proofs and simulated proofs that are built without the knowledge of w and so the real proof does not reveal anything about the witness.

In our algorithms, we will typically rerandomize the output of an algorithm. If this value was accompanied with a proof, it is not valid anymore. Hence we need a way to adapt this proof to be compatible with the rerandomized input. A malleable NIZK proof system [Cha+12] is a proof system with an additional algorithm that allows to adapt a proof to another proof for another (controlled) statement.

Definition 6 (Malleable NIZK)**.** *A malleable (NIZK) proof system for a statement adapting algorithm f is a NIKZ proof system with an additional algorithm* PAdapt*:*

$\mathsf{PAdapt}(\mathsf{crs}, x, \pi, w')$*: Given a valid proof π for a statement x, outputs a valid proof π' for a statement $x' \leftarrow f(x, w')$.*

The correctness and soundness can be easily extended to the malleable case, especially the PAdapt algorithm should not be able to create proofs for adapted statement not in the language. We can also extend the zero-knowledge property as follows:

Zero-Knowledge Adaptation There exists two PPT algorithms SimGen and $\mathsf{SimAdapt}$ such that for any adversary $\mathcal{A}$:

$$\left| \begin{array}{l} Pr\left(\mathsf{crs} \leftarrow\!\$\, \mathsf{Gen}(1^\lambda); \mathcal{A}^{\mathsf{P}(\cdot,\cdot)}(\mathsf{crs}) = 1\right) \\ -Pr\left(\mathsf{crs}, \mathsf{tk} \leftarrow\!\$\, \mathsf{SimGen}(1^\lambda); \mathcal{A}^{\mathsf{P}^{\mathsf{sim}}(\cdot,\cdot)}(\mathsf{crs}) = 1\right) \end{array} \right|$$

is negligible in λ, where $\mathsf{P}(x, \pi, w)$ and $\mathsf{P}^{\mathsf{sim}}(x, \pi, w)$ are oracles that respectively return $\mathsf{PAdapt}(\mathsf{crs}, x, w)$ in the first line and $\mathsf{SimAdapt}(\mathsf{crs}, \mathsf{tk}, x, \pi)$ in the second line. We require that the SimGen algorithm is the same as in the zero-knowledge definition.

For a randomizable encryption scheme, we set $f(c, r')$ as the $\mathsf{Rand}(c; r')$ algorithm. We stress that this PAdapt algorithm does not need to know an initial witness of the original input value.

2.2.6. Divertible Proof

Another strategy to build a proof for an updated statement is to use an interactive algorithm [OO89]. In an interactive proof, the prover, who computed the original statement, and the intermediate, who adapted it, compute together the proof such that none of them learns anything about the witness of the other party. A transcript of this proof can then be used non-interactively to convince another party. Interestingly, we can use the Fiat-Shamir heuristic [FS86] as the interactivity between the two parties is enough to build the proof. Hence, a crs is not needed for this protocol.

Definition 7 (Divertible proof system)**.** *A divertible zero-knowledge (DZK) proof system for a relation R is a tuple of algorithms* ($\mathsf{DivProve_P}$, $\mathsf{DivProve_I}$, $\mathsf{DivVerify}$)*.*

$[\mathsf{DivProve_P}(x, w) \leftrightarrow \mathsf{DivProve_I}(x, w')]$ *is an interactive protocol between a prover and an intermediate. Given a statement x and a witness w such that $(x, w) \in R$, and an additional witness w' such that $(x' = f(x, w'), w') \in R$ as well, the protocol outputs a proof π.*

$\mathsf{PVerify}(x', \pi)$*: Given a statement x' and a proof π, outputs 1 if π is a convincing proof that x' is in the language induced by R.*

In addition of the typical completeness, soundness, and zero-knowledge properties, we also require that an adversarial prover cannot prove any relation between the witness w that it used and the final proof:

Prover obliviousness There exists a PPT algorithm $\mathsf{SimDivProve_I}$ such that for any adversary $\mathcal{A}$:

$$\left| \; Pr\left(\mathcal{A}^{\mathsf{P}(\cdot,\cdot)}(1^\lambda) = 1\right) - Pr\left(\mathcal{A}^{\mathsf{P^{sim}}(\cdot)}(1^\lambda) = 1\right) \; \right|$$

is negligible in λ, where the $\mathsf{P}(x, w)$ and $\mathsf{P^{sim}}(x, w)$ oracles are respectively instantiations of the algorithms $\mathsf{DivProve_I}(x, w')$ and $\mathsf{SimDivProve_I}(x)$.

This additional property, that we refer as prover obliviousness, is already used in prior works aiming to achieve receipt-freeness and is instantiated in the random oracle model by efficient Σ-protocols [Hir01] that uses three interactions.

2.2.7. Plaintext Equality Proof

There is a last flavor of proof that we will use, called plaintext equality proof [MPT20]. In one of our protocol, we will want to test if two ciphertexts contain the same message or not, which is a special case of the proof system defined above.

Definition 8 (Plaintext Equality Proof)**.** *A plaintext equality proof (PEP) of an encryption scheme is a pair of algorithms* ($\mathsf{PEPProve}$, $\mathsf{PEPVerify}$)*:*

$\mathsf{PEPProve}(\mathsf{sk}, c_0, c_1)$*: Given two ciphertexts c_0 and c_1, outputs a bit b indicating if $\mathsf{Dec}(\mathsf{sk}, c_0) = \mathsf{Dec}(\mathsf{sk}, c_1)$ and a proof π that this is the case.*

$\mathsf{PEPVerify}(\mathsf{pk}, c_0, c_1, b, \pi)$: *Given two ciphertexts, outputs* 1 *if* π *is a convincing proof that the content of these ciphertexts are the same (if* $b = 1$*) or different (if* $b = 0$*).*

In particular, if the encryption is threshold, we desire the holders of the secret keys to be able to compute this proof without learning anything more than if the messages are equal or not. Especially, they should not be able to learn any other relation between those messages. This can be instantiated easily by a distributed PEPProve protocol as in [MPT20].

2.2.8. Tag-based cryptography

Finally, our last building block is related to the tag-based encryption notion of MacKenzie et al. [MRY04]. Intuitively, a *tag-based encryption scheme* is an encryption scheme where both the encryption and decryption algorithms takes an additional input, called the tag.

This is an interesting notion, because this tag enables to partition the ciphertexts and to control their malleability up to the tag [Kil06]. Indeed, the chosen-ciphertext attack security (CCA) is not always the security notion that we want to obtain, especially in our voting context with rerandomization which needs some malleability. In this thesis, we require our tag-based encryption scheme to offer weak CCA security in the sense of MacKenzie et al. [MRY04].

Definition 9. *A* tag-based encryption *scheme* $(\mathsf{TBGen}, \mathsf{TBEnc}, \mathsf{TBDec})$ *is a standard encryption scheme in which the encryption and decryption algorithms take an extra parameter* t*. That is,* $\mathsf{TBEnc}(\mathsf{pk}, t, m)$ *encrypts* m *with public key* pk *and tag* t*, and* $\mathsf{TBDec}(\mathsf{sk}, t, c)$ *decrypts* c *with secret key* sk *and tag* t*.*

wCCA security *A tag-based encryption scheme is said to be weakly secure against chosen ciphertext attacks (*TBE-wCCA*) if for every adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, the experiment* $\mathsf{Exp}^{\text{tbe-wcca}}_{\mathcal{A}}(\lambda)$ *defined in Figure 2.3 returns* 1 *with a probability negligibly close in* λ *to* $\frac{1}{2}$*.*

This tag-based notion can also be applied to proof systems [KW15], where we augment the PProve (and SimProve) and the PVerify algorithms with a tag τ. Along with this extended definition, we can define the concept of unbounded simulation soundness [Lib+14], which will be at the heart of our constructions:

Simulation Soundess For any adversary PPT $\mathcal{A}$, the following probabil-

$\mathsf{Exp}^{\text{tbe-wcca}}_{\mathcal{A}}(\lambda)$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow\$\, \mathsf{TBGen}(1^\lambda)$
$(m_0, m_1, t^\star, \mathsf{st}) \leftarrow\$\, \mathcal{A}_1^{\mathsf{TBDec}(\mathsf{sk},\cdot,\cdot)}(\mathsf{pk})$
$b \leftarrow\$\, \{0,1\}$
$c^\star \leftarrow\$\, \mathsf{TBEnc}(\mathsf{pk}, t^\star, m_b)$
$b' \leftarrow\$\, \mathcal{A}_2^{\mathsf{TBDec}^\star(\mathsf{sk},\cdot,\cdot)}(c^\star, \mathsf{st})$
return $b = b'$

Figure 2.3.: Weakly CCA security experiment for tag-based encryption. In the experiment, $\mathcal{A}_2$ has access to a decryption oracle $\mathsf{TBDec}^\star$ which, on input (t, c), returns $\mathsf{TBDec}(\mathsf{sk}, t, c)$ if $t \neq t^\star$ and test otherwise.

ity:

$$Pr\left(\begin{array}{c} \mathsf{crs}, \mathsf{tk} \leftarrow\$\, \mathsf{SimGen}(1^\lambda); \\ (x^*, \tau^*, \pi^*) \leftarrow \mathcal{A}^P(\cdot,\cdot)(\mathsf{crs}) : \\ \forall w^*\, (x^*, w^*) \notin R \wedge \tau^* \notin Q_{tags} \\ \wedge \mathsf{PVerify}(\mathsf{crs}, x^*, \tau^*, \pi^*) = 1 \end{array}\right)$$

is negligible in λ, where the adversary has access to an oracle $P(x, \tau)$ which returns $\mathsf{SimProve}(\mathsf{crs}, \mathsf{tk}, x, \tau)$ and add τ to a set of tags Q_{tags}.

Intuitively, simulation soundness guarantees that an adversary cannot produce proofs for fresh fake statements, even if it sees simulated proofs of other unrelated (with respect to the tag) statements. This is particularly useful for example to build CCA secure encryption scheme [Lib+14], and we will use this tool to reach our security notions.

2.3. Verifiable voting definitions

2.3.1. A blueprint for single-pass verifiable voting protocols

Many voting protocols follow a methodology pioneered by Benaloh [CF85] and refined by Cramer et al. [CGS97]. These protocols include, among others, VoteBox [SDW08], Helios [Adi08; Adi+09], STAR-Vote [Ben+13], Belenios [CGG19], and most of ElectionGuard applications [Ele24]. From this general methodology, we formulate a blueprint for this family of verifiable voting protocols as follows:

1. First, an official party generates the public key of the election, which is the public key of a threshold encryption scheme. This party, called the tallier or trustee, will be responsible of computing the tally of the election.

2. Then, in the voting phase, voters craft their ballots by encrypting their votes with the public key of the election, and possibly authenticating it using private credentials obtained from a prior registration. This ballot also typically contains a zero-knowledge proof that the ballot contains a valid encrypted choice. This ballot is then sent to a casting server, processed, and published on a public bulletin board.

3. After every voter has voted, the tallier computes the result of the election using the distributed secret key of the encryption scheme. It also publishes a zero-knowledge proof that this result has been correctly computed.

4. Finally, every voter verifies, if they desire so, that their ballot is correctly recorded on the bulletin board and that the result of the election is correctly computed from the ballots on the bulletin board.

Security notions Jumping ahead, the main security properties we expect from a voting system are the privacy ("nobody should know the choices of a voter") and the verifiability ("if the result of the election is altered, it should be detectable through some verification process"). Guaranteeing both privacy and verifiability is no easy task as these two properties seem to be of antagonistic nature, hence the use of cryptographic tools.

The encryption of votes aims at providing privacy, as it hides the voters' choices from the casting server, but also from anyone since these encryptions are published on a public bulletin board. Naturally, the votes are not completely hidden, since the trustee can decipher these votes with the secret key of the encryption scheme, and this is desirable since it must be feasible to compute the result of the election.

On the other hand, the bulletin board helps at convincing voters that their votes have been correctly recorded by checking if the ballots that they sent indeed appear on the bulletin board. A malicious casting server could still drop some voters' ballots, but in this case, conscientious voters would notice this fraud and alert the election authorities.

Additionally, the zero-knowledge proofs help at convincing everyone checking them that the tally is computed correctly from the valid ballots on the bulletin board. Also, the zero-knowledge property of these proofs is crucial, since it preserves the voters' privacy.

One of the focuses of this thesis is receipt-freeness, a stronger notion related to the voters' privacy. This notion captures that even if voters are willing to cooperate with an adversary, they should not be able

to produce a proof (i.e. a *receipt*) of how they voted. This notion is motivated to prevent vote-selling (or other forms of coercion), a scenario in which a buyer gives money to a voter in exchange for a proof that they voted in a given way. However, if there is no way for the voter to produce this proof, they could always accept the money and vote as they desire.

It is the processing of the ballot before publishing it that helps achieve receipt-freeness. Without this processing, and if a voter is able to retrieve all the computation steps that lead to the ballot published on the bulletin board, then there is a trivial receipt of the content of the ballot. This processing, which will typically consist in rerandomizing the encryption in this thesis, helps introduce fresh randomness in the generation of the ballot to prevent such attacks.

Like the building blocks presented above, we advocate for a provable security approach. In this regard, these security notions are formally captured such that one can prove that a protocol satisfies them rigorously. The formal definitions are presented in Section 2.3.5.

The single-pass flow This family of protocols is coined *single-pass* since the flow of the voting process consists in a single interaction between the voter and the voting server in the casting process. This notably implies that it is possible to generate a valid ballot using only the public key of the election, the voting intention, and, if they exist, private credentials.[5]

Restraining the number of interactions can be beneficial, as an interactionless protocol leads to fewer mistake during its execution and to an overall simpler implementation. Also, this flow is similar to more classical paper voting protocols in which voters fill their votes on paper ballots and cast them directly in the ballot box. Given that many voting protocols follow this flow, it is natural to study this general family of protocols.

We stress that this restriction does not involve the interactions between the "human" voter and the voting device. Indeed, human voters would typically make use of an electronic device to perform all the complex cryptographic computations that are needed to craft a ballot. In this process, there may be multiple interactions between the human voter and the device. These interactions include for example all the ways the voter engages with the voting device's UI such as pressing a button, which is usually not formalized, but also any *cast-or-challenge* mechanism that

[5] One could argue that for the voter to receive the public key of the election, we need at least one interaction with the voting server. We assume that this interaction only delivers the public key and does not contain additional materials that need to be used for the generation of a ballot.

can be used to offer protection against a corrupted voting device. We consider that a protocol with such interactions can still be considered single-pass, as limiting these interactions would drastically deteriorate the design space of the protocols. Hence, we can see the creation of a ballot as an offline algorithm that takes as input only the public key of the election, the voter's voting intent, and optionally any auxiliary data obtained from a registration procedure, and outputs a unique ballot to be sent to the casting server.

We also emphasize that there might be additional interactions between the voters and the election officials, depending on the design of the protocol. For example, the voter can interact with an official registration service that delivers some private credentials or with the bulletin board for verification purposes. These interactions are allowed even in single-pass protocols, since they are independent from the casting process.

Finally, in the design of voting systems, it is common to give the voters the opportunity to *revote*. Revoting can be used to cast a new ballot if for the voter suspect that their previously cast ballot fails to reflect their voting intention (because of a malicious voting device for example) or to escape a coercer that was present when the voter cast a first ballot but is not present anymore. We argue that a voting system with revoting can still be considered single-pass, if subsequent ballots can be computed independently of the previously cast ballots, and hence solely from the public key, the voting intention, and potentially voter credentials.

2.3.2. Trust model

Depending on the context of the election for which the voting protocol is used, the level of trust in the involved parties might vary. In governmental elections, for instance, there are often many security concerns as the stakes are high. In contrast, such concerns may be less critical in lower-stakes scenarios, such as elections for student representatives at a university.

The level of trust we place in the involved parties has a direct impact on the design of the voting protocol. Intuitively, the less we trust these parties, the more complex and expensive the required techniques become. In this section, we describe and motivate the trust assumptions made about each party and each security property throughout this thesis.

Verifiability In the settings that require the highest level of caution, the election authorities themselves might not be trusted for the election integrity, since they might have strong incentives to influence the result of the election. Throughout most of this thesis, we focus on the design of voting protocols that are verifiable even in the extreme case where

all official parties are corrupted and control the generation of the secret key of the election. In other words, even if all those parties collude, they cannot change the result of the election without being detected.

To reach this low level of trust, we assume that the voters have a secure way to authenticate themselves in order to receive election materials and to cast a ballot. In polling stations, this authentication can be done by showing an ID to an election official. Implementing a secure authentication mechanism is harder in a remote setting, but in Estonia, for example, voters can authenticate themselves with their physical ID through a card reader.[6]

Likewise, we assume that only *eligible* voters can cast a ballot, and their vote is counted only once in the tally. In elections where the information whether or not a voter has cast a vote is public, this can be enforced by printing on the bulletin board the identity of the voter next to the encrypted ballot. While the authentication and eligibility of voters is crucial to ensure the integrity of an election, we do not address these particular issues in this thesis and trust the authentication mechanism in this regard.

Moreover, human voters typically make use of an electronic device to perform all the complex cryptographic computations that are needed to craft a ballot. If this voting device were to be corrupted by an adversary, it could alter the voter's intention, and it is thus necessary to have a verifiability mechanism to prevent that. This notion of verifiability against a corrupted voting device is often called *cast-as-intended* verifiability. Since the first chapter of this thesis focuses on the receipt-freeness property, which is hard to obtain at the same time as the cast-as-intended verifiability property, we assume first that this voting device is honest and perfectly executes whatever code it received as input. In our case, the code would be the algorithm that crafts a ballot for a given vote.

To help in our formalism, we will thus denote by "the voter" as a single entity combining both the human being and the voting device, which simplifies the formalism of this ballot creation algorithm. When we say in a protocol that the voter executes some algorithm, it means that the voter uses the voting device to perform these computations. Chapter 5, however, aims at designing protocols attaining both receipt-freeness and cast-as-intended verifiability. In this regard, we will naturally not keep on with this simplifying view anymore and extend our model to handle interactions between the human voter and a possibly malicious voting device.

[6] `https://www.valimised.ee/sites/default/files/2023-02/IVXV-protocols.pdf`

Privacy Regarding privacy, we relax our trust model. Namely, we allow the privacy of the schemes to be broken if one of the official parties is corrupted. This is motivated by the fact that the holder of the secret key alone can decrypt all the ballots on the go and thus already break a weak form of privacy. There exist strategies to circumvent this single choke point, typically by using a threshold encryption scheme and distributing the keys to a set trustees. However, it is easier to treat the key as if it were generated by a central authority in the modeling and the security analysis of our schemes. Hence, we leave the distributed treatment of our algorithms as out of scope for this thesis, even though they could easily be adapted without compromising the security of the protocols. Likewise, we assume the voting device to be honest in this regard as well, as a corrupted device would trivially break the voters' privacy. We thus consider an honest single trustee and an honest voting device in our treatment of privacy.

We still require that the bulletin board does not leak any information about the voters' choices, besides what can be learned from the result of the election.[7] Even more, we want to offer protection against an adversary that can access this bulletin board at will, but also adaptively controls other voters participating in the election and crafts malicious ballots on their behalf. This is a reasonable scenario, since we could imagine that a motivated adversary could hack the voting devices of many voters, and still offer protection to voters that are not hacked. Furthermore, we assume that such an adversary may also gain access to voters' private credentials used during registration. These sensitive materials are typically generated prior to the election and are inherently more difficult to protect against potential leaks.

Receipt-Freeness Since receipt-freeness is a flavor of privacy, we keep the same trust model for this property, with a one key exception. Indeed, we can no longer afford the casting server to be malicious since this is the party that processes the ballot and brings the fresh entropy the voter cannot predict to craft a receipt. This casting server is thus trusted in our protocols to not share its view to a receipt-freeness adversary.

We also emphasize again that our assumption on the honesty of the voting device is that it correctly executes whatever code is given as input by the voter, in the fashion of a universal Turing machine. By default, if no input code is given, the device runs the voting algorithm specified by

[7] If the result of an election is the percentage of voters that voted for a given candidate, and if everyone voted for the same candidate in this election, then the result naturally leaks the choice of every voter. This leak is inherent to the election outcome itself and cannot be prevented.

the protocol. This is different from requiring the voting device to always correctly execute the voting algorithm, as coerced voters could ask their voting device to deviate from this algorithm to produce a receipt.

For example, a typical voting algorithm encrypts the voters' choices but does not return the randomness used in this encryption process. If the resulting ballot is not processed and is published as-is, this randomness, combined with the ballot, can uniquely identify how they voted, effectively acting as a receipt. In a setting where the voting device always executes the voting algorithm, the voter would never have access to this randomness and the protocol could be considered receipt-free. This is particularly relevant in a setting where the voters have little control on the device, such as in polling stations. However, if the voters can execute any instruction on their devices, they can ask their device to return this randomness and send it back to the adversary. Such a protocol would not be considered receipt-free anymore. This is particularly relevant in a remote voting setting, where voters use their own devices and can control what is run on them.

A remote voting setting brings other threats: an attacker could physically coerce voters and ask them to vote in their presence without any protection from the voting booth, or even directly request their credentials and vote on their behalf. The first scenario does not scale well and is hardly practical for an adversary to coerce a large number of voters. On the other hand, the second scenario can be mitigated through an authentication mechanism relying on a physical ID or a SIM card. Especially, these mechanisms are sometimes also used to authenticate users to their bank account, and in such case voters would be deterred to give access to an adversary to their authentication credentials. Still, these threats can endanger the integrity of an election, and they must be considered in a high stakes election. We emphasize that the protocols studied in this thesis do not offer convincing protection against such threats in, for example, a governmental election.

Our threat model, which ensures unconditional verifiability while maintaining privacy with stronger trust assumption, is a design choice that is common in the literature, such as in CGS [CGS97], Helios [Adi+09] or ElectionGuard [Ele24] for example.

2.3.3. Model and syntax

We define voting protocols in a way that largely follows the SoK from Bernhard et al. [Ber+15] and BeleniosRF [Cha+16]. In this thesis, we focus on "single-pass" protocols, in which voters can cast their ballot by sending a single message. Further interactions are only needed for

verifiability purposes. In our voting protocols, we consider the following parties:

- The *voters* are participating in the election and are willing to cast ballots representing their vote intent. The identities of these voters are denoted by a set $\mathsf{ID} = \{\mathsf{id}_1, \ldots, \mathsf{id}_m\}$.
- The *election administrator* is organizing the election and is responsible for coordinating the protocol execution.
- The *registrar* distributes private credentials to voters and registers the corresponding public credentials.
- The *ballot box manager*, or casting server, gathers the ballots of the voters on a bulletin board BB and provides a public view PBB of those ballots, for verifiability.
- The *trustee* is responsible for genreating the key of the election, correctly tallying the ballot box and providing a proof of the correctness of the tally.

These parties are standard entities in the voting literature. We call *official parties* all the parties except the voters. In some cases, we will also refer to the ballot box manager as the rerandomizing server, in order to make its receipt-freeness related role more visible.

We also define a result function $\rho : \mathbb{V}^\star \to \mathbb{R}$ which given a set of votes, returns the result of the election for these votes. The following definition encompasses the procedures used in a voting system.

Definition 10 (Voting System)**.** *A Voting System is a tuple of probabilistic polynomial-time algorithms* (Setup, $\mathsf{Register}$, Vote, $\mathsf{ProcessBallot}$, $\mathsf{TraceBallot}$, Valid, $\mathsf{Publish}$, $\mathsf{VerifyVote}$, Tally, $\mathsf{VerifyResult}$) *associated to a result function* $\rho : \mathbb{V}^\star \to \mathbb{R}$ *where* $\mathbb{V}$ *is the set of valid votes and* $\mathbb{R}$ *is the result space such that:*

- $\mathsf{Setup}(1^\lambda)$*: on input a security parameter* λ*, generate the public parameters of the election algorithms, as well as the public and secret keys* $(\mathsf{pk}, \mathsf{sk})$ *of the election. We will consider the public key of the election* pk *to be an implicit argument for the following algorithms.*
- $\mathsf{Register}(\mathsf{id})$*: on input an identifier* $\mathsf{id} \in \mathsf{ID}$*, outputs a public/private credential pairs* $(\mathsf{upk}, \mathsf{usk})$ *and add* upk *to a list of credentials* L*. We call the voting system registration-free if* usk *is set to* $\perp$*.*

- $\mathsf{Vote}(\mathsf{id}, \mathsf{usk}, \mathsf{v})$*: when receiving a voter* $\mathsf{id} \in \mathsf{ID}$*, private credentials* usk *and a vote* v*, outputs a ballot* b *and auxiliary data* aux*. It will also be possible to call* $\mathsf{Vote}(\mathsf{id}, \mathsf{usk}, \mathsf{v}, \mathsf{aux})$ *in order to obtain a ballot (which does not output auxiliary data this time) for vote* v *using* aux*. This auxiliary data will be useful to define security and enables the creation of ballots that share the same* aux*.*

- $\mathsf{ProcessBallot}(\mathsf{b})$*: on input ballot* b*, outputs an updated ballot* b'*. Typically,* b' *can be* b *itself or a rerandomization of* b*.*

- $\mathsf{TraceBallot}(\mathsf{b})$*: on input ballot* b*, outputs a tag* t*. The tag is the information that a voter can use to trace his ballot, using* $\mathsf{VerifyVote}$*.*

- $\mathsf{Valid}(\mathsf{BB}, \mathsf{b})$*: on input ballot box* BB *and ballot* b*, outputs* 0 *or* 1*. The algorithm outputs* 1 *if and only if the ballot is valid.*

- $\mathsf{Publish}(\mathsf{BB})$*: on input ballot box* BB*, outputs the public view* PBB *of* BB*, which is the one that is used to verify the election. Depending on the context, it may be used to remove some voter id for instance.*

- $\mathsf{VerifyVote}(\mathsf{PBB}, t)$*: on input public ballot box* PBB *and tracing tag* t*, outputs* 0 *or* 1*. This algorithm is used by voters to check if their ballot has been processed and recorded properly.*

- $\mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$*: on input ballot box* BB *and secret key of the election* sk*, outputs the tally* r *and a proof* Π *that the tally is correct w.r.t. the result function* ρ_m*.*

- $\mathsf{VerifyResult}(\mathsf{PBB}, \mathsf{r}, \Pi)$*: on input public ballot box* PBB*, result of the tally* r *and proof of the tally* Π*, outputs* 0 *or* 1*. The algorithm outputs* 1 *if and only if* Π *is a valid proof that* r *is the election result, computed w.r.t.* ρ_m*, corresponding to the ballots on* PBB*.*

These algorithms are used as follows in a typical election: First, the trustee generates the public and secret keys of the election with Setup. Then, voters authenticate themselves to the registrar and receive their credentials. If the $\mathsf{Register}$ algorithm is run with an input id different from the identity of the voter that is running it, then the algorithm aborts and returns $\perp$. Voters can then prepare a ballot b with the Vote algorithm using the public key of the election. This execution of the Vote algorithm is performed through a voting device that takes as input the Vote algorithm and the inputs of the algorithm, and outputs a ballot. This ballot is then sent to the ballot box manager, after another authentication procedure that fails if this is not the voter with identity id who is sending the ballot.

The voter also keeps $\mathsf{TraceBallot}(\mathsf{b})$ in order to be able to trace its ballot on the election public bulletin board. Each time the ballot box manager receives a ballot, it checks if it is valid with the Valid algorithm. If this is the case, it runs the $\mathsf{ProcessBallot}$ algorithm on it and appends the resulting ballot to the ballot box. This procedure of checking the validity of a ballot, processing it and appending it to the ballot box is captured by an extra algorithm Append for our convenience:

$$
\begin{aligned}
&\mathsf{Append}(\mathsf{BB}, \mathsf{b}) := \\
&\quad \text{if } \mathsf{Valid}(\mathsf{BB}, \mathsf{b}) \text{ then} \\
&\qquad \text{return } \mathsf{BB} || \mathsf{ProcessBallot}(\mathsf{b}) \\
&\quad \text{else return } \mathsf{BB}
\end{aligned}
$$

The ballot box manager also applies $\mathsf{Publish}$ on the ballot box in order to obtain the content that is made available on a public bulletin board PBB. Voters can check that their ballots have been correctly recorded on PBB using $\mathsf{VerifyVote}$ and the trace of their ballots.

Eventually, the trustee runs the Tally algorithm on the ballot box in order to compute the election result and a proof of correctness of this result. Anyone can use these, together with the content of PBB, in order to verify the election result using $\mathsf{VerifyResult}$.

This definition slightly differs from [Ber+15] and [Cha+16]. The main difference is the introduction of the $\mathsf{TraceBallot}$ algorithm. Such a procedure is implicit in other voting system descriptions, because $\mathsf{ProcessBallot}$ is often the identity function (such as in Helios) and voters directly check the presence of their ballot on PBB (in other words, $\mathsf{TraceBallot}$ is the identity function). However, some protocols modify the ballot before publishing it on the bulletin board, by rerandomizing it for example (which is motivated by receipt-freeness). It is thus useful to have a more expressive tracing algorithm for such cases.

With these algorithms of Definition 10, we can formally define an execution of a single-pass voting system in Figure 2.4. While this execution involves the aforementioned parties in practice, we present a streamlined flow of the execution for the sake of simplicity. The input of this execution is the security parameter λ and a sequence S of m pairs $(\mathsf{id}, \mathsf{v}_{\mathsf{id}})_{\mathsf{id} \in \mathsf{ID}}$ representing the voting intention of each voter and the order in which they cast their vote, where each of these ids is unique.

We expect from a voting system to satisfy some correctness properties guaranteeing that its execution produces a correct result.

Definition 11 (Correctness of a voting system)**.** *A voting system is correct for a result function ρ if for every security parameter λ and*

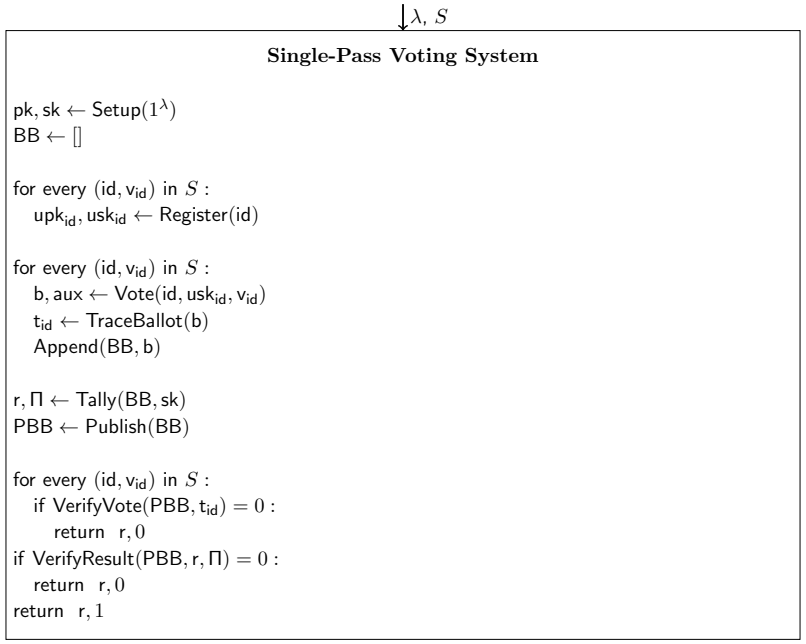


Figure 2.4.: Execution of a a single-pass voting system. The parameters of the execution are the security parameter λ and a sequence of pairs of (unique) voters' identity and voting intention S.

sequence of votes $S = (\mathsf{id}, \mathsf{v}_{\mathsf{id}})_{\mathsf{id}}$*, the result computed as described in Figure 2.4 is equal to* $\rho(V)$ *where* $V = (\mathsf{v}_{\mathsf{id}})_{(\star,\mathsf{v}_{\mathsf{id}})\in S}$.

Our formalism also allows for revoting policies, and even though it is not captured in Figure 2.4 for the sake of simplicity, it is possible to formulate a correctness notion for single-pass voting system with revoting.

2.3.4. Generic Tally Algorithm

In this thesis, we design voting protocols that produce bulletin boards that are compatible with standard tallying process. To compute the result of the election, there are two big families of voting protocols: homomorphic tallying and mixing networks.

Homomorphic tallying is particularly well suited for approval voting elections, where voters are given a list of candidates and needs to either approve or disapprove each candidate. In this family of protocols, voters encrypt one bit for each candidate and provide a zero-knowledge proof that each of the resulting ciphertexts indeed contains a bit. To compute the tally of election, the trustee homomorphically aggregates all the ballots to compute an encryption of the sum of the votes for each candidate, which can be done if the encryption scheme is additively homomorphic, and then decrypt the result. Protocols based on homomorphic voting have a simple tallying process, but require a result that can be easily computed homomorphically and is thus not adapted to every electoral system.

For example, in a instant runoff voting (IRV) election, voters are asked to rank the candidates, and then the result of the election is computed by eliminating the least popular candidate until the winner is determined. As the number of possibilities to express a ranking is the factorial of the number of candidates, it becomes soon impractical to homomorphically compute the result of the election with this approach.[8]

On the other hand, the second family of protocols is based on mix networks [Cha81]. The main idea is to provably shuffle the ballots multiple times to break their original link with the voters. Then, every ballot can be decrypted individually and the result directly computed from these decrypted votes. The privacy of the voters is thus respected as long as the mixing servers do not collude. This second family of protocols allows for more expressive votes, but requires a more complex and expensive tally procedure.

[8] Even worse, the blow up can increase faster than a factorial if the voter is not required to rank all the candidates.

In the CGS voting scheme [CGS97] for example, voters encrypt their vote bit-by-bit with an ElGamal encryption scheme. To prove that the resulting ciphertexts actually encrypts bits, the voters run a disjunctive Chaum-Pedersen protocol [CDS94; CP92a] in conjunction with the Fiat-Shamir transformation [FS86].

Throughout this work, we build algorithms that are as agnostic to the tallying process as possible, and assume we have access to a tally algorithm that can be used in a threshold manner and that has generic properties.

Especially, we assume that the Tally algorithm can be seen as a tuple $\mathsf{TCompute}, \mathsf{TProveGen}, \mathsf{TProve}, \mathsf{TVerify}$, such that:

- TCompute: On input $(\mathsf{sk}, \mathsf{BB})$, TCompute outputs the result of the election r.
- TProveGen: TProveGen outputs a CRS for the proof system. If the proof system is based on the ROM, the CRS can be empty.
- TProve: On input $(\mathsf{crs}, \mathsf{sk}, \mathsf{BB}, r)$, TProve outputs the proof Π that r is computed from an honest execution of TCompute.
- TVerify On input $(\mathsf{crs}, \mathsf{BB}, r, \Pi)$, TVerify returns 1 if and only if Π is a valid proof for this bulletin board and this result.

In general, TProveGen will be run in the Setup algorithm and the resulting CRS will be included in the public key. Likewise, TVerify will be run in the VerifyResult algorithm.

Furthermore, we assume that this proof is zero-knowledge. That is, there exists two PPT simulators TSimGen and TSimProve such that:

1. TSimGen outputs a CRS as well as a trapdoor τ.
2. For any bulletin board BB, the distributions $(\mathsf{crs}, \mathsf{TProve}(\mathsf{crs}, \mathsf{sk}, \mathsf{BB}, r))$ and $(\mathsf{crs}_{sim}, \mathsf{TSimProve}(\mathsf{crs}, \tau, \mathsf{BB}, r))$ are computationally indistinguishable, where crs is generated honestly from TProveGen and crs_{sim} is generated from TSimGen.

This Tally algorithm can be efficiently realized for both elections with mixnet or homomorphic tallying.

2.3.5. Security notions

Ballot Privacy The ballot privacy notion essentially guarantees the privacy of the voters. More precisely, no one should learn anything about the voters' intents besides what can be learned from the result of the

election, and this should hold despite the presence of a public bulletin board in which an encryption of these voting intents are published. This notion also offers protection against an active adversary that can cast ballots crafted as a funtion of the information the adversary can see on the board, such as in replay attacks [CS13].

BPRIV is a popular ballot privacy definition proposed by Bernhard et al. [Ber+15] which captures this intuition through a game-based approach. This privacy notion focus on what an outside observer can learn from the bulletin board, hence the trustee and casting server are considered honest in this definition. We will adapt the ballot privacy definition in Chapter 3 to also offer protection against a malicious casting server.

In this definition, an adversary attempts to distinguish between two worlds. In the first world, the adversary has access to a real bulletin board containing the ballots of the voters, as well as maliciously crafted ballots. In the second world, the adversary has access to a fake bulletin board containing dummy votes and the same maliciously crafted ballots. However in this second world, the fake bulletin board contains the result of the tally of the real bulletin board, as well as a simulated proof of the validity of the tally. By only getting the view of one of those bulletin boards, the adversary aims at guessing which world it sees. In case of a registration-free scheme, the ballot privacy can be formulated as the following:

Definition 12 (Ballot Privacy)**.** *A voting system $\mathcal{V}$ has ballot privacy if there exists algorithms* $\mathsf{SimSetup}$ *and* $\mathsf{SimProve}$ *such that no* PPT *adversary $\mathcal{A}$ can distinguish between games* $\mathsf{Exp}^{\mathsf{bpriv},0}_{\mathcal{A},\mathcal{V}}(\lambda)$ *and* $\mathsf{Exp}^{\mathsf{bpriv},1}_{\mathcal{A},\mathcal{V}}(\lambda)$ *defined by the oracles in Figure 2.5. That is, for any efficient algorithm $\mathcal{A}$:*

$$\left| Pr\left[\mathsf{Exp}^{\mathsf{bpriv},0}_{\mathcal{A},\mathcal{V}}(\lambda) = 1\right] - Pr\left[\mathsf{Exp}^{\mathsf{bpriv},1}_{\mathcal{A},\mathcal{V}}(\lambda) = 1\right] \right|$$

is negligible in λ.

BPRIV alone is not enough to guarantee the privacy of the voters, but when it is supplemented by two additional security properties, strong consistency and strong correctness, it has been shown that for a voting scheme secure in this regard, an adversary can extract as much information as it can extract from only seeing the result and nothing more. For the sake of thoroughness, we adapt these two notions to take into account the non-trivial $\mathsf{ProcessBallot}$ algorithm that was not present in their initial formalism.

Definition 13 (Strong consistency)**.** *A voting scheme $\mathcal{V}$ has strong consistency if there is*

$\mathcal{O}\mathsf{init}(\lambda)$

if $\beta = 0$ **then** $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
else $(\mathsf{pk}, \mathsf{sk}, \tau) \leftarrow \mathsf{SimSetup}(1^\lambda)$
$\mathsf{BB}_0 \leftarrow \bot; \mathsf{BB}_1 \leftarrow \bot$
return pk

$\mathcal{O}\mathsf{voteLR}(\mathsf{id}, \mathsf{v}_0, \mathsf{v}_1)$

if $\mathsf{v}_0 \notin \mathbb{V}$ or $\mathsf{v}_1 \notin \mathbb{V}$ **then return** $\bot$
$\mathsf{b}_0, \mathsf{aux}_0 = \mathsf{Vote}(\mathsf{id}, \mathsf{usk}_{\mathsf{id}}, \mathsf{v}_0)$
$\mathsf{b}_1, \mathsf{aux}_1 = \mathsf{Vote}(\mathsf{id}, \mathsf{usk}_{\mathsf{id}}, \mathsf{v}_1)$
$\mathsf{Append}(\mathsf{BB}_0, \mathsf{b}_0); \mathsf{Append}(\mathsf{BB}_1, \mathsf{b}_1)$

$\mathcal{O}\mathsf{register}(\mathsf{id})$

if $\mathsf{id} \notin \mathsf{ID}$:
then return $\bot$
$\mathsf{upk}_{\mathsf{id}}, \mathsf{usk}_{\mathsf{id}} \leftarrow \mathsf{Register}(\mathsf{id})$
return $\mathsf{upk}_{\mathsf{id}}, \mathsf{usk}_{\mathsf{id}}$

$\mathcal{O}\mathsf{cast}(\mathsf{b})$

if $\mathsf{Valid}(\mathsf{BB}_0, \mathsf{b}) = 0$ or $\mathsf{Valid}(\mathsf{BB}_1, \mathsf{b}) = 0$
then return $\bot$
else $\mathsf{Append}(\mathsf{BB}_0, \mathsf{b}); \mathsf{Append}(\mathsf{BB}_1, \mathsf{b})$

$\mathcal{O}\mathsf{tally}()$

$(\mathsf{r}, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}_0, \mathsf{sk})$
if $\beta = 1$ **then** $\Pi \leftarrow \mathsf{SimProve}(\mathsf{BB}_1, \mathsf{r}, \tau)$
return (r, Π)

$\mathcal{O}\mathsf{board}()$

return $\mathsf{Publish}(\mathsf{BB}_\beta)$

Figure 2.5.: Oracles used in the $\mathsf{Exp}^{\mathsf{bpriv},\beta}_{\mathcal{A},\mathcal{V}}(\lambda)$ experiment. The adversary first calls $\mathcal{O}\mathsf{init}$ and as many $\mathcal{O}\mathsf{register}$ requests as it wants, then can call $\mathcal{O}\mathsf{voteLR}$, $\mathcal{O}\mathsf{cast}$ and $\mathcal{O}\mathsf{board}$ as much as it wants. Finally, the adversary calls $\mathcal{O}\mathsf{tally}$, receives the result of the election and must return its guess, which is the output of the experiment.

$\mathsf{Exp}^{\text{strong_consistency}}_{\mathcal{A},\mathcal{V}}(\lambda)$

$(\mathsf{pk},\mathsf{sk}) \leftarrow\$\, \mathsf{Setup}(1^\lambda)$
$\mathsf{BB} \leftarrow\$\, \mathcal{A}(\mathsf{pk})$
$(\mathsf{r},\Pi) \leftarrow\$\, \mathsf{Tally}(\mathsf{BB},\mathsf{sk})$
if $r \neq \rho_n(\mathsf{Extract}(\mathsf{sk},\mathsf{b}_1),\ldots,\mathsf{Extract}(\mathsf{sk},\mathsf{b}_n))_{\mathsf{b}_i \in \mathsf{BB}\text{ s.t. }\mathsf{ValidInd}(\mathsf{b}_i)=1}$
then return 1 **else return** 0;

Figure 2.6.: Strong consistency experiment.

1. *An extraction algorithm* Extract *that takes as input a secret key* sk *and a ballot* b *and outputs* $\mathsf{v} \in \mathbb{V}$.
2. *A ballot validation algorithm* ValidInd *that takes as input the public key of the election* pk *and a ballot* b *and outputs* 0 *or* 1.

which satisfy these conditions:

- *For any* $(\mathsf{pk},\mathsf{sk})$ *in the range of* Setup*, for any vote* v *and for any* b,aux *in the range of* $\mathsf{Vote}(\mathsf{id},\mathsf{v})$, $\mathsf{Extract}(\mathsf{sk},\mathsf{b}) = \mathsf{v}$ *as well as* $\mathsf{Extract}(\mathsf{sk},\mathsf{ProcessBallot}(\mathsf{b})) = \mathsf{v}$.
- *For any* $(\mathsf{BB},\mathsf{b}) \leftarrow\$\, \mathcal{A}$, $\mathsf{Valid}(\mathsf{BB},\mathsf{b}) = 1$ *implies* $\mathsf{ValidInd}(b) = 1$.
- *Consider an adversary* $\mathcal{A}$ *which is given* pk *and consider the experiment* $\mathsf{Exp}^{\text{strong_consistency}}_{\mathcal{A},\mathcal{V}}(\lambda)$ *given in Figure 2.6. We require that the probability* $Pr[\mathsf{Exp}^{\text{strong_consistency}}_{\mathcal{A},\mathcal{V}}(\lambda) = 1]$ *is negligible in* λ.

The second point of the strong consistency is novel, as in the model of [Ber+15], ProcessBallot was modeled as a the identity function. We simply require the ProcessBallot algorithm to keep the same vote contained in the ballot. We require perfect correctness for the sake of simplicity, but these definitions could be adapted to a computational variant for voting protocols based on imperfect encryption schemes.

Intuitively, strong correctness guarantees that the validity of honestly generated ballot should be independent of the bulletin board, and hence a maliciously crafted bulletin board cannot discard ballots of honest voters. Likewise, ProcessBallot should preserve the validity of honestly generated ballots, otherwise a pathological ProcessBallot could also discard ballots of honest voters.

Definition 14 (Strong correctness)**.** *A voting scheme* $\mathcal{V}$ *has strong correctness if for any adversary* $\mathcal{A}$ *that takes as input* pk*, the probability*

$$Pr\left[\begin{array}{c}(\mathsf{id},\mathsf{v},\mathsf{BB}) \leftarrow\$\, \mathcal{A}(\mathsf{pk});\, \mathsf{b},\mathsf{aux} \leftarrow\$\, \mathsf{Vote}(\mathsf{id},\mathsf{v}) : \\ v \in \mathbb{V} \text{ and } (\mathsf{Valid}(\mathsf{BB},b) = 0 \text{ or } \mathsf{Valid}(\mathsf{BB},\mathsf{ProcessBallot}(b)) = 0)\end{array}\right]$$

is negligible in λ.

Finally, we need an extra definition on the Valid algorithm to avoid yet another pathological case. Indeed, there could exist valid ballots that are not in the range of Vote but would trigger an adversarial behavior in the ProcessBallot algorithm. This is especially important since in the receipt-freeness game defined later in this section, the adversary can send ballots of its choice. This definition ensures that it is computationally hard to compute such pathological ballots.

Definition 15 (Strong validity)**.** *A voting scheme $\mathcal{V}$ has strong validity if for any adversary $\mathcal{A}$ that takes as input* pk*, the probability*

$$Pr\left[\begin{array}{c}(\mathsf{b},\mathsf{BB}) \leftarrow\!\$\, \mathcal{A}(\mathsf{pk}) : \\ \mathsf{Valid}(\mathsf{BB},\mathsf{b}) = 1 \text{ and } \mathsf{b} \text{ not in the range of } \mathsf{Vote}(\mathsf{pk},\cdot)\end{array}\right]$$

is negligible in λ.

Receipt-Freeness We essentially consider a strong form of receipt-freeness in this work, in which the adversary can interact with the voters before the voting process and hence give them instructions just before they vote. Besides, we consider that the voters have total control of the voting device on which their ballot is computed and can thus execute the adversary's instruction on this device.

We adopt here a definition of receipt-freeness that is similar to the one of Chaidos et al [Cha+16], which was itself inspired by the BPRIV definition. This definition is designed with single-pass voting scheme in mind, as the absence of interactions simplifies the formalization of receipt-freeness. Indeed, in this case, the adversary's instruction can be considered as the ballot to be cast and the voter does not have any receipt as this ballot is the only message that is sent in the voting process.

Our definition is slightly different but takes into account some attacks that are highlighted from our formalization of the TraceBallot algorithm. Various other definitions exist, but they are either too informal for our purpose (e.g., [HS00]), or focus on the stronger notion of coercion resistance, in which voters need to adopt a specific counter-strategy depending on instructions of the coercer (e.g., [MN06; UM10; KTV10; Alw+15]).

This definition ensures that voters should not be able to pick a pair of ballots in such a way that it is possible to distinguish which one of them has been uploaded on the bulletin board. We can see one of the ballots has the adversary's instruction and the other ballot as encoding the voter's true intention. We stress that these ballots can possibly be unexplainable by an honest execution of the voting algorithm and contain some subliminal information to help the tracking of the ballot.

$\mathcal{O}\mathsf{init}(\lambda)$

if $\beta = 0$ **then** $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
else $(\mathsf{pk}, \mathsf{sk}, \tau) \leftarrow \mathsf{SimSetup}(1^\lambda)$
$\mathsf{BB}_0 \leftarrow \bot; \mathsf{BB}_1 \leftarrow \bot$
return pk

$\mathcal{O}\mathsf{receiptLR}(\mathsf{b}_0, \mathsf{b}_1)$

if $\mathsf{TraceBallot}(\mathsf{b}_0) \neq \mathsf{TraceBallot}(\mathsf{b}_1)$
or $\mathsf{Valid}(\mathsf{BB}_0, \mathsf{b}_0) = 0$
or $\mathsf{Valid}(\mathsf{BB}_1, \mathsf{b}_1) = 0$
then return $\bot$
else $\mathsf{Append}(\mathsf{BB}_0, \mathsf{b}_0); \mathsf{Append}(\mathsf{BB}_1, \mathsf{b}_1)$

$\mathcal{O}\mathsf{register}(\mathsf{id})$

if $\mathsf{id} \notin \mathsf{ID}$:
then return $\bot$
$\mathsf{upk}_{\mathsf{id}}, \mathsf{usk}_{\mathsf{id}} \leftarrow \mathsf{Register}(\mathsf{id})$
return $\mathsf{upk}_{\mathsf{id}}, \mathsf{usk}_{\mathsf{id}}$

$\mathcal{O}\mathsf{board}()$

return $\mathsf{Publish}(\mathsf{BB}_\beta)$

$\mathcal{O}\mathsf{tally}()$

$(\mathsf{r}, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}_0, \mathsf{sk})$
if $\beta = 1$ **then** $\Pi \leftarrow \mathsf{SimProve}(\mathsf{BB}_1, \mathsf{r})$
return (r, Π)

Figure 2.7.: Oracles used in the $\mathsf{Exp}^{\mathsf{srf},\beta}_{\mathcal{A},\mathcal{V}}(\lambda)$ experiment. The adversary first calls $\mathcal{O}\mathsf{init}$ and then can call $\mathsf{Oregister}$, Oboard and $\mathsf{OreceiptLR}$ as much as it wants. Finally, the adversary calls $\mathcal{O}\mathsf{tally}$, receives the result of the election and must return its guess, which is the output of the experiment.

This definition also considers that the channels between the voters and the ballot box manager is private and, indeed, without the assumption that such private channels are available, achieving receipt-freeness in a verifiable election is impossible [Che+10].

Definition 16 (Receipt-Freeness)**.** *A voting system $\mathcal{V}$ has receipt-freeness if there exists algorithms* $\mathsf{SimSetup}$ *and* $\mathsf{SimProve}$ *such that no* PPT *adversary $\mathcal{A}$ can distinguish between games $\mathsf{Exp}^{\mathsf{srf},0}_{\mathcal{A},\mathcal{V}}(\lambda)$ and $\mathsf{Exp}^{\mathsf{srf},1}_{\mathcal{A},\mathcal{V}}(\lambda)$ defined by the oracles in Figure 2.7, that is for any efficient algorithm $\mathcal{A}$:*

$$\left| Pr\left[\mathsf{Exp}^{\mathsf{srf},0}_{\mathcal{A},\mathcal{V}}(\lambda) = 1\right] - Pr\left[\mathsf{Exp}^{\mathsf{srf},1}_{\mathcal{A},\mathcal{V}}(\lambda) = 1\right] \right|$$

is negligible in λ.

In this game, parameterized by a bit β, the adversary has access to the following oracles:

- $\mathcal{O}\mathsf{init}$: initializes the voting system. It generates the public and privates keys of election and returns the public key to the adversary. When $\beta = 1$, a simulated setup may be performed, depending on the computational model, which will offer some trapdoor information

that may be needed to produced a simulated tally correctness proof for instance. Eventually, two empty ballot boxes are created: the real ballot box BB_0 that will be tallied and the fake ballot box BB_1. Both boxes will be populated during the game, but the adversary will only see $\mathsf{Publish}(\mathsf{BB}_\beta)$.

- $\mathcal{O}\mathsf{receiptLR}(\mathsf{b}_0, \mathsf{b}_1)$: Lets the adversary cast ballots b_0 in the real ballot box BB_0 and b_1 in the fake ballot box BB_1, as long as both ballots are valid and have the same trace. This oracle is the central one for receipt-freeness which deviates from BPRIV.

- $\mathcal{O}\mathsf{board}$: Returns $\mathsf{Publish}(\mathsf{BB}_\beta)$, which represents its view of the public bulletin board.

- $\mathcal{O}\mathsf{tally}$: Returns the result of the election based on the ballots on BB_0, as well as a proof of correctness of the tally. If $\beta = 1$, this proof is simulated w.r.t. the content available from BB_1.

Compared to Chaidos et al.'s definition, we removed the $\mathcal{O}\mathsf{corrupt}$ oracle, as we simply assume that all the voters are under adversarial control. We also omitted their $\mathcal{O}\mathsf{cast}$ and $\mathcal{O}\mathsf{voteLR}$ oracles, because $\mathcal{O}\mathsf{receiptLR}$ subsumes them.

Several observations can be made about this game. First, and as expected, it can be seen that the ballot box manager is considered to behave honestly. A dishonest ballot box manager could simply replace $\mathsf{ProcessBallot}$ and $\mathsf{Publish}$ with the identity function, which would make the game trivial to win, independently of any cryptographic operation. The Tally operation is also performed honestly: a dishonest truste could decrypt all the ballots individually, which would again make the game trivial to win. In practice, this assumption can be mitigated by using a distributed decryption process, which is always possible using MPC but can typically be done more efficiently.

Secondly, we modified the game in a non-trivial way. The related constraint that $\mathsf{TraceBallot}(\mathsf{b}_0) = \mathsf{TraceBallot}(\mathsf{b}_1)$ is actually missing from Chaidos' definition[Cha+16], and we noticed that it makes their game trivial to win in most natural case, including with their own protocol: an adversary could simply submit two ballots that have different traces (or are signed with different keys in the wording of their paper), and immediately identify which bulletin board he sees.[9]

[9] Of course, this is easily fixable with the Valid algorithm that could check the ballot with respect to the id of the voter since the "trace" of the voter is fixed at the registration.

We insist that this definition does not consider forced-abstention attacks, so that an adversary can still instruct a voter not to vote, even if the voting protocol is receipt free. In many protocols, the identity of a voter is tied to their ballot on the public bulletin board for an eligibility reason, which makes such forced-abstention attacks impossible to evade.

Finally, this game prompts for the introduction of an extra correctness requirement on the definition of Vote and $\mathsf{TraceBallot}$, in order to make sure that it is possible to compute ballots that encode different votes and have the same tag.

Definition 17 (Ballot traceability for receipt freeness)**.** *For every public key* pk *in the range of* Setup*, the ballots produced for every pair of voting choices* $(\mathsf{v}_0, \mathsf{v}_1)$ *with the same auxiliary data trace to the same tag. That is, for* $\mathsf{b}_0, aux \leftarrow\!\$\, \mathsf{Vote}(id, \mathsf{v}_0)$*,* $\mathsf{b}_1 \leftarrow\!\$\, \mathsf{Vote}(id, \mathsf{v}_1, \mathsf{aux})$*, we have* $\mathsf{TraceBallot}(\mathsf{b}_0) = \mathsf{TraceBallot}(b_1)$*.*

Without this extra constraint, we could imagine a $\mathsf{TraceBallot}$ algorithm which returns a tag depending on the vote inside the ballot. For example if $\mathsf{TraceBallot}(\mathsf{b}) = \mathsf{b}$ as we discussed earlier, then $\mathcal{O}\mathsf{receipt}(\mathsf{b}_0, \mathsf{b}_1)$ does nothing except if the two ballots are identical and the adversary can never win the receipt-freeness game, while in practice the voting scheme does not satisfy our intuitive meaning of receipt-freeness. It is thus natural to require that a tag returned by $\mathsf{TraceBallot}$ can be reached with any possible voting choice, or in other words, even if an adversary is waiting for a chosen tag, the voter can still cast a vote for any choice.

Verifiability Intuitively, a protocol is verifiable if, by performing a determined set of verifications, the participants have the guarantee about the integrity of the election. Verifiability is typically split into two different notions: individual verifiability, which convinces a particular voter that their vote is correctly counted in the tally and universal verifiability, which convinces everyone that the result of the election is correctly computed. While individual verifiability typically englobes cast-as-intended verifiability, or in other words offers verifiability against a malicious voting device, we consider first that this voting device is honest. Chapter 5 will propose another definition of individual verifiability where voters can verify that their vote is correctly counted even in the presence of a malicious voting device.

In this work, we use a definition inspired from the notion of individual verifiability with careful voters from Cortier et al. [CL18]. Besides some small syntactic modifications in order to be compatible with the syntax of the voting protocol of Section 2.3.3, we also give the secret key to the

verifiability adversary in order to have stronger guarantees as we do not want that the possession of the secret key sk eases fooling voters.

In this game, the adversary can instruct the voters to cast ballot with a given vote. Then, the adversary outputs a bulletin board of its choice. If every voter is convinced that their ballot in included in the bulletin board (using the $\mathsf{VerifyVote}$ algorithm) and if the result of the election when running an honest execution of the tally algorithm is different from the result obtained by an aggregation of the votes given to the voters, then the adversary wins the game. This definition effectively captures individual verifiability.

Definition 18 (Individual verifiability)**.** *A voting system $\mathcal{V}$ is individually verifiable if for all PPT $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the following is negligible in λ:*

$$\mathsf{Adv}^{\mathrm{verif}}_{\mathcal{A},\mathcal{V}}(\lambda) = \Pr\left(\mathsf{Exp}^{\mathrm{verif}}_{\mathcal{A},\mathcal{V}}(\lambda) = 1\right) \leq \mu(\lambda)$$

where $\mathsf{Exp}^{\mathrm{verif}}_{\mathcal{A},\mathcal{V}}(\lambda)$ is defined in Figure 2.8.

Ideally, we would want protection against a malicious trustee who does not execute the Tally algorithm correctly. In this regard, this work does not bring any novelty since we use a generic tally algorithm. Hence we assume that the tally provides a proof that it correctly computes the result function on the content of the ballot. This can be done in a generic way, or with more efficient algorithms for classical tally procedures (such as homomorphic tallying or mixnet based tallying).

$\mathsf{Exp}^{\mathrm{verif}}_{\mathcal{A},\mathcal{V}}(\lambda)$

$\mathsf{pk}, \mathsf{sk} \leftarrow \mathsf{Setup}(1^\lambda)$;
$\mathsf{U} \leftarrow \{\}$;
$\mathsf{st} \leftarrow \mathcal{A}_1^{\mathcal{O}\mathsf{register}}(\mathsf{pk}, \mathsf{sk})$;
$\mathsf{Voted} \leftarrow \{\}$;
$\mathsf{BB} \leftarrow \mathcal{A}_2^{\mathcal{O}\mathsf{vote}}(\mathsf{st})$;
if $\forall \mathsf{id}, (\mathsf{id}, \star, \mathsf{t}) \in \mathsf{Voted} \implies \mathsf{VerifyVote}(\mathsf{BB}, \mathsf{t}) = 1$
then
  $r, \Pi \leftarrow \mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$;
  if $r \neq \perp$ and $\forall V_C$ s.t. $r \neq \rho(V_H \cup V_C)$ **then return** 1;
return 0;

where $V_H = \{(\mathsf{v_{id}})\}$ for each $(\mathsf{id}, \star, \mathsf{v_{id}}) \in \mathsf{Voted}$

$\mathcal{O}\mathsf{register}(\mathsf{id}, \mathsf{upk_{id}}, \mathsf{usk_{id}})$

$\mathsf{U} \leftarrow \mathsf{U} \cup \{(\mathsf{id}, \mathsf{upk_{id}}, \mathsf{usk_{id}})\}$
return $\mathsf{upk_{id}}, \mathsf{usk_{id}}$

$\mathcal{O}\mathsf{vote}(\mathsf{id}, \mathsf{v})$

if $(\mathsf{id}, \star, \star) \in \mathsf{U}$
then
  $\mathsf{b}, \mathsf{aux} \leftarrow \mathsf{Vote}(\mathsf{id}, \mathsf{usk_{id}}, \mathsf{v})$;
  $\mathsf{t} \leftarrow \mathsf{TraceBallot}(\mathsf{b})$;
  $\mathsf{Voted} \leftarrow \mathsf{Voted}' \cup \{(\mathsf{id}, \mathsf{v}, \mathsf{t})\}$;
return b

where Voted' is obtained
by removing all instances of $(\mathsf{id}, \star, \star)$
from Voted

Figure 2.8.: Oracles used in the $\mathsf{Exp}^{\mathrm{verif}}_{\mathcal{A},\mathcal{V}}(\lambda)$ experiment. The adversary can first register as many voters as it want through $\mathcal{O}$register calls, and then can make these voters vote through $\mathcal{O}$vote calls. The adversary finally outputs a bulletin board and wins the game if the bulletin board is tallied into a result different from an aggregation of the votes inputed in the $\mathcal{O}$vote calls by the result function ρ.

Part I.

Receipt-Freeness

3. Traceable Receipt-Free Encryption

In the preliminary chapter, we precisely defined the receipt-freeness of a voting protocol which captures that voters should be unable to demonstrate to a third party how they voted. In this chapter, we focus on the blueprint for single-pass voting protocol of Section 2.3 and explore how we can generically reach this notion. Especially, it is natural to search which encryption primitives are exactly needed to attain receipt-freeness, regardless of the tallying process or individual verifiability mechanism (that we will explore in Chapter 5). This generic treatment allows for a more modular approach to build voting protocols.

For the weaker notion of ballot privacy [Ber+15], given in the preliminaries, this question has been answered: an encryption scheme which is indistinguishable against chosen-ciphertext attacks (IND-CCA security) can be used to obtain a private ballot submission, a requirement that can be relaxed to non-malleability against a chosen-plaintext attacks (NM-CPA security) when the tally takes place in a single decryption round [Wik08; BPW12]. These works highlight the need of non-malleable ballots, especially to avoid replay-attacks. That is, if an adversary can select a ballot on the bulletin board and craft their own ballot non-independently from the content of the original (by rerandomizing it for example), then they could learn in some cases some information about the vote contained in this ballot after the tally.

CCA security is however problematic in the context of receipt-free ballot submission, since we need to re-randomize encrypted votes, so that the voter cannot explain the content of a ballot anymore. We thus need to allow some malleability and prevent replay attacks in an other way. Furthermore, in a receipt-freeness game, the adversary can select the ballots which will be rerandomized. Hence, we need a security definition where the adversary sends two ciphertexts as a challenge and receives a rerandomization of one of them (instead of sending two messages and receiving an encryption of one of them in a more classical CCA-like definition).

In this chapter, we propose a new primitive, called Traceable Receipt-free Encryption (TREnc). This primitive is equipped with a new CCA-

like notion of security, *traceable* CCA security (TCCA). In this notion, adversarially-chosen ciphertexts are submitted in the challenge phase, and then rerandomized to produce the challenge ciphertext. Moreover, to allow some non-malleability, we recognize derivatives of the challenge ciphertext thanks to a non-malleable public *trace* which is present in any ciphertext. To avoid trivial attacks, both ciphertexts given in the challenge phase must trace to each other, i.e., they must have the same trace.

Furthermore, we equip the TREnc primitive with an additional security property, traceability. This property, related to unforgeability, ensures that if an adversary sees a ciphertext, it cannot produce another ciphertext with the same trace that decrypts to a different message, and this even if it knows the secret key of the scheme. Indeed, in our voting application, we desire that even if the rerandomizing entity colludes with the talliers, it remains unable modify the content of a voter's ballot, for the sake of recorded-as-cast verifiability.

From this new primitive, we will then show how we can build a generic single-pass voting scheme inspired by the blueprint described in Chapter 2. This resulting voting scheme provably satisfies a notion of receipt-freeness according to Definition 16.

Structure of the chapter We start the chapter with a brief literature review of receipt-freeness through the use of a rerandomising server and CCA-like definitions with controlled malleability in Section 3.1. Then in Section 3.2, we detail our Traceable Receipt-Free Encryption primitive, along with all the companion security properties discussed above. Section 3.3 explores the link between this new primitive and more classical notions such as tag-based encryption and a security notion known as Replayable CCA security. Eventually, Section 3.4 describes our generic single-pass protocol built from TREncs and demonstrates its receipt-freeness.

3.1. Related works

3.1.1. Receipt-Freeness through rerandomization

Since the seminal work of Benaloh [BT94], numerous protocols explored mechanisms that would guarantee that the random coins used by a voter are insufficient to explain the content of their ballot that is posted on the bulletin board for the needs of verification. In a first line of works [BT94; SK95; HS00], every possible voting choice is encrypted, the resulting ciphertexts are rerandomized and shuffled by the election authorities

and made available to the voter. Furthermore, the permutations applied during the shuffle are also transmitted to the voter using secure channels. The voter then picks the ciphertext encoding their choice, and submits it for display on the bulletin board. Such a protocol guarantees that the voter does not know the randomness used encrypting a ballot, and the protocol is designed in such a way that the voter is unable to prove which permutation was received, typically using designated-verifier zero-knowledge proofs. Such interactive protocols are however quite demanding in terms of resources, as they require to encrypt a number of ciphertext proportional to the number of voting options, and a communication bandwidth for each voter that it is proportional to the number of authorities.

More recently, Blazy et al. [Bla+11] proposed a simpler voting flow supporting receipt-freeness based on signatures on randomizable ciphertexts (SRC): voters encrypt their votes and sign the resulting ciphertexts, which are then transmitted to a re-encryption authority that re-randomizes the ciphertexts, adapts the signatures accordingly, and posts the result on the bulletin board. The voters remain able to verify that a vote with a valid signature are posted on the board on their behalf, but is unable to explain the vote content thanks to the re-randomization step. Furthermore the SRC guarantees that the contents of encrypted ballots cannot be modified during the re-encryption process. This approach was further refined by Chaidos et al. [Cha+16], who also proposed a simple game-based definition of receipt-freeness, which we slightly modified as explained in the previous chapter, and more efficient SRCs keep being proposed [Cha+16; BF20].

This approach makes the ballot submission process asymptotically optimal for the voter, in the sense of Cramer et al. [CGS97]: the protocol complexity for the voter becomes logarithmic in the number of voting options and independent of the number of election authorities, contrary to a dependency that is at least linear in both these factors when the approaches of [BT94; SK95; HS00] are used.

3.1.2. CCA security with controlled malleability

The exploration of CCA-like security notions that would support some form of controlled malleability has been a fertile research area, which resulted in the definition of the notions of replayable-CCA (RCCA) security[CKN03], homomorphic-CCA (HCCA) security [PR08], and controlled-malleable CCA (CM-CCA) security [Cha+12] for instance. As far as we know, all these works rely on the same CCA blueprint, in which an adversary submits one or more *messages* to a challenger, who answers

either with a honest encryption of the messages or with something else, and the adversary must decide what was received with the help of a decryption oracle that decrypts any ciphertext that is not "recognizably" related to the challenge ciphertexts. The same holds in any other encryption primitives with CCA-like security with enhanced decryption capabilities. While they give more flexible ways to decrypt ciphertexts (based on identities, attributes and so on [Sha84; HL02; SW05; BSW11]), the challenge ciphertext is computed when the adversary sends a chosen *message*.

This blueprint is however inadequate when turning to encryption schemes that would support the design of protocols offering receipt-freeness: in such a setting, we need to consider an adversary who sends chosen *ciphertexts*, that may not be honestly computed as a random encryption of a plaintext vote.

3.2. Traceable Receipt-Free Encryption

We propose a new public key encryption primitive and associated security notions that would support the receipt-free submission of votes in a protocol. As a first task, we identify the fundamental ingredients we wish to capture.

An encryption scheme. We expect voters to submit their vote in an encrypted form, in order to guarantee the privacy of the votes. Specifically, we desire a BPRIV-like notion against the rerandomizing authority.

Receipt-free encryption. Voters willing to sell their votes may choose to submit arbitrary encrypted votes, which may be in the range of honestly produced ciphertexts but sampled according to a different distribution, or even just a sequence of bits that would not be within the range of the encryption mechanism. By deviating from the normal encryption process, the voters hope to obtain receipts that could be used to demonstrate their vote intent to a third party.

If the encrypted vote that is tallied is produced by the voter only, then the voter will always have a receipt: the random coins used to encrypt the ballot. In order to avoid this, we rely on the existence of a semi-trusted authority: that authority will be trusted to prevent a dishonest voter from obtaining a receipt for his vote, but will not be trusted for the correctness of the election result, and will not be trusted for the privacy of votes encrypted by honest voters as already explained.

Concretely, in order to achieve receipt-freeness, this semi-trusted authority tests the validity of a voter submitted ciphertext (without the need of any secret key) and rerandomizes every valid ciphertext

before posting it on a public bulletin board.

Traceable Receipt-Free Encryption. In order to make it possible for a voter to check that his ballot has not been unduly modified by this semi-trusted rerandomizing authority, it must be possible to extract a *trace* from any valid ciphertext. An honest rerandomization process would keep the trace unchanged, hence making ciphertexts *traceable*, while no corrupted authority should be able to modify a ciphertext in such a way that it would decrypt to a different vote while keeping the trace unchanged.

Furthermore, we need to make sure that this trace cannot serve as a receipt for the vote. In order to make sure that it is the case, we split the encryption process in two steps, that guarantee that any trace can be associated to any possible vote intent. Concretely, an encryption starts with the generation of a secret *link key*,[10] which is then used, together with the encryption public key, to encrypt any possible vote. This guarantees that, even if a voter leaks the link key associated to a ballot as a receipt, the ballot could still encrypt any vote.[11]

3.2.1. Syntax

With all these ingredients in mind, we define a Traceable Receipt-Free Encryption scheme, or TREnc.

Definition 19 (Traceable Receipt-Free Encryption)**.** *A* Traceable Receipt-Free Encryption *scheme (TREnc) is a public key encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *that is augmented with a 5-tuple of algorithms* $(\mathsf{LGen}, \mathsf{LEnc}, \mathsf{Trace}, \mathsf{Rand}, \mathsf{Ver})$*:*

- $\mathsf{LGen}(\mathsf{pk}; r)$*: The link generation algorithm takes as input a public encryption key* pk *in the range of* Gen *and randomness* r*, and outputs an ephemeral secret link key* lk*.*

- $\mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, m; r)$*: The linked encryption algorithm takes as input a pair of public/link keys* $(\mathsf{pk}, \mathsf{lk})$*, a message* m *and randomness* r *and outputs a ciphertext.*

- $\mathsf{Trace}(\mathsf{pk}, c)$ *: The tracing algorithm takes as input a public key* pk*, a ciphertext* c *and outputs a trace* t*. We call* t *the trace of* c*.*

[10] Jumping ahead, this link key will be the aux value returned by the Vote algorithm in our generic construction.

[11] Of course, this also means that, if a corrupted rerandomizing authority obtains a voter's secret link key (e.g., by corrupting the voter's voting client), then it might be able to produce a ciphertext that encrypts a different vote intent but would still trace to the original voter trace.

- $\mathsf{Rand}(\mathsf{pk}, c; r)$*: The randomization algorithm takes as input a public key* pk*, a ciphertext* c *and randomness* r *and outputs another ciphertext.*

- $\mathsf{Ver}(\mathsf{pk}, c)$*: The verification algorithm takes as input a public key* pk*, a ciphertext* c *and outputs* 1 *if the ciphertext is valid,* 0 *otherwise.*

In many cases, we will omit the randomness r *from our notations. It is then assumed that it is selected uniformly at random.*

We require several correctness properties from the additional algorithms of a TREnc. The first requires that encrypting a message m by picking a link key lk using LGen and computing $\mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, m)$ produces a ciphertext that is identically distributed to a fresh encryption of m using Enc. The second requires that the Trace of a ciphertext does not depend on the message that is encrypted. The third requires that randomizing a ciphertext does not change either the corresponding plaintext or the corresponding trace. The last requires that every honestly computed ciphertext passes the verification algorithm.

Definition 20 (TREnc correctness)**.** *We require that a TREnc scheme satisfies the following correctness requirements.*

Encryption compatibility *For every* pk *in the range of* Gen *and message* m*, the distributions of* $\mathsf{Enc}(\mathsf{pk}, m)$ *and* $\mathsf{LEnc}(\mathsf{pk}, \mathsf{LGen}(\mathsf{pk}), m)$ *are identical.*

Link traceability *For every* pk *in the range of* Gen*, every* lk *in the range of* $\mathsf{LGen}(\mathsf{pk})$*, the encryptions of every pair of messages* (m_0, m_1) *trace to the same trace, that is, the following always holds:*

$$\mathsf{Trace}(\mathsf{pk}, \mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, m_0)) = \mathsf{Trace}(\mathsf{pk}, \mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, m_1)).$$

Publicly Traceable Randomization *For every* pk *in the range of* Gen*, every message* m *and every* c *in the range of* $\mathsf{Enc}(\mathsf{pk}, m)$*,*

$$\mathsf{Dec}(\mathsf{sk}, c) = \mathsf{Dec}(\mathsf{sk}, \mathsf{Rand}(\mathsf{pk}, c))$$

$$\mathsf{Trace}(\mathsf{pk}, c) = \mathsf{Trace}(\mathsf{pk}, \mathsf{Rand}(\mathsf{pk}, c)).$$

Honest verifiability *For every* pk *in the range of* Gen*, every message* m*, and every* c *in the range of* $\mathsf{Enc}(\mathsf{pk}, m)$*, it holds that* $\mathsf{Ver}(\mathsf{pk}, c) = 1$ *and* $\mathsf{Ver}(\mathsf{pk}, \mathsf{Rand}(\mathsf{pk}, c)) = 1$*.*

3.2.2. Security definitions

Verifiability We require several security properties from a TREnc. Our first property is fairly standard: a TREnc is verifiable if the Ver algorithm guarantees that a ciphertext is within the range of Enc. In other words, the ciphertext can be explained by some message m, some link key lk, and some coins, even if they are not easily computable.

Definition 21 (Verifiability)**.** *A TREnc is* verifiable *if for every* PPT *adversary, the following probability:*

$$\Pr\left(\begin{array}{c}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda); c \leftarrow \mathcal{A}(\mathsf{pk},\mathsf{sk}) : \\ \mathsf{Ver}(\mathsf{pk},c) = 1 \text{ and } c \text{ not in the range of } \mathsf{Enc}(\mathsf{pk},\cdot)\end{array}\right)$$

is negligible in λ*.*

TCCA **security.** We now turn to our central security definition, *security against traceable chosen ciphertexts attacks*, or TCCA security, which differs from all existing CCA-like notions by letting the adversary submit pairs of ciphertexts instead of pairs of messages, reflecting that we need security in front of adversarially chosen ciphertexts. In the TCCA security game (Fig. 3.1), the adversary receives the public key and has access to a decryption oracle, as usual. It then submits a pair of ciphertexts that must be valid and have identical traces. One of the ciphertexts is randomized and returned to the adversary, who must decide which one it is. After receiving this challenge ciphertext, the adversary can still query the decryption oracle, but only on ciphertexts that have a trace different from the challenge ciphertext. So, the challenger must faithfully decrypt pre-challenge ciphertexts that have the same trace as the challenge ciphertext. Looking ahead, this decryption capability offers an easy but necessary means of simulating the result of an election when proving receipt-freeness, as we need to extract votes without the secret key.

TCCA security guarantees that, if a voter submits a ciphertext that is randomized before it is posted on a public bulletin board, then the resulting ciphertext becomes indistinguishable from any other ciphertext that would have the same trace, and we know from the link traceability that the encryption of any vote could have that trace. This essentially guarantees the absence of a vote receipt.

Definition 22 (TCCA)**.** *A TREnc is* TCCA *secure if for every* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *the experiment* $\mathsf{Exp}^{\mathrm{tcca}}_{\mathcal{A}}(\lambda)$ *defined in Figure 3.1 (left) returns* 1 *with a probability negligibly close in* λ *to* $\frac{1}{2}$*.*

$\mathsf{Exp}^{\mathrm{tcca}}_{\mathcal{A}}(\lambda)$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow\$\, \mathsf{Gen}(1^\lambda)$
$(c_0, c_1, \mathsf{st}) \leftarrow\$\, \mathcal{A}_1^{\mathsf{Dec}(\cdot)}(\mathsf{pk})$
$b \leftarrow\$\, \{0,1\}$
if $\mathsf{Trace}(\mathsf{pk}, c_0) \neq \mathsf{Trace}(\mathsf{pk}, c_1)$ or
$\mathsf{Ver}(\mathsf{pk}, c_0) = 0$ or $\mathsf{Ver}(\mathsf{pk}, c_1) = 0$
then return 0
$c^\star \leftarrow\$\, \mathsf{Rand}(\mathsf{pk}, c_b)$
$b' \leftarrow\$\, \mathcal{A}_2^{\mathsf{Dec}^\star(\cdot)}(c^\star, \mathsf{st})$
return $b' = b$

$\mathsf{Exp}^{\mathrm{trace}}_{\mathcal{A}}(\lambda)$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow\$\, \mathsf{Gen}(1^\lambda)$
$(m, \mathsf{st}) \leftarrow\$\, \mathcal{A}_1(\mathsf{pk}, \mathsf{sk})$
$c \leftarrow\$\, \mathsf{Enc}(\mathsf{pk}, m)$
$c^\star \leftarrow\$\, \mathcal{A}_2(c, \mathsf{st})$
if $\mathsf{Trace}(\mathsf{pk}, c) = \mathsf{Trace}(\mathsf{pk}, c^\star)$ and
$\mathsf{Ver}(\mathsf{pk}, c^\star) = 1$ and $\mathsf{Dec}(\mathsf{sk}, c^\star) \neq m$
then return 1
else return 0

Figure 3.1.: TCCA and trace experiments. In the TCCA experiment, $\mathcal{A}_2$ has access to a decryption oracle $\mathsf{Dec}^\star(\cdot)$ which, on input c, returns $\mathsf{Dec}(c)$ if $\mathsf{Trace}(\mathsf{pk}, c) \neq \mathsf{Trace}(\mathsf{pk}, c^\star)$ and $\perp$ otherwise.

We stress that in the challenge query the adversary may know the random coins underlying c_0 and c_1 and may have drawn them from a specific secret distribution. The randomization leading to the challenge ciphertext $c^\star$ should thus erase any subliminal information binding $c^\star$ to the message in c_b. This definition introduces some technical difficulty when it comes to proving the TCCA security as it becomes harder to program the public key to ease the transition toward a game where we are able to inject an independent message in the plaintext in an undetectable way. Indeed, we have no clue at the setup time about the distribution of (c_0, c_1) and their common trace while the emulation of $\mathsf{Rand}(\mathsf{pk}, c_b)$ must preserve it without even knowing the underlying link keys.

TCCA security is reminiscent of the notion of weak CCA security of tag-based encryption schemes proposed by MacKenzie [MRY04] and publicly detectable replayable-CCA (pd-RCCA) security proposed by Canetti et al. [CKN03]. Beyond the dissimilarity of plaintext indistinguishability from encryption in contrast with the ciphertext indistinguishability from rerandomization, the differences between these notions are further discussed in Section 3.3.

Traceability and Strong Randomization. While TCCA security relates to a model in which the voting client may be corrupted but the rerandomization server is honest, we now focus on two central properties that are important when the voting client is honest and the rerandomization server might be corrupted.

The traceability property guarantees to the sender of an honestly encrypted message that no efficient adversary would be able to produce

another ciphertext that traces to the same trace and would decrypt to a different message, even if the adversary knows the secret decryption key. So, even if a TREnc offers some form of ciphertext malleability, its traceability implies the non-malleability of the plaintexts. This is an important feature for the verifiability of a voting system: as long as the link key used to encrypt a vote remains secret, and the voter submits a single ciphertext encrypted with that link key, the voter is guaranteed that any ciphertext that would trace to the original ciphertext encrypts the original vote. (But, of course, using the link key, it remains possible to produce ciphertexts with the same trace that would decrypt to any vote.)

Definition 23 (Traceability)**.** *A TREnc is* traceable *if for every* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, the experiment* $\mathsf{Exp}^{\mathrm{trace}}_{\mathcal{A}}(\lambda)$ *defined in Figure 3.1 (right) returns* 1 *with a probability negligible in* λ*.*

The second property, strong randomization, requires that the output of the Rand algorithm applied to any valid ciphertext is distributed just as a random encryption of the same message with the same link key.

Definition 24 (Strong Randomization)**.** *A TREnc is* strongly randomizable *if for every* $c \in \mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, m)$ *with* pk *in the range of* Gen *and* lk *in the range of* $\mathsf{LGen}(\mathsf{pk})$*, the following computational indistinguishability relation holds:*

$$\mathsf{Rand}(\mathsf{pk}, c) \approx_c \mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, m)$$

So far, nothing guaranteed that the encryption algorithm actually hides the message. An extreme case could define Enc as the identity function, Trace as mapping to a single constant trace, and Rand actually performing the encryption work, and this could still offer a TCCA secure scheme. Requiring strong randomization together with traceability and TCCA security guarantees that Enc actually hides messages. CPA security comes easily: when the CPA adversary sends (m_0, m_1) to the TCCA adversary, the TCCA adversary can encrypt the 2 messages using a single random link key and send them to the TCCA challenger, which will return a randomization of one of them. Strong randomization guarantees that this is distributed exactly like an encryption of one of the two messages, and we can send the result to the CPA adversary, who will then offer the answer expected for the TCCA game. In the next section, we will show a stronger implication to a variant of weak CCA security and to RCCA security.

$\mathsf{Exp}^{\mathrm{m-tcca}}_{\mathcal{A}}(\lambda)$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow\$\ \mathsf{Gen}(1^\lambda)$
$b \leftarrow\$\ \{0,1\}$
$L \leftarrow \{\}$
$b' \leftarrow\$\ \mathcal{A}^{\mathsf{Dec}(\cdot),\mathsf{Chall}(\cdot)}(\mathsf{pk})$
return $b' = b$

$\mathsf{Chall}(c_0, c_1)$

if $\mathsf{Trace}(c_0) \in L$ or
$\mathsf{Trace}(\mathsf{pk}, c_0) \neq \mathsf{Trace}(\mathsf{pk}, c_1)$or
$\mathsf{Ver}(\mathsf{pk}, c_0) = 0$ or $\mathsf{Ver}(\mathsf{pk}, c_1) = 0$:
  return $\perp$
else $L \leftarrow L \cup \{\mathsf{Trace}(c_b)\}$
return $\mathsf{Rand}(\mathsf{pk}, c_b)$

$\mathsf{Dec}(c)$

if $\mathsf{Trace}(c) \in L$: **return** $\perp$
else return $\mathsf{Dec}(\mathsf{sk}, c)$

Figure 3.2.: mTCCA experiment.

3.2.3. Additional definitions

A TREnc equipped with the TCCA, traceability and strong randomizability is already sufficient to generically build receipt-free protocols, as we will show later. However, it is worth mentioning the following extra properties that will be useful in this thesis. The first property, mTCCA, is a natural extension of the TCCA security to a multi-challenge setting. The second property is an extension of TREncs that essentially provides a trapdoor algorithm to extract link keys of a ciphertext, and is especially useful to build TREncs from more classical primitives.

Multi-challenge TCCA security We introduce here the natural notion of mTCCA security, in which the adversary can make a polynomial number of challenge queries. It can be observed that, in this definition, the adversary can always request decryption on any ciphertext as long as its trace was not already involved in a challenge query. That is, decryption is allowed for any trace even if they later occur in a challenge query. This property will be crucial in the original definition to allow reaching receipt-freeness when a TREnc is used in a voting system because the decryption oracle always make it possible to compute (or simulate) the result of the election. Our new experiment respects this fundamental feature.

Definition 25 (mTCCA)**.** *A TREnc is secure against traceable chosen-ciphertext attacks in the multi-challenge setting if for every efficient adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *the experiment* $\mathsf{Exp}^{\mathrm{m-tcca}}_{\mathcal{A}}(\lambda)$ *defined in Figure 3.2 (left) returns* 1 *with a probability negligibly close in* λ *to* $\frac{1}{2}$*.*

Obviously, the mTCCA notion implies the TCCA notion. Conversely, we state that the mTCCA notion is implied by the the TCCA notion.

Lemma 1. *If a TREnc is TCCA, then it is* mTCCA. *More precisely, for any adversary $\mathcal{A}$ with advantage $\varepsilon_{\text{TCCA}}$ in the* mTCCA *game and making q challenge queries, we have $\varepsilon_{\text{mTCCA}} \leq q \cdot \varepsilon_{\text{TCCA}}$, where* TCCA *is the maximal success probability of any adversary in the TCCA experiment.*

Proof. We build a sequence of $q + 1$ hybrid games by modifying the rerandomization requests of the mTCCA game in the following way. In the i-th hybrid, the game rerandomizes c_0 for the first i requests and c_1 for the remaining requests. It is clear that the first and last games correspond to the two games that the mTCCA adversary tries to distinguish. We define S_i as the event that the adversary wins in the i-th hybrid.

We observe that an adversary $\mathcal{A}$ distinguishing the i-th hybrid from the $i + 1$-th hybrid can be used to build a TCCA adversary $\mathcal{B}$ with identical advantage. $\mathcal{B}$ simulates $\mathcal{A}$ and answers its first $i - 1$ requests with a rerandomization of c_0. Then, for the i-th request, it forwards the requests to the TCCA challenger and returns the output to $\mathcal{A}$. Finally, it answers the remaining requests by randomizing the c_1 ciphertext, and stores the trace of this ciphertext in a list L. Regarding the decryption request, $\mathcal{B}$ forwards the request to the TCCA challengers. If one of the post-challenge requests has a trace already appearing in L, $\mathcal{B}$ returns 0.

It is clear that if the challenge bit of the TCCA game is 0, then $\mathcal{B}$ simulates the $i - th$ hybrid. Otherwise, it simulates the $i + 1$-th hybrid. Hence $Pr[S_{i+1}] - Pr[S_i] \leq \varepsilon_{TCCA}$ for each $i \in [q-1]$, where ε_{TCCA} is the advantage of an efficient adversary playing the TCCA games. Finally,

$$|Pr[S_q] - Pr[S_0]| = |\sum_{i=0}^{q-1} Pr[S_{i+1} - S_i]| \leq q\varepsilon,$$

which concludes the proof that TCCA implies mTCCA. □

Extractable TREncs Essentially, an extractable TREnc enables the generation of encryption keys with a trapdoor using a TrapGen algorithm. Using that trapdoor, it becomes possible to extract, from any ciphertext, a link key that makes it possible to produce new ciphertexts with the same trace as the original one. This in turn implies the possibility of breaking the traceability of the scheme.

While this link extraction can be seen as an extra cost compared to a TREnc and something that we would want to avoid when building a TREnc for voting purposes, it is actually useful to generically design a TREnc. As we said the TCCA property is very reminiscent of the more classical wCCA property of a tag-based encryption scheme. In

the next section, it will be shown that for a TREnc equipped with this extraction property, the TCCA security is implied by a close variant of wCCA adapted to TREncs. Then, we will build in Chapter 4 a generic TREnc construction from this implication and we will thus need to be able to extract link keys.

Definition 26 (Extractable TREnc)**.** *An extractable TREnc is a TREnc with three additional algorithms* $\mathsf{TrapGen}$*,* $\mathsf{TrapAdapt}$ *and* LExtr*:*

- $\mathsf{TrapGen}(\mathsf{pk})$*: The trapdoor generation algorithm takes as input the public key generated as* $\mathsf{pk}, \mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda)$ *and outputs a pair of public/trapdoor keys* $(\mathsf{pk}', \mathsf{tk})$*. We require the distribution of* $(\mathsf{pk}, \mathsf{sk})$ *and* $(\mathsf{pk}', \mathsf{sk})$ *to be identical.*

- $\mathsf{TrapAdapt}(\mathsf{tk}, \mathsf{pk}, c)$*: The trapdoor adaptation algorithm takes as input the trapdoor and a ciphertext* c *in the range of* $\mathsf{Enc}(\mathsf{pk}, \cdot)$ *and outputs another ciphertext* c' *in the range of* $\mathsf{Enc}(\mathsf{pk}', \cdot)$ *such that for any message* m *and link key* lk*,* $\mathsf{TrapAdapt}(\mathsf{tk}, \mathsf{Enc}(\mathsf{pk}, \mathsf{lk}, m))$ *and* $\mathsf{LEnc}(\mathsf{pk}', \mathsf{lk}, m)$ *have the same distribution.*

  Conversely, it can also take as input the trapdoor and any ciphertext c' *in the range of* $\mathsf{Enc}(\mathsf{pk}', \cdot)$ *and output another valid ciphertext* c *in the range of* $\mathsf{Enc}(\mathsf{pk}, \cdot)$ *such that for any message* m *and link key* lk*,* $\mathsf{TrapAdapt}(\mathsf{tk}, \mathsf{Enc}(\mathsf{pk}', \mathsf{lk}, m))$ *and* $\mathsf{Enc}(\mathsf{pk}, \mathsf{lk}, m)$ *have the same distribution.*

- $\mathsf{LExtr}(\mathsf{tk}, c)$*: The link extraction algorithm takes as input the trapdoor key and a ciphertext and returns a link key* lk *such that, if* c *is in the range of* $\mathsf{Enc}(\mathsf{pk}, \cdot)$ *with* pk *in the range of* Gen*, then* c *is in the range of* $\mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, \cdot)$*.*

This link extraction mechanism can be seen as a man-in-the-middle generation of a trapdoor. It would be more natural to directly generate the trapdoor as the public and secret keys, but such trapdoor generation is not powerful enough to obtain TCCA security from wCCA security. Indeed, we need to extract somehow the link key of the TCCA challenge ciphertexts, which would not be possible in the wCCA game. Here, our wCCA adversary can run the trapdoor algorithm and run as a man-in-the-middle.

This more restrictive notion of extraction is still realizable rather intuitively. We can imagine an additional use of an encryption scheme to encrypt the link key and a zero-knowledge proof that this link key is consistent with the trace of the ciphertext. The trapdoor would simply be the secret key of the encryption scheme, which we do not include in

$\mathsf{Exp}^{\mathrm{trenc-wcca}}_{\mathcal{A}}(\lambda)$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow\$\ \mathsf{Gen}(1^\lambda)$
$(m_0, m_1, \mathsf{lk}^\star, \mathsf{st}) \leftarrow\$\ \mathcal{A}_1^{\mathsf{Dec}(\mathsf{sk},\cdot,\cdot)}(\mathsf{pk})$
$b \leftarrow\$\ \{0,1\}$
$c^\star \leftarrow\$\ \mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}^\star, m_b)$
$b' \leftarrow\$\ \mathcal{A}_2^{\mathsf{Dec}^\star(\mathsf{sk},\cdot)}(c^\star, \mathsf{st})$
return $b = b'$

Figure 3.3.: Weakly CCA security experiment for TREncs. In the experiment, $\mathcal{A}_2$ has access to a decryption oracle $\mathsf{Dec}^\star$ which, on input c, returns $\mathsf{Dec}(\mathsf{sk}, c)$ if $\mathsf{Trace}(c) \neq \mathsf{Trace}(c^\star)$.

the secret key of the TREnc. We rely on this similar technique in our generic construction.

Finally, it is fairly natural to require that ciphertexts can only be consistent with one single link key, hence guaranteeing a unique link key extraction.

Definition 27 (Unique Extraction)**.** *An extractable TREnc has* unique extraction *if, for every* $(\mathsf{pk}, \mathsf{sk})$ *in the range of* $\mathsf{Gen}(1^\lambda)$*,* $(\mathsf{pk}', \mathsf{tk})$ *in the range of* $\mathsf{TrapGen}(\mathsf{pk})$ *and* lk *in the range of* $\mathsf{LGen}(\mathsf{pk})$*, we have that:*

- $\mathsf{LExtr}(c, \mathsf{tk}) = \mathsf{lk}$ *whenever* $c \in \mathsf{LEnc}(\mathsf{pk}', \mathsf{lk}, \cdot)$*;*
- $\mathsf{LExtr}(c_0, \mathsf{tk}) = \mathsf{LExtr}(c_1, \mathsf{tk})$ *whenever we have* $\mathsf{Trace}(\mathsf{pk}', c_0) = \mathsf{Trace}(\mathsf{pk}', c_1)$ *and* $c_0, c_1 \in \mathsf{Enc}(\mathsf{pk}', \cdot)$*.*

3.3. Relations with more classical security definitions

In this section, we look at more classical security definitions related to tag-based encryption schemes and to rerandomizable encryption and their relationship with our TCCA-security definition.

3.3.1. wCCA security of Tag-based encryption

TREncs share similarities with tag-based encryption schemes (See Section 2.2.8). Instead of using a unique tag for both the encryption and the decryption, we can see the link key and the Trace as a public/secret variant of TBE tag. In that regard, the weak CCA security of a tag-based encryption scheme (Figure 2.3) can be reformulated as in Figure 3.3.

This notion is appealing for our voting application since it allows for some unmalleability, while keeping a way to recognize the challenge ciphertexts. It can also be seen as a TCCA game where the challenge

is a more classical left-or-right encryption instead of a left-or-right re-randomization. Besides that, there are more central differences, which motivate the introduction of the TCCA security game:

1. The challenge ciphertexts of the wCCA security game are always honestly computed and, as such, wCCA security does not offer any guarantee in front of maliciously produced ciphertexts, as would be the case when a voter tries to obtain a receipt for a vote.

2. Contrary to wCCA security, we observed that TCCA security says nothing about the hiding property of the Enc algorithm, since the adversary must distinguish based on outputs of Rand. An extreme case could define Enc as the identity function, Trace as mapping to a single constant trace, and Rand actually performing the encryption work, and this could still offer a TCCA secure scheme. The confidentiality requirements on Enc are handled through the combination of the TCCA and strong randomization properties.

3. In the TCCA game, the adversary sends ciphertext challenges for which the challenger does not know the link key. Especially, the challenger does not have a direct way to produce a ciphertext with the same trace that encrypts another message if the TREnc is traceable. This is crucial to prove the TCCA security.

However, we can build a TCCA secure TREnc from a wCCA-secure TREnc if the TREnc is verifiable and strongly randomizable (which solve the first two points) and if we have access to the extraction algorithm of Definition 26 (which solves the third point).

Theorem 1. *If a TREnc scheme is extractable, verifiable, strongly randomizable, and* wCCA*-secure, then it is* TCCA *secure. More precisely, if the advantages of any PPT adversary at the strong randomization and* wCCA *experiment are respectively bounded by ε_{SR} and $\varepsilon_{\mathrm{wCCA}}$, then for any* PPT *adversary $\mathcal{A}$, we have $Pr[\mathsf{Exp}^{\mathrm{tcca}}_{\mathcal{A},T}(\lambda) = 1] \leq \frac{1}{2} + \varepsilon_{SR} + 2\varepsilon_{VER} + \varepsilon_{\mathrm{wCCA}}$.*

Proof. Let $\mathcal{A}$ be an adversary against the TCCA security of the TREnc, and $\varepsilon_{\mathrm{TCCA}}$ be its winning probability in the TCCA game as defined in Figure 3.1 of Definition 22. We give a succession of hybrid games, where the first is the real TCCA game. For each of those games, we denote S_i the event that the i-th game outputs 1. When we say that the challenger aborts, we mean that the experiment outputs a random bit.

Game 0: This is the original TCCA game. By definition of the TCCA game, $Pr[\mathsf{Exp}^{\mathrm{tcca}}_{\mathcal{A},T}(\lambda) = 1] = Pr[S_0]$.

Game 1: After generating the keys $(\mathsf{pk}, \mathsf{sk})$ through an execution of Gen, we now use $\mathsf{TrapGen}$ from Definition 26 and obtain a pair $(\mathsf{pk}', \mathsf{tk})$ that we use in the remainder of the game.

Since the public key has the same distribution, $Pr[S_1] = Pr[S_0]$.

Game 2: This game is as $\mathsf{Game}_1(\lambda)$, except that the challenger now extracts the link keys of the challenge ciphertexts and aborts in the event that they are different. More precisely, the challenger computes $\mathsf{lk}_0 = \mathsf{LExtr}(\mathsf{tk}, c_0)$ and $\mathsf{lk}_1 = \mathsf{LExtr}(\mathsf{tk}, c_1)$ and aborts if $\mathsf{lk}_0 \neq \mathsf{lk}_1$.

The unique extraction property of the link key extraction algorithm directly implies that $Pr[S_2] = Pr[S_1]$.

Game 3: In this game, we introduce the failure event F if one of the challenge ciphertext to be randomized c_0 or c_1 is valid but not in the range of $\mathsf{Enc}(\mathsf{pk}, \cdot)$. From the verifiability of the TREnc, $|Pr[S_3] - Pr[S_2]| \leq 2\varepsilon_{VER}$

Game 4: In this game, we transform $\mathsf{Game}_3(\lambda)$ by modifying the way we create the challenge ciphertext. Instead of randomizing c_b, we first decrypt it to receive a message m and then compute $c_b^\star = \mathsf{LEnc}(\mathsf{pk}', \mathsf{lk}_b, m)$.

Using the strong randomization of the TREnc (since the ciphertext of the challenge is in the range of the encryption algorithm), we have that $|Pr[S_4] - Pr[S_3]| \leq \varepsilon_{SR}$.

Game 5: In this last game, we use the $\mathsf{TrapAdapt}$ algorithm to adapt all the ciphertexts to be compatible with pk. Especially, for every decryption request of a ciphertext c, we first run the $\mathsf{TrapAdapt}$ algorithm to get a ciphertext in the range of $\mathsf{Enc}(\mathsf{pk}, \cdot)$ and then run the decryption algorithm.

Similarly for the challenge, we compute $c_b^\star$ as $\mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}_b, m)$ and then run the $\mathsf{TrapAdapt}$ algorithm to get a ciphertext in the range of $\mathsf{LEnc}(\mathsf{pk}', m)$ that we return to the adversary.

From the definition of the link extraction algorithm, these ciphertexts have the same distribution as the last game and $Pr[S_5] = Pr[S_4]$.

We finally reduce the last game to the wCCA experiment of Figure 3.3. We build an adversary for $\mathsf{Exp}^{\mathrm{trenc-wcca}}_{\mathcal{A}}(\lambda)$ as follows. First, we run the $\mathsf{TrapGen}$ algorithm on the public key received from the wCCA challenger

to obtain the trapdoor to extract link keys. Every decryption query is adapted and forwarded to $\mathsf{Exp}_{\mathcal{A}}^{\text{trenc}-\text{wcca}}(\lambda)$'s challenger, which proceeds identically as $\mathsf{Game}_5(\lambda)$ since the post-challenge decryption fails in the case where the trace of the ciphertext is the same as the challenge ciphertext. Regarding the challenge, our adversary receives at some point the challenge c_0, c_1. It asks the challenger to decrypt these ciphertexts (which always work because this happens before our own challenge) and extracts the link key lk of these ciphertexts via the link extraction algorithm. Then, we simply send to the challenger the pair of decrypted plaintext and the extracted link key, and finally adapt the resulting challenge ciphertext.

It follows that $Pr[S_5] \leq \frac{1}{2} + \varepsilon_{\text{wCCA}}$. □

Unsurprisingly, we can also prove the implication in the other direction as well. This time, we do not require the TREnc to be extractable.

Theorem 2. *If a TREnc scheme is verifiable, strongly randomizable, and* TCCA*-secure, then it is* wCCA*-secure. More precisely, if the advantages of any PPT adversary at the strong randomization and* TCCA *experiment are respectively bounded by ε_{SR} and $\varepsilon_{\text{TCCA}}$, then for any* PPT *adversary $\mathcal{A}$, we have $Pr[\mathsf{Exp}_{\mathcal{A},T}^{\text{trenc}-\text{wcca}}(\lambda) = 1] \leq \frac{1}{2} + \varepsilon_{SR} + \varepsilon_{\text{TCCA}}$.*

Proof. Let $\mathcal{A}$ be an adversary against the wCCA security of the TREnc, and $\varepsilon_{\text{wCCA}}$ be its winning probability in the TCCA game as defined in Figure 3.1 of Definition 22. We give a succession of hybrid games, where the first one is the real TCCA game. For each of those games, we note S_i the event that the i-th game outputs 1. When we say that the challenger aborts, we mean that the experiment outputs a random bit.

We create $\mathsf{Game}_1(\lambda)$ by modifying how the challenge ciphertext is computed. After computing the encryption of m_b, we simply randomize the ciphertext. Using the strong randomization of the TREnc, we directly have that $|Pr[S_0] - Pr[S_1]| \leq \varepsilon_{SR}$.

We can then reduce $\mathsf{Game}_1(\lambda)$ to the TCCA experiment. We build an adversary for $\mathsf{Exp}_{\mathcal{A}}^{\text{tcca}}(\lambda)$ as follow. Every decryption query is forwarded to $\mathsf{Exp}_{\mathcal{A}}^{\text{tcca}}(\lambda)$'s challenger, which proceeds identically as $\mathsf{Game}_4(\lambda)$ since the post-challenge decryption fails in the case where the trace of the ciphertext is the same as the challenge ciphertext. Regarding the challenge, our adversary receives at some point the challenge (m_0, m_1, lk). It encrypts each of the m_i into a ciphertext using lk and forwards these ciphertexts to the TCCA challenger as its own challenge. Clearly, this simulates perfectly $\mathsf{Game}_1(\lambda)$ and it follows that $Pr[S_1] \leq \frac{1}{2} + \varepsilon_{\text{TCCA}}$. □

$\mathsf{Exp}^{\mathrm{rcca}}_{\mathcal{A}}(\lambda)$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow\$\, \mathsf{Gen}(1^\lambda)$
$(m_0, m_1, \mathsf{st}) \leftarrow\$\, \mathcal{A}_1^{\mathsf{Dec}(\mathsf{sk},\cdot,\cdot)}(\mathsf{pk})$
$b \leftarrow\$\, \{0,1\}$
$c^\star \leftarrow\$\, \mathsf{Enc}(\mathsf{pk}, m_b)$
$b' \leftarrow\$\, \mathcal{A}_2^{\mathsf{Dec}^\star(\mathsf{sk},\cdot)}(c^\star, \mathsf{st})$
return $b = b'$

Figure 3.4.: Replayable CCA security experiment. In the experiment, $\mathcal{A}_2$ has access to a decryption oracle $\mathsf{Dec}^\star$ which, on input c, returns $m = \mathsf{Dec}(\mathsf{sk}, c)$ if $m \notin \{m_0, m_1\}$.

3.3.2. RCCA and pd-RCCA security

The replayable CCA security game (RCCA) [CKN03] is essentially the same as the CCA game, except for one main difference. In the post-challenge decryption queries, the challenger will decrypt the queried ciphertext and refuse to return the resulting plaintext if it is equal to one of the challenge messages. It is defined in Figure 3.4. The RCCA security game looks appealing in the context of voting, because it is a classical notion that captures this idea of having the possibility to re-randomize ciphertexts. And, indeed, RCCA-secure encryption has been used in previous proposals of receipt-free voting schemes [Cha+16]. However, there is little hope to build generic receipt-free voting schemes for RCCA-secure encryption schemes alone because of the need to be able to decrypt the ballots before rerandomizing them, as we need to know the content of the ballots to compute the simulated tally. RCCA-security does not offer enough flexibility.

Alternatively, the publicly-detectable RCCA security game (pd-RCCA) [CKN03] is a variant of RCCA security. It is essentially the same as the CCA game, except for two main differences: a publicly computable equivalence relation is defined on ciphertexts and, after the challenge ciphertext has been received, the challenger will refuse to decrypt any ciphertext that is equivalent to this challenge ciphertext. Furthermore, ciphertexts that are in the same equivalence class must decrypt to the same message. The pd-RCCA security game still offers the possibility to re-randomize ciphertexts but also provides a trace that could be detected through the equivalence relation.

If we define the relation between the ciphertexts as having the same trace or not and if we naturally allow the adversary to set the link key that will be used in the challenge ciphertext, we quickly find a notion that is almost equivalent to wCCA security of TREnc. However, the

difference is crucial: in the pd-RCCA game, all ciphertexts that have the same trace must decrypt to the same plaintext. This unfortunately clashes with the link traceability of the TREnc. Removing this constraint, which is well motivated for receipt-freeness, makes this game equivalent to wCCA security and thus implied by TCCA security.

Regarding RCCA security, it is proven that this is implied by TCCA and traceable security.

Theorem 3. *If a TREnc is traceable and* TCCA *secure, then it is* RCCA *secure. More precisely, if the advantages of any PPT adversary at the traceability, strong randomizability, and* TCCA *experiment are respectively bounded by ε_{TR}, ε_{SR} and $\varepsilon_{\mathrm{TCCA}}$, then for any* PPT *adversary $\mathcal{A}$, we have $Pr[\mathsf{Exp}^{\mathrm{rcca}}_{\mathcal{A},T}(\lambda) = 1] \leq \frac{1}{2} + \varepsilon_{TR} + \varepsilon_{SR} + \varepsilon_{\mathrm{TCCA}}$.*

Proof. Let $\mathcal{A}$ be an adversary against the RCCA security of the TREnc, and $\varepsilon_{\mathrm{RCCA}}$ be its winning probability as defined in Figure 3.1.

We give a succession of hybrid games, where the first one is the real RCCA game. For each of those games, we denote S_i the event that the i-th game outputs 1. When we say that the challenger aborts, we mean that the experiment outputs a random bit.

Game 0: This is the original RCCA game. By definition, we have that $Pr[\mathsf{Exp}^{\mathrm{rcca}}_{\mathcal{A},T}(\lambda) = 1] = Pr[S_0]$.

Game 1: We define this game as $\mathsf{Game}_0(\lambda)$, but we modify the post-challenge decryption oracle. For a ciphertext c, we return $\perp$ if $\mathsf{Dec}(\mathsf{sk}, c) \in \{m_0, m_1\}$ but also if $\mathsf{Trace}(c) = \mathsf{Trace}(c^\star)$.

Let F be the event in $\mathsf{Game}_0(\lambda)$ that at least one of the post-challenge decryption request is such that $\mathsf{Trace}(c) = \mathsf{Trace}(c^\star)$ and $\mathsf{Dec}(c) \neq m_b$. It is clear that the two games are identical unless F occurs and $|Pr[S_1] - Pr[S_0]| = Pr[F]$. We reduce to the traceability of the TREnc to show that this probability is negligible. We build an adversary $\mathcal{B}$ for $\mathsf{Exp}^{\mathrm{trace}}_{\mathcal{B}}(\lambda)$ as follows. We receive the pair of public-secret keys from the challenger, and simulate an adversary of $\mathsf{Game}_1(\lambda)$. For each pre-challenge decryption requests, we use the secret key to answer the request. For the challenge $(m_0^\star, m_1^\star)$, we pick a random b, send $m_b^\star$ to its challenger and forward the resulting ciphertext to the simulated adversary. Then for each of the post-challenge decryption requests, we decrypt the ciphertext and check if it has the same trace as the challenge ciphertext and if it is decrypted into a message different than m_b. In that case, we send it to the traceability challenger and win the game. Hence $Pr[F] \leq \varepsilon_{TR}$.

Game 2: In this game, we transform $\mathsf{Game}_1(\lambda)$ by modifying the way we create the challenge ciphertext. Instead of returning directly $c_b^\star$, we randomize it first.

Using the strong randomization of the TREnc, we have that $|Pr[S_2] - Pr[S_1]| \leq \varepsilon_{SR}$.

We finally reduce the last game to the TCCA experiment of Figure 3.1. We build an adversary $\mathcal{B}$ for $\mathsf{Exp}_{\mathcal{B}}^{\mathrm{tcca}}(\lambda)$ as follow and simulate an adversary $\mathcal{A}$ for $\mathsf{Game}_2(\lambda)$. First, we receive the public key from the challenger. When we receive the challenge (m_0, m_1) from $\mathcal{A}$, we generate a link key and encrypt both messages with the same link key. We then use these ciphertexts as our TCCA challenge and return the resulting challenge ciphertext to the adversary. Every decryption query is forwarded to the $\mathsf{Exp}_{\mathcal{A}}^{\mathrm{tcca}}(\lambda)$'s challenger. In the case of a ciphertext that has a trace different from the trace of the challenge ciphertext but which contains a message identical to m_0 or m_1, we return $\perp$. The decryption requests clearly proceed identically as $\mathsf{Game}_2(\lambda)$.

It finally follows that $Pr[S_2] \leq \frac{1}{2} + \varepsilon_{\mathrm{TCCA}}$. □

3.4. A generic voting scheme based on Traceable Receipt-Free Encryption

As we will demonstrate in this section, TREncs are particularly well suited for the design of voting systems offering receipt freeness, that is, systems in which voters cannot demonstrate how they voted to a third party. In this section, we mostly use the definitions introduced in Section 2.3 of the preliminaries

3.4.1. Voting scheme

We first describe how a generic registration-free voting system can be built from a TREnc. The protocol in itself is of little interest: it essentially follows previous proposals [Bla+11; Cha+16]. Its central interest is that defining it from a TREnc makes the proof of its receipt-freeness almost immediate, and independent of any specific TREnc instance.

A detailed pseudocode description is proposed in Figure 3.5. The protocol executes as follow. The election authorities set-up the election in the following way. They create an empty bulletin board and create the pair of public and private keys $(\mathsf{pk}, \mathsf{sk})$ of the election by running the Gen algorithm of the TREnc scheme with the desired security parameter. pk is distributed to every party taking part to the election and sk is given

Setup (λ)

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$
$\mathsf{crs} \leftarrow \mathsf{TProveGen}(1^\lambda)$
return $(\mathsf{pk}, \mathsf{crs})$

Vote $(\mathsf{id}, \mathsf{v}[, \mathsf{aux}])$

if aux is specified **then** $\mathsf{lk} \leftarrow \mathsf{aux}$
else $\mathsf{lk} \leftarrow\$ \mathsf{LGen}(\mathsf{pk})$
$\mathsf{b} \leftarrow \mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, \mathsf{v})$
if aux is not specified **then return** b
else return $\mathsf{b}, \mathsf{Trace}(\mathsf{b}), \mathsf{lk}$

TraceBallot (b)

return $\mathsf{Trace}(\mathsf{b})$

Valid $(\mathsf{BB}, \mathsf{b})$

if $\mathsf{Ver}(\mathsf{b}) \wedge \forall \mathsf{b}' \in \mathsf{BB}$,
$\mathsf{TraceBallot}(\mathsf{b}') \neq \mathsf{TraceBallot}(\mathsf{b})$
then return 1 **else return** 0

ProcessBallot (b)

return $\mathsf{Rand}(\mathsf{pk}, \mathsf{b})$

VerifyVote $(\mathsf{PBB}, \mathsf{t})$

if $\exists b \in \mathsf{PBB} : \mathsf{Valid}(b) \wedge$
$\mathsf{t} == \mathsf{TraceBallot}(\mathsf{b})$
then return 1 **else return** 0

Tally $(\mathsf{sk}, \mathsf{BB})$

$r \leftarrow \mathsf{TCompute}(\mathsf{sk}, \mathsf{BB})$;
$\Pi \leftarrow \mathsf{TProve}(\mathsf{crs}, \mathsf{sk}, \mathsf{BB}, r)$;
return r, Π

VerifyResult (BB, r, Π)

return $\mathsf{TVerify}(\mathsf{crs}, \mathsf{BB}, r, \Pi)$

Figure 3.5.: Instantiation of our generic voting scheme from a generic TREnc scheme. The algorithms (TCompute, TProveGen, TProve, TVerify) form a generic tallying algorithm discussed in Section 2.3.4 and can be instantiated via standard techniques, depending on the result function (homomorphic tallying, verifiable mixnet, ...).

to the tallier. Note that sk can be generated in a distributed way, so that decryption requires the contribution of multiple trustees – our TREnc constructions are compatible with standard threshold key generation protocols for discrete log based cryptosystems, which can be used as usual since the decryption of a ciphertext is independent from the link key and the Trace algorithm [Cor+13; Gen+07]. We consider a unique tallier in the following.

When a user wants to cast a vote v, they first generate a link key lk by running the $\mathsf{LGen}(\mathsf{pk})$ algorithm, then encrypts the vote with the $\mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, \mathsf{v})$ algorithm in order to obtain a ballot b, while aux is defined as lk. The voter then sends the encrypted ballot to the ballot box manager of the voting system. It is extremely important that the user erases the link key as soon as possible, as the integrity of their vote may rely on the secrecy of this key. The voter will however store $\mathsf{TraceBallot}(\mathsf{b}) = \mathsf{Trace}(\mathsf{b})$ in order to verify that a ballot with the correct trace eventually appears on the public bulletin board.

When the ballot box manager receives a new ballot b, it verifies the validity of the ballot by checking that $\mathsf{Ver}(\mathsf{b})$ succeeds and that no ballot with the same trace was recorded before. Invalid ballots are dropped and valid ones are put through $\mathsf{Append}(\mathsf{BB}, \mathsf{b})$, which runs $\mathsf{ProcessBallot}(\mathsf{b}) = \mathsf{Rand}(\mathsf{pk}, \mathsf{b})$ and appends the result to BB.

The user can verify that their vote is on the bulletin board by checking with the $\mathsf{TraceBallot}$ algorithm if any entry in the public bulletin board has the same trace as the one they recorded when they produced their ballot. The traceability property of the TREnc then guarantees that nobody (including the rerandomizing server and the election authorities who hold the decryption key) could have forged another valid ciphertext of another vote linked to this ballot, except with non-negligible probability.

Once every voter has cast a vote, the tallier can gather the ballots on the bulletin board and compute the result of the election r, as well as a proof of correctness Π. The exact details of this process will depend on the ballot format and the result function ρ_m that the voting protocol requires. We use the generic tally procedure proposed in Section 2.3.4 to model this tally, without detailing how to instantiate it. This could be done for example with a verifiable mixnet, as our TREnc ciphertexts are compatible with various standard options that operate on votes encrypted as vectors of group elements, including the Verificatum mixnet [Wik09].

3.4.2. Security of the voting scheme

Receipt-freeness This voting scheme has receipt freeness, as claimed in the following theorem.

Theorem 4. *If the TREnc used in the voting scheme is* TCCA *secure and verifiable, and if the proof system used to prove the correctness of the tally is zero-knowledge, then the voting scheme has receipt freeness.*

Proof. Let $\mathcal{A}$ be an adversary against the receipt-freeness of the voting scheme, and ε_{RF} be its winning probability in the receipt-freeness game as defined in Figure 2.7 of Definition 16. We give a succession of hybrid games, where the first one is the real RF game for $\beta = 0$. For each of those games, we denote by S_i the event that the i-th game outputs 1. When we say that the challenger aborts, we mean that the experiment outputs a random bit.

$\mathsf{Game}_0(\lambda)$**:** This it the original receipt-freeness game. By definition, $Pr[\mathsf{Exp}^{\mathsf{srf},0}_{\mathcal{A},\mathcal{V}}(\lambda) = 1] = Pr[S_0]$. As a reminder, the adversary in this game can ask a polynomially bounded number of $\mathcal{O}\mathsf{receiptLR}(\mathsf{b}_0, \mathsf{b}_1)$ requests and a final $\mathcal{O}\mathsf{tally}$ request.

$\mathsf{Game}_1(\lambda)$**:** We define $\mathsf{Game}_1(\lambda)$ as $\mathsf{Game}_0(\lambda)$ with the following modification. Instead of computing the proof of the tally honestly with TProve, we run the $\mathsf{TSimProve}$ algorithm and define $\mathsf{SimProve}$ accordingly.

From the zero-knowledge property of the proof of the tally, this transition is indistinguishable and $|Pr[S_1] - Pr[S_0]| \leq \varepsilon_{ZK}$.

$\mathsf{Game}_2(\lambda)$**:** In this game, we add the failure event F that for at least one of the ballots in the $\mathcal{O}\mathsf{receiptLR}$ requests, the ballot contains a ciphertext which is valid but not in the range of $\mathsf{Enc}(\mathsf{pk}, \cdot)$. If the event happens, we abort and return a random bit.

Using a standard hybrid argument, $|Pr[S_2] - Pr[S_1]| \leq 2q_r\varepsilon_{VER}$.

$\mathsf{Game}_3(\lambda)$**:** In the next game, we compute the result of the election in another way. Instead of running the $\mathsf{TCompute}$ algorithm, we run the TREnc decryption algorithm on each valid ballot b_0 received in each of the $\mathcal{O}\mathsf{receiptLR}$ request and compute directly the tally with the result function ρ applied on those votes.

From the correctness of the TREnc and the tallying algorithm, we have that $Pr[S_3] = Pr[S_2]$.

$\mathsf{Game}_4(\lambda)$**:** In $\mathsf{Game}_4(\lambda)$, we now set β to 1. That is, we return BB_1 instead of BB_0 in the $\mathcal{O}\mathsf{board}$ oracle and generate the simulated proof Π from BB_1 in the $\mathcal{O}\mathsf{tally}$ oracle.

We reduce to the mTCCA game of the TREnc, which was proven to be equivalent to TCCA up to a factor q, where q is the number of

challenges (see Lemma 1). From an adversary $\mathcal{B}$ distinguishing between $\mathsf{Game}_3(\lambda)$ and $\mathsf{Game}_4(\lambda)$, we build the mTCCA adversary as follows. First, we define BB as an empty bulletin board and forward the public key of the TREnc to $\mathcal{B}$. Each time we receive an $\mathcal{O}\mathsf{board}$ request, we send BB to the adversary. Then, for each $\mathcal{O}\mathsf{receiptLR}(\mathsf{b}_0, \mathsf{b}_1)$ request, we request a decryption of b_0 to the challenger and then ask for a challenge with $(\mathsf{b}_0, \mathsf{b}_1)$.[12] We append the resulting ballot to BB. Finally, when we receive the $\mathcal{O}\mathsf{tally}$ request, we directly compute the result of the election from the decrypted ballots and use the $\mathsf{TSimProve}$ algorithm to prove the validity of the result. This adversary perfectly simulates Game_1 or Game_2 depending on the mTCCA challenge bit and hence, $|Pr[S_4] - Pr[S_3]| \leq q\varepsilon_{TCCA}$.

$\mathsf{Game}_5(\lambda)$: Finally in the last game $\mathsf{Game}_5(\lambda)$, we recompute the result from $\mathsf{TCompute}$ again on the secret view of BB_0. Again, from the correctness of the TREnc and the tallying algorithm, we have that $Pr[S_5] = Pr[S_4]$.

The last game is equivalent to the receipt-freeness experiment for $\beta = 1$, namely $\mathsf{Exp}^{\mathsf{srf},1}_{\mathcal{A},\mathcal{V}}(\lambda)$. Hence, we conclude that the probability that the adversary wins the receipt-freeness experiment is thus bounded by $\varepsilon_{ZK} + q(2\varepsilon_{VER} + \varepsilon_{TCCA})$.

□

As discussed in Section. 2.3, the receipt-freeness game alone is not enough to guarantee that a voter can escape an adversary. Indeed, we could imagine pathological TREncs such that for each trace, only one message can produce a ciphertext with this trace. Hence, even if the voting protocol is receipt-free in the sense of Definition 16, an adversary could force a voter to use a given trace related to a single message. To counter such attacks, we introduced the ballot traceability to ensure that ballots can encode different votes with the same trace. Thanks to the link traceability (Definition 20) of the TREnc, our protocol trivially satisfies this notion.

Ballot privacy Receipt-freeness can naturally be seen as a stronger form of ballot privacy as replacing the $\mathcal{O}\mathsf{receiptLR}$ oracle in the receipt-freeness game with the $\mathcal{O}\mathsf{voteLR}$ and $\mathcal{O}\mathsf{cast}$ of Figure 2.5 yields the BPRIV game [Ber+15], and those two oracles can be simulated by $\mathcal{O}\mathsf{receiptLR}$. Hence, the voting scheme has ballot privacy against every

[12] This is the reason that we need to be able in the TCCA game to ask for decryption of ballots for which the trace can be used in a challenge afterwards

party except the rerandomizing server, who is supposed to be honest in the receipt-freeness game.

Fortunately, our protocol also offers privacy against a rerandomizing server. As can be expected, proving privacy against such an adversary requires taking advantage of the strong-randomization property of the TREnc, which was not necessary for receipt-freeness.

If the rerandomizing server is corrupted, we have to slightly modify the BPRIV game since it has an additional view of the election (it sees the ballots sent by the voters before the $\mathsf{ProcessBallot}$ algorithm) and it can maliciously process those ballots. The main difference is in the $\mathcal{O}\mathsf{voteLR}$ oracle. Instead of honestly processing the ballots, the oracle appends the ballot of both worlds to a queue of unprocessed ballots SBB, and return the ballot relative to the adversary's view. The adversary also has access to an $\mathcal{O}\mathsf{process}$ oracle which allows it to send a processing $\tilde{\mathsf{b}}$ of a ballot b. If the ballot is indeed in the list of unprocessed ballots SBB and if the processing keeps the same trace, we append $\tilde{\mathsf{b}}$ to the adversary's bulletin board and an honest processing of the other related ballot to the other board. The adversary can also send $\mathcal{O}\mathsf{cast}$ requests as before, but the trace of such ballots needs to be different from the trace of any ballot on SBB for the definition to remain meaningful.

Definition 28 (Ballot privacy against the ballot box manager)**.** *A voting system $\mathcal{V}$ has ballot privacy against the ballot box manager if there exists algorithms* $\mathsf{SimProof}$ *and* $\mathsf{SimSetup}$ *such that no* PPT *adversary $\mathcal{A}$ can distinguish between games $\mathsf{Exp}^{\mathsf{bpriv-bb},0}_{\mathcal{A},\mathcal{V}}(\lambda)$ and $\mathsf{Exp}^{\mathsf{bpriv-bb},1}_{\mathcal{A},\mathcal{V}}(\lambda)$ defined by the oracles in Figure 3.6, that is for any efficient algorithm $\mathcal{A}$:*

$$\left| Pr\left[\mathsf{Exp}^{\mathsf{bpriv-bb},0}_{\mathcal{A},\mathcal{V}}(\lambda) = 1\right] - Pr\left[\mathsf{Exp}^{\mathsf{bpriv-bb},1}_{\mathcal{A},\mathcal{V}}(\lambda) = 1\right] \right|$$

is negligible in λ.

Theorem 5. *If the TREnc used in the voting scheme is* TCCA *secure, verifiable, has traceability, and strong randomization and if the proof system used to prove the correctness of the tally is zero-knowledge, then the voting scheme has ballot privacy against the ballot box manager.*

Proof. The proof is similar to the receipt-freeness, except that we first need to use the strong randomization of the TREnc before the TCCA transition.

Let $\mathcal{A}$ be an adversary against ballot privacy with a malicious casting server of the voting scheme, and ε_{BPRIV} be its winning probability in the experiment defined in Figure 3.6. We give a succession of hybrid

$\mathcal{O}\mathsf{init}(\lambda)$

if $\beta = 0$ **then** $(\mathsf{pk}, \mathsf{sk}) \leftarrow\$\, \mathsf{Setup}(1^\lambda)$
else $(\mathsf{pk}, \mathsf{sk}, \tau) \leftarrow\$\, \mathsf{SimSetup}(1^\lambda)$
return pk

$\mathcal{O}\mathsf{voteLR}(\mathsf{id}, \mathsf{v}_0, \mathsf{v}_1)$

if $\mathsf{v}_0 \notin \mathbb{V}$ or $\mathsf{v}_1 \notin \mathbb{V}$ **then return** $\perp$
$\mathsf{b}_0, \Gamma_0 \leftarrow\$\, \mathsf{Vote}(\mathsf{id}, \mathsf{v}_0)$
$\mathsf{b}_1, \Gamma_1 \leftarrow\$\, \mathsf{Vote}(\mathsf{id}, \mathsf{v}_1, \mathsf{aux})$
$\mathsf{SBB} \leftarrow \mathsf{SBB} \cup \{(\mathsf{b}_0, \mathsf{b}_1)\}$
return b_β

$\mathcal{O}\mathsf{process}(\mathsf{b}, \tilde{\mathsf{b}})$

if $\mathsf{TraceBallot}(\mathsf{b}) = \mathsf{TraceBallot}(\tilde{\mathsf{b}})$
and $\mathsf{Valid}(\mathsf{BB}_\beta, \tilde{\mathsf{b}}) = 1$
and $\exists (\mathsf{b}_0, \mathsf{b}_1) \in \mathsf{SBB}$ s.t. $\mathsf{b} = \mathsf{b}_\beta$, **then**
  $\mathsf{BB}_\beta \leftarrow \mathsf{BB}_\beta \cup \{\tilde{\mathsf{b}}\}$
  $\mathsf{b}' \leftarrow\$\, \mathsf{ProcessBallot}(\mathsf{b}_{1-\beta})$
  $\mathsf{BB}_{1-\beta} \leftarrow \mathsf{BB}_{1-\beta} \cup \{\mathsf{b}'\}$

$\mathcal{O}\mathsf{cast}(\mathsf{b})$

if $\mathsf{Valid}(\mathsf{BB}_\beta, \mathsf{b}) = 1$ and$\forall (\mathsf{b}_0, \mathsf{b}_1) \in \mathsf{SBB}$
$\mathsf{TraceBallot}(\mathsf{b}_\beta) \neq \mathsf{TraceBallot}(\mathsf{b})$ **then**
  $\mathsf{Append}(\mathsf{BB}_0, \mathsf{b}); \mathsf{Append}(\mathsf{BB}_1, \mathsf{b})$

$\mathcal{O}\mathsf{board}()$

return BB_β

$\mathcal{O}\mathsf{tally}()$

$(\mathsf{r}, \Pi) \leftarrow\$\, \mathsf{Tally}(\mathsf{BB}_0, \mathsf{sk})$
if $\beta = 1$ **then** $\Pi \leftarrow\$\, \mathsf{SimProve}(\mathsf{BB}_1, \mathsf{r}, \tau)$
return (r, Π)

Figure 3.6.: Oracles used in the ballot privacy against the ballot box manager games. The $\beta = 0$ game corresponds to the real ballot box, while the $\beta = 1$ game correspond to the fake ballot box. The only other differences with the BPRIV oracles of Figure 2.5 are that the $\mathcal{O}\mathsf{board}$ oracle returns BB instead of $\mathsf{Publish}(\mathsf{BB})$ and Append is replaced by Append' which runs Append and returns to the adversary the randomness used in the execution. Here, since the rerandomizing server is corrupted, the $\mathcal{O}\mathsf{voteLR}$ oracle returns to the adversary the ballot that will end up in the adversary's bulletin board after $\mathsf{ProcessBallot}$.

games, where the first one is the real BPRIV game for $\beta = 0$. For each of those games, we denote by S_i the event that the i-th game outputs 1. When we say that the challenger aborts, we mean that the experiment outputs a random bit. Also, we denote respectively by q_v and q_c the number of calls to the $\mathcal{O}\mathsf{voteLR}$ and $\mathcal{O}\mathsf{cast}$ oracles.

$\mathsf{Game}_0(\lambda)$**:** We define $\mathsf{Game}_0(\lambda)$ as the original BPRIV experiment. In this game, the adversary in this game can ask a polynomially bounded number of $\mathcal{O}\mathsf{voteLR}(\mathsf{v}_0, \mathsf{v}_1)$ and $\mathcal{O}\mathsf{cast}(\mathsf{b})$ requests and a final $\mathcal{O}\mathsf{tally}$ request. Also, the computation of BB_1 can be ignored since it never appears in the view of the adversary. Especially, we do not compute b_1 in the $\mathcal{O}\mathsf{voteLR}$ oracle.

$\mathsf{Game}_1(\lambda)$**:** We define $\mathsf{Game}_1(\lambda)$ as $\mathsf{Game}_0(\lambda)$ with the following modification. Instead of computing the proof of the tally honestly with TProve, we run the $\mathsf{TSimProve}$ algorithm on BB_0 and define $\mathsf{SimProve}$ accordingly.

From the zero-knowledge property of the tallying algorithm, this transition is indistinguishable and $|Pr[S_1] - Pr[S_0]| \leq \varepsilon_{ZK}$.

$\mathsf{Game}_2(\lambda)$**:** We introduce in $\mathsf{Game}_2(\lambda)$ the following failing event F that happens if in any of the $\mathcal{O}\mathsf{cast}$ requests or any of the $\mathcal{O}\mathsf{process}$, the adversarial ballot to be added to the bulletin board is valid but not in the range of Vote.

Using a standard hybrid argument, $|Pr[S_2] - Pr[S_1]| \leq (q_c + q_v)\varepsilon_{VER}$.

$\mathsf{Game}_3(\lambda)$**:** In $\mathsf{Game}_3(\lambda)$, we compute the result of the election in another way. Instead of running the $\mathsf{TCompute}$ algorithm, we run the TREnc decryption algorithm on each valid ballot b received in a $\mathcal{O}\mathsf{cast}$. Together with all the votes v_0 received in the $\mathcal{O}\mathsf{voteLR}$ requests such that there has been a corresponding $\mathcal{O}\mathsf{process}$ request afterwards, we directly compute the result of the election with ρ.

Let us consider the event that for the ballot generated in the i-th $\mathcal{O}\mathsf{voteLR}$ request, the corresponding "processed" ballot $\tilde{\mathsf{b}}$ does not have the same content. This event happens with at most probability ε_{trace}.

Indeed, we can reduce to the traceability of the TREnc. From an adversary $\mathcal{B}$ for $\mathsf{Game}_3(\lambda)$, we build the traceability adversary as follows. We first receive the secret key of the TREnc, which is used for the decryption of the $\mathcal{O}\mathsf{cast}$ ballots. During the i-th $\mathcal{O}\mathsf{voteLR}$ request, we do not use the Vote algorithm and instead send it to the traceability challenger. We receive in return a ballot that we append to the set SBB. When $\mathcal{B}$ sends a $\mathsf{ProcessBallot}$ request for this ballot, we simply return

the corresponding processed ballot to the traceability challenger. Since both ballots have the same trace, we win the game whenever this event happens.

Thus, if this event does not happen, the result is computed identically from the correctness of the TREnc and the tallying algorithm. Hence we have $|Pr[S_3] - Pr[S_2]| = q_v \varepsilon_{trace}$.

$\mathsf{Game}_4(\lambda)$**:** In the next hybrid $\mathsf{Game}_4(\lambda)$, we now compute the ballots relative to the BB_1 bulletin board. Moreover, we modify the $\mathcal{O}\mathsf{voteLR}$ oracle. After computing b_0 and b_1, we rerandomize them. We add these ballots to the set SBB and return the randomized ballot to the adversary.

Since the ballots are in the range of the encryption algorithm of the TREnc, the strong randomizability ensures that the distribution of these ballots are computationally indistinguishable and thus $|Pr[S_4] - Pr[S_3]| \leq \varepsilon_{SR}$.

$\mathsf{Game}_5(\lambda)$**:** In $\mathsf{Game}_5(\lambda)$, we now set β to 1. That is, we return BB_1 instead of BB_0 in the $\mathcal{O}\mathsf{board}$ oracle. We already generate the simulated proof Π from BB_1 in the $\mathcal{O}\mathsf{tally}$ oracle since the first hybrid.

We reduce to the mTCCA game of the TREnc, which was proven to be equivalent to TCCA up to a factor q, where q is the number of challenges (see Lemma 1). From an adversary $\mathcal{B}$ distinguishing between games $\mathsf{Game}_4(\lambda)$ and $\mathsf{Game}_5(\lambda)$, we build the mTCCA adversary as follows. First, we define BB as an empty bulletin board and forward the public key of the TREnc to $\mathcal{B}$. Each time we receive an $\mathcal{O}\mathsf{board}$ request, we send BB to the adversary. Also, for each $\mathcal{O}\mathsf{voteLR}(\mathsf{v}_0, \mathsf{v}_1)$ request, we compute the ballots b_0 and b_1 from an execution of the Vote algorithm and then send them as a challenge to the challenger. We send the resulting ballot to the adversary. When we we receive the corresponding $\mathcal{O}\mathsf{process}$ request, we append the adversary's ballot to BB. Moreover, each time we receive an $\mathcal{O}\mathsf{cast}$ request, we ask the mTCCA challenger to decrypt the ballot (to be able to compute the tally) and append the ballot to BB. It is never an issue to ask for such decryption as these traces are always different from the traces of the ballots in the $\mathcal{O}\mathsf{voteLR}$ calls for the request to be valid. Finally, when we receive the $\mathcal{O}\mathsf{tally}$ request, we directly compute the result of the election from the decrypted ballots and use the $\mathsf{TSimProve}$ algorithm to prove the validity of the result. This adversary perfectly simulates Game_3 or Game_4 depending on the mTCCA challenge bit and hence, $|Pr[S_5] - Pr[S_4]| \leq q_v \varepsilon_{TCCA}$.

$\mathsf{Game}_6(\lambda)$**:** In the second to last game, we do not rerandomize the ballots in the $\mathcal{O}\mathsf{voteLR}$ oracle. Again from the strong randomization,

$|Pr[S_6] - Pr[S_5]| \leq \varepsilon_{SR}$.

$\mathsf{Game}_7(\lambda)$**:** Finally in the last game $\mathsf{Game}_7(\lambda)$, we recompute the result from $\mathsf{TCompute}$ again. Since BB_0 is computed from an honest execution of $\mathsf{ProcessBallot}$, we have that $Pr[S_7] = Pr[S_6]$.

The last game is equivalent to the BPRIV experiment for $\beta = 1$. Hence, we conclude that the probability that the adversary wins the ballot privacy experiment against the ballot box manager is bounded by $\varepsilon_{ZK} + 2\varepsilon_{SR} + (q_c + q_v)\varepsilon_{VER} + q_v(\varepsilon_{TCCA} + \varepsilon_{trace})$. □

It is worth noting that the strong randomizability is not *necessary*. We can prove this result more directly from a reduction to the wCCA-security of a TREnc defined in Figure 3.3, which is implied by the TCCA security and the strong randomizability. Actually, we can also use an even weaker notion where the adversary does not choose adaptively the link key of the challenge, as the link keys of the ballots in the $\mathcal{O}\mathsf{voteLR}$ oracle are generated by the challenger.

It is also important to note that the notions of receipt-freeness and ballot privacy only make sense when applied to voting protocols that satisfy some extra correctness requirements (see Bernhard et al. [Ber+15] for instance) – a pathological Valid process that would just drop all but one ballot could result in this ballot been tallied alone, which could satisfy the definition or receipt-freeness but would obviously be problematic from a privacy point of view. The notions of strong consistency, strong correctness, and strong validity defined in Section 2.3 address these questions. Fortunately, the protocol satisfies these notions.

Lemma 2. *If the TREnc used in the voting scheme is verifiable and strongly randomizable, then the voting scheme has strong consistency, strong correctness and strong validity.*

Proof.
The argument is split into 3 steps.

Strong consistency: We define the $\mathsf{Extract}$ algorithm as the decryption algorithm applied on the ciphertext. Also, we define the $\mathsf{ValidInd}$ algorithm as the Ver algorithm applied on the ciphertext. For the first two points of the definition, the correctness of the encryption scheme ensures that the decryption of a ciphertext returns the encrypted message. For the third point, Valid outputs 1 only if the Ver algorithm also outputs 1. Finally the last point results from the correctness of the tally algorithm.
Strong correctness: The ballot $\mathsf{b} \leftarrow\!\$\, \mathsf{Vote}(\mathsf{id}, \mathsf{v})$ of any voter id and any vote v is not valid only if

- $\mathsf{Ver}(b) = 0$, which can only happen with a negligible probability on λ because of the honest verifiability of the TREnc (Definition 20).
- There is already a ballot on BB that has the same trace, which can only happen with a negligible probability because of the traceability property.

Also, it is computationally hard to distinguish a randomized ballot from a fresh encryption from the strong randomization property (Definition 24). Together with the honest verifiability property (Definition 20), the output of $\mathsf{ProcessBallot}$ will be valid as well with overwhelming probability.
Strong validity: This is directly implied by the TREnc's verifiability (Definition 21). □

Verifiability We do not detail the whole verifiability of our voting system, which would require us to introduce a substantial machinery. We outline how this could work here:

- Eligibility verifiability could proceed by adding the voter's name next to each ballot on the public bulletin board, and let auditors check whether these are legitimate voters. Weaker but more convenient options include relying on a trusted authentication server and/or on a PKI.
- Universal verifiability, which guarantees that the tally is computed correctly, would result from the tallying process, e.g., from a verifiable mix-net.
- Individual verifiability is captured in Definition 18 and is discussed in the remaider of this section. Intuitively, if the voter's voting client is honest, this follows from the traceability property of the TREnc and the single use of lk, which guarantee that any ballot with the same trace as the ballot submitted by the voter would decrypt to the right vote. Detecting a malicious voting client that may encrypt a vote different from the one chosen by the voter is more tricky and will be explored in Chapter 5.

Theorem 6. *If the TREnc used in the voting scheme is traceable and verifiable, then the voting scheme is verifiable in the sense of Definition 18.*

Proof. The proof uses four different games, where the first one is the BPRIV experiment with $\beta = 0$. We denote by S_i the event that the adversary returns 1 in the i-th game. Since our protocol is registration free, we assume that there are no calls to the $\mathcal{O}\mathsf{register}$ and $\mathcal{O}\mathsf{corrupt}$

oracles. Instead, the adversary fixes the number u and cu of honest and corrupted users.

$\mathsf{Game}_1(\lambda)$**:** We define $\mathsf{Game}_1(\lambda)$ as the original verifiability experiment. In this game, the adversary can ask a polynomially bounded number of $\mathcal{O}\mathsf{vote}(\mathsf{id}, \mathsf{v})$ requests.

$\mathsf{Game}_2(\lambda)$**:** $\mathsf{Game}_2(\lambda)$ proceeds as $\mathsf{Game}_1(\lambda)$, but we compute the tally differently. Instead of computing the tally through the Tally algorithm, we decrypt each ballot and compute the results r from the result function ρ.

The correctness of the tally and the decryption guarantee that the result is computed identically if all ballots are in the range of $\mathsf{Vote}(\cdot)$. Hence from the verifiability of the TREnc, we have $|Pr[S_2] - Pr[S_1]| \leq (u + cu)\varepsilon_{VER}$.

$\mathsf{Game}_3(\lambda)$**:** Before defining $\mathsf{Game}_3(\lambda)$, we define some variables. We define v_{id} (resp. b_{id}) as the vote (resp. the ballot) of an honest voter id computed in the $\mathcal{O}\mathsf{vote}$ oracle and $\tilde{\mathsf{v}}_{\mathsf{id}}$ as the vote contained in the ballot $\tilde{\mathsf{b}}_{\mathsf{id}}$ in BB with the same trace as b_{id} (or $\perp$ if it does not exist). Finally, we define $\tilde{V}_H$ and $\tilde{V}_C$ as the set of $\{\tilde{\mathsf{v}}_{\mathsf{id}}\}$ and as the set of the decryption of the others valid ballots on BB.

Now, we define $\mathsf{Game}_3(\lambda)$ as $\mathsf{Game}_2(\lambda)$, but we let the adversary win the game if all the $\mathsf{VerifyVote}$ checks go through and if for at least one honest voter, $\tilde{\mathsf{v}}_{\mathsf{id}} \neq \mathsf{v}_{\mathsf{id}}$.

This happens with a negligible probability. We reduce to a traceability game of the TREnc for the i-th voter. In the reduction, instead of computing the b_{id} from the vote algorithm for the i-th voter, we send the vote to traceability challenger and return the resulting ballot. Then, if $\tilde{\mathsf{b}}_{\mathsf{id}} \neq \perp$, we use this ballot as our forgery. Otherwise, $\mathsf{VerifyVote}$ returns 0 for this voter.

Combining all the voters together, we have thus $|Pr[S_3] - Pr[S_2]| \leq u\varepsilon_{trace}$.

Finally, since $V_H = \tilde{V}_H$ and since the result is computed as $\rho(\tilde{V}_H \cup \tilde{V}_C)$, the adversary loses the game and its advantage is bounded by $u(\varepsilon_{VER} + \varepsilon_{trace})$. □

3.5. Concluding remarks

This chapter proposed the first cryptographic primitive that can be used to generically obtain receipt-free and single-pass voting protocols. For

instantiations of this primitive, we refer to Chapter 4, where we will design several constructions that realize the security notions that we defined.

The central result of this chapter is that it is sufficient to have a TCCA encryption scheme to attain receipt-freeness, using the blueprint voting protocol presented in Section 2.3. The analogous result for ballot privacy is stronger: an NM-CPA encryption scheme (or equivalently an IND-1-CPA encryption scheme) is necessary and sufficient [BPW12]. Intuitively, TCCA security cannot be equivalent to receipt-freeness: indeed, in the receipt-freeness game we only have one "decryption" query after all the ballots have been cast, while the TCCA adversary can ask for a polynomial number of decryption queries. On the other hand, we cannot adapt the TCCA security to only have one decryption request of a vector of ciphertexts at the end of the game, since we need to be able to extract the vote of the ballots in the $\mathcal{O}\mathsf{receiptLR}$ and in the reduction to the TCCA experiment, this is made through a decryption request before each challenge. Capturing a natural encryption primitive which is necessary and sufficient to provide receipt-freeness is thus still an open question in the quest of simplification, efficiency and better understanding of security.

Arguably, this line of work is more of a theoretical nature, since the generic protocol is not fully described and hardly practical. Crucially, the protocol assumes that the voter knows the link key of their ballot when they cast it. If this is not the case, the receipt-freeness adversary can force them to use a ballot of a given trace, and the traceability of the TREnc ensures that the voter cannot compute another ballot for the vote of their choice.

It is not trivial to ensure the voter knows this link key, as a cryptographic proof of knowledge would require interactions with the casting server. Since in practice a voter needs to download the public parameters from the casting server, we could imagine that it receives an unpredictable nonce as well that can be used to prove the knowledge of the link key, but this would need to modify the voting protocol and this strategy would be hard to instantiate generically. Another approach would be to rely on a registration procedure to provide the link key without a joint trust assumption that at most one of the registrar or the rerandomizing server is malicious. Otherwise, with the knowledge of the link key, they could replace the voter's ballot by whichever vote they desire. However, this awkward assumption on the generation of the link key seems inherent to single-pass voting protocols. This will be investigated more in Chapter 5.

Finally, while the clear focus is on the verifiable voting applications, it is still unclear if TREncs have another use case.

4. TREnc constructions

In the previous chapter, we introduced the new Traceable Receipt-Free Encryption (TREnc) primitive and motivated it for its application to non-interactive receipt-free voting protocols. We now show in this chapter how to build this primitive.

First, we will start with a generic construction [DPP22b] that is built on top of classic cryptographic primitives. This construction is based on signatures on randomizable ciphertexts and on tag-based encryption and relies on the additional link extraction feature to attain the TCCA security from wCCA security of a TREnc. However, this extraction feature is superfluous, and we can actually directly build a TREnc without it.

We will thus propose more efficient constructions based on a bilinear group and which only relies on the SXDH assumption. The first one [DPP22b] is mainly described for chronological reasons, as this is the first efficient TREnc that has ever been proposed and the original design greatly influenced the following constructions. However, this construction unfortunately uses a structured CRS, which can be problematic in a voting setting. Indeed, the CRS is a set of public parameters that are used in the protocols to prove the correctness of some statements, such as the fact that a ballot contains a valid vote. A structured CRS enjoys from some algebraic structure to help the design of the protocol, but in turn, it relies on a trusted setup. That is, the party that generates the CRS often obtains a trapdoor from the randomness used to generate the CRS, and must then be trusted to erase this trapdoor. Otherwise, this party could generate convincing proofs of false statements and, in our voting context, could craft invalid ballots. To address this issue, a common solution is to distribute the trust among multiple parties and generate the CRS distributively with a multi-party computation protocol. Unfortunately, this solution is not satisfying since we aim at building voting schemes that are verifiable even if all the official parties are corrupted.

The next two constructions solve this problem, as they use a public coin CRS. Compared to a structured CRS, a public coin CRS can be generated by a public hash function, and hence does not require a trusted setup. These two constructions differ in the way they encode a message.

One construction directly encodes group elements as the original TREnc construction and is perfectly adapted to be used with a mixing network, while the other one encodes a message in a bit-by-bit fashion and can be used in a voting scheme with a homomorphic tally. Last but not least, these constructions achieve a tight TCCA security property where the security of the scheme does not decrease as the number of challenges increases. This is important since our proof of receipt-freeness in the voting protocol of Chapter 3 relies on a single mTCCA transition.

Structure of the chapter The structure of this chapter is straightforward. Each section focuses on one construction, where we give an intuition of our design choices, the formal description and then the proof of security (except for the last construction, for which the proof can be found in [DPP24a]). Finally, we propose in Section 4.5 a comparison of all these constructions with the previous state-of-the-art, [Cha+16].

4.1. Generic construction

We are first exploring how a TREnc can be designed from existing tools. For this first construction, we actually build a wCCA-secure extractable TREnc. From Theorem 1, such a TREnc is TCCA-secure if it is strongly randomizable as well. We target an extractable TREnc instead of a more basic TREnc for a practical reason. In the proof of TCCA-security, one has to embed at some point a DDH challenge hiding the original challenge ballot to be randomized. However, the traceable security makes it hard to build a convincing ciphertext that has the same trace as this is an unusual properties that we do not naturally find in existing notions. With the additional link key extraction algorithm, this solves the issue even if the extractibility was not our goal.

4.1.1. Intuition

As a first natural building block, we use a signature on randomizable ciphertexts (SRC), as introduced by Blazy et al. [Bla+11]. In an SRC, any signed ciphertext can be publicly re-randomized, and the signature can be publicly adapted so that it remains valid for the new ciphertext.

We can easily obtain the structure of a TREnc from an SRC by defining the LGen function as setting lk as a fresh signing key for the SRC, and the LEnc function as encrypting the plaintext using a randomizable encryption scheme, then signing that ciphertext using lk. The trace of a ciphertext would then be the signature verification key.

This offers a promising skeleton, but it is not sufficient to obtain wCCA security: as it is, the adversary could simply remove the signature from the challenge ciphertext, sign that ciphertext with a fresh key in order to obtain a different trace, and ask for the decryption of the result, which would be granted.

A natural solution to this problem is to link the trace to the ciphertext using tag-based encryption [Kil06] mechanism and use a CHK transformation [CHK04]. In a tag-based encryption scheme, the encryption and decryption functions take an arbitrary tag as an extra input, and the decryption of a ciphertext with an incorrect tag will fail. We rely on the standard notion of weak-CCA security for tag based encryption [MRY04], which is the CCA security game excepted that the challenge ciphertext is produced using an *adversarially chosen* tag, and that no decryption query can be made using that tag (only) *after* the challenge phase. This security game nicely fits our wCCA security game, in which the trace derived from the link key submitted by the adversary can be used as a tag, and guarantees that no ciphertext can be modified in such a way that it successfully decrypts with a tag that is different from the original one. We note that we must be able to decrypt pre-challenge queries that already contain the "challenge tag" of the adversary, which prevents us from only relying on (weak) selective-tag security.

But we still need to be able to extract the link key from a tag-based ciphertext. This can be done fairly easily, by augmenting our encryption process with the requirement to encrypt the link key using a CPA-secure encryption scheme, and to add a non-interactive zero-knowledge proof that the encrypted link key is indeed the one that is used as tag for the tag-based encryption. Extraction would then simply proceed by decrypting that CPA ciphertext. We do not need this part to be rerandomizable, as everything is only related to the trace and not to the encrypted message. This full separation between the message and the trace extraction is exactly what allows us to reach the trapdoor adaptation property.

4.1.2. Building Blocks

More precisely, we build an extractable TREnc from the following ingredients:

A tag-based encryption scheme $(\mathsf{TBGen}, \mathsf{TBEnc}, \mathsf{TBDec})$.

We use a TBE scheme as defined in Definition 9 of Section 2.2.8. We also require the scheme to have two additional algorithms, a randomizing algorithm TBRand and a verification algorithm TBVer. More precisely,

TBRand provides statistical randomizability for the same tag of a given ciphertext as long as it is in the range of honest encryptions.

Regarding the verifiability, we desire an analogous property to the TREnc's verifiability. That is, for any PPT adversary that receives a pair of public/secret keys as input and outputs a ciphertext c_{tbe}, the probability that $\mathsf{TBVer}(\mathsf{pk}_{tbe}, c_{tbe}) = 1$ and c_{tbe} is not in the range of $\mathsf{TBEnc}(\mathsf{pk}, \cdot, \cdot)$ is bounded by a negligible function $\varepsilon_{tbe-ver}$.

For confidentiality, the tag-based encryption must be TBE-wCCA secure. More precisely, the advantage of any PPT adversary in the TBE-wCCA experiment (see Figure 2.3) is bounded by a negligible function $\varepsilon_{tbe-wCCA}$.

A signature on randomizable ciphertexts scheme $(\mathsf{SGen}, \mathsf{Sign}, \mathsf{SVer})$.

Altogether with the tag-based encryption scheme, we require the signature on ciphertext scheme to form an SRC as defined in Definition 3 of Section 2.2.3 that has the signature adaptation property (regarding a signature adapting algorithm SAdapt).

Additionally, we desire a signature scheme for which there exists an injective key derivation function ϕ that computes the verification key vk from the signing key sk: $\mathsf{vk} = \phi(\mathsf{sk})$. Since the key derivation function is injective, every verification key is associated with an unique signing key.

We further require the signature scheme to be verifiable in a stronger sense. More precisely, for any PPT adversary the following probability:

$$\Pr\left[\begin{array}{c}(\mathsf{ek}_{tbe}, \mathsf{dk}_{tbe}) \leftarrow \mathsf{TBGen}(1^\lambda); (\mathsf{vk}, c_{tbe}, \sigma) \leftarrow \mathcal{A}(\mathsf{ek}_{tbe}, \mathsf{dk}_{tbe}) : \\ \mathsf{SVer}(\mathsf{vk}, \mathsf{ek}_{tbe}, c_{tbe}) = 1 \text{ and} \\ \forall \mathsf{sk} \in \mathsf{SGen}(1^\lambda), \mathsf{vk} \neq \phi(\mathsf{sk}) \text{ or } \sigma \notin \mathsf{Sign}(\mathsf{sk}, \mathsf{ek}_{tbe}, c_{tbe})\end{array}\right]$$

is bounded by a negligible function ε_{S-ver}.

Finally, the signature on ciphertexts scheme should be unforgeable: the advantage of any PPT adversary in the unforgeability experiment is bounded by a negligible function ε_{UF}.

A randomizable CPA-secure encryption scheme $(\mathsf{EGen}, \mathsf{EEnc}, \mathsf{EDec})$.

We also use an IND-CPA-secure encryption scheme as described in Definition 2.2.1. We note the bound on the advantage of any PPT adversary at the IND-CPA experiment ε_{CPA}.

A NIZK proof system $(\mathsf{PGen}, \mathsf{PProve}, \mathsf{PVerify})$.

We use a NIZK proof system as defined in Section 2.2.5. We note SimGen and $\mathsf{SimProve}$ the zero-knowledge simulators and ε_{ZK} as an upper bound of the advantage of any adversary at the zero-knowledge game. We

also require a similar verifiability notion as the TREnc. More precisely, for any PPT adversary that receives an honestly generated sound crs as input and outputs a proof π for a given statement st, the probability that $\mathsf{PVerify}(\mathsf{crs}, x, \pi) = 1$ and π is not in the range of $\mathsf{PProve}(\mathsf{crs}, st, w)$ for a valid witness is bounded by a negligible function ε_{P-ver}. This is a stronger form of soundness.

We need this proof system to prove statements of the form: $((\mathsf{ek}_{extr}, c_{extr}, \mathsf{vk}), (\mathsf{sk}, r)) \in R \iff \mathsf{vk} = \phi(\mathsf{sk})$ and $c_{extr} = \mathsf{Enc}(\mathsf{ek}_{extr}, \mathsf{sk}; r)$.

We stress that we do not need any form of malleability or simulation-soundness, in contrast to the proof systems that we will use in the following constructions.

4.1.3. Description

Our generic extractable TREnc Π_G follows.

Setup(1^λ): Generate the public parameters pp of the building blocks.

Gen(1^λ, pp): Generate the keys $(\mathsf{ek}_{tbe}, \mathsf{dk}_{tbe}) \leftarrow\!\$\ \mathsf{TBGen}(1^\lambda, \mathsf{pp})$, $(\mathsf{ek}_{extr}, \mathsf{dk}_{extr}) \leftarrow\!\$\ \mathsf{EGen}(1^\lambda, \mathsf{pp})$, and finally a CRS $\mathsf{crs} \leftarrow\!\$\ \mathsf{PGen}(1^\lambda, \mathsf{pp})$ of the proof system. Set $\mathsf{SK} = (\mathsf{dk}_{tbe})$, and $\mathsf{PK} = (\mathsf{ek}_{tbe}, \mathsf{ek}_{extr}, \mathsf{crs})$.

Enc(PK, m): This is computed as $\mathsf{LEnc}(\mathsf{PK}, \mathsf{LGen}(\mathsf{PK}), m)$.
LGen(PK): Generate a verifying/signing key $\mathsf{vk}, \mathsf{sk} \leftarrow\!\$\ \mathsf{SGen}(1^\lambda, \mathsf{pp})$ of the SRC. Output $\mathsf{lk} = \mathsf{sk}$.
LEnc(PK, lk = sk, m):

1. Compute $\mathsf{vk} = \phi(\mathsf{sk})$, and $c_{tbe} \leftarrow \mathsf{TBEnc}(\mathsf{ek}_{tbe}, \mathsf{vk}, m)$.
2. Compute $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, c_{tbe})$ to get an SRC ciphertext.
3. Pick a random coin r and compute $c_{extr} = \mathsf{EEnc}(\mathsf{ek}_{\mathsf{extr}}, \mathsf{sk}; r)$.
4. Generate a proof $\pi \leftarrow \mathsf{PProve}(\mathsf{crs}, (\mathsf{ek}_{extr}, c_{extr}, \mathsf{vk}), (\mathsf{sk}, r))$ given crs, the encrypted message sk inside c_{extr} under the randomness r, to ensure that $((\mathsf{ek}_{extr}, c_{extr}, \mathsf{vk}), (\mathsf{sk}, r)) \in R$.
5. Output the ciphertext $\mathsf{CT} = (\mathsf{vk}, c_{tbe}, \sigma, c_{extr}, \pi)$.

Ver(PK, CT): Run the verification algorithm of each scheme and output 1 if and only if they all returned 1.

Dec(sk, CT): If $\mathsf{Ver}(\mathsf{PK}, \mathsf{CT}) = 0$, output $\perp$. Otherwise, return $\mathsf{Dec}(\mathsf{dk}_{tbe}, \mathsf{vk}, c_{tbe})$.

Trace(PK, CT): If $\mathsf{Ver}(\mathsf{PK}, \mathsf{CT}) = 0$, output $\perp$. Otherwise, output vk.

Rand(PK, CT): Randomize c_{tbe}: $c'_{tbe} \leftarrow \mathsf{TBRand}(\mathsf{ek}_{tbe}, c_{tbe}; r'_{tbe})$ with random coin r'_{tbe} and adapt the signature: $\sigma' \leftarrow \mathsf{SAdapt}(\mathsf{vk}, \mathsf{ek}_{tbe}, \sigma, c_{tbe}; r'_{tbe}, s'_{tbe})$ with random coin s'_{tbe}. vk, c_{extr} and π do not need to be re-randomized as they are only related to the trace and not to the encrypted message. Return $\mathsf{CT}' = (\mathsf{vk}, c'_{tbe}, \sigma', c_{extr}, \pi)$.

Extractability:

TrapGen(pk): Generate a new key $(\mathsf{ek}'_{extr}, \mathsf{dk}'_{extr}) \leftarrow\!\!\$\, \mathsf{EGen}(1^\lambda, \mathsf{pp})$ and from $\mathsf{PK} = (\mathsf{ek}_{tbe}, \mathsf{ek}_{extr}, \mathsf{crs})$, set $\mathsf{PK}' = (\mathsf{ek}_{tbe}, \mathsf{ek}'_{extr}, \mathsf{crs})$ and $\mathsf{TK} = \mathsf{dk}'_{extr}$.

LExtr(TK, CT): Output $\mathsf{lk} = \mathsf{EDec}(\mathsf{dk}_{extr}, c_{extr})$.

TrapAdapt(TK, CT, pk′): For $\mathsf{CT} = (\mathsf{vk}, c_{tbe}, \sigma, c_{extr}, \pi)$, run $\mathsf{lk} \leftarrow \mathsf{LExtr}(\mathsf{TK}, \mathsf{CT})$, compute $c'_{extr} = \mathsf{Enc}(\mathsf{ek}'_{extr}, \mathsf{sk}; r)$ and generate a proof $\pi' \leftarrow \mathsf{PProve}(\mathsf{crs}, (\mathsf{ek}'_{extr}, c'_{extr}, \mathsf{vk}), (\mathsf{sk}, r))$.

4.1.4. Security

Theorem 7. *Π_G is verifiable. More precisely, for any PPT adversary $\mathcal{A}$, we have $Pr[\mathsf{Exp}^{\mathrm{ver}}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \varepsilon_{tbe-ver} + \varepsilon_{S-ver} + \varepsilon_{P-ver}$.*

Proof. Let $\mathcal{A}$ be a PPT adversary. Given $\mathsf{PK}, \mathsf{SK} \leftarrow\!\!\$\, \mathsf{Gen}(1^\lambda)$, we have to show that the probability that $\mathcal{A}$ outputs a ciphertext CT such that $\mathsf{Ver}(\mathsf{PK}, \mathsf{CT}) = 1$ but that is not in the range of $\mathsf{Enc}(\mathsf{PK}, \cdot)$ is negligible. First, if $\mathsf{Ver}(\mathsf{PK}, \mathsf{CT}) = 1$ then $\mathsf{TBVer}(\mathsf{ek}_{tbe}, \mathsf{vk}, c_{tbe}) = 1$, $\mathsf{SVer}(\mathsf{vk}, \mathsf{ek}, c, \sigma) = 1$ and $\mathsf{PVerify}(\mathsf{crs}, \pi, (\mathsf{ek}_{extr}, c_{extr}, \mathsf{vk})) = 1$. We can then reduce each of the following points to the corresponding verifiability game:

1. There exists an unique $\mathsf{sk} \in \mathsf{SGen}(1^\lambda)$ such that $\mathsf{vk} = \phi(\mathsf{sk})$ and σ is in the range of $\mathsf{Sign}(\mathsf{sk}, \mathsf{ek}, c_{tbe})$, except with a probability bounded by ε_{S-ver}.

2. c_{tbe} is in the range of $\mathsf{TBEnc}(\mathsf{pk}, \mathsf{vk}, \cdot)$, except with a probability bounded by $\varepsilon_{tbe-ver}$.

3. π is in the range of $\mathsf{PProve}(\mathsf{crs}, (\mathsf{ek}_{extr}, c_{extr}, \mathsf{vk}), w)$ for a valid witness $w = (\mathsf{sk}', r')$, except with a probability bounded by ε_{P-ver}. Since ϕ is injective, $\mathsf{sk}' = \mathsf{sk}$ and $c_{extr} = \mathsf{Enc}(\mathsf{ek}_{extr}, \mathsf{sk}; r')$.

Putting everything together, we conclude that c is in the range of $\mathsf{Enc}(\mathsf{PK}, \cdot)$ except if one of those events happens. Thus, $Pr[\mathsf{Exp}^{\mathrm{ver}}_{\mathcal{A},\Pi_G}(\lambda) = 1] \leq \varepsilon_{tbe-ver} + \varepsilon_{S-ver} + \varepsilon_{P-ver}$. □

Theorem 8. *Π_G is traceable. More precisely, for any PPT adversary $\mathcal{A}$, we have $Pr[\mathsf{Exp}^{\mathrm{trace}}_{\mathcal{A},\Pi_G}(\lambda) = 1] \leq \varepsilon_{ZK} + \varepsilon_{CPA} + \varepsilon_{UF}$.*

Proof. Let $\mathcal{A}$ be an efficient adversary against the traceability of our TREnc Π_G. We consider a sequence of games. In Game i, we denote by S_i the event that $\mathcal{A}(\mathsf{PK}, \mathsf{SK})$ wins the game.

Game 0: We define $\mathsf{Game}_0(\lambda)$ as the original traceability experiment $\mathsf{Exp}^{\mathsf{trace}}_{\mathcal{A},\Pi_G}(\lambda)$ as defined in Definition 23. By definition, $\mathcal{A}$ wins the game $Pr[S_0] = \mathsf{Exp}^{\mathsf{trace}}_{\mathcal{A},\Pi_G}(\lambda)$. In this game, the adversary sends a message m and we return a ciphertext $\mathsf{CT} \leftarrow \mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, m)$ for a random $\mathsf{lk} \leftarrow \mathsf{LGen}(\mathsf{pk})$. The adversary wins the game by producing a valid ciphertext $\mathsf{CT}^* = (\mathsf{vk}^\star, c^\star_{tbe}, \sigma^\star, c^\star_{extr}, \pi^\star)$ that traces to CT but $\mathsf{Dec}(\mathsf{SK}, \mathsf{CT}^*) \neq m$.

Game 1: In this first game, we use the zero-knowledge simulators $\mathsf{SimSetup}$ and $\mathsf{SimProve}$ to simulate the proof π of CT. From the zero-knowledge property of the proof system, we directly have that $|Pr[S_1] - Pr[S_0]| \leq \varepsilon_{ZK}$.

Game 2: We modify $\mathsf{Game}_1(\lambda)$ into $\mathsf{Game}_2(\lambda)$ by modifying the way c_{extr} is computed. Instead of an encrypting sk, we now encrypt a random signing key. We can directly reduce to the CPA security of the encryption scheme as we do not need a witness to compute π anymore, thus $|Pr[S_2] - Pr[S_1]| \leq \varepsilon_{CPA}$.

We finally reduce to the unforgeability of the signature scheme. We build our adversary as follows. First, we receive $(\mathsf{ek}, \mathsf{dk})$ as well as a verification key from the unforgeability challenger. We run the parts of the Gen algorithm of the TREnc that have not been generated yet, that is we generate $(\mathsf{ek}_{extr}, \mathsf{dk}_{extr})$ and the crs. Then, we send the resulting public key to $\mathsf{Game}_2(\lambda)$ adversary and receive a message m.

We now run the encryption algorithm with m, but instead of computing ourselves the signature in step 2, we ask the unforgeability challenger to produce the signature. Finally, we receive the forged ciphertext CT^* from the adversary and extract the signature $\sigma^\star$ which we use as our forgery. If the traceability adversary returns a valid ciphertext with the same trace as CT, then $\mathsf{SVer}(\mathsf{vk}, \mathsf{ek}^\star_{tbe}, c^\star_{tbe}, \sigma^\star) = 1$. Also, if it wins and returns a ciphertext that does not decrypt into m, then we win our experiment as well since this message has not been queried .

Finally, $Pr[S_4]$ is bounded by $\varepsilon_{UF}(\lambda)$ and the TREnc is traceable. □

Theorem 9. *Π_G is strongly randomizable. More precisely, for every $c \in \mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, m)$ with pk in the range of Gen and lk in the range of $\mathsf{LGen}(\mathsf{pk})$, the following statistical indistinguishability relation holds:*

$$\mathsf{Rand}(\mathsf{pk}, c) \approx_s \mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, m)$$

Proof. Since pk and c are respectively in the range of Gen and Enc, we parse $\mathsf{pk} = (\mathsf{ek}_{tbe}, \mathsf{ek}_{extr}, \mathsf{crs})$ and c as $(\mathsf{vk}, c_{tbe}, \sigma, c_{extr}, \pi)$ where $\mathsf{vk} = \phi(\mathsf{lk})$ for some link key lk.

First, $(\mathsf{vk}, c_{extr}, \pi)$ is left untouched in the Rand algorithm and is computed identically as in a fresh encryption.

Then, (c_{tbe}, σ) is in the range of $(\mathsf{TBEnc}(\mathsf{ek}_{tbe}, \mathsf{vk}, m), \mathsf{Sign}(\mathsf{lk}, c_{tbe}))$. As the tag-based encryption scheme coupled to the signature scheme form a SRC, the signature adaptation property guarantees that the distributions $\{\mathsf{TBEnc}(\mathsf{ek}_{tbe}, \phi(\mathsf{lk}), m; r'), \mathsf{Sign}(\mathsf{lk}, \mathsf{ek}_{tbe}, c'; s')\}$ and $\{\mathsf{TBRand}(\mathsf{ek}_{tbe}, \mathsf{vk}, c; r'), \mathsf{SAdapt}(\mathsf{vk}, \mathsf{ek}_{tbe}, c_{tbe}, \sigma; r', s')\}$ are statistically indistinguishable.

□

Theorem 10. *Π_G is extractable. More precisely, Π_G satisfies the trapdoor indistinguishability, trapdoor adaptation and unique extractability properties.*

Proof.

Trapdoor indistinguishability The public key can be decomposed as $(\mathsf{pk}_{trenc}, \mathsf{pk}_{extr}) = ((\mathsf{ek}_{tbe}, \mathsf{crs}), \mathsf{ek}_{extr})$, where $\mathsf{sk} = \mathsf{dk}_{tbe}$ is the secret key associated to ek_{tbe} and is independent of ek_{extr}. Hence, $(\mathsf{pk}', \mathsf{sk}) = (((\mathsf{ek}_{tbe}, \mathsf{crs}), \mathsf{ek}'_{extr}), \mathsf{sk})$ outputted by $\mathsf{TrapGen}$ has the same distribution as $(\mathsf{pk}, \mathsf{sk})$.

Trapdoor adaptation Given $c = \mathsf{TrapAdapt}(\mathsf{lk}, \mathsf{Enc}(\mathsf{pk}, \mathsf{lk}, m))$ and $c' = \mathsf{LEnc}(\mathsf{pk}', \mathsf{lk}, m)$, the c_{extr} and π part are computed identically, either in the $\mathsf{TrapAdapt}$ in the former case, or directly in the latter case.

Also the $(\mathsf{vk}, c_{tbe}, \sigma)$ parts of the ciphertexts are computed identically in the Enc algorithm since it does not use the adapted key.

Unique Extraction For $(\mathsf{pk}, \mathsf{tk}) = ((\mathsf{ek}_{tbe}, \mathsf{ek}_{extr}, \mathsf{crs}), \mathsf{dk}_{extr})$ in the range of $\mathsf{TrapGen}$, the second entry of pk and tk form a pair of public and secret keys of the CPA-secure scheme.

- If a ciphertext c is in the range of $\mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, \cdot)$, then the c_{extr} part of c is also in the range of $\mathsf{EEnc}(\mathsf{ek}_{extr}, \mathsf{lk})$. From the correctness of the CPA-secure encryption scheme, $\mathsf{LExtr}(c, \mathsf{tk}) = \mathsf{EDec}(\mathsf{dk}_{extr}, c_{extr}) = \mathsf{lk}$.
- Since c_0 and c_1 are both in the range of $\mathsf{Enc}(\mathsf{pk}, \cdot)$, there are two link keys lk_0 and lk_1 in the range of $\mathsf{LGen}(\mathsf{pk})$ such that $c_i \in \mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}_i, \cdot)$ for $i = 0, 1$. Also, they both have a component vk_i such that $\mathsf{vk}_i = \phi(\mathsf{lk}_i)$. Since $\mathsf{Trace}(\mathsf{pk}, c_0) = \mathsf{Trace}(\mathsf{pk}, c_1)$, $\mathsf{vk}_0 = \mathsf{vk}_1$ and the injectivity of ϕ guarantee that $\mathsf{lk}_0 = \mathsf{lk}_1$. Finally, the first item of the definition implies that $\mathsf{LExtr}(\mathsf{tk}, c_0) = \mathsf{lk}_0 = \mathsf{lk}_1 = \mathsf{LExtr}(\mathsf{tk}, c_1)$.

□

Theorem 11. Π_G *is* wCCA*-secure. More precisely, for any adversary* PPT $\mathcal{A}$*, we have* $Pr[\mathsf{Exp}^{\mathrm{wcca}}_{\mathcal{A},\Pi_G}(\lambda) = 1] \leq \varepsilon_{tbe-wCCA}$.

Proof. The proof is a direct reduction from the TBE-wCCA property of the tag-based encryption scheme used in the construction. Let $\mathcal{A}$ be an adversary for the wCCA experiment of Π_G. Using $\mathcal{A}$, we build an adversary for the TBE-wCCA experiment that also wins with the same advantage.

At the beginning of the experiment, we receive from the challenger the pair of public/private key of the tag-based encryption scheme used by the challenger. In addition to these keys, we run the parts of the Gen algorithm of the TREnc that have not been generated yet, that is we generate $(\mathsf{ek}_{extr}, \mathsf{dk}_{extr})$ and the crs.

Now for each decryption request that we receive from $\mathcal{A}$, we send vk, c_{tbe} to the TBE-wCCA challenger and return the output to $\mathcal{A}$. As we define our trace as the vk component of a ciphertext, the post-challenge decryption requests abort in the same cases.

At some point, we receive the challenge (m_0, m_1, lk) from $\mathcal{A}$ and compute $\mathsf{vk} = \phi(\mathsf{lk})$. We then forward (m_0, m_1, vk) to the challenger as our challenge and receive a tag-based encryption $c^{\star}_{tbe}$ of either m_0 or m_1. We then generate a signature σ on $c^{\star}_{tbe}$ using lk and compute an encryption of lk as well as the accompanying proof respectively with $c_{extr} = \mathsf{EEnc}(\mathsf{ek}_{extr}, \mathsf{lk}; r)$ and $\pi \leftarrow\!\$\, \mathsf{PProve}(\mathsf{crs}, (\mathsf{ek}_{extr}, c_{extr}, \mathsf{vk}), (\mathsf{lk}, r))$. The ciphertext $\mathsf{CT}^{\star} = (\mathsf{vk}, c^{\star}_{tbe}, \sigma, c_{extr}, \pi)$ forms a challenge ciphertext that has the same distribution as ciphertexts directly computed with the Enc algorithm of the TREnc.

As the probability of our adversary to win the TBE-wCCA experiment is the same as the probability of $\mathcal{A}$ to win the wCCA experiment of the TREnc, we conclude that the advantage of $\mathcal{A}$ has to be bounded by $\varepsilon_{wCCA}(\lambda)$. □

Since Π_G is extractable and wCCA-secure, it follows from Theorem 1 that Π_G is TCCA-secure as well.

4.1.5. Instantiation

To build an extractable TREnc, we briefly sketch how to instantiate each building blocks in asymmetric bilinear groups. For the SRC, we start from BeleniosRF and use the Waters' signature scheme [Wat05]

adapted to asymmetric pairings [Bla+11].[13] For the weak CCA secure randomizable tag-based encryption, we depart from BeleniosRF [Cha+16] as their TBE is only selective-tag weakly CCA secure. While it is easy to show that this is enough to generically imply RCCA security, it is not enough to reach our natural variant of wCCA, even without extractor, which is necessary to get TCCA security. We thus borrow the scheme due to [MRY04], which is the Cramer-Shoup ciphertext where the hashed value is replaced by the tag. It is easy to see that this provides our desired notion of randomizability and that we can turn it into a randomizable SRC by adding the same additional components of [Cha+16] to allow adapting the signature after the randomization of the TBE. However, we also have to turn it into a publicly verifiable TBE/CRS, and include a randomizable simulation-sound proof that we indeed encrypt the Waters' hash of the signed message that is decomposed bit by bit. As in BeleniosRF, we can rely on the Groth-Sahai proof system [GS08] which provides perfect NIWI proofs for quadratic statement of pairing-product equations (when the CRS is generated in the hiding mode). To turn them into a simulation-sound zero-knowledge (ZK) proof, we can combine them with a structure-preserving signature on a Waters' public key, and prove that either the statement is true or that we have such a valid signature as suggested in [Cha+12].

The remaining part is more straightforward. We can use a perfectly randomizable ElGamal encryption to encrypt the Waters' signing key (which is a single group element) and use its decryption key for extraction. For the randomizable NIZK proof that we indeed faithfully encrypt the signing key, we can take the same modified Groth-Sahai proof system described above.

This section showed that the notion of extractable TREnc offers a convenient companion for a TREnc: it is possible to build an extractable TREnc from relatively common, yet strong, building blocks, and the proof of TCCA security of this TREnc comes relatively easily because we can naturally use a wCCA security notion for an extractable TREnc that implies our new TCCA security notion. The resulting construction is however expected to be fairly expensive since, in the standard model, all known instantiations of the building blocks rely on a bit-by-bit decomposition of the message or the secret singing key of which the ciphertext must contain a ZK proof. Nevertheless, providing this extractability feature is an artifice for the construction that is not necessary for the security of the TREnc, but as far as we know there is no obvious generic

[13] It is easy to see that the signature has all the security guarantees that we want. Valid signatures are always in the range of honest signatures, and the public key unequivocally defines the secret key.

construction leading to a TREnc without extractability. In the next section we turn to the construction of an ad-hoc efficient instance of a TREnc based on a standard computational assumption that also avoids the costly bit-by-bit decomposition.

4.2. Trusted setup construction

This section provides a secure TREnc in the standard model, only relying on the SXDH assumption and on a CRS. Contrary to our previous construction, this one is not extractable – extractability was just a convenience but does not offer any security benefit. This allows us to get a more efficient solution, here, in asymmetric bilinear groups.

Moreover, our construction enjoys a short public-key and short ciphertexts as they only contain a constant number of group elements to encrypt a full group element, contrary to previous proposals that required processing the message bit by bit [Bla+11; Cha+16].

We will later describe in this chapter a TREnc construction with better properties, such as a public coin CRS and a tight TCCA security. The construction described here is only slightly more efficient, but is still relevant in this thesis from a historical point of view since it was the first efficient TREnc to be described.

4.2.1. Intuition

A ciphertext of our construction has the form $\mathsf{CT} = (\boldsymbol{c}, \mathit{trace}, \boldsymbol{\sigma}_{\mathsf{trace}}, \boldsymbol{\pi}_{\mathsf{valid}})$, where we defined $\mathsf{Trace}(c)$ as outputting trace. The $\boldsymbol{c}$ part is a randomizable CPA encryption of the message $m \in \mathbb{G}$ and $\boldsymbol{\pi}_{\mathsf{valid}}$ is a simulation-sound proof that the $\boldsymbol{c}$ part is well-formed. Finally, $\boldsymbol{\sigma}_{\mathsf{trace}}$ is related to the traceability property of the TREnc and can be seen as a signature on $\boldsymbol{c}$, for which the verification key is actually the trace of the ciphertext. For the TCCA property, we need of course to be able to randomize $\boldsymbol{c}, \boldsymbol{\sigma}_{\mathsf{trace}}$ and $\boldsymbol{\pi}_{\mathsf{valid}}$, but for the last two elements, we also need them to adapt to the randomization of $\boldsymbol{c}$.

We build the CPA part as a vector $\boldsymbol{c} = (c_0, c_1, c_2)$ of the form $c_0 = m \cdot f^\theta$, $c_1 = g^\theta$, $c_2 = h^\theta$. For this part to be well-formed, we need that $(c_1, c_2) \in \langle (g, h) \rangle$.

For the proof of validity, we give the intuition in several steps.

- The first idea is to include a one-time LHSP signature on top of $\boldsymbol{c}$ as first suggested in [Lib+13]. That means that the public key contains an LHSP signature $\boldsymbol{\Sigma}$ on (g, h) so that we can derive a signature on $(g, h)^\theta$ if indeed (c_1, c_2) lies in $\mathrm{span}\langle (g, h) \rangle$ by computing $\boldsymbol{\pi}_{\mathsf{valid}}$

as $\boldsymbol{\pi} = \boldsymbol{\Sigma}^{\theta}$. Here, the public key includes a CRS that contains a signature on the basis of the linear subspace $\mathrm{span}\langle(g, h)\rangle$ of $\mathbb{G}^2$. Given $\boldsymbol{c}$ and the LHSP signature $\boldsymbol{\pi}_{\mathsf{valid}}$ one can easily randomize the ciphertext as follows: compute $\boldsymbol{c}' = \boldsymbol{c} \cdot (f, g, h)^{\theta'}$, and adapt the proof $\boldsymbol{\pi}' = \boldsymbol{\pi} \cdot \boldsymbol{\Sigma}^{\theta'}$. While this solution is perfectly randomizable and the signing key allows us to perfectly simulate the proof, it only provides a CCA1 security.[14] Indeed, the proof is not simulation sound: if we create a ciphertext $(c_1^\star, c_2^\star) \notin \mathrm{span}\langle(g, h)\rangle$ in one our hybrid in the hope of hiding the challenge ciphertext, $\boldsymbol{\pi}_{\mathsf{valid}}$ would give a signature of a linearly independent vector. An adversary could then ask decryption of badly formed ciphertexts as a linear combination of (g, h) and $(c_1^\star, c_2^\star)$ that span $\mathbb{G}^2$. That is, the adversary could theoritically create fake proofs which could leak the secret key.

- While the first idea does not work, this technique has been enhanced to provide a tag-based simulation-sound proof system which is reminiscent to building CCA-like secure encryption. The underlying technique is to generate a tag from the trace of the ciphertext as $\tau = H(trace)$ for some collision-resistant hash function H, and use this tag to build the proof.

  A natural solution would be to borrow the first solution due to [Lib+14] but it only provides selective-tag simulation soundness. TCCA security implies that our underlying encryption must achieve tag-based wCCA security, as the tag $\tau^* = H(\boldsymbol{opk}^*)$ involved in the challenge ciphertext may be chosen by the adversary at any time. Furthermore, we must be able to answer *any* pre-challenge decryption queries, including even those that already used τ^*. That means that we cannot program the public key to embed τ^* that will help us to incorporate an SXDH instance in the computation of the challenge ciphertext. Specifically, we cannot use "partitionning" techniques, and in particular, programmable hash function [HK08].

- Instead, we include a signature $\boldsymbol{\Sigma}_u$ on $(g, h, 1, 1)$ and another signature $\boldsymbol{\Sigma}_v$ on $(1, 1, g, h)$ in the public key. Given a tag τ, the computation of $\boldsymbol{\pi} = (\boldsymbol{\Sigma}_u^{\tau}\boldsymbol{\Sigma}_v)^{\theta}$ due to [KW15] is an the LHSP signature on $(c_1^{\tau}, c_2^{\tau}, c_1, c_2)$ which gives us the expected security. If we give a fake proof to the TCCA adversary for a ciphertext such

[14] CCA1 security is a weaker variant of CCA security, in which the adversary cannot make decryption request after the challenge. Comparatively, in the TCCA game, the adversary cannot make decryption requests of ciphertexts with the same trace as the challenge.

that $(c_1^{\star\tau}, c_2^{\star\tau}, c_1^{\star}, c_2^{\star})$ is not in the span of $\mathrm{span}\langle(g^\tau, h^\tau, g, h)\rangle$, the adversary still does not have signatures for the full space of $\mathbb{G}^4$ and will not be able to build a fake signature for another τ. It can thus only create fake proofs for ciphertexts that have the same trace as the challenge ciphertext, which will not be decrypted in the TCCA game anymore.

Now, we come back on the signature $\boldsymbol{\sigma}_{\mathsf{trace}}$ of $(\boldsymbol{c}, \boldsymbol{\pi})$, for which we define a pair of signing and verification keys $(\mathbf{ovk}, \mathbf{osk})$ generated by the LEnc algorithm. We set the *trace* as $\mathbf{ovk}$

We again give the intuition in several steps.

- It would be easy to use a strongly unforgeable signature scheme providing non-malleability of ciphertext and thus the traceability of the TREnc. Unfortunately, we need to be able to adapt the signature to the randomized CPA part when we randomize the ciphertext.

- Here again, LHSP signature scheme comes in handy. If our fresh key pair $(\mathbf{ovk}, \mathbf{osk})$ is generated from a one-time LHSP signature scheme, we can fix the message and preserve the randomizability of $(\boldsymbol{c}, \boldsymbol{\pi})$ by computing one-time LHSP signatures $\boldsymbol{\sigma}_1$ on (g, c_0, c_1, c_2) and $\boldsymbol{\sigma}_2$ on $(1, f, g, h)$. This way, when we randomize $(\boldsymbol{c}, \pi)$ as $\boldsymbol{c'} = \boldsymbol{c} \cdot (f, g, h)^{\theta'}$, $\pi' = \pi \cdot (\boldsymbol{\Sigma}_u^\tau \boldsymbol{\Sigma}_v)^{\theta'}$ we can also adapt the signature $(\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2)$ as $\boldsymbol{\sigma}'_1 = \boldsymbol{\sigma}_1 \cdot \boldsymbol{\sigma}_2^{\theta'}$ to remain valid for the vector $(g, c'_0, c'_1, c'_2) = (g, c_0, c_1, c_2) \cdot (1, f, g, h)^{\theta'}$, and simply keep $\boldsymbol{\sigma}_2$.

  While the correctness follows by inspection, the non-malleability of the message is due to the presence of the constant g that must be the first component of the signed vector associated to $\boldsymbol{\sigma}_1$ and any adaptation $\boldsymbol{\sigma}'_1$. Modifying m requires computing a one-time LHSP signature on a vector necessarily outside the span generated by (g, c_0, c_1, c_2) and $(1, f, g, h)$ as long as g is the first component.

- However, this signature brings an issue in the proof of TCCA security. At some point, we want to embed a random triple (F, G, H) into the randomization of the challenge (c_0^*, c_1^*, c_2^*) such that the last two entries are not in the span of $\langle(g, h)\rangle$ anymore. While we can simulate the proof $\boldsymbol{\pi}_{\mathsf{valid}}$, it is statistically hard to craft a valid looking adaptation $\boldsymbol{\sigma}_1$ without knowing the signature key. In the generic construction, we proceeded by extracting this key, which would be expensive in our case.

  So instead, we include a third signature $\boldsymbol{\sigma}_3$ on $(1, F, G, H)$ in the ciphertext, where $(1, F, G, H)$ is given in the public key. This

allows us to derive a valid $\boldsymbol{\sigma}_1^* = \boldsymbol{\sigma}_1 \cdot \boldsymbol{\sigma}_2^{\theta^*} \cdot \boldsymbol{\sigma}_3^{\rho^*}$ when we compute $(g, c_0^*, c_1^*, c_2^*) = (g, c_0, c_1, c_2) \cdot (1, f, g, h)^{\theta^*} \cdot (1, F, G, H)^{\rho^*}$. Crucially, the third factor $(1, F, G, H)$ cannot be used by the adversary wihtout providing a related fake proof π as explained above.

(F, G) is in the $\mathrm{span}\langle(f, g)\rangle$, so it does not help the adversary to forge a signature for another message.

- Still, if we reveal $(1, F, G, H)$, $\boldsymbol{c}^* = (c_0^*, c_1^*, c_2^*)$ will not be random even if (F, G, H) is random in the TCCA game.

  Fortunately, it is sufficient for the traceability to use $(\mathsf{ovk}, \mathsf{osk})$ to sign the shorter vectors (g, c_0, c_1), $(1, f, g)$ and $(1, F, G)$, and keep H away from the adversary's view to have a statistically random $\boldsymbol{c}^*$ in the reduction. Thanks to the proof of validity, (c_1, c_2) is guaranteed to be in $\mathrm{span}\langle(g, h)\rangle$ and the non-malleability is preserved.

- Finally, we also need to hide $\boldsymbol{\sigma}_1$ by using a randomizable and adaptable non-interactive witness-indistinguishable Groth-Sahai proof. Indeed, given two ciphertexts in the TCCA game, we cannot be sure that the adversary used an honest execution of the signing algorithm with osk. Instead, it could put some hidden information in this $\boldsymbol{\sigma}_1$ signature which would remain after the randomization and use this to distinguish between the two randomized ciphertexts. Therefore, $\boldsymbol{\sigma}_{\mathsf{Trace}}$ contains a Groth-Sahai proof of the LHSP signature on (g, c, d) instead of the signature itself, which is perfectly randomizable.

  Fortunately, this is not an issue for σ_2 and σ_3 as for two ciphertexts with the same trace, they should be respectively equal (or we could break a double pairing problem) as the LHSP signature is deterministic.

We can now describe formally our construction called Π_{TS}, where TS stands for the *trusted setup* to differentiate with our next constructions that use a public coin CRS.

4.2.2. Description

$\mathsf{Gen}(1^\lambda)$: Run $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$. together with $g, h \leftarrow\!\$\, \mathbb{G}$ and $\hat{g}, \hat{h} \leftarrow\!\$\, \hat{\mathbb{G}}$.

1. Pick a random pair $\mathsf{sk} = (\alpha, \beta) \leftarrow\!\$\, \mathbb{Z}_q^2$ and set $f = g^\alpha h^\beta$.
2. Pick $\delta \leftarrow\!\$\, \mathbb{Z}_q$ and compute $(F, G, H) = (f, g, h)^\delta$.
3. Generate a tuple of 4 random group elements $\mathsf{crs}_\sigma = (\boldsymbol{w}_1, \boldsymbol{w}_2) \leftarrow\!\$\, \mathbb{G}^4$ seen as a Groth-Sahai CRS to commit to groups

elements over $\mathbb{G}$, where $\boldsymbol{w}_1 = (w_{11}, w_{12})$ and $\boldsymbol{w}_w = (w_{21}, w_{22})$ are generated in the perfect NIWI mode. For simplicity, we write Com for the commitment algorithm $\mathsf{Com}(\mathbf{crs}_\sigma, \cdot)$.

4. Define the vector $\boldsymbol{v} = (g, h)$ and generate 2 key pairs $(\mathbf{sk}_u, \mathbf{pk}_u)$ and $(\mathbf{sk}_u, \mathbf{pk}_u)$ for the one-time linearly homomorphic signature of Section 2.2.4 in order to sign vectors of dimension $n = 2$, given the common public parameters $\hat{g}, \hat{h}$. Let $\mathbf{pk}_u = \{\hat{u}_1, \hat{u}_2\}$ and $\mathbf{pk}_v = \{\hat{v}_1, \hat{v}_2\}$. Using $\mathbf{sk}_u$ (resp. $\mathbf{sk}_v$), generate a one-time LHSP signature $\boldsymbol{\Sigma}_u = (Z_u, R_u)$ (resp. $\boldsymbol{\Sigma}_v = (Z_v, R_v)$) on $\boldsymbol{v}$. In other words, for $\mathbf{pk}_{\mathsf{lhsp}}^{\mathsf{qazk}} = \{\hat{u}_1, \hat{u}_2, \hat{v}_1, \hat{v}_2\}$, $\boldsymbol{\Sigma}_u, \boldsymbol{\Sigma}_v$ are one-time LHSP signatures on the rows of the matrix

$$\boldsymbol{P} = \begin{pmatrix} g & h & 1 & 1 \\ 1 & 1 & g & h \end{pmatrix}.$$

The private key consists of $\mathsf{SK} = (\alpha, \beta)$ and the public key $\mathsf{PK} \in \mathbb{G}^{13} \times \hat{\mathbb{G}}^6$ is

$$\mathsf{PK} = \left(f,\ g,\ h,\ F,\ G,\ \mathbf{crs}_\sigma,\ \boldsymbol{\Sigma}_u,\ \boldsymbol{\Sigma}_v,\ \mathbf{pk}_{\mathsf{lhsp}}^{\mathsf{qazk}},\ \hat{g},\ \hat{h} \right).$$

$\mathsf{Enc}(\mathsf{PK}, \boldsymbol{m})$: Given a message $m \in \mathbb{G}$ to encrypt, first run $\mathsf{LGen}(\mathsf{PK})$: Generate a key pair $(\mathbf{osk}, \mathbf{ovk})$ for the one-time linearly homomorphic signature of Section 2.2.4 from the public generators $\hat{g}, \hat{h}$ in order to sign vectors of dimension 3. Let $\mathsf{lk} = \mathbf{osk} = \{(\eta_i, \zeta_i)\}_{i=1}^3$ be the private key, of which the corresponding public key is $\mathbf{ovk} = \{\hat{f}_i\}_{i=1}^3$. Then, conduct the following steps of $\mathsf{LEnc}(\mathsf{PK}, \mathsf{lk}, m)$:

1. Pick $\theta \leftarrow\!\!\$\, \mathbb{Z}_q$ and compute the CPA encryption $\boldsymbol{c} = (c_0, c_1, c_2)$, where $c_0 = mf^\theta$, $c_1 = g^\theta$ and $c_2 = h^\theta$. Keep the random coin θ.
   Next steps 2-3 are dedicated to the tracing part.
2. To allow tracing, authenticate the row space of the matrix $\boldsymbol{T} = \left(T_{i,j}\right)_{1 \le i,j \le 3}$

$$\boldsymbol{T} = \begin{pmatrix} g & c_0 & c_1 \\ 1 & f & g \\ 1 & F & G \end{pmatrix} \tag{4.1}$$

   by using $\mathsf{lk} = \mathbf{osk}$. Namely, sign each row $\boldsymbol{T}_i$ of $\boldsymbol{T}$ resulting in $\boldsymbol{\sigma} = (\boldsymbol{\sigma}_i)_{i=1}^3$, where $\boldsymbol{\sigma}_i = (Z_i, R_i) \in \mathbb{G}^2$.

3. To allow strong randomizability, commit to $\boldsymbol{\sigma}_1 = (Z_1, R_1)$ using the Groth-Sahai CRS $\mathbf{crs}_\sigma$. That is, compute $\boldsymbol{C}_Z = \iota(Z_1)\boldsymbol{w}_1^{z_1}\boldsymbol{w}_2^{z_2}$ and $\boldsymbol{C}_R = \iota(R_1)\boldsymbol{w}_1^{r_1}\boldsymbol{w}_2^{r_2}$. To ensure that $\boldsymbol{\sigma}_1$ is a valid one-time LHSP signature on (g, c_0, c_1) compute the proof $\hat{\boldsymbol{\pi}}_{\mathsf{sig}} = (\hat{P}_1, \hat{P}_2) \in \hat{\mathbb{G}}^2$ such that $\hat{P}_1 = \hat{g}^{z_1}\hat{h}^{r_1}$ and $\hat{P}_2 = \hat{g}^{z_2}\hat{h}^{r_2}$. *The next step shows the validity of* $\boldsymbol{c}$ *associated to* $\mathbf{ovk}$.
4. Given θ and $\tau = H(\mathbf{ovk})$, compute a randomizable simulation-sound proof that $(c_1, c_2) \in \mathrm{span}\langle(g, h)\rangle$. Namely, derive the LHSP signature $\pi = (\boldsymbol{\Sigma}_u^\tau \boldsymbol{\Sigma}_v)^\theta =: (Z_\pi, R_\pi)$ on the vector $(c_1^\tau, c_2^\tau, c_1, c_2) = ((g, h, 1, 1)^\tau (1, 1, g, h))^\theta$. We note $\boldsymbol{\pi}_{\mathsf{valid}} = (Z_\pi, R_\pi) \in \mathbb{G}$.

Output the ciphertext

$$\mathsf{CT} = \left(\boldsymbol{c}, \mathbf{ovk}, \boldsymbol{\sigma}_{\mathsf{trace}}, \boldsymbol{\pi}_{\mathsf{valid}}\right) \in \mathbb{G}^{13} \times \hat{\mathbb{G}}^5$$

where $\boldsymbol{\sigma}_{\mathsf{trace}} = (\boldsymbol{C}_Z, \boldsymbol{C}_R, \boldsymbol{\sigma}_2, \boldsymbol{\sigma}_3, \hat{\boldsymbol{\pi}}_{\mathsf{sig}}) \in \mathbb{G}^8 \times \hat{\mathbb{G}}^2$ and $\boldsymbol{\pi}_{\mathsf{valid}} \in \mathbb{G}^2$.

Trace(PK, CT): Parse PK and CT as above, and output $\mathbf{ovk}$ in the obvious way.

Rand(PK, CT): Parse the ciphertext as above and conduct the following steps:

1. Randomize the CPA encryption part $\boldsymbol{c} = (c_0, c_1, c_2)$: Pick random $\theta' \leftarrow\!\$\, \mathbb{Z}_q$ and compute $\boldsymbol{c'} = (c_0', c_1', c_2') = (c_0 \cdot f^{\theta'}, c_1 \cdot g^{\theta'}, c_2 \cdot h^{\theta'}) = \boldsymbol{c} \cdot (f, g, h)^{\theta'}$. *Next steps 2-3 are dedicated to the tracing part* $\boldsymbol{\sigma}'_{\mathsf{trace}}$.
2. Adapt the commitments $\boldsymbol{C}_Z$ and $\boldsymbol{C}_R$ to the signature $\boldsymbol{\sigma}_1$ so that they become commitments $\tilde{\boldsymbol{C}}_Z$ and $\tilde{\boldsymbol{C}}_R$ to the signature $\boldsymbol{\sigma}'_1 = \boldsymbol{\sigma}_1 \boldsymbol{\sigma}_2^{\theta'}$. That is, parse $\boldsymbol{\sigma}_2$ as (Z_2, R_2) and compute $\tilde{\boldsymbol{C}}_Z = \boldsymbol{C}_Z \cdot \iota(Z_2^{\theta'})$ and $\tilde{\boldsymbol{C}}_R = \boldsymbol{C}_R \cdot \iota(R_2^{\theta'})$. Note: $\boldsymbol{\sigma}'_1$ is a valid signature on (g, c_0', c_1') for $\mathbf{ovk}$, and $\hat{\boldsymbol{\pi}}_{sig}$ remains a proof that $\boldsymbol{\sigma}'_1 = (Z_1', R_1')$ satisfies

$$\begin{aligned} &e(R_1', \hat{g}) \cdot e(Z_1', \hat{h}) \\ &= e(g, \hat{l}_1) \cdot e(c_0, \hat{l}_2) \cdot e(c_1, \hat{l}_3) \end{aligned}$$

   where $(Z_1', R_1') = (Z_1, R_1) \cdot (Z_2, R_2)^{\theta'}$.
3. Randomize the GS proof $(\tilde{\boldsymbol{C}}_Z, \tilde{\boldsymbol{C}}_R, \hat{\boldsymbol{\pi}}_{sig})$ for the CRS $\mathbf{crs}_\sigma$ leading to $(\boldsymbol{C}'_Z, \boldsymbol{C}'_R, \hat{\boldsymbol{\pi}}'_{sig})$. Let the randomized tracing part be $\boldsymbol{\sigma}'_{\mathsf{trace}} = (\boldsymbol{C}'_Z, \boldsymbol{C}'_R, \boldsymbol{\sigma}_2, \boldsymbol{\sigma}_3, \hat{\boldsymbol{\pi}}'_{sig})$

   The next step is dedicated to the validity proof $\boldsymbol{\pi}'_{\mathsf{valid}}$.

4. Adapt the proof of the validity of the CPA ciphertext. Namely, compute $\boldsymbol{\pi}'_{\mathsf{valid}} = \pi \cdot (\boldsymbol{\Sigma}_u^{\tau}\boldsymbol{\Sigma}_v)^{\theta'} = (Z_\pi (Z_u^{\tau} Z_v)^{\theta'}, R_\pi (R_u^{\tau} R_v)^{\theta'})$, where $\tau = H(\mathbf{ovk})$.

Output the randomized ciphertext

$$\mathsf{CT} = \left(\boldsymbol{c}', \mathbf{ovk}, \boldsymbol{\sigma}_{\mathsf{trace}}, \boldsymbol{\pi}_{\mathsf{valid}}\right) .$$

Ver(PK, CT): Conduct the following checks:

1. Verify if PK and CT parse properly. IF not, output 0. Else compute (C, F, G) from $\mathcal{H}(\hat{g}, \mathbf{ovk})$ and PK as above.
2. Verify the validity of the signatures $\boldsymbol{\sigma}_2$ and $\boldsymbol{\sigma}_3$ on the last two rows $\boldsymbol{T}_1, \boldsymbol{T}_2$ of the matrix $\boldsymbol{T}$ in (4.1) with respect to $\mathbf{ovk} = \{\hat{l}_i\}_{i=1}^3$. Namely, parse $\boldsymbol{\sigma}_i = (Z_i, R_i)$ for $i = 2, 3$ and check the next equations:

$$e(Z_2, \hat{g})e(R_2, \hat{h}) = e(f, \hat{l}_2) \cdot e(g, \hat{l}_3)$$
$$e(Z_3, \hat{g})e(R_3, \hat{h}) = e(F, \hat{l}_2) \cdot e(G, \hat{l}_3)$$

3. Verify the validity of the proof $\boldsymbol{\pi}_{\mathsf{valid}}$. Namely, verify that $\boldsymbol{\pi}_{\mathsf{valid}} = (Z_\pi, R_\pi)$ is a valid one-time LHSP signature on the vector $(c_1^{\tau}, c_2^{\tau}, c_1, c_2)$, which must satisfy

$$e(Z_\pi, \hat{g}) \cdot e(R_\pi, \hat{h}) = e(c_1, \hat{u}_1^{\tau}\hat{v}_1) \cdot e(c_2, \hat{u}_2^{\tau}\hat{v}_2), \tag{4.2}$$

where $\tau = H(\mathbf{ovk})$.

Output 1 if all these checks pass, otherwise, output 0.

Dec(SK, PK, CT): If $\mathsf{Ver}(\mathsf{PK}, \mathsf{CT}) = 0$, output $\perp$. Otherwise, given $\mathsf{SK} = (\alpha, \beta)$ and $\boldsymbol{c} = (c_0, c_1, c_2)$ included in CT, compute and output $m = c_0 \cdot c_1^{-\alpha} \cdot c_2^{-\beta}$.

The correctness follows by inspection.

4.2.3. Security

Theorem 12. *In the standard model, the above TREnc Π_{TS} is computationally verifiable, strongly randomizable, traceable, and TCCA secure under the SXDH assumption relative to* Setup *if the hash function $\mathcal{H}$ is collision-resistant.*

For the proof, we refer to [DPP22b]. It follows the intuition given prior to the description of the construction.

4.3. Public coin construction for encrypting a group element

In this section, we propose another construction using again an asymmetric bilinear group in the standard model. Compared to the previous construction, the first main advantage of this new construction is that it relies on a public coin CRS, as the main application of TREnc is the design of voting protocols. For the sake of verifiability, we want as few trust assumptions as possible on the parties, including the ones generating the CRS.

Secondly, we will propose a variant of the construction that achieves in the ROM a tight mTCCA notion, or in other words, the security of the scheme does not decrease when we increase the number of challenges. This tightness is practically helpful, since in our generic voting protocol of Section 3.4, we prove the receipt-freeness security with a single reduction to the mTCCA security, where the number of challenges is given by the number of voters.

Since the constructions in the standard model and in the random oracle are very close, we provide a single description and highlight where the schemes diverge.

4.3.1. Intuition

We first describe the non-tight construction in the standard model. We will build ciphertexts that have the same structure as the previous construction, with a CPA encryption part $\boldsymbol{c}$, a tracing part $\boldsymbol{\sigma}_{\mathsf{trace}}$ and a proof of validity $\boldsymbol{\pi}_{\mathsf{valid}}$. The CPA part $\boldsymbol{c}$ is computed identically as before, as well as the tracing part which is the same LHSP signature of $\boldsymbol{c}$ and the trace which is again the verification key of the signature. Hence, only $\boldsymbol{\pi}_{\mathsf{valid}}$ is different.

We still need to construct a randomizable simulation-sound proof that $(c_1, c_2) \in \langle (g, h) \rangle$. Previously, $\boldsymbol{\pi}_{\mathsf{valid}}$ was built using another LHSP signature included in pk, where the generation is secret-coin. Again, we give the intuition of this proof in several steps:

- Instead of using an LHSP signature, we use a slightly more expensive technique that we adapt from Couteau and Hartmann [CH20], which aim at compiling a Σ-protocol into a non-interactive proof through the use of pairings. Given $(c_1, c_2) = (g, h)^\theta$ where θ is the witness, we proceed as follows. To build the proof, we first generate in the public key of the scheme a value $\hat{E} \in \hat{\mathbb{G}}$ uniformly at random. Then, in the encryption algorithm, we generate a pair

of commitments $(X_1, X_2) = (g, h)^a$ and the response $\hat{Z} = \hat{E}^\theta \hat{g}^a$. These values verify the pairing equations:

$$e(c_1, \hat{E}) \cdot e(X_1, \hat{g}) = e(g, \hat{Z})$$
$$e(c_2, \hat{E}) \cdot e(X_2, \hat{g}) = e(h, \hat{Z})$$

- However, this proof is not simulation-sound yet as it is not linked to the trace. An adversary can reuse a simulated proof for a false statement, as in the case of the LHSP signature discussed in the intuition of the previous construction. Instead, we define in the public key two random group elements $\hat{E}_0$ and $\hat{E}_1$ and set $\hat{E}$ to $\hat{E}_0^\tau \hat{E}_1$ in the Enc algorithm, where τ is defined as in the previous construction as the hash of the trace.

  For the simulation-soundness to work in the security proof, we also add another pair of commitmentts $(Y_1, Y_2) = (g, h)^b$ in the ciphertext, which is original to this construction. The new pairing equations are

$$e(c_1, \hat{E}) \cdot e(X_1, \hat{g}) \cdot e(Y_1, \hat{h}) = e(g, \hat{Z})$$
$$e(c_2, \hat{E}) \cdot e(X_2, \hat{g}) \cdot e(Y_2, \hat{h}) = e(h, \hat{Z})$$

- We can simulate this proof by using a trapdoor in $\hat{E}$. Namely, we generate $\hat{E}_i$ as a commitment $\hat{g}^{x_i} \hat{h}^{y_i}$ for $i = 0$ and $i = 1$. Then, for a random $\hat{Z} = \hat{g}^s \hat{h}^t$, we can define $x = x_0 \tau + x_1$ and $y = y_0 \tau + y_1$ such that x and y is a valid opening of $\hat{E}$. Then,

$$X_1 = g^s / c_1^x \quad Y_1 = g^t / c_1^y$$
$$X_2 = h^s / c_2^x \quad Y_2 = h^t / c_2^y$$

  form a valid simulated proof. To show the soundness, we also need a language specific trapdoor ω allowing to check wheter the word (c_1, c_2) is indeed in $\mathrm{span}\langle(g, h)\rangle$. We simply have $h = g^\omega$ so that $c_2 = c_1^\omega$ in the honest case. That is, the proof system is witness samplable.

  The adversary, however, cannot simulate this proof. If it manages to find a proof $(X_1, X_2, Y_1, Y_2, \hat{Z})$ for a fresh trace that has not been queried in the challenge[15] for a statement $(c_1, c_2) \notin \langle(g, h)\rangle$, then we have:

$$e(c_1^\omega / c_2, \hat{E}) \cdot e(X_1^\omega / X_2, \hat{g}) \cdot e(Y_1^\omega / Y_2, \hat{h}) = e(g^\omega / h, \hat{Z}) = 1$$

[15] For queries with a trace queried in the challenge, the TCCA game will refuse such decryption request.

for a non-trivial $D = c_1^\omega / c_2$. Also, at least one of $X = X_1^\omega / X_2$ or $Y = Y_1^\omega / Y_2$ is non trivial and we can rewrite the equation as

$$e(D, \hat{g}^x \hat{h}^y) \cdot e(X, \hat{g}) \cdot e(Y, \hat{h}) = e(XD^x, \hat{g}) \cdot e(YD^y, \hat{h}) = 1.$$

Since the x is perfectly hidden from the adversary when it chooses the proof, XD^x will be non-trivial with overwhelming probability and we can reduce to SXDH. But also, since $\hat{E}$ is linked to the trace, even if an unbounded adversary sees a simulated proof for $\hat{E}' = \hat{g}^{x'} \hat{h}^{y'} = \hat{g}^{x_0 \tau' + x_1} \hat{h}^{y_0 \tau' + y_1}$, it will only see one relation on the two random variables x_0 and x_1, and $x = x_0 + \tau x_1$ is still perfectly unpredictable for any $\tau \neq \tau'$.

We stress that without the second pair of commitment, XD^x would be trivial to satisfy the last equation and so the argument would not work anymore. This motivates our deviation from the technique of Couteau and Hartmann.

Regarding the tight variant in the ROM, the construction is very similar. First, instead of generating $\hat{E} = \hat{E}_0^\tau \hat{E}_1$, we directly generate $\hat{E}$ as the hash of τ to be able to refresh the commitment $\hat{E}$ for every challenge and thus to have unbounded simulation soundness almost for free.

Moreover, we cannot use the same (F, G, H) tuple when we try to embed an SXDH instance in the randomization of each challenge in the TCCA proof, as the resulting ciphertexts will not be independent random tuples of $\mathbb{G}^3$. Instead, we generate randomly fresh (F, G) from the hash of the trace and take advantage of the random self-reducibility of SXDH to fully randomize these challenges.

Finally, it is worth mentioning that this technique is not compatible with the previous secret coin construction as π_{valid} is *one-time* simulation sound. For two challenges (and two simulated proofs), the securityproof does not work anymore.

We now fully describe the construction, called $\Pi_{PC}^{\mathbb{G}}$. We simultaneously describe the construction in the standard model and a tight variant in the ROM.

4.3.2. Description

Gen(1^λ): Run $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$.

1. Generate $h, F, G \leftarrow\!\$\, \mathbb{G}^3, \hat{h}, \hat{E}_0, \hat{E}_1 \leftarrow\!\$\, \hat{\mathbb{G}}^3$. In the ROM construction, ignore $\hat{E}_0, \hat{E}_1, F, G$ and generate all the other values from $\mathcal{H}(g, \hat{g})$ instead, where $g, \hat{g} \in \mathsf{pp}$.

2. Pick a random pair $\mathsf{sk} = (\alpha, \beta) \leftarrow\!\!\$\, \mathbb{Z}_q^2$ of scalars and compute $f = g^\alpha h^\beta$.
3. Generate a tuple of 4 random group elements $\mathsf{crs}_\sigma \leftarrow\!\!\$\, \mathbb{G}^4$ is seen as a Groth-Sahai CRS to commit to group elements over $\mathbb{G}$. For simplicity, we write Com for the commitment algorithm $\mathsf{Com}(\mathsf{crs}_\sigma, \cdot)$.

   In the ROM construction, generate this CRS from $\mathcal{H}(g, \hat{g}, f)$ instead.

The private key consists of $\mathsf{SK} = \mathsf{sk} = (\alpha, \beta)$ and the public key $\mathsf{PK} \in \mathbb{G}^9 \times \hat{\mathbb{G}}^4$ that can be derived from pp and f only is

$$\mathsf{PK} = \left(g, h, f, \hat{E}_0, \hat{E}_1, F, G, \mathsf{crs}_\sigma, \hat{g}, \hat{h}\right).$$

In the ROM construction, the public key can be derived from pp and f only and the values $\hat{E}_0, \hat{E}_1, F, G$ are stripped from PK.

Enc(PK, m): Given a message $m \in \mathbb{G}$ to encrypt, first run $\underline{\mathsf{LGen}(\mathsf{PK})}$: Generate a key pair $(\mathbf{osk}, \mathbf{ovk})$ for the one-time linearly homomorphic signature of Section 2.2.4 from the public generators $\hat{g}, \hat{h}$ in order to sign vectors of dimension 3. Let $\mathsf{lk} = \mathbf{osk} = \{(\eta_i, \zeta_i)\}_{i=1}^3$ be the private key, of which the corresponding public key is $\mathbf{ovk} = \{\hat{l}_i\}_{i=1}^3$. From $\mathbf{ovk}$, compute $\tau = \mathcal{H}(\hat{g}, \mathbf{ovk})$ and $\hat{E} = \hat{E}_0^\tau \hat{E}_1$. In the ROM construction, derive $(\hat{E}, F, G) = \mathcal{H}(\hat{g}, \mathbf{ovk})$.
Then, conduct the following steps of $\underline{\mathsf{LEnc}(\mathsf{PK}, \mathsf{lk}, m)}$:

1. Pick $\theta \leftarrow\!\!\$\, \mathbb{Z}_q$ and compute the CPA encryption $\boldsymbol{c} = (c_0, c_1, c_2)$, where $c_0 = mf^\theta$, $c_1 = g^\theta$, and $c_2 = h^\theta$. Keep the random coin θ.
   Next steps 2-3 are dedicated to the tracing part $\boldsymbol{\sigma}_{\mathsf{trace}}$.
2. To allow tracing, use $\mathsf{lk} = \mathbf{osk}$ to authenticate the row space of the matrix $\mathbf{T} = \left(T_{i,j}\right)_{1 \le i,j \le 3}$

$$\mathbf{T} = \begin{pmatrix} g & c_0 & c_1 \\ 1 & f & g \\ 1 & F & G \end{pmatrix}, \tag{4.3}$$

   by using $\mathsf{lk} = \mathsf{osk}$. Namely, sign each row $\boldsymbol{T}_i$ of $\mathbf{T}$ resulting in $\boldsymbol{\sigma} = (\boldsymbol{\sigma}_i)_{i=1}^3$, where $\boldsymbol{\sigma}_i = (Z_i, R_i) \in \mathbb{G}^2$.
3. Commit to $\boldsymbol{\sigma}_1 = (Z_1, R_1)$ using $\mathbf{crs}_\sigma$ as $\boldsymbol{C}_Z = \mathsf{Com}(Z_1)$, $\boldsymbol{C}_R = \mathsf{Com}(R_1)$. To ensure that each $\boldsymbol{\sigma}_1$ is a valid one-time

LHSP signature on (g, c_0, c_1), compute the Groth-Sahai proof $\hat{\boldsymbol{\pi}}_{sig} \in \hat{\mathbb{G}}^2$ that

$$e(Z_1, \hat{g}) \cdot e(R_1, \hat{h}) = e(g, \hat{l}_1) \cdot e(c_0, \hat{l}_2) \cdot e(c_1, \hat{l}_3)$$

The next step shows the validity of $\boldsymbol{c}$ *associated to* ovk.

4. Given θ, compute the randomizable simulation-sound proof π that $(c_1, c_2) \in \langle g, h \rangle$. More precisely, compute the commitments $(X_1, X_2) = (g, h)^a$ and $(Y_1, Y_2) = (g, h)^b$ for random $a, b \in \mathbb{Z}_q$.

   Then, compute the response $\hat{Z} = \hat{E}^\theta \hat{g}^a \hat{h}^b$ that verifies the equations:

$$e(c_1, \hat{E}) \cdot e(X_1, \hat{g}) \cdot e(Y_1, \hat{h}) = e(g, \hat{Z})$$
$$e(c_2, \hat{E}) \cdot e(X_2, \hat{g}) \cdot e(Y_2, \hat{h}) = e(h, \hat{Z})$$

   Let $\boldsymbol{\pi}_{\mathsf{valid}} = (X_1, X_2, Y_1, Y_2, \hat{Z}) \in \mathbb{G}^4 \times \hat{\mathbb{G}}$

Output the ciphertext

$$\mathsf{CT} = \Big(\boldsymbol{c}, \mathbf{ovk}, \boldsymbol{\sigma}_{\mathsf{trace}}, \boldsymbol{\pi}_{\mathsf{valid}}\Big),$$

where $\boldsymbol{\sigma}_{\mathsf{trace}} = (\boldsymbol{C}_Z, \boldsymbol{C}_R, \boldsymbol{\sigma}_2, \boldsymbol{\sigma}_3, \hat{\boldsymbol{\pi}}_{sig}) \in \mathbb{G}^8 \times \hat{\mathbb{G}}^2$ and $\boldsymbol{\pi}_{\mathsf{valid}} \in \mathbb{G}^4 \times \hat{\mathbb{G}}$.

Trace(PK, CT): Parse PK and CT as above, and output $\mathbf{ovk}$.

Rand(PK, CT): Parse the ciphertext as above, and compute (C, F, G) from $\mathcal{H}(\hat{g}, \mathbf{ovk})$ and PK as above and conduct the following steps:

1. Randomize the CPA part $\boldsymbol{c} = (c_0, c_1, c_2)$ by picking a random $\theta' \leftarrow\!\!\$\, \mathbb{Z}_q$ and compute $\boldsymbol{c'} = (c'_0, c'_1, c'_2) = (c_0 \cdot f^{\theta'}, c_1 \cdot g^{\theta'}, c_2 \cdot h^{\theta'}) = \boldsymbol{c} \cdot (f, g, h)^{\theta'}$.
   Next steps 2-3 are dedicated to the tracing part $\boldsymbol{\sigma}'_{\mathsf{trace}}$.
2. Adapt the commitments $\boldsymbol{C}_Z$ and $\boldsymbol{C}_R$ to the signature $\boldsymbol{\sigma}_1$ so that they become commitments $\tilde{\boldsymbol{C}}_Z$ and $\tilde{\boldsymbol{C}}_R$ to the signature $\boldsymbol{\sigma}'_1 = \boldsymbol{\sigma}_1 \boldsymbol{\sigma}_2^{\theta'}$. That is, parse $\boldsymbol{\sigma}_2$ as (Z_2, R_2) and compute $\tilde{\boldsymbol{C}}_Z = \boldsymbol{C}_Z \cdot \iota(Z_2^{\theta'})$ and $\tilde{\boldsymbol{C}}_R = \boldsymbol{C}_R \cdot \iota(R_2^{\theta'})$. Note: $\boldsymbol{\sigma}'_1$ is a valid signature on (g, c'_0, c'_1) for $\mathbf{ovk}$, and $\hat{\boldsymbol{\pi}}_{sig}$ remains a proof that $\boldsymbol{\sigma}'_1 = (Z'_1, R'_1)$ satisfies

$$e(Z'_1, \hat{g}) \cdot e(R'_1, \hat{h}) = e(g, \hat{l}_1) \cdot e(c_0, \hat{l}_2) \cdot e(c_1, \hat{l}_3)$$

   where $(Z'_1, R'_1) = (Z_1, R_1) \cdot (Z_2, R_2)^{\theta'}$.

3. Randomize the GS proof $(\tilde{\boldsymbol{C}}_Z, \tilde{\boldsymbol{C}}_R, \hat{\boldsymbol{\pi}}_{sig})$ for the CRS crs_σ leading to $(\boldsymbol{C}'_Z, \boldsymbol{C}'_R, \hat{\boldsymbol{\pi}}'_{sig})$. Let the randomized tracing part be $\boldsymbol{\sigma}'_{\mathsf{trace}} = (\boldsymbol{C}'_Z, \boldsymbol{C}'_R, \boldsymbol{\sigma}_2, \boldsymbol{\sigma}_3, \hat{\boldsymbol{\pi}}'_{sig})$. [16]

   The next step is dedicated to the validity proof $\boldsymbol{\pi}'_{\mathsf{valid}}$.

4. Adapt the proof $\boldsymbol{\pi}_{\mathsf{valid}} = (X_1, X_2, Y_1, Y_2, \hat{Z})$ to (c'_1, c'_2) using θ'. Namely, compute $(X_1, X_2)' = (X_1, X_2) \cdot (g, h)^{a'}$, $(Y_1, Y_2)' = (Y_1, Y_2) \cdot (g, h)^{b'}$ and $\hat{Z}' = \hat{Z} \cdot \hat{E}^{\theta'} \hat{g}^{a'} \hat{h}^{b'}$ for random $a', b' \in \mathbb{Z}_q$. Let $\boldsymbol{\pi}'_{\mathsf{valid}} = (X'_1, X'_2, Y'_1, Y'_2, \hat{Z}')$

Return the randomized ciphertext

$$\mathsf{CT}' = \left(\boldsymbol{c}', \mathbf{ovk}, \boldsymbol{\sigma}'_{\mathsf{trace}}, \boldsymbol{\pi}'_{\mathsf{valid}}\right).$$

Ver(PK, CT): Conduct the following checks:

1. Verify whether PK and CT parse properly. If not, output 0. Else compute (C, F, G) from $\mathcal{H}(\hat{g}, \mathbf{ovk})$ and PK as above.

   Next steps 2-3 verify the tracing part $\boldsymbol{\sigma}_{\mathsf{trace}}$.

2. Verify the validity of the signatures σ_2 and σ_3 on the last two rows $\boldsymbol{T}_1, \boldsymbol{T}_2$ of the matrix $\mathbf{T}$ in (4.3) with respect to $\mathbf{ovk} = \{\hat{l}_i\}_{i=1}^3$. Namely, parse $\boldsymbol{\sigma}_i = (Z_i, R_i)$ for $i = 2, 3$ and check the next equations:

$$e(Z_2, \hat{g})e(R_2, \hat{h}) = e(f, \hat{l}_2) \cdot e(g, \hat{l}_3)$$
$$e(Z_3, \hat{g})e(R_3, \hat{h}) = e(F, \hat{l}_2) \cdot e(G, \hat{l}_3)$$

3. Verify the validity of the proof $\hat{\boldsymbol{\pi}}_{sig}$ that committed variables in $(\boldsymbol{C}_Z, \boldsymbol{C}_R)$ consist of a signature on the first row (g, c_0, c_1) of $\mathbf{T}$ under $\mathbf{ovk}$.
   Next step verifies the validity proof part $\boldsymbol{\pi}_{\mathsf{valid}}$.

4. Verify the proof $\boldsymbol{\pi}_{\mathsf{valid}}$. Namely, parse the proof $\boldsymbol{\pi}_{\mathsf{valid}} = (X_1, X_2, Y_1, Y_2, \hat{Z})$, and check

$$e(c_1, \hat{E}) \cdot e(X_1, \hat{g}) \cdot e(Y_1, \hat{h}) = e(g, \hat{Z})$$
$$e(c_2, \hat{E}) \cdot e(X_2, \hat{g}) \cdot e(Y_2, \hat{h}) = e(h, \hat{Z})$$

Output 1 if all these checks pass, otherwise, output 0.

Dec(SK, PK, CT): If $\mathsf{Ver}(\mathsf{PK}, \mathsf{CT}) = 0$, output $\perp$. Otherwise, given $\mathsf{SK} = (\alpha, \beta)$ and $\boldsymbol{c} = (c_0, c_1, c_2)$ included in CT, compute and output $m = c_0 \cdot c_1^{-\alpha} c_2^{-\beta}$.

[16] Since computing these LHSP signatures is deterministic in $\mathsf{lk} = \mathbf{osk}$, they cannot be randomized, and we do not need to randomize them to satisfy the strong randomizability notion.

4.3.3. Security

Theorem 13. *The above TREnc* $\Pi_{PC}^{\mathbb{G}}$ *is verifiable under the* SXDH *assumption. More precisely, for any adversary* $\mathcal{A}$*, we have:*

$$\Pr[\mathsf{Exp}^{\mathrm{ver}}_{\mathcal{A},\Pi_{PC}^{\mathbb{G}}}(\lambda) = 1] \leq 2 \cdot \varepsilon_{\mathrm{SXDH}} + 2/q$$

Proof. Let $\mathcal{A}$ be an adversary against the verifiability notion of Definition 21 such that, given $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{Gen}(1^\lambda)$, $\mathcal{A}$ produces a ciphertext CT. Next, we show that if CT is valid then $\mathsf{CT} \in \mathsf{Enc}(\mathsf{PK})$ but with probability bounded by $2 \cdot \varepsilon_{\mathrm{SXDH}} + 2/q$, where $q > 2^{\mathsf{poly}(\lambda)}$.

1. Given (c_1, c_2) and the proof of validity satisfying the verification equations, either $(c_1, c_2) \in \mathrm{span}\langle(g, h)\rangle$ or we can build a DP solver.

   We first define $h = g^\omega$ and $\hat{E} = \hat{g}^x \hat{h}^y$. If the adversary outputs $(c_1, c_2) \notin \langle(g, h)\rangle$ and a proof $(X_1, X_2, Y_1, Y_2, \hat{Z})$ verifying the equations, we have:

   $$e(c_1^\omega / c_2, \hat{E}) \cdot e(X_1^\omega / X_2, \hat{g}) \cdot e(Y_1^\omega / Y_2, \hat{h}) = e(g^\omega / h, \hat{Z}) = 1$$

   for a non-trivial $D = c_1^\omega / c_2$ such that $h = g^\omega$ by carefully dividing the equations. Also, at least one of $X = X_1^\omega / X_2$ or $Y = Y_1^\omega / Y_2$ is non trivial and we can rewrite the previous equation as

   $$e(D, \hat{g}^x \hat{h}^y) \cdot e(X, \hat{g}) \cdot e(Y, \hat{h}) = e(XD^x, \hat{g}) \cdot e(YD^y, \hat{h}) = 1.$$

   where XD^x is non-trivial with probability $1 - 1/q$ as x is independent of the view of the adversary. Hence, we built a DP solver from this proof of validity, so this event cannot happen with a probability bigger than $1/q + \varepsilon_{\mathrm{SXDH}}$.

2. Also, for a correct $(c_1, c_2) = (g, h)^\theta$, we can write $X_1 = g^a$ and $Y_1 = g^b$. From the first equation of verification, $\hat{Z}$ is constrained to be computed as $\hat{E}^\theta \hat{g}^a \hat{h}^b$. If we now compute

   $$\begin{aligned} e(c_1^\omega / c_2, \hat{E}) \cdot e(X_1^\omega / X_2, \hat{g}) \cdot e(Y_1^\omega / Y_2, \hat{h}) &= e(g^\omega / h, \hat{Z}) \\ e(X_1^\omega / X_2, \hat{g}) \cdot e(Y_1^\omega / Y_2, \hat{h}) &= 1 \end{aligned}$$

   either $X_2 = X_1^\omega = h^a$ and $Y_2 = Y_1^\omega = h^b$ or we have again a DP solver.

3. Now, we turn to **ovk** that should lie in the output of LGen and to the remaining items 2-3 of the encryption algorithm with the LHSP signatures and the GS proof $\boldsymbol{\sigma}_{\mathsf{trace}} = (\boldsymbol{C}_Z, \boldsymbol{C}_R, \boldsymbol{\sigma}_2, \boldsymbol{\sigma}_3, \hat{\boldsymbol{\pi}}_{sig})$.

Since $\mathbf{crs}_\sigma$ is hiding as a commitment key to commit to group elements, except if it is a DH tuple, which can only occur with negligible probability $1/q$, $(\boldsymbol{C}_Z, \boldsymbol{C}_R, \hat{\boldsymbol{\pi}}_{sig})$ can always be explained as an honest run for any witness satisfying the LHSP equation in (3). Now, let us consider the 6 variables forming the potential $\mathsf{lk} = \mathbf{osk} = \{(\eta_i, \zeta_i)\}_{i=1}^3$ explaining $\mathbf{ovk}$. This public information imposes a single linear relation between η_i and ζ_i for each index. Moreover, the verification equations of item 2 of the verification algorithm impose 2 linear conditions (since the two last rows of the matrix $\mathbf{T}$ in Equation 4.3 are linearly independent with overwhelming probability $(q-1)/q$ due to the uniform generation of $(C, F, G) = \mathcal{H}(\hat{g}, \mathbf{ovk})$ by the random oracle). All these conditions leave one degree of freedom for either η_1 or ζ_1. Therefore, for any chosen value of η_1, we can explain $\mathbf{ovk}$, and then $(\boldsymbol{C}_Z, \boldsymbol{C}_S, \hat{\boldsymbol{\pi}}_{sig})$.

Consequently, the scheme is verifiable. More precisely, we have $\Pr[\mathsf{Exp}^{\mathrm{ver}}_{\mathcal{A},\Pi^{\mathbb{G}}_{PC}}(\lambda) = 1] \leq 2 \cdot \varepsilon_{\mathrm{SXDH}} + 2/q$, with $q > 2^{\mathsf{poly}(\lambda)}$. □

Theorem 14. *The above TREnc $\Pi^{\mathbb{G}}_{PC}$ is statistically strongly randomizable. More precisely, for any adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_2$ is possibly unbounded, we have:*

$$\Pr[\mathsf{Exp}^{\mathrm{sr}}_{\mathcal{A},\Pi^{\mathbb{G}}_{PC}}(\lambda) = 1] \leq \frac{1}{q}$$

Proof. Our TREnc is *statistically* strongly randomizable as defined in Definition 24. We only fully explain the proof for the construction in the standard model, as the proof for the construction in the ROM is similar. Let pp be generated from the algorithm Setup with $q > 2^{\mathsf{poly}(\lambda)}$ and $\mathsf{PK} = (g, h, f, \hat{E}_0, \hat{E}_1, F, G, \mathbf{crs}_\sigma, \hat{g}, \hat{h})$ be generated from Gen. Also, let $m \in \mathbb{G}$ be a message and $\mathsf{lk} = \mathbf{osk} = \{(\eta_i, \zeta_i)\}_{i=1}^3 \in \mathbb{Z}_q^6$ be a link key. For any ciphertext $\mathsf{CT} \in \mathsf{LEnc}(\mathsf{PK}, \mathsf{lk}, m)$, we show that $\mathsf{Rand}(\mathsf{PK}, \mathsf{CT}) \approx_s \mathsf{LEnc}(\mathsf{PK}, \mathbf{osk}, m)$, where $\approx_s$ refers to statistical indistinguishability. Let ε_{SR} be the probability of distinguishing the two distributions.

By assumption, CT can be parsed as $(\boldsymbol{c}, \mathbf{ovk}, \boldsymbol{\sigma}_{\mathsf{trace}}, \boldsymbol{\pi}_{\mathsf{valid}})$, where $\mathbf{ovk}$, the tracing part $\boldsymbol{\sigma}_{\mathsf{trace}} = (\boldsymbol{C}_R, \boldsymbol{C}_S, \boldsymbol{\sigma}_2, \boldsymbol{\sigma}_3, \hat{\boldsymbol{\pi}}_{sig})$ and the validity proof $\boldsymbol{\pi}_{\mathsf{valid}} = (X_1, X_2, Y_1, Y_2, \hat{Z})$ respect the computation of LEnc as described in the encryption algorithm, for some values playing the role of the random coins. Moreover, as in the previous proof of verifiability, $\mathbf{crs}_\sigma$ of PK is in the hiding mode to commit to group elements, except with probability $1/q$. Then, we must have:

1. The trace $\mathbf{ovk}$ is deterministic in $\mathsf{lk} = \mathbf{osk}$ as the verification key of the LHSP signature scheme of Section 2.2.4.

2. In $\boldsymbol{\sigma}_{\mathsf{trace}}$, the LHSP signatures $\boldsymbol{\sigma}_2, \boldsymbol{\sigma}_3$ on the vectors $(1, f, g)$ and $(1, F, G)$ that are the rows of the matrix $\mathbf{T}$ in Equation 2 satisfy the relation of the deterministic signing algorithm of Section 2.2.4.

3. The GS proof $(\boldsymbol{C}_Z, \boldsymbol{C}_R, \hat{\boldsymbol{\pi}}_{sig})$ in $\boldsymbol{\sigma}_{\mathsf{trace}}$ is a valid proof of knowledge of a signature $\boldsymbol{\sigma}_1 = (Z_1, R_1)$ on the vector in the first row of $\mathbf{T}$ composed of elements from the CPA part $\boldsymbol{c}$ that can be written as (c_0, c_1, c_2), where $c_0 = mf^\theta$, $c_1 = g^\theta$ and $c_2 = h^\theta$ for some $\theta \in \mathbb{Z}_q$. This proof is perfectly witness indistinguishable and randomizable.

4. The proof of validity $(X_1, X_2, Y_1, Y_2, \hat{Z})$ is zero-knowledge and perfectly randomizable.

Based on these observations, we next show that the distribution of honest randomization of CT and the distribution of honest re-encryption of m with the link key lk above are identical.

On the one hand, independently of the precise values taken by θ, a, b and the hidden scalars in all the GS proofs (so, even if $\theta = a = b = 0$, for instance) the algorithm Rand perfectly randomizes the CPA part $\boldsymbol{c}$ in item 1 as well as $(X_1, X_2, Y_1, Y_2, \hat{Z})$ in item 4. Then, the randomized tracing part is fully redistributed among all the valid proofs with the adapted $\boldsymbol{c}$ (but $\boldsymbol{\sigma}_2, \boldsymbol{\sigma}_3$ are unchanged as detailed in item 3).

On the other hand, with a fresh execution of LGen on input PK, lk and m, m is unchanged, the link key lk leads to the same trace $\mathbf{ovk}$ as well as trace-dependent value $\hat{E} = \hat{E}_0^\tau \hat{E}_1$, and the signatures on $(1, f, g)$ and $(1, F, G)$ are uniquely determined since the LHSP signing algorithm is deterministic. That is LGen produces the same $\boldsymbol{\sigma}_2, \boldsymbol{\sigma}_3$ as above.

Moreover, the CPA part on m, $\boldsymbol{\pi}_{\mathsf{valid}}$, and all the GS commitments and proofs are honestly distributed by definition. So these distributions are the same as the output of Rand as justified above when analyzing the distribution of the randomized ciphertext.

To conclude, the scheme is statistically strongly randomizable. More precisely, we have $\Pr[\mathsf{Exp}^{\mathrm{sr}}_{\mathcal{A},\Pi^{\mathbb{G}}_{PC}}(\lambda) = 1] \leq \frac{1}{q}$. □

Theorem 15. *The above TREnc $\Pi^{\mathbb{G}}_{PC}$ is traceable under the SXDH assumption. More precisely, for any adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we have* $\Pr[\mathsf{Exp}^{\mathrm{trace}}_{\mathcal{A},Pi^{\mathbb{G}}_{PC}}(\lambda) = 1] \leq 4 \cdot \varepsilon_{\mathrm{SXDH}} + 3/q$.

Proof. Let $\mathcal{A}$ be an adversary against the traceability of our TREnc, and ε_{trace} be its winning probability in the traceability game as defined in Figure 3.1 (right) of Definition 23. We give a succession of hybrid games, where the first one is the real traceability game. For each of those games, we denote by S_i the event that the i-th game outputs 1 in the security

parameter λ. Also, we define q_{ovk} as the number of distinct $\mathcal{H}$-queries of the form $(\hat{E}, F, G) \leftarrow \mathcal{H}(\hat{g}, \mathsf{ovk})$ made by the adversary for any triple $\mathsf{ovk} \in \hat{\mathbb{G}}^3$ in the ROM-based construction.

$\mathsf{Game}_0(\lambda)$: This is the real traceability game where, given $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{Gen}(1^\lambda)$, the adversary $\mathcal{A}$ chooses $m \in \mathbb{G}$. It gets $\mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{PK}, m)$ and outputs CT^*. By definition, $\mathsf{Game}_0(\lambda)$ outputs 1 if $\mathsf{Trace}(\mathsf{PK}, \mathsf{CT}) = \mathsf{Trace}(\mathsf{PK}, \mathsf{CT}^*)$, $\mathsf{Ver}(\mathsf{PK}, \mathsf{CT}^*) = 1$, but $\mathsf{Dec}(\mathsf{SK}, \mathsf{CT}^*) \neq m$. By definition, we have $Pr[S_0] = \varepsilon_{trace}$.

$\mathsf{Game}_1(\lambda)$: In this game, we modify $\mathsf{Game}_0(\lambda)$ in the way we produce CT. Instead of directly computing $\mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{PK}, m)$, we first run $\mathsf{lk} = \mathbf{osk} = \{(\eta_i, \zeta_i)\}_{i=1}^3 \leftarrow \mathsf{LGen}(\mathsf{PK})$ at the beginning of the game and keep $\mathbf{osk}$ throughout all the game(s). Then, when the adversary requests the encryption, we compute $\mathsf{CT} \leftarrow \mathsf{LEnc}(\mathsf{PK}, \mathbf{osk}, m)$. The trace is then the public key $\mathbf{ovk} = \{\hat{l}_i\}_{i=1}^3$ of the LHSP signature of Section 2.2.4 computed from $\mathbf{osk}$. For the ROM-based case, we also assume that the challenger computes $(\hat{E}, F, G) = \mathcal{H}(\hat{g}, \mathbf{ovk})$ at the beginning of the game. By definition of TREnc, we have $Pr[S_1] = Pr[S_0]$.

$\mathsf{Game}_2(\lambda)$: In this game, we modify the generation of the Groth-Sahai CRS $\mathbf{crs}_\sigma = (\boldsymbol{u}_1, \boldsymbol{u}_2) \in \mathbb{G}^4$ in item 3 of the key generation by picking a random exponent $\nu \leftarrow\!\!\$\, \mathbb{Z}_q$, $\boldsymbol{u}_1 \leftarrow\!\!\$\, \mathbb{G}^2$ and computing $\boldsymbol{u}_2 = \boldsymbol{u}_1^\nu$. In the ROM variant, we program the random oracle to output this CRS: $(\mathbf{crs}_\sigma) \leftarrow \mathcal{H}(g, \hat{g}, f)$. Clearly, $|Pr[S_2] - Pr[S_1]| \leq \varepsilon_{\mathrm{SXDH}}$.

$\mathsf{Game}_3(\lambda)$: This game is as $\mathsf{Game}_2(\lambda)$ except in the way we generate $\boldsymbol{u}_1 = (u_{1,1}, u_{1,2})$ in $\mathbf{crs}_\sigma$. Now, we pick a trapdoor extracting key $\tau_\sigma \leftarrow\!\!\$\, \mathbb{Z}_q$ and compute $u_{1,1} = u_{1,2}^{\tau_\sigma}$. As long as $u_{1,2} \neq 1_\mathbb{G}$, $u_{1,1}$ remains independent of $u_{1,2}$. We have $|Pr[S_3] - Pr[S_2]| \leq 1/q$.

$\mathsf{Game}_4(\lambda)$: In this game, if the adversary $\mathcal{A}$ outputs a *valid* ciphertext $\mathsf{CT}^* = (\boldsymbol{c}^*, \mathbf{ovk}^*, \boldsymbol{\sigma}^*_{\mathsf{trace}}, \boldsymbol{\pi}^*_{\mathsf{valid}})$ such that $\mathbf{ovk}^* = \mathbf{ovk}$, we use the trapdoor key τ_σ to extract a valid signature $\boldsymbol{\sigma}_1^* = (Z_1^*, R_1^*)$ from the commitments $\boldsymbol{C}_Z, \boldsymbol{C}_R$. This is always possible since $\mathbf{crs}_\sigma$ is in the binding mode for group elements, and $\hat{\boldsymbol{\pi}}_{sig}$ is then perfectly sound. This signature $\boldsymbol{\sigma}_1^* = (Z_1^*, R_1^*)$ satisfies $e(Z_1^*, \hat{g}) \cdot e(R_1^*, \hat{h}) = e(g, \hat{l}_1) \cdot e(c_0^*, \hat{l}_2) \cdot e(c_1^*, \hat{l}_3)$, where $\mathbf{ovk} = (\hat{l}_i)_{i=1}^3$. Now, given the vector (g, c_0^*, c_1^*) we compute a fresh LHSP signature with $\mathbf{osk}$ resulting in $\boldsymbol{\sigma}_1^\dagger = (Z_1^\dagger, R_1^\dagger)$. In the event that $\boldsymbol{\sigma}_1^\dagger \neq \boldsymbol{\sigma}_1^*$, we abort the game and output 0. If we let this event be $\mathcal{F}_4$, we have $|Pr[S_4] - Pr[S_3]| \leq \Pr[\mathcal{F}_4]$.

It is easy to see that with two distinct valid LHSP signatures on the same vector we have $e(Z_1^*/Z_1^\dagger, \hat{g}) \cdot e(R_1^*/R_1^\dagger, \hat{h}) = 1_{\mathbb{G}_T}$. Given $\hat{g}, \hat{h} \in \hat{\mathbb{G}}$,

computing these elements of $\mathbb{G}$ breaks the SXDH assumption. We thus have $\Pr[\mathcal{F}_4] \leq \varepsilon_{\text{SXDH}}$.

$\mathsf{Game}_5(\lambda)$: This games is as $\mathsf{Game}_4(\lambda)$ except that we introduce a failing event $\mathcal{F}_5$ which causes the game to abort and outputs 0. $\mathcal{F}_5$ occurs when the adversary $\mathcal{A}$ outputs a *valid* ciphertext $\mathsf{CT}^* = (\boldsymbol{c}^*, \mathbf{ovk}^*, \boldsymbol{\sigma}^*_{\mathsf{trace}}, \boldsymbol{\pi}^*_{\mathsf{valid}})$ such that $(c_1^*)^\omega \neq c_2^*$ for $\boldsymbol{c}^* = (c_0^*, c_1^*, c_2^*)$ and $h = g^\omega$. Similarly to point 1 of the verifiability proof, if this event happens, then we can build a DP solver. Hence, we have $|Pr[S_5] - Pr[S_4]| \leq \Pr[\mathcal{F}_5] \leq 1/q + \varepsilon_{\text{SXDH}}$.

$\mathsf{Game}_6(\lambda)$: This game is like $\mathsf{Game}_5(\lambda)$, but we bring a last modification. In the key generation algorithm of the standard model construction, we define $F = f^\mu$ and $G = g^\mu$ for a random $\mu \in \mathbb{Z}_q$. In the ROM variant, the challenger simply programs $(\hat{E}, F, G) = \mathcal{H}(\hat{g}, \mathbf{ovk})$ with those parameters at the beginning of the game. In both cases, we have $|Pr[S_6] - Pr[S_5]| \leq \varepsilon_{\text{SXDH}}$.

Conclusion. Now, we argue that $\Pr[\mathsf{Game}_6(\lambda) = 1] = Pr[S_6] \leq 1/q$. To see that, let us assume that $\mathcal{A}$ wins in $\mathsf{Game}_6(\lambda)$. From $\mathsf{Game}_4(\lambda)$, we know that the extracted signature $\boldsymbol{\sigma}_1^*$ on (g, c_0^*, c_1^*), from $\boldsymbol{c}^* = (c_0^*, c_1*, c_2)$ is equal to the honestly computed one $\boldsymbol{\sigma}_1^\dagger$ from $\mathbf{osk}$. Moreover, during the whole game the adversary only sees two LHSP signatures related to $\mathbf{osk}$ on two independent vectors. That is because the committed signature $\boldsymbol{\sigma}_1$ authenticates the vector $(g, c_0, c_1) = (g, mf^\theta, g^\theta)$ and, $\boldsymbol{\sigma}_2$ authenticates the vector $(1, f, g)$ since the third vector $(1, F, G) = (1, f, g)^\mu$ from $\mathsf{Game}_6(\lambda)$. Therefore, the LHSP signatures authenticate a subspace of dimension 2 in $\mathbb{G}^3$.

To conclude, we show that the authenticated vector $(g, c_0^*, c_1^*) = (g, m^* f^{\theta^*}, g^{\theta^*})$ of the adversary is outside the linear span generated by (g, c_0, c_1) and $(1, f, g)$, and then $\boldsymbol{\sigma}_1^\dagger$ remains unpredictable, but with probability $1/q$. This is because $(g, m^*, 1)$ is not a linear combination of $(g, m, 1)$ and $(1, f, g)$, as $m \neq m^\star$. So, $\boldsymbol{\sigma}_1^\dagger = (g^{\eta_1}(c_0^*)^{\eta_2}(c_1^*)^{\eta_3}, g^{\zeta_1}(c_0^*)^{\zeta_2}(c_1^*)^{\zeta_3})$ is independent of $\mathcal{A}$'s view and remains uniform (from linear algebra).

All in all, in both constructions we have $\Pr[\mathsf{Exp}^{\text{trace}}_{\mathcal{A}, Pi^{\mathbb{G}}_{PC}}(\lambda) = 1] \leq 4 \cdot \varepsilon_{\text{SXDH}} + 3/q$. □

Theorem 16. *The above TREnc $\Pi^{\mathbb{G}}_{PC}$ is TCCA-secure under the SXDH assumption and the collision resistance of the hash function. More precisely, we have*

$$\Pr[\mathsf{Exp}^{\text{mtcca}}_{\mathcal{A}, \Pi^{\mathbb{G}}_{PC}}(\lambda) = 1] \leq \frac{1}{2} + 4\varepsilon_{\text{SXDH}} + \frac{q_{\mathcal{H}}(q_{\mathcal{H}} - 1)/2 + (2m + 8)}{q}$$

for the construction in the ROM and

$$\Pr[\mathsf{Exp}^{\mathrm{tcca}}_{\mathcal{A},\Pi^{\mathbb{G}}_{PC}}(\lambda)=1] \leq \frac{1}{2} + 4\varepsilon_{\mathrm{SXDH}} + \varepsilon_{cr} + \frac{9}{q}$$

for the construction in the standard model. m is the number of challenges in the mTCCA *game, $q_{\mathcal{H}}$ is bounded by the number of evaluation of $\mathcal{H}$ made by the adversary, and ε_{cr} is the collision-resistance advantage of the hash function.*

Proof. Let $\mathcal{A}$ be an adversary against the mTCCA security of our TREnc, and TCCA be its winning probability in the mTCCA game as defined in Figure 3.2 of Definition 25. We give a succession of hybrid games, where the first one is the real mTCCA game. In each of those games, the challenger always knows the secret key SK to answer any decryption queries appropriately. For each of those games, we denote by S_i the event that the i-th game outputs 1 in the security parameter λ.

Let m be the number of challenge queries (which is equal to 1 in the ROM construction) made by $\mathcal{A}$. Let also $q_{\mathcal{H}}$ be the total number of $\mathcal{H}$-queries of the form $(\hat{E}, F, G) \leftarrow \mathcal{H}(\hat{g}, \mathbf{ovk})$ for any triple $\mathbf{ovk} \in \hat{\mathbb{G}}^3$ in the ROM construction.

$\mathsf{Game}_0(\lambda)$: This is the real TCCA game where, the challenger runs $\mathsf{Gen}(1^\lambda)$ to generate $(\mathsf{PK}, \mathsf{SK})$ with $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ such that $\mathsf{SK} = (\mathsf{sk}) = (\alpha, \beta) \leftarrow\!\!\$\, \mathbb{Z}_q^2$ and $\mathsf{PK} = (g, h, f, \hat{E}_0, \hat{E}_1, F, G, \mathbf{crs}_\sigma, \hat{g}, \hat{h})$, where $f = g^\alpha h^\beta$. In the ROM construction, $\hat{E}_0, \hat{E}_1, F, G$ do not appear in PK as they are generated from an hash of the trace for every ciphertext. Also $\mathbf{crs}_\sigma \in \mathbb{G}^4$ is uniform, meaning that $\mathbf{crs}_\sigma$ is in the perfectly hiding/WI mode for group elements in $\mathbb{G}$. The challenger hands the adversary PK, and also flips a random bit $b \leftarrow\!\!\$\, \{0,1\}$. At each decryption query, given CT_i the challenger checks whether its trace is in the list of the traces that already appeared in a challenge query. If so, the challenger outputs $\perp$. If not, it sends $\mathsf{Dec}(\mathsf{SK}, \mathsf{CT}_i)$ to the adversary $\mathcal{A}$. At each challenge query, given $(\mathsf{CT}_j^0, \mathsf{CT}_j^1)$ the challenger checks whether both ciphertexts are valid and share the same trace, and if this trace never appeared in a previous challenge query. If so, we say that the challenge query is *valid*. In that case, the challenger adds the common trace, denoted $\mathbf{ovk}_j^*$, to the list L of challenge traces, computes $\mathsf{CT}_j^* \leftarrow \mathsf{Rand}(\mathsf{PK}, \mathsf{CT}_j^b)$, and sends CT_j^* to $\mathcal{A}$. If the query is invalid, it simply outputs $\perp$. At the end of the game, $\mathcal{A}$ outputs b', and the challenger outputs 1 if $b = b$, and 0, otherwise. This latter bit is the output of $\mathsf{Game}_0(\lambda)$. By definition, $Pr[S_0] = \mathrm{mTCCA}$.

In the next games, we assume that the challenger programs the random oracle and consistently replies to repeating $\mathcal{H}$-queries.

$\mathsf{Game}_1(\lambda)$: In this game, the challenger exactly follows $\mathsf{Game}_0(\lambda)$ except in the ROM variant. In this case, it directly outputs $\perp$ if the adversary makes a challenge query $(\mathsf{CT}_j^0, \mathsf{CT}_j^1)$ with a common trace $\mathbf{ovk}_j^*$ but for which there is still no entry in the $\mathcal{H}$-list. That is, by emulating the random oracle, the challenger never defines an input-output pair for $(\hat{g}, \mathbf{ovk}_j^*)$. Since, when defining $(\hat{E}_j^*, F_j^*, G_j^*) \leftarrow \mathcal{H}(\hat{g}, \mathbf{ovk}_j^*)$, for any choice of G_j^* there is a single group element F_j^* that fulfills the verification equation related to the LHSP signature $\boldsymbol{\sigma}_3^{j,0} \in \boldsymbol{\sigma}_{\mathsf{trace}}^{j,0}$ on the last row $(1, F_j^*, G_j^*)$ of the matrix $\mathbf{T}_j^0$ recomputed in item 2 of the verification algorithm, the probability of getting a valid CT_j^0 is bounded by $1/q$. Therefore, $|Pr[S_1] - Pr[S_0]| \leq m/q$ in the ROM variant and $Pr[S_1] = Pr[S_0]$otherwise.

Without loss of generality, from now on we assume that all the q challenge queries are valid, which also means that the output $(\hat{E}_j^*, F_j^*, G_j^*)$ of $\mathcal{H}$ on input $(\hat{g}, \mathbf{ovk}_j^*)$ is already defined prior to the challenge query $(\mathsf{CT}_j^0, \mathsf{CT}_j^1)$.

$\mathsf{Game}_2(\lambda)$: For the standard model construction, this game is as $\mathsf{Game}_1(\lambda)$ except that the challenger aborts the game outputs a random bit if, the adversary outputs two different ciphertexts containing distinct $\mathbf{ovk}$ and $\mathbf{ovk}'$ such that $\mathcal{H}(\hat{g}, \mathbf{ovk}) =: \tau = \tau' := \mathcal{H}(\hat{g}, \mathbf{ovk}')$. Clearly, this would contradict the collision-resistance of $\mathcal{H}$ and thus $|Pr[S_2] - Pr[S_1]| \leq \varepsilon_{cr}$.

For the ROM variant, this game is as $\mathsf{Game}_1(\lambda)$ except that the challenger aborts the game and outputs a random bit if, when it defines a new output $(\hat{E}, F, G)$ to answer a $\mathcal{H}$-query on input a new $(\hat{g}, \mathbf{ovk})$, $\hat{E}$ is already the first component of another output. That is, the challenger directly aborts if there is a collision on the first component $\hat{E}$ among the triples $(\hat{E}, F, G)$ programmed as the outputs of the $\mathcal{H}$ function on input of the form $(\hat{g}, \mathbf{ovk}) \in \hat{\mathbb{G}}^4$.

Obviously, $|Pr[S_2] - Pr[S_1]| \leq q_{\mathcal{H}}(q_{\mathcal{H}} - 1)/2q$.

$\mathsf{Game}_3(\lambda)$: This game is as $\mathsf{Game}_2(\lambda)$ except that the challenger aborts the game and outputs a random bit if the adversary managed to make a valid challenge query on input $(\mathsf{CT}_j^0, \mathsf{CT}_j^1)$ such that both ciphertexts are valid for their common trace $\mathbf{ovk}_j^*$, but the valid LHSP signatures $\boldsymbol{\sigma}_2^{j,0}, \boldsymbol{\sigma}_3^{j,0} \in \boldsymbol{\sigma}_{\mathsf{trace}}^{j,0}$ and $\boldsymbol{\sigma}_2^{j,1}, \boldsymbol{\sigma}_3^{j,1} \in \boldsymbol{\sigma}_{\mathsf{trace}}^{j,1}$ on the last two (common) rows of their respective matrix $\mathbf{T}_j^0$ and $\mathbf{T}_j^1$ (as recomputed in item 2 of the verification algorithm) are such that $\boldsymbol{\sigma}_2^{j,0} \neq \boldsymbol{\sigma}_2^{j,1}$ or $\boldsymbol{\sigma}_3^{j,0} \neq \boldsymbol{\sigma}_3^{j,1}$. Since these LHSP signatures $\boldsymbol{\sigma}_2^{j,0}, \boldsymbol{\sigma}_2^{j,1}$ and $\boldsymbol{\sigma}_3^{j,0}, \boldsymbol{\sigma}_3^{j,1}$ are valid respectively on $(1, f, g)$ and either $(1, F, G)$ (or $(1, F_j^*, G_j^*)$ in the ROM variant) for both

ciphertexts, the challenger gets two distinct signatures on the same vector.

Since the validity of the LHSP signatures are all related to the same parameter $(\hat{g}, \hat{h})$ for all the $\mathbf{ovk}_j^*$ and the hardness of SXDH implies the infeasibility of producing two distinct LHSP signatures on the same vectors, we have $|Pr[S_3] - Pr[S_2]| \leq \varepsilon_{\text{SXDH}}$.

$\mathsf{Game}_4(\lambda)$: In this game, we generate the public PK differently. We now generate a random $f \leftarrow\!\$\, \mathbb{G}$ and instead, compute $h = (fg^{\alpha})^{\beta^{-1}}$. Moreover, we need to to update the decryption algorithm. If a ciphertext is valid, we now compute $m = c_0 \cdot c_1^{\alpha} \cdot c_2^{-\beta}$.

Clearly, (f, g, h) has the same distribution in $\mathsf{Game}_3(\lambda)$ and $\mathsf{Game}_4(\lambda)$ and m is computed identically with a rewriting of the secret key. We thus have $Pr[S_4] = Pr[S_3]$.

$\mathsf{Game}_5(\lambda)$: This game proceeds as $\mathsf{Game}_4(\lambda)$, except the generation of the values F, G. In the standard model construction, instead of generating random $(F, G) \leftarrow\!\$\, \mathbb{G}^2$, we generate $\delta \leftarrow \mathbb{Z}_q$ and set $(F, G) = (f, g)^{\delta}$. Also, we define $H = h^{\delta}$ that we will use later and we now have $(F, G, H) = (f, g, h)^{\delta}$. In the ROM construction, we generate these values similarly each time the challenger responds to one of the $q_{\mathcal{H}}$ $\mathcal{H}$-queries of the form $\mathcal{H}(\hat{g}, \mathbf{ovk})$ for any triple $\mathbf{ovk}$ in $\hat{\mathbb{G}}^3$.

For the standard construction, clearly $|Pr[S_5] - Pr[S_4]| \leq \varepsilon_{\text{SXDH}}$. For the ROM construction, we show that $|Pr[S_5] - Pr[S_4]| \leq \varepsilon_{\text{SXDH}} + 3/q$.

To see why a single reduction to SXDH works for all the $\mathcal{H}$-queries of this form, we observe that they can all be answered given a single SXDH challenge (g, f, G_0, F_0), where g is used to generate the public parameters of PK as in both games $\mathsf{Game}_4(\lambda)$ and $\mathsf{Game}_5(\lambda)$. Here, sk is unknown even if PK has the same distribution in both games.

Now, to answer each fresh $\mathcal{H}$-query on input $(\hat{g}, \mathbf{ovk})$, the reduction simply picks fresh $u, v \leftarrow\!\$\, \mathbb{Z}_q$, computes $G = g^u G_0^v$ and $F = f^u F_0^v$, and programs $(\hat{E}, F, G) \leftarrow \mathcal{H}(\hat{g}, \mathbf{ovk})$ with $\hat{E}$.

Assuming that $g, f \neq 1_{\mathbb{G}}$, let $(G_0, F_0) = (g, f)^{\kappa}(1, g^{\upsilon})^{\gamma}$ for some unknown random $\kappa, \upsilon \in \mathbb{Z}_q$ and $\gamma \in \{0, 1\}$. Then, the pair (F, G) in the output of any $\mathcal{H}$-query can be written as $(F, G) = (f, g)^{u+\kappa v} \cdot (g^{\upsilon v}, 1)^{\gamma}$, where $g^{\upsilon v}$ is a random element independent the remaining view even given $w = u + \kappa v$ conditioned on $\upsilon \neq 0$. Therefore, if $\gamma = 1$ the view is the one of the adversary in $\mathsf{Game}_4(\lambda)$, and if $\gamma = 0$ the view is the one of the adversary in $\mathsf{Game}_5(\lambda)$.

$\mathsf{Game}_6(\lambda)$: This game is as $\mathsf{Game}_5(\lambda)$ except that the challenger modifies the way it answers the valid non-aborting challenge queries. Given such $(\mathsf{CT}_j^0, \mathsf{CT}_j^1)$, the challenger retrieves in the ROM construction the δ_j^* that defines $(F_j^*, G_j^*) = (f,g)^{\delta_j^*}$ in the answer to the $\mathcal{H}$-query on input $(\hat{g}, \mathbf{ovk}_j^*)$ from the modification introduced in $\mathsf{Game}_5(\lambda)$. In the standard model construction, the following modifications can be easily adapted for one challenge and one tuple (F, G, H).

Next, instead of randomizing the CPA part $\boldsymbol{c}_j^b = (c_0^{j,b}, c_1^{j,b}, c_2^{j,b})$ and the tracing part $\boldsymbol{\sigma}_{\mathsf{trace}}^{j,b} = (\boldsymbol{C}_Z^{j,b}, \boldsymbol{C}_R^{j,b}, \boldsymbol{\sigma}_2^{j,b}, \boldsymbol{\sigma}_3^{j,b}, \hat{\boldsymbol{\pi}}_{sig}^{j,b})$ of the ciphertext CT_j^b according to the real randomizing algorithm, the challenger now conducts the following steps. First, it picks random $\theta_j', \theta_j'' \leftarrow\!\!\$\, \mathbb{Z}_q$ and computes

$$\begin{aligned} \boldsymbol{c}_j^* &= \big(c_0^{j,*}, c_2^{j,*}, c_2^{j,*}\big) \\ &= \big(c_0^{j,b}, c_2^{j,b}, c_2^{j,b}\big) \quad = \big(d^{j,b}, (c_i^{j,b})_{i=1}^{\ell}\big) \cdot \big(f,g,h\big)^{\theta_j'} \cdot \big(F_j^*, G_j^*, H_j^*\big)^{\theta_j''}. \end{aligned} \tag{4.4}$$

That is, with respect to item 1 of the randomizing algorithm, it uses the additional vector (F_j^*, G_j^*, H_j^*) of intermediate values as a support to introduce the additional randomness θ_j''. Second, it adapts the commitments $\boldsymbol{C}_Z^{j,b}, \boldsymbol{C}_R^{j,b}$ into

$$\begin{aligned} \tilde{\boldsymbol{C}}_Z^{j,b} &= \boldsymbol{C}_Z^{j,b} \cdot \iota(Z_2^{j,b})^{\theta'} \cdot \iota(Z_3^{j,b})^{\theta''} \\ \tilde{\boldsymbol{C}}_R^{j,b} &= \boldsymbol{C}_R^{j,b} \cdot \iota(R_2^{j,b})^{\theta'} \cdot \iota(R_3^{j,b})^{\theta''}, \end{aligned} \tag{4.5}$$

where, with respect to the item 2 of the randomizing algorithm, it not only uses $\boldsymbol{\sigma}_2^{j,b} = (Z_2^{j,b}, R_2^{j,b})$ to adapt the committed signature but also $\boldsymbol{\sigma}_3^{j,b} = (Z_3^{j,b}, R_3^{j,b})$. In other words, if $\boldsymbol{\sigma}_1^{j,b}$ was the committed LHSP signature relative to the first row of the matrix $\mathbf{T}_j^b$, the adapted LHSP signature would be $\boldsymbol{\sigma}_1^{j,*} = \boldsymbol{\sigma}_1^{j,b}(\boldsymbol{\sigma}_2^{j,b})^{\theta'}(\boldsymbol{\sigma}_3^{j,b})^{\theta''}$. Finally, the challenger follows the steps of item 3 by honestly randomizing the commitments and the proof $\hat{\boldsymbol{\pi}}_{sig}^{j,b}$ leading to $\boldsymbol{\sigma}_{\mathsf{trace}}^{j,*} = (\boldsymbol{C}_Z^{j,*}, \boldsymbol{C}_R^{j,*}, \boldsymbol{\sigma}_2^{j,*}, \boldsymbol{\sigma}_3^{j,*}, \hat{\boldsymbol{\pi}}_{sig}^{j,*})$, where $\boldsymbol{\sigma}_2^{j,*} = \boldsymbol{\sigma}_2^{j,b}$ and $\boldsymbol{\sigma}_3^{j,*} = \boldsymbol{\sigma}_3^{j,b}$. The validity proof part $\boldsymbol{\pi}_{\mathsf{valid}}^{j,*}$ is computed exactly as in $\mathsf{Game}_5(\lambda)$.

Since the committed $\boldsymbol{\sigma}_1^{j,b}$ would be an LHSP signature on the first row $(g, c_0^{j,b}, c_1^{j,b})$ of $\mathbf{T}_j^b$, together with $\boldsymbol{\sigma}_2^{j,b}$ on the second row $(1, f, g)$ and $\boldsymbol{\sigma}_3^{j,b}$ on the last row $(1, F_j^*, G_j^*)$, $\boldsymbol{\sigma}_1^{j,*}$ would be an LHSP signature on

$$\big(g, c_0^{j,b}, c_1^{j,b}\big) \cdot \big(1, f, g\big)^{\theta_j'} \cdot \big(1, F_j^*, G_j^*\big)^{\theta_j''}$$

that is nothing but the vector $(g, c_0^{j,*}, c_1^{j,*})$ computed from $\boldsymbol{c}_j^*$. Moreover, due to the definition of the intermediate tuple (G_j^*, F_j^*) and because

$(1, F_j^*, G_j^*) = (1, f, g)^{\delta_j^*}$ from $\mathsf{Game}_5(\lambda)$, the CPA part satisfies

$$\boldsymbol{c}_j^* = \left(c_0^{j,b}, c_1^{j,b}, c_2^{j,b},\right) \cdot \left(g, f, h\right)^{\theta'_j + \delta_j^* \theta''_j}.$$

Therefore, since $\mathbf{crs}_\sigma$ is in the hiding mode, the perfect witness indistinguishable proof $\hat{\boldsymbol{\pi}}_{sig}^{j,*}$ is distributed as an honest proof of knowledge of the signature $\boldsymbol{\sigma}_1^{j,b}(\boldsymbol{\sigma}_2^{j,b})^{\theta' + w_j^* \theta''}$ on the vector $(g, c_0^{j,*}, c_1^{j,*})$ that is distributed exactly as in $\mathsf{Game}_5(\lambda)$. Therefore, $Pr[S_6] = Pr[S_5]$.

$\mathsf{Game}_7(\lambda)$: In this game, we now generate the (F, G) values uniformly at random again. That is, in the standard model construction, we generate $(F, G) \leftarrow\!\$\, \mathbb{G}^2$, $H = (FG^\alpha)^{\beta^{-1}}$ In the ROM construction, we generate all the (F_j, G_j) as random tuples of $\mathbb{G}^2$ and define $H_i = (F_i G_i^\alpha)^{\beta^{-1}}$ as well each time the challenger responds to one of the $q_{\mathcal{H}}$ $\mathcal{H}$-queries of the form $\mathcal{H}(\hat{g}, \mathbf{ovk})$ for any triple $\mathbf{ovk}$ in $\hat{\mathbb{G}}^3$. Now, $(1, F, G)$ is no more a random vector in the linear span generated by $(1, f, g)$ and turns back $\mathbf{T}$ as a full rank with overwhelming probability.

Following the same arguments as the transition in $\mathsf{Game}_5(\lambda)$, $|Pr[S_7] - Pr[S_6]| \leq \varepsilon_{\mathrm{SXDH}} + 3/q$.

$\mathsf{Game}_8(\lambda)$: The public PK is computed again as in the original Gen algorithm, that is as $f = g^\alpha h^\beta$ for random $\alpha, \beta \in \mathbb{Z}_q$. Also, we decrypt ciphertexts again with the original Dec algorithm. Finally, we now generate h as g^ω for a random $\omega \in \mathbb{Z}_q$.

For the same reasons as $\mathsf{Game}_4(\lambda)$, $Pr[S_8] = Pr[S_7]$.

$\mathsf{Game}_9(\lambda)$: This game is computed as $\mathsf{Game}_8(\lambda)$, with the following modifications.

In the standard model construction, instead of generating $\hat{E}_0, \hat{E}_1$ uniformly at random directly from $\mathbb{G}^2$, the challenger now picks x_0, x_1, $y_0, y_1 \leftarrow\!\$\, \mathbb{Z}_q$ and computes $\hat{E}_0 = \hat{g}^{x_0} \hat{h}^{y_0}$ and $\hat{E}_1 = \hat{g}^{x_1} \hat{h}^{y_1}$.

Analogously in the ROM variant, for all the hash queries of the form $\mathcal{H}(\hat{g}, \mathbf{ovk})$ with $\mathbf{ovk} \in \hat{\mathbb{G}}^3$, the challenger still picks $F, G \leftarrow\!\$\, \mathbb{G}$ but also takes random $x, y \leftarrow\!\$\, \mathbb{Z}_q$ to compute $\hat{E} = \hat{g}^x \hat{h}^y$. Then, it programs $(\hat{E}, F, G) \leftarrow \mathcal{H}(\hat{g}, \mathbf{ovk})$.

Since all the so-generated triples are still uniformly distributed as long as $\hat{g} \neq 1$ and $\hat{h} \neq 1$, we have $|Pr[S_9] - Pr[S_8]| \leq 1/q$.

$\mathsf{Game}_{10}(\lambda)$: In this game, we introduce a new event, $\mathcal{F}_{10}$. If at some point during a decryption query or a challenge the adversary outputs a valid ciphertext $\mathsf{CT} = (\boldsymbol{c}, \mathbf{ovk}, \boldsymbol{\sigma}_{\mathsf{trace}}, \boldsymbol{\pi}_{\mathsf{valid}})$ with $\boldsymbol{c} = (c_0, c_1, c_2)$ such

that $c_1^{\omega} \neq c_2$, then we abort and output a random bit. If we call this event $\mathcal{F}_{10}$, we clearly have $|Pr[S_{10}] - Pr[S_9]| = \Pr[\mathcal{F}_{10}]$. We claim that $\Pr[\mathcal{F}_{10}] \leq \varepsilon_{\mathrm{SXDH}} + 1/q$ using the same idea as point 1 of the proof of verifiability

If this event happens, we look at the first time such a ciphertext is submitted by the adversary. This ciphertext has a $\boldsymbol{\pi}_{\mathsf{valid}}$ component which has the structure $(X_1, X_2, Y_1, Y_2, \hat{Z})$, from which we define $(X, Y, D) = (X_1, Y_1, c_1)^{\omega}/(X_2, Y_2, c_2)$. We also note $\hat{E} = \hat{E_0}^{\tau}\hat{E_1} = \hat{g}^{\tau x_0 + x_1}\hat{h}^{\tau y_0 + y_1} = \hat{g}^x\hat{h}^y$ in the standard model construction or $\hat{E} = \hat{g}^x\hat{h}^y$ directly from the $\mathcal{H}(\hat{g}, \mathbf{ovk})$ in the ROM construction, and $\boldsymbol{\pi}_{\mathsf{valid}} = (X_1, X_2, Y_1, Y_2, \hat{Z})$, Clearly, $D \neq 1_{\mathbb{G}}$.

Since the ciphertext is valid, the proof $(X_1, X_2, Y_1, Y_2, \hat{Z})$ must verify the equations of the point 4 of the encryption algorithm, and we have:

$$e(c_1^{\omega}/c_2, \hat{E}) \cdot e(X_1^{\omega}/X_2, \hat{g}) \cdot e(Y_1^{\omega}/Y_2, \hat{h}) = e(g^{\omega}/h, Z) = 1$$

for a non-trivial $D = c_1^{\omega}/c_2$ such that $h = g^{\omega}$ by carefully dividing the equations. Also, at least one of $X = X_1^{\omega}/X_2$ or $Y = Y_1^{\omega}/Y_2$ is non trivial and we can rewrite the previous equation as

$$e(D, \hat{g}^x\hat{h}^y) \cdot e(X, \hat{g}) \cdot e(Y, \hat{h}) = e(XD^x, \hat{g}) \cdot e(YD^y, \hat{h}) = 1.$$

With probability $1 - 1/q$, we argue that XD^x is non-trivial and thus that we can build a DP solver from this event. If this ciphertext originates from a pre-challenge decryption query, then x is clearly independent of the view of the adversary (it is the opening of a perfectly hiding commitment scheme).

Regarding the post-challenge ciphertexts, either the trace is the same as a previous challenge and in that case the ciphertext is not valid, or again the value x is independent from the view of the adversary. In any case, the adversary cannot make use of a challenger's simulated proof to trigger this event.

Conclusion: To conclude, we argue that the random bit b is statistically hidden in $\mathsf{Game}_{10}(\lambda)$. First, the public-key component h is generated as g^{ω} for random scalar ω and for each $\mathcal{H}$-query $(\hat{g}, \mathbf{ovk})$ that generates the intermediate tuple and defines the output $(\hat{E}, G, F)$ let us write $(F, G, H) = (g^z f^w, g^w, h^w))$ for random $w, z \in \mathbb{Z}_q$.

Second, the CPA part $\boldsymbol{c}_j^*$ of the answer CT_j^* to the valid and non-aborting challenge query $(\mathsf{CT}_j^0, \mathsf{CT}_j^1)$ can be written as

$$\begin{aligned}\boldsymbol{c}_j^* &= \left(c_0^{j,b}, c_1^{j,b}, c_2^{j,b}\right) \cdot (f, g, h)^{\theta'_j} \cdot \left(F_j^*, G_j^*, H_j^*\right)^{\theta''_j} \\ &= \left(c_0^{j,b}, c_1^{j,b}, c_2^{j,b}\right) \cdot (f, g, h)^{\theta'_j + w_j^*\theta''_j} \cdot \left(g^{z_j^*\theta''_j}, 1, 1\right)\end{aligned}$$

for the random coins $w_j^*, z_j^* \in \mathbb{Z}_q$ defined previously in the $\mathcal{H}$-query $(\hat{g}, \mathbf{ovk}_j^*)$ and that can be used a single time since the trace $\mathbf{ovk}_j^*$ cannot be repeated through different challenge queries. Hence, given $(G_j^*, H_j^*) = (g,h)^{w_j^*}$ and $F_j^* = g^{z_j^*} f^{w_j^*}$, the computation of $\boldsymbol{c}_j^*$ still perfectly hides $\boldsymbol{c}_j^b$. Indeed, w_j^* and z_j^* makes G_j^* and F_j^* uniform in $\mathbb{G}$ and $H_j^* = (G_j^*)^{\omega}$. Moreover, $\theta_j' + w_j^*\theta_j''$ perfectly randomizes $(c_1^{j,b}, c_2^{j,b})$ with the component (g,h) which makes it independent of b. That is because all valid non-aborting challenge queries $(\mathsf{CT}_j^0, \mathsf{CT}_j^1)$ are also computed with perfectly sound proofs that their respective CPA part $(\boldsymbol{c}_j^0, \boldsymbol{c}_j^1)$ are honest due to the change introduced in $\mathsf{Game}_{10}(\lambda)$. Then, $(c_1^{j,0}, c_2^{j,0}), (c_1^{j,1}, c_2^{j,1}) \in \mathrm{span}\langle(g,h)\rangle$. Furthermore, all the $c_0^{j,b}$ components are also perfectly hidden by the corresponding z_j as long as $z_j^* \neq 0$, which always holds but with probability $1/q$.

Third, the validity proof $\pi_{\mathsf{Valid}}^{j,*}$ is simulated independently of b since the proof is given the above $\boldsymbol{c}_j^*$ and uses the witness (x_j^*, y_j^*) of $\hat{E}_j^* = \hat{g}^{x_j^*}\hat{h}^{y_j^*}$.

Fourth, the tracing part $\boldsymbol{\sigma}_{\mathsf{Trace}}^{j,*} = (\boldsymbol{C}_Z^{j,*}, \boldsymbol{C}_R^{j,*}, \boldsymbol{\sigma}_2^{j,*}, \boldsymbol{\sigma}_3^{j,*}, \hat{\boldsymbol{\pi}}_{sig}^{j,*})$ is computed such that $\boldsymbol{\sigma}_2^{j,*} = \boldsymbol{\sigma}_2^{j,0} = \boldsymbol{\sigma}_2^{j,1}$ and $\boldsymbol{\sigma}_3^{j,*} = \boldsymbol{\sigma}_3^{j,0} = \boldsymbol{\sigma}_3^{j,1}$ since CT_j^* is computed only for non-aborting query as defined from $\mathsf{Game}_3(\lambda)$. Also, the Groth-Sahai proof $(\boldsymbol{C}_Z^{j,*}, \boldsymbol{C}_R^{j,*}, \hat{\boldsymbol{\pi}}_{sig}^{j,*})$ is a valid proof that there exists a valid LHSP signature for $\mathbf{ovk}_j^*$ on the vector $(g, c_0^{j,*}, c_1^{j,*})$ composed from $\boldsymbol{c}_j^*$. Since $\mathbf{crs}_\sigma$ is in the hiding mode, the perfect witness indistinguishability thus reveals nothing about how the underlying signatures has been performed, and then remains independent of $\boldsymbol{\sigma}_{\mathsf{trace}}^{j,b}$ (because the ability of computing such a proof was also independent of b). Consequently, $Pr[S_{10}] \leq 1/2 + m/q$ for the ROM-based construction, and $Pr[S_{10}] \leq 1/2 + 1/q$ for the standard model construction. For the ROM-based construction, we find

$$\Pr[\mathsf{Exp}^{\mathrm{mtcca}}_{\mathcal{A},\Pi^{\mathbb{G}}_{PC}}(\lambda) = 1] \leq \frac{1}{2} + 4\varepsilon_{\mathrm{SXDH}} + \frac{q_{\mathcal{H}}(q_{\mathcal{H}} - 1)/2 + (2m+8)}{q}$$

where m is the number of challenge queries and $q_{\mathcal{H}}$ is the number of $\mathcal{H}$-queries of the form $(\hat{g}, \mathbf{ovk})$ made by the adversary. For the standard model construction, we have $\Pr[\mathsf{Exp}^{\mathrm{tcca}}_{\mathcal{A},\Pi^{\mathbb{G}}_{PC}}(\lambda) = 1] \leq \frac{1}{2} + 4\varepsilon_{\mathrm{SXDH}} + \varepsilon_{cr} + \frac{9}{q}$ □

4.4. Public coin construction for encrypting a sequence of bits

So far, all of our constructions are tailored to the encryption of a group element. While this works well for an election based on a mixnet tally,

these TREncs are not compatible for an election based on homomorphic encryption. We thus propose another TREnc that supports the encryption of ℓ bits.[17] As the previous construction, this construction uses a public coin CRS and can be adapted in the ROM to achieve tight mTCCA security.

4.4.1. Intuition

We now want to encrypt a vector of messages $\boldsymbol{m} = (m_1, \ldots, m_\ell)$ where each of the m_i are either 0 or 1. Our ciphertext will again have the structure $\mathsf{CT} = (\boldsymbol{c}, \mathit{trace}, \boldsymbol{\sigma}_{\mathsf{trace}}, \boldsymbol{\pi}_{\mathsf{valid}})$. Naturally, the CPA part $\boldsymbol{c}$ needs to be adapted to contain one component for each bit and the proof of validity $\boldsymbol{\pi}_{\mathsf{valid}}$ needs to prove that each of the bits in the CPA part is indeed a bit. The $\boldsymbol{\sigma}_{\mathsf{trace}}$ component will be very similar as in $\Pi^{\mathbb{G}}_{PC}$ again with a LHSP signature whose public key osk is the *trace*, but is still adapted to sign each of the message bits in a compact manner.

We start with the description of the CPA part. We compute a homomorphic multi-base ElGamal-like encryption with randomness-reuse as $\boldsymbol{c} = (d_1, d_2, c_1, \ldots, c_\ell) \in \mathbb{G}^{\ell+2}$ such that

$$\boldsymbol{c} = (1, 1, g_1^{m_1}, \ldots, g_\ell^{m_\ell}) \cdot (g, h, f_1, \ldots, f_\ell)^\theta,$$

where $f_i = g^{\alpha_i} h^{\beta_i}$ and g_i are part of the public key, for all $1 \leq i \leq \ell$, and the secret key is $\mathsf{SK} = (\mathsf{sk}_i)_{i=1}^{\ell} = (\alpha_1, \beta_1, \ldots, \alpha_\ell, \beta_\ell)$ which allows decrypting $\boldsymbol{c}$ to $g_1^{m_1}, \ldots, g_\ell^{m_\ell}$. Hence, the CPA part can be seen as the previous CPA parts, with every bit having its own f_i and secrets α_i, β_i. While we do not need at first glance a different base g_i to encode each bit, it will be useful in the $\boldsymbol{\sigma}_{\mathsf{trace}}$ to compress the signature, which we describe now:

- Naively, we can easily adapt the LHSP signature of the previous constructions with a longer row authenticating each of $(c_i)_{i=1}^{\ell}$, or in other words use $\mathsf{lk} = \mathsf{osk}$ to authenticate the row space of the matrix $\mathbf{T} = (T_{i,j})_{1 \leq i,j \leq \ell+2}$

$$\mathbf{T} = \begin{pmatrix} g & c_1 & \ldots & c_\ell & d_1 \\ 1 & f_1 & \ldots & f_\ell & g \\ 1 & F_1 & \ldots & F_\ell & G \end{pmatrix}, \tag{4.6}$$

[17] While we focus in this section on a proof that a ciphertext contains a sequence of bits, the techniques that we use can be adapted to prove more general relations on these bits. For example, it is possible to adapt the construction to common constraints, such as enforcing a lower and/or an upper bound on the number of ones in a ballot.

with random $(F_i)_{i=1}^{\ell}$ in the public key.

- However, we can compress each row into a row of only 3 elements to shorthen **ovk** down to 3 elements of $\hat{\mathbb{G}}$. Indeed, we compute $c = \prod_{i=1}^{\ell} c_1$ and $f = \prod_{i=1}^{\ell} f_i$ and authenticate the rows (g, c, d_1) and $(1, f, g)$. This is possible because $c = g_1^{m_1} \dots g_\ell^{m_\ell} f^\theta = M f^\theta$ is a computationally binding commitment on $\boldsymbol{m}$, which is the reason we are using a different base for each message. In other words, the security of the LHSP prevents an adversary from computing a signature on a different message M' (without breaking the SXDH assumption) and the discrete logarithm assumption prevents an adversary from finding another opening of M.

  Accordingly, we also use $(1, F, G)$ as the last row of the matrix, where (F, G) is either generated randomly in the key generation, or it is generated as the hash of the *trace* of each ciphertext for the tight variant in the ROM.

- Also, even if we have only the pair (F, G) by challenge ciphertext to hide ℓ components $(c_i)_{i=1}^{\ell}$ of the CPA part, we can again use the random-self reducibility of the SXDH assumption to have enough random group elements to hide all of these components (but here, for each challenge encryption).

  Regarding the tightness, we also use this random-self reducibility to generate all the pairs (F_i, G_i) for all the challenges from only one SXDH instance, similarly to our previous construction.

- Finally, we use Groth-Sahai proofs to hide the signature σ_1 similarly to the previous constructions and for the same reasons.

Now, we turn to the randomizable proof $\boldsymbol{\pi}_{\mathsf{Valid}}$ that $\boldsymbol{c}$ is well-formed. To ensure that each triple (d_1, d_2, c_i) for $i \in [\ell]$ is indeed an ElGamal-like encryption of a bit m_i, we can prove the following relations:

$$\begin{aligned} d_1 &= g^\theta & d_2 &= h^\theta & c_i &= g_i^{m_i} f_i^\theta && (4.7) \\ &= d_1^{m_i} g^{\vartheta_i} & & & &= c_i^{m_i} f_i^{\vartheta_i}, && (4.8) \end{aligned}$$

where $\vartheta_i = \theta(1 - m_i)$. It turns out that satisfying these equations implies that (d_1, d_2, c_i) is indeed an encryption of a bit. Intuitively, Equation 4.7 implies that the ciphertext (d_1, d_2, c_i) encrypts some message $m_i \in \mathbb{Z}_q$, and Equation 4.8 then implies that $(d_1, c_i)^{(1-m_i)}$ is an ElGamal encryption of 0. This leads to $m_i(1 - m_i) = 0$ and thus $m_i \in \{0, 1\}$ since $\mathbb{Z}_q$ is a field. This overall implication essentially holds because there is

no g_i component in the equation $c_i = c_i^{m_i} f_i^{\vartheta_i}$ and there is no need to prove that $\vartheta_i = \theta(1-m_i)$; only the existence of some common ϑ_i matters. Next, we show how we can build a proof system for the above statement that is compatible with the TCCA notion.

- First, we could use the same simulation proof technique as the previous construction, as all the relations of the equations 4.7 and 4.8 are linear. However, this would be expensive. Indeed, to take advantage of the perfectly hiding property of the Pedersen commitment E, we need two commitments $X_i \in \mathbb{G}^2$ and $Y_i \in \mathbb{G}^2$ for each relation that needs to be proven, which results in a proof size of $6l + 4$ elements in $\mathbb{G}$ (for the commitments) and $2l + 1$ elements in $\hat{\mathbb{G}}$ (for the responses).

  We would like to build a more efficient proof.

- To do so, we use the randomizable Groth-Sahai system for scalars with another CRS, where the CRS can also be derived from the random oracle to ensure no one knows any hidden discrete-log relation. For these linear relations, the Groth-Sahai proofs are computationally zero-knowledge, as desired. Also, these proofs can be adapted to a rerandomization of $\boldsymbol{c}$ and perfectly randomized.

- However, these proofs alone are not simulation-sound, and we thus combine them with the well-known following technique. We make a disjunctive proof that: 1) either the relations of the equations 4.7 and 4.8 hold, 2) or we know an opening of a random commitment that we program with the hash function evaluated on the trace of the ciphertext.[18] Of course, in a normal behavior of the encryption algorithm, we always prove honestly the first branch and simulate the second branch as we cannot know an opening without breaking the hiding property of the commitment. In our reductions though, we will switch to proving the second branch when we embed our SXDH instance in our randomized $\boldsymbol{c}$ as the relations of the first branch do not hold anymore.[19] Such simulated proofs are tied to a given trace as the commitment is derived from the trace, and hence we achieve simulation-soundness.

  More precisely, the commitment C of the second branch of the disjunctive proof is a Pedersen commitment $g^x h^y$ for random $x, y \in$

[18] In the standard model case, we prove instead the knowledge of an unique commitment given in pk.

[19] Likewise in the standard model case, we will switch to proving the second branch. This will naturally be possible for only one challenge in this case, hence the non-tightness.

$\mathbb{Z}_q$ and is derived from $\mathcal{H}(\hat{g}, \mathsf{ovk})$, or $C_0^{\tau} C_1$ for the standard model version where C_0 and C_1 are contained in the public key. It is reminiscent of our $\hat{E}$ commitment of the previous TREnc and similarly, if the adversary manages to simulate the first branch and prove the second branch for a fresh trace (that is, a trace which does not appear yet in any challenge), then with overwhelming probability this opening is different from the x we programmed and we will break the binding property of the commitment. For ciphertexts with a trace appearing in a challenge, the challenger will simply refuse to decrypt the ciphertext.

- To instantiate this disjunctive proof technique, we rely on a correlated Groth-Sahai CRS technique [Ràf15]. The main idea is to generate in the public key a CRS $\mathbf{crs}_{\pi}$ and to generate in the ciphertext two CRSes $\mathbf{crs}_0$ and $\mathbf{crs}_1$ used respectively in the first and second branches of the disjunctive proof such that $\mathbf{crs}_{\pi} = \mathbf{crs}_0 \cdot \mathbf{crs}_1$. $\mathbf{crs}_{\pi}$ is generated such that for any pair of these correlated CRSes at most one of them can be generated in the "simulation" mode and the other one must then produce sound proofs. Naturally, an honest ciphertext will set the $\mathbf{crs}_0$ in the "sound" mode (to prove equations 4.7 and 4.8) and $\mathbf{crs}_1$ in the "simulation" mode (to simulate the proof of knowledge of the opening of C).

  This technique is particularly appealing because for the TCCA security, we need to be able to rerandomize these CRSes. Fortunately, the Groth-Sahai proof system allows us to rerandomize them and to adapt the corresponding proofs, which answer our needs. We stress that this general correlated CRS proof system was not designed with rerandomizability. Fortunately, we observe that it is easily compatible with our needs. Indeed, we can rerandomize the two correlated CRSes $\mathbf{crs_0}$ and $\mathbf{crs_1}$ and adapt their related proofs for the branches accordingly and then proceed with the usual adaptation and rerandomization of these proofs based and the known Groth-Sahai properties.

  In more detail, we say that a $\mathbf{crs} = (\boldsymbol{\varphi}, \boldsymbol{\psi})$ is in the simulation mode if there exists a trapdoor $\kappa \in \mathbb{Z}_q$ such that $\boldsymbol{\varphi} = \boldsymbol{\psi}^{\kappa}$. The Groth-Sahai allows us to simulate the proofs for such $\mathbf{crs}$ using κ. On the other hand, we say that a $\mathbf{crs} = (\boldsymbol{\varphi}, \boldsymbol{\psi})$ is in the sound mode if there is no such trapdoor, or in other words if $\mathbf{crs}$ is uniform[20] in

[20] In the security proof, an uniform $\mathbf{crs}$ can be generated so that for any $\mathbf{crs_0} \cdot \mathbf{crs_1} = \mathbf{crs}$ and valid proofs in a ciphertext, as long as either $\mathbf{crs_0}$ or $\mathbf{crs_1}$ is in the simulation mode, the other one allows extracting a witness.

$\hat{\mathbb{G}}^4$.

So in the public key generation, we generate $\mathbf{crs}_\pi$ in the sound mode. Hence with overwhelming probability there is no trapdoor κ such that $\boldsymbol{\varphi}_\pi = \boldsymbol{\psi}_\pi^\kappa$ and it is not possible to generate both $\mathbf{crs}_0$ and $\mathbf{crs}_1$ in the simulation mode. At some point in our reduction, we will switch $\mathbf{crs}_\pi$ to the simulation mode, so that $\mathbf{crs}_0^*$ and $\mathbf{crs}_1^*$ of the challenges can be simultaneously in the simulation mode, thanks to an SXDH transition. From there, we will simulate the proof related to the validity of the CPA encryption and then switch back $\mathbf{crs}_\pi$ to the the sound mode again.

- However, we need to be careful when we switch between the modes of theses CRSes in the TCCA reduction. Especially, after $\mathbf{crs}_\pi$ is switched to the simulation mode, we want to generate fresh CRSes $\mathbf{crs}_0^*$ and $\mathbf{crs}_1^*$ in the simulation mode instead of randomizing them. If those CRSes are both already in the simulation mode, then this transition is perfect as the fresh CRSes have the same distribution as the randomized ones.

  Unfortunately, an issue arises when these $\mathbf{crs}_0$ and $\mathbf{crs}_1$ of a challenge ciphertext are both in the sound mode, which does prevent their product from being in the simulation mode. In that case, we cannot replace them by fresh CRSes in the simulation mode and we are stuck in the reduction. Furthermore, we cannot rely at this point on the fact that the adversary cannot produce an opening of the commitment with our programmed x committed value. This is because, before this transition, $\mathbf{crs}_1$ is still in the simulation mode and we cannot extract an opening from such ciphertexts and then compute the solution of the double pairing problem.

  To solve this issue where $\mathbf{crs_0}$ and $\mathbf{crs_1}$ would both be in the sound mode when $\mathbf{crs}$ is in the simulation mode, we modify the second branch of the OR proof to enforce the simulation of a false statement. Therefore, it is impossible to have $\mathbf{crs_1}$ in the sound mode as the extraction would lead to a true statement. The reason why we have to modify the second branch is because there always exists an opening of C so we really have to add additional statement in the second branch.

  Hence, we force the adversary to simulate the second branch (and thus to use $\boldsymbol{crs}_2$ in the simulation mode) by ensuring that the statement in the second branch is initially false. We proceed by generating random $(A, B) \in \mathbb{G}^2$ and prove the statement that they are in $\mathrm{span}\langle(g, h)\rangle$, which should be false with overwhelming

probability.

Still, we need to be able to create a sound proof of the second branch when we simulate the first branch and we switch back to $\mathbf{crs}_\pi$ in the sound mode. To do so, we generate (A, B) in $\text{span}\langle(g, h)\rangle$ before the next transition to switch the distribution of $\mathbf{crs}_\pi$ batck to the sound mode with an SXDH reduction. That way, $(A, B) \in \text{span}\langle(g, h)\rangle$ is a true statement.

This concludes our proof of validity, which consists of a (sound) proof $\boldsymbol{\pi}_0$ of the statements in equations 4.7 and 4.8, and a (fake) proof $\boldsymbol{\pi}_1$ of knowledge of an opening of a random commitment C derived from the trace and a proof that for some random values, (g, h, A, B) is a DDH tuple.

Finally, one could notice that since we have Groth-Sahai commitments of each of the m_i to prove their validity, we could extract the messages to answer the decryption requests of the TCCA game and hence we could use a shorter key $\mathsf{sk}_i = \alpha_i$ and $f_i = g^{\alpha_i}$ to save some elements. However, this is not possible for the traceability game. To show traceability, we at least have to ensure that $M = g_1^{m_1} \cdots g_\ell^{m_\ell}$ in $c = \prod_{i=1}^{\ell} c_i$ cannot be modified. Even if $\boldsymbol{\pi}_{\mathsf{Valid}}$ must imply the validity of the apparently randomized $\boldsymbol{c}$, that does not mean that it cannot be of the form $(d_1', (c_i')_{i=}^{\ell}) = (g^{\theta+\theta'}, (g_i^{m_i'} f_i^{\theta+\theta'})_{i=1}^{\ell}$ with $M \neq M' = g_1^{m_1'} \cdots g_\ell^{m_\ell'}$. To show that this reduces to forging an LHSP signature, we first have to make sure that (g, c', d_1') with $c' = \prod_{i=1}^{\ell} c_i'$ is outside the linear vector space spanned by the signed three vectors. However, if $(1, F, G)$ is random it is not in the linear vector space spanned by (g, c, d_1) and $(1, f, g)$ with overwhelming probability. That means that by definition, there is no forgery as all the vector space $\mathbb{G}^3$ is already authenticated. We thus have to turn back $(1, F, G)$ in the two-dimensional vector space generated by (g, c, d_1) and $(1, f, g)$ in the proof, which leaves us with a single possibility of $(1, F, G) = (1, f, g)^\mu$ for any (random) $\mu \in \mathbb{Z}_q$.

Now, it is easy to see why traceability cannot hold since we would have $F = G^\alpha$ with $\alpha = \sum_{i=1}^{\ell} \alpha_i$ and since the adversary is given $\mathsf{SK} = (\mathsf{sk}_i)_{i=1}^{\ell}$, it would distinguish the transition. Our additional elements $h \in \mathsf{PK}$ and thus $d_2 \in \boldsymbol{c}$ basically ensure that no party knows neither the discrete logarithm of f_i in base g nor the one of F in base G.

4.4.2. Description

We simultaneously define a construction in the standard model Π_{PC}^{bit} and its variant in the ROM where $\mathcal{H}$ is modeled as a random oracle.

Gen(1^λ): Run $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and let $\ell = \mathsf{poly}'(\lambda)$.

1. Generate $A, B, C_0, C_1, F, G, (g_i)_{i=1}^{\ell}, h \leftarrow\!\$\, \mathbb{G}^{\ell+7}$ and $\hat{h} \leftarrow\!\$\, \hat{\mathbb{G}}$. *In the ROM construction*, ignore C_0, C_1, F, G and generate all the other values from $\mathcal{H}(g, \hat{g}, \ell)$ instead, where $g, \hat{g} \in \mathsf{pp}$.
2. Pick a random pair $\mathsf{sk}_i = (\alpha_i, \beta_i) \leftarrow\!\$\, \mathbb{Z}_q^2$ and compute $f_i = g^{\alpha_i} h^{\beta_i} \in \mathbb{G}$, for each $i \in [\ell]$.
3. Generate two tuples of four random group elements $(\mathbf{crs}_\sigma, \mathbf{crs}_\pi) \leftarrow\!\$\, \mathbb{G}^4 \times \hat{\mathbb{G}}^4$ such that $\mathbf{crs}_\sigma = (\boldsymbol{u}_1, \boldsymbol{u}_2)$ is seen as a Groth-Sahai CRS to commit to group elements over $\mathbb{G}$ and $\mathbf{crs}_\pi = (\boldsymbol{\varphi}, \boldsymbol{\psi})$ is seen as a Groth-Sahai CRS to commit to scalars over $\hat{\mathbb{G}}$. *In the ROM construction*, generate these CRSes from $\mathcal{H}(g, \hat{g}, (f_i)_{i=1}^{\ell})$ instead.

   For simplicity, we write Com for the commitment algorithm $\mathsf{Com}(\mathbf{crs}_\sigma, \cdot)$.

The private key consists of $\mathsf{SK} = (\mathsf{sk}_i)_{i=1}^{\ell} = (\alpha_i, \beta_i)_{i=1}^{\ell}$ and the public key $\mathsf{PK} \in \mathbb{G}^{12+2\ell} \times \hat{\mathbb{G}}^6$ is

$$\mathsf{PK} = \left(g, h, (g_i, f_i)_{i=1}^{\ell}, A, B, C_0, C_1, F, G, \mathbf{crs}_\sigma, \mathbf{crs}_\pi, \hat{g}, \hat{h}\right).$$

In the ROM construction, the public key can be derived from pp and $(f_i)_{i=1}^{\ell} \in \mathbb{G}^\ell$ only and there is no value C_0, C_1, F, G in PK.

Enc($\mathsf{PK}, \boldsymbol{m}$): Given a message $\boldsymbol{m} = (m_1, \ldots, m_\ell) \in \{0,1\}^\ell$ to encrypt, first run $\underline{\mathsf{LGen}(\mathsf{PK})}$: Generate a key pair $(\mathbf{osk}, \mathbf{ovk})$ for the one-time LHSP recalled in Section 2.2.4 from the public generators $\hat{g}, \hat{h}$ in order to sign vectors of dimension 3. Let $\mathsf{lk} = \mathbf{osk} = \{(\eta_i, \zeta_i)\}_{i=1}^{3}$ be the private key, of which the corresponding public key is $\mathbf{ovk} = \{\hat{l}_i\}_{i=1}^{3}$. From $\mathbf{ovk}$, compute $\tau = \mathcal{H}(\hat{g}, \mathbf{ovk})$ and $C = C_0^\tau C_1$. *In the ROM construction*, derive $(C, F, G) = \mathcal{H}(\hat{g}, \mathbf{ovk})$.

Then, conduct the following steps of $\underline{\mathsf{LEnc}(\mathsf{PK}, \mathsf{lk}, \boldsymbol{m})}$:

1. Pick $\theta \leftarrow\!\$\, \mathbb{Z}_q$ and compute the CPA encryption $\mathbf{c} = (d_1, d_2, (c_i)_{i=1}^{\ell})$, where $d_1 = g^\theta$, $d_2 = h^\theta$ and $c_i = g_i^{m_i} f_i^\theta$ for each $i \in [\ell]$. Keep the random coin θ.
   Next steps 2-3 are dedicated to the tracing part $\boldsymbol{\sigma}_{\mathsf{trace}}$.
2. To allow tracing, use $\mathsf{lk} = \mathbf{osk}$ to authenticate the row space

of the matrix $\mathbf{T} = \left(T_{i,j}\right)_{1 \le i,j \le 3}$

$$\mathbf{T} = \begin{pmatrix} g & c & d_1 \\ 1 & f & g \\ 1 & F & G \end{pmatrix} , \tag{4.9}$$

where $c = \prod_{i=1}^{\ell} c_i$ and $f = \prod_{i=1}^{\ell} f_i$. by using $\mathsf{lk} = \mathsf{osk}$. Namely, sign each row $\boldsymbol{T}_i = (T_{i,1}, T_{i,2}, T_{i,3})$ resulting in $\boldsymbol{\sigma} = (\boldsymbol{\sigma}_i)_{i=1}^{3} \in \mathbb{G}^6$, where $\boldsymbol{\sigma}_i = (R_i, S_i) \in \mathbb{G}^2$.

3. Commit to $\boldsymbol{\sigma}_1 = (R_1, S_1)$ using $\mathbf{crs}_\sigma$ as $\boldsymbol{C}_R = \mathsf{Com}(R_1)$, $\boldsymbol{C}_S = \mathsf{Com}(S_1)$. To ensure that $\boldsymbol{\sigma}_1$ is a valid one-time LHSP signature on (g, c, d_1), compute the Groth-Sahai proof $\hat{\boldsymbol{\pi}}_{sig} \in \hat{\mathbb{G}}^2$ that

$$e(R_1, \hat{g}) \cdot e(S_1, \hat{h}) = e(g, \hat{l}_1) \cdot e(c, \hat{l}_2) \cdot e(d_1, \hat{l}_3) .$$

Next steps are dedicated to the validity proof $\boldsymbol{\pi}_{\mathsf{valid}}$.

4. For the OR-proof, generate two correlated Groth-Sahai CRSes $\mathbf{crs}_i = (\boldsymbol{\varphi}_i, \boldsymbol{\psi}) \in \hat{\mathbb{G}}^4$ to later produce the proof $\boldsymbol{\pi}_i$, for $i \in \{0, 1\}$: Initialize a simulation bit b_{sim} to 1 and pick a random $\gamma \in \mathbb{Z}_q$. Then, compute $\boldsymbol{\varphi}_0$ and $\boldsymbol{\varphi}_1$ such that $\boldsymbol{\varphi} = \boldsymbol{\varphi}_0 \boldsymbol{\varphi}_1$ and $\boldsymbol{\varphi}_{b_{\mathsf{sim}}} = \boldsymbol{\psi}^{\gamma}$ over $\hat{\mathbb{G}}^2$.

5. To compute the proof $\boldsymbol{\pi}_0 = (\boldsymbol{\pi}_{\mathsf{ciph}}, \hat{\boldsymbol{c}}_{\mathsf{ciph}})$, commit to the $2\ell + 1$ (witness) scalars θ, m_i and $\vartheta_i = \theta(1 - m_i)$ with $\mathbf{crs}_0$, for all $i \in [\ell]$, resulting in $\hat{\boldsymbol{c}}_0 = \mathsf{Com}_0(\theta; \rho_0)$, $\hat{\boldsymbol{c}}_i = \mathsf{Com}_0(m_i; \rho_i)$ and $\hat{\boldsymbol{d}}_i = \mathsf{Com}_0(\vartheta_i; \tau_i)$, for some random coin $\boldsymbol{\rho}_{\mathsf{ciph}} = (\rho_0, \{\rho_i, \tau_i\}_{i=1}^{\ell})$, where Com_0 denotes $\mathsf{Com}_{\mathbf{crs}_0}$. Let $\hat{\boldsymbol{c}}_{\mathsf{ciph}} = (\hat{\boldsymbol{c}}_0, \{\hat{\boldsymbol{c}}_i, \hat{\boldsymbol{d}}_i\}_{i=1}^{\ell}) \in \hat{\mathbb{G}}^{4\ell+2}$.

6. Compute the Groth-Sahai proof $\boldsymbol{\pi}_{\mathsf{ciph}} \in \mathbb{G}^{3\ell+2}$ on input $\boldsymbol{\rho}_{\mathsf{ciph}}$ to show that the equations (4.7) and (4.8) hold for all $i \in [\ell]$, resulting in $3\ell + 2$ equations. To be explicit, $\boldsymbol{\pi}_{\mathsf{ciph}} = (\pi_{1,0}, \pi_{2,0}, (\pi_{1,i}, \pi_{2,i}, \pi_{3,i})_{i=1}^{\ell})$, where

$$\pi_{1,0} = g^{\rho_0} \qquad \pi_{2,0} = h^{\rho_0} \qquad \pi_{2,i} = g_i^{\rho_i} f_i^{\rho_0} \tag{4.10}$$

$$\pi_{1,i} = d_1^{\rho_i} g^{\tau_i} \qquad \pi_{3,i} = c_i^{\rho_i} f_i^{\tau_i}. \tag{4.11}$$

7. To compute the proof $\boldsymbol{\pi}_1 = (\hat{\boldsymbol{c}}_{\mathsf{Enc}}, \boldsymbol{\pi}_{\mathsf{Enc}}, \hat{\boldsymbol{c}}_{\mathsf{Com}}, \pi_{\mathsf{Com}})$, on input $\mathbf{crs}_1$ and the simulation key γ in $\boldsymbol{\varphi}_1 = \boldsymbol{\psi}^{\gamma}$, simulate that $(A, B) \in \langle (g, h) \rangle$ and a knowledge of an opening of the commitment C.

- For the encryption (A, B), compute a fake commitment $\hat{\boldsymbol{c}}_{\mathsf{Enc}} = \boldsymbol{\psi}^{\rho_z}$ and the simulated proof $\boldsymbol{\pi}_{\mathsf{Enc}} = (\pi_A, \pi_B) = (A, B)^{-\gamma}(g, h)^{\rho_z} \in \mathbb{G}^2$ as if $\hat{\boldsymbol{c}}_{\mathsf{Enc}}$ was a commitment to z such that $(A, B) = (g, h)^z$.
- For the commitment C, compute fake commitments $\hat{\boldsymbol{c}}_x = \boldsymbol{\psi}^{\rho_x}$ and $\hat{\boldsymbol{c}}_y = \boldsymbol{\psi}^{\rho_y}$ for random $\rho_x, \rho_y \in \mathbb{Z}_q$ as well as the simulated proof $\pi_{\mathsf{Com}} = C^{-\gamma} g^{\rho_x} h^{\rho_y} \in \mathbb{G}$ as if $\hat{\boldsymbol{c}}_x$ and $\hat{\boldsymbol{c}}_y$ were commitments to an opening (x, y) of $C = g^x h^y$. Let $\hat{\boldsymbol{c}}_{\mathsf{Com}} = (\hat{\boldsymbol{c}}_x, \hat{\boldsymbol{c}}_y)$.

Output the ciphertext

$$\mathsf{CT} = \Big(\mathbf{c}, \mathbf{ovk}, \boldsymbol{\sigma}_{\mathsf{trace}}, \boldsymbol{\pi}_{\mathsf{valid}}\Big),$$

where $\boldsymbol{\sigma}_{\mathsf{trace}} = (\boldsymbol{C}_R, \boldsymbol{C}_S, \boldsymbol{\sigma}_2, \boldsymbol{\sigma}_3, \hat{\boldsymbol{\pi}}_{sig}) \in \mathbb{G}^8 \times \hat{\mathbb{G}}^2$ and $\boldsymbol{\pi}_{\mathsf{valid}} = (\boldsymbol{\varphi}_0, \boldsymbol{\pi}_0, \boldsymbol{\pi}_1)$ with $\boldsymbol{\pi}_0 \in \mathbb{G}^{3\ell+2} \times \hat{\mathbb{G}}^{4\ell+2}$ and $\boldsymbol{\pi}_1 \in \mathbb{G}^3 \times \hat{\mathbb{G}}^6$ up to reordering.

Trace(PK, CT): Parse PK and CT as above, and output **ovk**.

Rand(PK, CT): Parse the ciphertext as above, and let (C, F, G) be computed from $\mathcal{H}(\hat{g}, \mathbf{ovk})$ and PK as above. Conduct the following steps:

1. Randomize the CPA part $\mathbf{c} = (d_1, d_2, (c_i)_{i=1}^{\ell})$: Pick $\theta' \leftarrow\!\!\$\, \mathbb{Z}_q$ and compute $\mathbf{c}' = (d_1', d_2', (c_i')_{i=1}^{\ell}) = (d_1 g^{\theta'}, d_2 h^{\theta'}, (c_i f_i^{\theta'})_{i=1}^{\ell}) = \mathbf{c} \cdot (g, h, (f_i)_{i=1}^{\ell})^{\theta'}$.
   Next steps 2-3 are dedicated to the tracing part $\boldsymbol{\sigma}'_{\mathsf{trace}}$.
2. Adapt the commitment $(\boldsymbol{C}_R, \boldsymbol{C}_S)$ to the signature $\boldsymbol{\sigma}_1$ so that it becomes a commitment $(\tilde{\boldsymbol{C}}_R, \tilde{\boldsymbol{C}}_S)$ to the signature $\boldsymbol{\sigma}_1' = \boldsymbol{\sigma}_1 \boldsymbol{\sigma}_2^{\theta'}$. That is, parse $\boldsymbol{\sigma}_2$ as (R_2, S_2) and compute $\tilde{\boldsymbol{C}}_R = \boldsymbol{C}_R \cdot \iota(R_2^{\theta'})$ and $\tilde{\boldsymbol{C}}_S = \boldsymbol{C}_S \cdot \iota(S_2^{\theta'})$. Note: $\boldsymbol{\sigma}_1'$ is a valid signature on $(g, \prod c_i', d_1')$ for **ovk**, and $\hat{\boldsymbol{\pi}}_{sig}$ remains a proof that $\boldsymbol{\sigma}_1' = (R_1', S_1')$ satisfies

$$e(R_1', \hat{g}) \cdot e(S_1', \hat{h}) = e(g, \hat{l}_1) \cdot e(c, \hat{l}_2) \cdot e(d_1, \hat{l}_3)\,,$$

   where $(R_1', S_1') = (R_1, S_1) \cdot (R_2, S_2)^{\theta'}$.
3. Randomize the GS proof $((\tilde{\boldsymbol{C}}_R, \tilde{\boldsymbol{C}}_S), \hat{\boldsymbol{\pi}}_{sig})$ for the CRS $\mathbf{crs}_\sigma$ leading to $((\boldsymbol{C}'_R, \boldsymbol{C}'_S), \hat{\boldsymbol{\pi}}'_{sig})$. Let $\boldsymbol{\sigma}'_{\mathsf{trace}} = (\boldsymbol{C}'_R, \boldsymbol{C}'_S, \boldsymbol{\sigma}_2, \boldsymbol{\sigma}_3, \hat{\boldsymbol{\pi}}'_{sig})$[21] be the randomized tracing part.

[21] Since computing these LHSP signatures is deterministic in $\mathsf{lk} = \mathbf{osk}$, they cannot be randomized, and we do not need to randomize them to satisfy the strong randomizability notion.

Next steps are dedicated to the validity proof $\boldsymbol{\pi}'_{\mathsf{valid}}$.

4. Randomize the correlated Groth-Sahai CRSes of the OR-proof: Given $\boldsymbol{\varphi}_0$, compute $\boldsymbol{\varphi}_1$ such that $\boldsymbol{\varphi} = \boldsymbol{\varphi}_0\boldsymbol{\varphi}_1$ and let $\mathbf{crs}_i = (\boldsymbol{\varphi}_i, \boldsymbol{\psi})$, for $i \in \{0,1\}$. Then, pick a random $\gamma' \in \mathbb{Z}_q$, and compute $\boldsymbol{\varphi}'_0 = \boldsymbol{\varphi}_0\boldsymbol{\psi}^{-\gamma'}$ as well as $\boldsymbol{\varphi}'_1 = \boldsymbol{\varphi}_1\boldsymbol{\psi}^{\gamma'}$. Let $\mathbf{crs}'_i = (\boldsymbol{\varphi}'_i, \boldsymbol{\psi})$, for $i \in \{0,1\}$.
5. Adapt the proof $\boldsymbol{\pi}_0 = (\boldsymbol{\pi}_{\mathsf{ciph}}, \hat{\boldsymbol{c}}_{\mathsf{ciph}})$ with respect to the randomized $\mathbf{c}'$ and $\mathbf{crs}'_0$ thanks to (θ', γ'):
   - Adapt the witness in $\hat{\boldsymbol{c}}_{\mathsf{ciph}} = (\hat{\boldsymbol{c}}_0, \{\hat{\boldsymbol{c}}_i, \hat{\boldsymbol{d}}_i\}_{i=1}^{\ell})$: Compute $\tilde{\boldsymbol{c}}_0 = \hat{\boldsymbol{c}}_0 \cdot \boldsymbol{\varphi}_0^{\theta'}$ and $\tilde{\boldsymbol{d}}_i = \hat{\boldsymbol{d}}_i \cdot (\boldsymbol{\varphi}_0/\hat{\boldsymbol{c}}_i)^{\theta'}$, for $i = 1$ to ℓ. The proof part $\boldsymbol{\pi}_{\mathsf{ciph}}$ remains valid for $\mathbf{crs}_0$ and $\tilde{\boldsymbol{c}}_{\mathsf{ciph}} = (\tilde{\boldsymbol{c}}_0, \{\hat{\boldsymbol{c}}_i, \tilde{\boldsymbol{d}}_i\}_{i=1}^{\ell})$.
   - Adapt $\boldsymbol{\pi}_{\mathsf{ciph}} = (\pi_{1,0}, \pi_{2,0}, (\pi_{1,i}, \pi_{2,i}, \pi_{3,i})_{i=1}^{\ell})$ to $\mathbf{crs}'_0$ by computing $\tilde{\boldsymbol{\pi}}_{\mathsf{ciph}} = \boldsymbol{\pi}_{\mathsf{ciph}} \cdot (d'_1, d'_2, (d'_1, c'_i, c'_i)_{i=1}^{\ell})^{\gamma'}$.
6. Randomize the proof $\tilde{\boldsymbol{\pi}}_0 = (\tilde{\boldsymbol{\pi}}_{\mathsf{ciph}}, \tilde{\boldsymbol{c}}_{\mathsf{ciph}})$ for the CRS $\mathbf{crs}'_0$ leading to $\boldsymbol{\pi}'_0 = (\boldsymbol{\pi}'_{\mathsf{ciph}}, \hat{\boldsymbol{c}}'_{\mathsf{ciph}})$.
7. Adapt the proof $\boldsymbol{\pi}_1 = (\hat{\boldsymbol{c}}_{\mathsf{Enc}}, \boldsymbol{\pi}_{\mathsf{Enc}}, \hat{\boldsymbol{c}}_{\mathsf{Com}}, \pi_{\mathsf{Com}})$ to the randomized $\mathbf{crs}'_1$ thanks to γ'. Namely, compute $(\tilde{\pi}_A, \tilde{\pi}_B, \tilde{\pi}_{\mathsf{Com}}) = (\pi_A, \pi_B, \pi_{\mathsf{Com}}) \cdot (A, B, C)^{-\gamma'}$, where $\boldsymbol{\pi}_{\mathsf{Enc}} = (\pi_A, \pi_B)$. Then, randomize the proof $\tilde{\boldsymbol{\pi}}_1 = (\hat{\boldsymbol{c}}_{\mathsf{Enc}}, \tilde{\boldsymbol{\pi}}_{\mathsf{Enc}}, \hat{\boldsymbol{c}}_{\mathsf{Com}}, \tilde{\pi}_{\mathsf{Com}})$ for the CRS $\mathbf{crs}'_1$, where $\tilde{\boldsymbol{\pi}}_{\mathsf{Enc}} = (\tilde{\pi}_A, \tilde{\pi}_B)$, leading to $\boldsymbol{\pi}'_1 = (\hat{\boldsymbol{c}}'_{\mathsf{Enc}}, \boldsymbol{\pi}'_{\mathsf{Enc}}, \hat{\boldsymbol{c}}'_{\mathsf{Com}}, \pi'_{\mathsf{Com}})$.

Return the randomized ciphertext

$$\mathsf{CT}' = \Big(\mathbf{c}', \mathbf{ovk}, \boldsymbol{\sigma}'_{\mathsf{trace}}, \boldsymbol{\pi}'_{\mathsf{valid}} = (\boldsymbol{\varphi}'_0, \boldsymbol{\pi}'_0, \boldsymbol{\pi}'_1)\Big).$$

Ver(PK, CT): Conduct the following checks:

1. Verify whether PK and CT parse properly. If not, output 0. Else compute (C, F, G) from $\mathcal{H}(\hat{g}, \mathbf{ovk})$ and PK as above.

   Next steps 2-3 verify the tracing part $\boldsymbol{\sigma}_{\mathsf{trace}}$.

2. Verify the validity of the signatures $\boldsymbol{\sigma}_2$ and $\boldsymbol{\sigma}_3$ on the 2 last rows $\{\boldsymbol{T}_i\}_{i=2}^{3}$ of the matrix $\mathbf{T}$ in (4.9) with respect to $\mathbf{ovk} = \{\hat{l}_i\}_{i=1}^{3}$. Namely, parse $\boldsymbol{\sigma}_2 = (R_2, S_2)$ and $\boldsymbol{\sigma}_3 = (R_3, S_3)$ and check the next equations:

$$e(R_2, \hat{g})e(S_2, \hat{h}) = e(f, \hat{l}_2)e(g, \hat{l}_3)$$
$$e(R_3, \hat{g})e(S_3, \hat{h}) = e(F, \hat{l}_2)e(G, \hat{l}_3)$$

3. Verify the validity of the proof $\hat{\boldsymbol{\pi}}_{sig}$ that committed variables in $\boldsymbol{C}_R$ and $\boldsymbol{C}_S$ consist of a signature on the first row (g, c, d) of $\mathbf{T}$ under ovk.
Next steps 4-6 verify the validity proof part $\boldsymbol{\pi}_{\mathsf{valid}}$.
4. Given $\boldsymbol{\varphi}_0$, compute $\boldsymbol{\varphi}_1$ such that $\boldsymbol{\varphi} = \boldsymbol{\varphi}_0\boldsymbol{\varphi}_1$ and let $\mathsf{crs}_i = (\boldsymbol{\varphi}_i, \boldsymbol{\psi})$, for $i \in \{0, 1\}$.
5. Verify the proof $\boldsymbol{\pi}_0 = (\boldsymbol{\pi}_{\mathsf{ciph}}, \hat{\boldsymbol{c}}_{\mathsf{ciph}})$. Namely, parse the commitment $\hat{\boldsymbol{c}}_{\mathsf{ciph}} = (\hat{\boldsymbol{c}}_0, \{\hat{\boldsymbol{c}}_i, \hat{\boldsymbol{d}}_i\}_{i=1}^{\ell})$ and the proof $\boldsymbol{\pi}_{\mathsf{ciph}} = (\pi_{1,0}, \pi_{2,0}, (\pi_{1,i}, \pi_{2,i}, \pi_{3,i})_{i=1}^{\ell})$, and check

$$E_2(d_1, \boldsymbol{\varphi}_0)E_2(\pi_{1,0}, \boldsymbol{\psi}) = E_2(g, \hat{\boldsymbol{c}}_0)$$
$$E_2(d_2, \boldsymbol{\varphi}_0)E_2(\pi_{2,0}, \boldsymbol{\psi}) = E_2(h, \hat{\boldsymbol{c}}_0)$$
$$E_2(c_i, \boldsymbol{\varphi}_0)E_2(\pi_{2,i}, \boldsymbol{\psi}) = E_2(g_i, \hat{\boldsymbol{c}}_i)E_2(f_i, \hat{\boldsymbol{c}}_0)$$

for Equation (4.7) as well as

$$E_2(d_1, \boldsymbol{\varphi}_0)E_2(\pi_{1,i}, \boldsymbol{\psi}) = E_2(d_1, \hat{\boldsymbol{c}}_i)E_2(g, \hat{\boldsymbol{d}}_i)$$
$$E_2(c_i, \boldsymbol{\varphi}_0)E_2(\pi_{3,i}, \boldsymbol{\psi}) = E_2(c_i, \hat{\boldsymbol{c}}_i)E_2(f_i, \hat{\boldsymbol{d}}_i)$$

for Equation (4.8).
6. Verify the proof $\boldsymbol{\pi}_1 = (\hat{\boldsymbol{c}}_{\mathsf{Enc}}, \boldsymbol{\pi}_{\mathsf{Enc}}, \hat{\boldsymbol{c}}_{\mathsf{Com}}, \pi_{\mathsf{Com}})$. Namely, parse $\boldsymbol{\pi}_{\mathsf{Enc}} = (\pi_A, \pi_B)$ and $\hat{\boldsymbol{c}}_{\mathsf{Com}} = (\hat{\boldsymbol{c}}_x, \hat{\boldsymbol{c}}_y)$, and check

$$E_2(A, \boldsymbol{\varphi}_1)E_2(\pi_A, \boldsymbol{\psi}) = E_2(g, \hat{\boldsymbol{c}}_{\mathsf{Enc}})$$
$$E_2(B, \boldsymbol{\varphi}_1)E_2(\pi_B, \boldsymbol{\psi}) = E_2(h, \hat{\boldsymbol{c}}_{\mathsf{Enc}})$$
$$E_2(C, \boldsymbol{\varphi}_1)E_2(\pi_{\mathsf{Com}}, \boldsymbol{\psi}) = E_2(g, \hat{\boldsymbol{c}}_x)E_2(h, \hat{\boldsymbol{c}}_y)$$

Output 1 if all these checks pass, otherwise, output 0.

Dec(SK, PK, CT): If $\mathsf{Ver}(\mathsf{PK}, \mathsf{CT}) = 0$, output $\perp$. Otherwise, given $\mathsf{SK} = (\alpha_i, \beta_i)_{i=1}^{\ell}$ and $\mathbf{c} = (d_1, d_2, (c_i)_{i=1}^{\ell})$ included in CT, compute and output $\mathbf{m} = (m_i)_{i=1}^{\ell}$ such that $m_i = 0$ if $c_i = d_1^{\alpha_i} d_2^{\beta_i}$, and $m_i = 1$, otherwise.

4.4.3. Security

Theorem 17. *In the standard model, the above TREnc* Π_{PC}^{bit} *is statistically verifiable and statistically strongly randomizable, and computationally traceable and TCCA secure under the SXDH assumption relative to* Setup *if the* $\mathcal{H}$ *function is collision-resistant. In the random oracle model, the TREnc variant is statistically verifiable and statistically strongly randomizable, and computationally traceable and tightly mTCCA secure under the SXDH assumption relative to* Setup.

Table 4.1.: Size of a ciphertext and number of exponentiations needed to compute it.

Construction	Group elements	Exponentiations
Group-encoded constructions		
Π_{TS} [DPP22b]	$\mathbb{G}^{13} \times \hat{\mathbb{G}}^{5}$	$\mathbb{G}^{29} \times \hat{\mathbb{G}}^{10}$
$\Pi_{PC}^{\mathbb{G}}$ [DPP24a]	$\mathbb{G}^{15} \times \hat{\mathbb{G}}^{6}$	$\mathbb{G}^{25} \times \hat{\mathbb{G}}^{13}$
Bit-by-bit constructions		
BeleniosRF [Cha+16]	$\mathbb{G}^{6\ell+14} \times \hat{\mathbb{G}}^{6\ell+7}$	$\mathbb{G}^{9\ell+23} \times \hat{\mathbb{G}}^{8\ell+17}$
Π_{PC}^{bit} [DPP24a]	$\mathbb{G}^{4\ell+13} \times \hat{\mathbb{G}}^{4\ell+15}$	$\mathbb{G}^{4\ell+33} \times \hat{\mathbb{G}}^{4\ell+26}$

For the proof, we refer to [DPP24a]. It follows the same structure as the security proofs of $\Pi_{PC}^{\mathbb{G}}$ with the additional steps discussed in the intuition of the construction.

4.5. Performance and comparisons

To compare our constructions, we summarized in a table the size of the ciphertexts of each construction and the number of exponentiations needed to compute one ciphertext. For our public coin constructions, we only included the tight constructions in the ROM, since the size of the ciphertexts are identical and the number of exponentiations is almost the same (the main difference is the generation of the $\hat{E}$ or C commit which cost one more exponentiation in the variant in the standard model). We also give these numbers for the previous state-of-the-art single-pass receipt-free protocol, BeleniosRF.

Since the main application of TREncs is the design of voting protocols, we assume that the critical computation steps are done by the voter client-side, as they could typically use a smartphone, a browser or a low-end voting device. This is why we focus on comparing the metrics related to the computation of a ciphertext. On the other hand, the operations on the rerandomization server can be done on the fly as ballots come in and the verification operations can be distributed among many parties.

Also, we implemented in Rust [Tre] these four constructions and benchmarked the time to encryptn one ciphertext of various lengths. We conducted our experiments on a laptop with a 1.8Ghz Intel i7 processor and 16GB of RAM running Ubuntu 20.04 LTS. We used the `bls12_381` Rust crate [Bls] with the pairing-friendly elliptic curve of the same name and the criterion library to perform the benchmarks. For our comparison with BeleniosRF, we reimplemented in Rust the referenced Javascript

Table 4.2.: Time to encrypt the vote of a ballot containing $\ell = 1, 2, 4, 8, 64$ bits or group element. The times reported are in milliseconds.

Group-encoded constructions					
Π_{PC} [DPP22b]	27.3				
$\Pi^{\mathbb{G}}_{SC}$[DPP24a]	32.6				
Bit-by-bit constructions					
Number of bits ℓ	1	2	4	8	64
BeleniosRF [Cha+16]	58.5	82.2	124.7	191.0	1211.7
Π^{bit}_{SC}[DPP24a]	67.3	78.4	95.1	132.6	628.7

implementation [Brf].

We see in Table 4.2 that the performance of the Π_{SC} and $\Pi^{\mathbb{G}}_{PC}$ are similar, with an advantage for the construction with a trusted setup Π_{SC} of around 15%. However, the slightly more expensive construction offers a public coin CRS (which is very desirable in a voting setting) and a tight TCCA security. Also, we see in the table that even for a 64-choices race, the encrypting time of the bit-by-bit TREnc is well beneath one second and outperforms BeleniosRF.

4.6. Concluding remarks

This chapter proposes four constructions of the TREnc primitive introduced in the last chapter. Overall, the last two constructions are the most promising, for encrypting respectively a group element and a sequence of bits, and this is because they use a public CRS for a relatively cheap cost.

All of our constructions are built to be strongly randomizable, which restricts the design space. Forcing the output of the rerandomizing algorithm to have the same distribution as fresh encryptions is a restriction that is not necessary to build a secure protocol, but more of a convenience to get wCCA for free (see Section 3.3). It is thus an open direction to explore the design of TREncs without this restriction on the rerandomizing algorithm.

Finally, our constructions are still fairly expensive compared to more practical and more simple encryptions schemes used in real voting protocols such as ElectionGuard [Ele24]. We need 1) a bilinear group 2) asymptotically more than four times as many exponentiations (in a more expensive group) than the solutions that we will explore in Chapter 6 to verifiably encrypt bits. Especially, this is the proof of validity that

is still expensive, as we need the proof to be rerandomizable. Hence, cheaper randomizable proofs that an ElGamal ciphertext contains indeed a sequence of bits would be of great interest.

5. Can we cast a ballot as intended and be Receipt-Free?

In the first two chapters, we explored a new solution to build receipt-free single-pass voting schemes. However this is not the only desirable property that a secure voting system should satisfy. Cast-as-intended verifiability, in particular, has been overlooked until now in the context of receipt-free voting protocols in which the voting device is not trusted to run the voting algorithm specified by the protocol.

We start by investigating the compatibility of single-pass voting protocols with these properties. And actually, we demonstrate that it is impossible for receipt-freeness and cast-as-intended to coexist without a trusted registration if the voting protocol has a non-interactive ballot casting process. Intuitively, if there is one way for voters to deceive a receipt-freeness adversary, a malicious device trying to modify the content of some voter's ballot can follow the same deceiving strategy to now fool the voter.

Motivated by this result, we then adapt the definitions of Section 2.3 to protocols with an interactive casting process. Indeed, the already existing game-based definition only makes sense if the voting process is non-interactive, which greatly simplifies the receipt-freeness definition. In a single-pass protocol, the receipt-freeness adversary's instruction can only take the form of the ballot to send to the casting server. Now if there are interactions, the instructions can adapt to each new interaction with the casting server. Thus, we alter Definition 16 by allowing the adversary to send instructions in the form of a Turing machine that can adapt to every interaction.

Finally, we build three protocols exhibiting receipt-freeness and cast-as-intended. The first two protocols rely on an interactive voting process: the first protocol makes strong computational requirements on the voter but needs a constant number of rounds, while the second protocol is more user-friendly and requires a number of rounds that is linear in the security parameter. Then, the third protocol is non-interactive and is built around a TREnc, but relies on the availability of a trusted registration authority.

Structure of the chapter We start the chapter with a brief literature review of impossibility results in voting protocols and cast-as-intended verifiability. Then in Section 5.2, we adapt the definitions of voting protocols to a ballot casting procedure which is now interactive, and in Section 5.3, we show that this is actually impossible to build a single-pass voting protocol that achieves both the receipt-freeness and the cast-as-intended verifiability properties. Section 5.4 proposes the three new protocols that achieve those two properties, with the corresponding proofs presented in Section 5.5. Finally, Section 5.6 concludes the chapter with a comparison of these protocols.

5.1. Related works

First impossibility results for voting were proposed by Chevallier-Mames et al. [Che+10]. In particular, they demonstrate that universal verifiability and receipt-freeness cannot be jointly obtained unless private channels are available between the voters and the voting authorities. Our result extends that by further demonstrating incompatibilities between cast-as-intended verifiability and non-interactive voting, in the absence of trusted authorities. More recently, Cortier and Lallemand showed that private elections cannot be obtained unless there is individual verifiability [CL18]. Receipt-freeness, on which we focus here, is itself implied by privacy (if my ballot is not private, it is itself a receipt), and our work sheds some light on the structure of the ballot submission process that is needed to achieve a strong form of privacy. But we see the main contribution of our work as the focus on cast-as-intended verifiability, while Cortier and Lallemand essentially express individual verifiability as a recorded-as-cast property, without separation between the voter and the voting device.

In terms of protocols, we believe that our protocols are the first to offer cast-as-intended verifiability and receipt-freeness without relying on further assumptions. Especially, we do not trust the voting device to honestly execute the voting algorithm specified by the protocol, as we allow the voter to have full control of the device. This trust assumption greatly helps the design of receipt-free protocol, since the official voting algorithm would typically not return the randomness used in the encryption of the vote. Helios, for example can be considered receipt-free in this regard.

In this chapter, we propose protocols for which the cast-as-intended verifiability can be obtained with overwhelming probability, contrary to previous protocols that rely on a Benaloh challenge, like Helios [Adi+09].

To this purpose, we borrow and extend techniques from the recent Themis protocol by Bougon et al [Bou+22], where ballots contain two ciphertexts that contain secret shares of the vote, and one of the ciphertexts is opened.

5.2. New definitions for interactive voting protocols

By introducing interactivity in the voting procedure and mechanisms to achieve cast-as-intended verification, we need to adapt our definitions of Section 2.3. Especially, our previous receipt-freeness definition was only compatible with a single-pass ballot casting and we propose a brand new definition to model this property when there are several interactions in the voting process. Moreover, we extend the verifiability definition (Definition 18) to deal with malicious voting devices in a new cast-as-intended definition in the computational model.

5.2.1. Interactive Voting system

First, we refine the model for the voter as we consider that the voter can use a voting device that can be malicious in the verifiability experiment. Hence, we consider the human voter H and the voting device VD as two independent entities. We also consider an extra entity, the auditing device AD which is used to run the verification algorithms.

We extend and write now Vote as an interactive protocol run by a voter id and the casting server CS: $\Gamma, \mathsf{b} \leftarrow [\mathsf{Vote}_{\mathsf{V}}(1^\lambda, \mathsf{usk}, \mathsf{v}) \leftrightarrow \mathsf{Vote}_{\mathsf{CS}}(1^\lambda, \mathsf{id})]$, where the voter has as input a choice $\mathsf{v} \in \mathbb{V}$ and credentials usk, and receives as output Γ as some verification materials. Indeed, since we consider now more involved verification mechanisms against the voting device, we open the possibility for the voter to acquire additional verification data that was not capture by the public $\mathsf{TraceBallot}$ algorithm. To simplify the notations, we write an execution of the voting protocol as $\Gamma, \mathsf{b} \leftarrow \mathsf{Vote}(\mathsf{id}, \mathsf{v})$. A voting scheme with multiple interactions is not considered single-pass anymore, but this formalism can still capture our previous interactionless setting used so far. Indeed, if there is a single interaction between $\mathsf{Vote}_{\mathsf{V}}$ and $\mathsf{Vote}_{\mathsf{CS}}$, then $\mathsf{Vote}_{\mathsf{CS}}$ can be seen as the $\mathsf{ProcessBallot}$ algorithm and the protocol is single-pass.

Without loss of generality, we assume that the output of CS is a value b that we call a ballot and that is appended to the bulletin board at the end of the protocol. Hence, this interactive Vote algorithms replace the previous Vote algorithm, but also the $\mathsf{ProcessBallot}$ aglorithm, and the Append algorithm is implicitly defined.

5. Can we cast a ballot as intended and be Receipt-Free?

Even if our model prevents CS from updating BB during the execution of the protocol, we can imagine that it sends future updates of BB to the voter during the protocol, which can appear in the verification data. If CS does not publish these updates, it will be caught by the voter in the $\mathsf{VerifyVote}$ procedure by comparing the verification data with the bulletin board.

Also, we model each round of the $\mathsf{Vote}_{\mathsf{V}}$ algorithm as a human-machine interaction to be able to model the corruption of a voting device. Hence, $\mathsf{Vote}_{\mathsf{V}}(1^\lambda, \mathsf{usk}, \mathsf{v})$ is an interactive protocol $[\mathsf{Vote}_{\mathsf{H}}(1^\lambda, \mathsf{v}, \mathsf{usk}) \leftrightarrow \mathsf{Vote}_{\mathsf{VD}}(1^\lambda)]$ between the actual human voter and the voting device VD. The input of the voter is the voting intention v but also the credentials usk, which we consider the voter has acquired in some trusted way before. Then, the messages sent by $\mathsf{Vote}_{\mathsf{H}}$ are real human inputs (eg. a choice on a screen) and the messages sent by $\mathsf{Vote}_{\mathsf{VD}}$ are data that the human can access (eg. information on a screen). Finally, the output of $\mathsf{Vote}_{\mathsf{VD}}$ is the message sent to CS in one round of $\mathsf{Vote}_{\mathsf{V}}$ and the output of $\mathsf{Vote}_{\mathsf{H}}$ in the last round of $\mathsf{Vote}_{\mathsf{V}}$ forms the verification data Γ. Both the voter and the voting device keep their states across iterations of $\mathsf{Vote}_{\mathsf{V}}$. Again, if there is only one message from the voting device to the casting server, we consider the protocol to be single-pass even if there are multiple interactions between the human voter and the voting device.

5.2.2. Receipt-freeness with interactive voting

The receipt-freeness definition (Definition 16) introduced in Section 2.3 is no longer suitable if the vote protocol is interactive. Indeed, for an interactionless procedure, the only sensible strategy for an adversary is to instruct a voter to send a given ballot as the only message to the ballot box manager. However, if there is at least one interaction, an attacker might instruct the voter to craft a message as a function of previous interactions.

We thus propose an alternative definition, Definition 29, that supports an interactive voting processes. The main difference is that the adversary's instruction now takes the form of an interactive Turing machine $\mathcal{I}$ that can give instructions to the voter at each step of the voting process execution. These instructions also output a final "receipt" returned to the adversary.

The Turing machine, which is possibly obfuscated, is expected to produce the messages sent to the server instead of the voter and to produce an arbitrary string (the receipt) at the end of the voting protocol, as a replacement of the $\mathsf{Vote}_{\mathsf{V}}$ algorithm. Crucially, we still assume that the adversary does not control the voting device , as a malicious voting

device would trivially leak the voters' choice.

Furthemore, we equip the voter with an extra algorithm $\mathcal{D}$, called the deceiving algorithm which outputs a strategy to escape from an adversary's instructions. More precisely, this algorithm takes as input the real voting intent of a voter and the instructions of a receipt-freeness adversary and returns for each interaction the message to be sent to CS. This deceiving algorithm also outputs the verification data to still enable the voter to verify that the vote is correctly recorded together with a fake receipt to fool the adversary that the voter followed its instructions.

Most importantly, the voter has the possibility to run $\mathcal{I}$ as a subroutine and to rewind it. Indeed, if $\mathcal{I}$ can send messages to CS directly, there is no hope of security. This is thus similar to organizational measures to ensure that the voter can interact with the instructions in this way.

Definition 29 (Receipt-freeness with interactive voting)**.** *A voting system $\mathcal{V}$ has receipt-freeness if there exists two PPT algorithms* $\mathsf{SimSetup}$*,* $\mathsf{SimProve}$ *and an interactive PPT deceiving algorithm $\mathcal{D}$ such that the two following conditions are met:*

- *no efficient adversary can distinguish between games* $\mathsf{Exp}^{\mathrm{rf},0}_{\mathcal{A},\mathcal{V},\mathcal{D}}(\lambda)$ *and* $\mathsf{Exp}^{\mathrm{rf},1}_{\mathcal{A},\mathcal{V},\mathcal{D}}(\lambda)$ *defined in Figure 5.1. That is, for any PPT algorithm $\mathcal{A}$, the following advantage* $\mathsf{Adv}^{\mathrm{rf}}_{\mathcal{A},\mathcal{V},\mathcal{D}}(\lambda)$ *is negligible in λ:*

$$\left|\Pr\left(\mathsf{Exp}^{\mathrm{rf},0}_{\mathcal{A},\mathcal{V},\mathcal{D}}(\lambda)=1\right)-\Pr\left(\mathsf{Exp}^{\mathrm{rf},1}_{\mathcal{A},\mathcal{V},\mathcal{D}}(\lambda)=1\right)\right|;$$

- *no efficient adversary can win the vote deviation correctness experiment* $\mathsf{Exp}^{\mathrm{dev}}_{\mathcal{A},\mathcal{V},\mathcal{D}}(\lambda)$ *defined in Figure 5.2. That is, for any PPT algorithm $\mathcal{A}$, the following advantage is negligible in λ:*

$$\mathsf{Adv}^{\mathrm{dev}}_{\mathcal{A},\mathcal{V},\mathcal{D}}(\lambda)=\Pr\left(\mathsf{Exp}^{\mathrm{dev}}_{\mathcal{A},\mathcal{V},\mathcal{D}}(\lambda)=1\right).$$

This definition is very similar to Definition 16, except for the main differences highlighted above. The adversary still has to distinguish between two worlds, where either the voters run the deceiving algorithms ($\beta = 0$) or the voters follow its instructions ($\beta = 1$). Compared to the previous definition, we added the oracles related to the registration as we will consider protocols with registration in this chapter, and we add an $\mathcal{O}\mathsf{cast}(\mathsf{b})$ oracle to let the adversary append malicious ballots on both boards, as this classic oracle is not superseded by $\mathcal{O}\mathsf{receiptLR}$ anymore.

There is an additional part in the definition related to Figure 5.2, which alleviates a shortcoming of the [Cha+16] definition. Indeed, we need to

$\mathsf{Exp}^{\mathrm{rf},\beta}_{\mathcal{A},\mathcal{V},\mathcal{D}}(\lambda)$

$\mathsf{pk} \leftarrow \mathcal{O}\mathsf{init}(1^\lambda)$;
$\mathsf{st} \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{register},\mathcal{O}\mathsf{corrupt},\mathcal{O}\mathsf{cast},\mathcal{O}\mathsf{board},\mathcal{O}\mathsf{voteLR},\mathcal{O}\mathsf{receiptLR}}(\mathsf{pk})$;
$(\mathsf{r},\Pi) \leftarrow \mathcal{O}\mathsf{tally}()$;
return $\mathcal{A}(\mathsf{st},\mathsf{r},\Pi)$;

$\mathcal{O}\mathsf{init}(1^\lambda)$

if $\beta = 0$
then $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
else $(\mathsf{pk},\mathsf{sk},\tau) \leftarrow \mathsf{SimSetup}(1^\lambda)$
$\mathsf{BB}_0 \leftarrow \varnothing$; $\mathsf{BB}_1 \leftarrow \varnothing$
return pk

$\mathcal{O}\mathsf{cast}(\mathsf{id},\mathcal{I})$

$\mathsf{rcpt},\mathsf{b} \leftarrow [\mathcal{I}() \leftrightarrow \mathsf{Vote}_{\mathsf{CS}}(\mathsf{id})]$
if $\mathsf{Valid}(\mathsf{BB}_\beta,\mathsf{b}) = 0$ **then return** 0
$\mathsf{BB}_0 \leftarrow \mathsf{BB}_0||\mathsf{b}$; $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1||\mathsf{b}$
return rcpt

$\mathcal{O}\mathsf{corrupt}(\mathsf{id})$

if $\mathsf{id} \in \mathsf{U}$:
then $\mathsf{CU} \leftarrow \mathcal{CU} \cup \{\mathsf{id}\}$
and **return** usk

$\mathcal{O}\mathsf{tally}()$

$(\mathsf{r},\Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}_0,\mathsf{sk})$
if $\beta = 1$ **then** $\Pi \leftarrow \mathsf{SimProve}(\mathsf{BB}_1,\mathsf{r},\tau)$
return (r,Π)

$\mathcal{O}\mathsf{receiptLR}(\mathsf{id},\mathcal{I},\mathsf{v})$

if $\mathsf{v} \notin \mathbb{V}$ **then return** 0
$(\mathsf{rcpt}_0,\Gamma),\mathsf{b}_0 \leftarrow [\mathcal{D}(\mathsf{id},\mathcal{I},\mathsf{v}) \leftrightarrow \mathsf{Vote}_{\mathsf{CS}}(\mathsf{id})]$
$\mathsf{rcpt}_1,\mathsf{b}_1 \leftarrow [\mathcal{I}() \leftrightarrow \mathsf{Vote}_{\mathsf{CS}}(\mathsf{id})]$
if $\mathsf{Valid}(\mathsf{BB}_1,\mathsf{b}_1) = 0$ **then return** 0
$\mathsf{BB}_0 \leftarrow \mathsf{BB}_0||\mathsf{b}_0$; $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1||\mathsf{b}_1$
return rcpt_β

$\mathcal{O}\mathsf{register}(\mathsf{id})$

if id has not been queried yet:
then $\mathsf{upk},\mathsf{usk} \leftarrow \mathsf{Register}(\mathsf{id})$
$\mathsf{U} \leftarrow \mathcal{U} \cup \{\mathsf{id}\}$
return upk

$\mathcal{O}\mathsf{board}()$

return BB_β

Figure 5.1.: The receipt-free experiment $\mathsf{Exp}^{\mathrm{rf},\beta}_{\mathcal{A},\mathcal{V},\mathcal{D}}(\lambda)$ for an interactive voting protocol. In the $\mathcal{O}\mathsf{receiptLR}$ oracle, the deceiving algorithm takes as input the instructions of the adversary, the true voting intention of the voter and potential information attached to the voter, such as an id and, if there is a registration, credentials. Also, the bulletin boards are not updated during any execution of the voting protocols. Instead, the ballots are appended to each of the bulletin boards if they are both valid.

$\mathsf{Exp}^{\mathrm{dev}}_{\mathcal{A},\mathcal{V},\mathcal{D}}(\lambda)$

$\mathsf{pk} \leftarrow \mathcal{O}\mathsf{init}(1^\lambda); \mathcal{A}^{\mathcal{O}\mathsf{register},\mathcal{O}\mathsf{voteLR}}(\mathsf{pk});$
$(\mathsf{r}_0, \Pi_0) \leftarrow \mathsf{Tally}(\mathsf{BB}_0, \mathsf{sk});$
$(\mathsf{r}_1, \Pi_1) \leftarrow \mathsf{Tally}(\mathsf{BB}_1, \mathsf{sk});$
if $\mathsf{r}_0 \neq \mathsf{r}_1$ **return** 1 **else return** 0;

$\mathcal{O}\mathsf{init}(1^\lambda)$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
$\mathsf{BB}_0 \leftarrow \varnothing; \mathsf{BB}_1 \leftarrow \varnothing$
return pk

$\mathcal{O}\mathsf{register}(\mathsf{id})$

if id has not been queried yet:
then $\mathsf{upk}, \mathsf{usk} \leftarrow \mathsf{Register}(\mathsf{id})$
$\mathsf{U} \leftarrow \mathsf{U} \cup \{\mathsf{id}\}$
return $\mathsf{upk}, \mathsf{usk}$

$\mathcal{O}\mathsf{voteLR}(\mathsf{id}, \mathcal{I}, \mathsf{v})$

if $\mathsf{v} \notin \mathbb{V}$ **or** $(\mathsf{id}, \star, \star) \notin \mathsf{U}$ **then return** 0
$\Gamma_0, \mathsf{b}_0 \leftarrow \mathsf{Vote}(\mathsf{id}, \mathsf{v})$
$(\mathsf{rcpt}_1, \Gamma_1), \mathsf{b}_1 \leftarrow [\mathcal{D}(\mathsf{id}, \mathcal{I}, \mathsf{v}) \leftrightarrow \mathsf{Vote}_{\mathsf{CS}}(\mathsf{id})]$
$\mathsf{rcpt}', \mathsf{b}' \leftarrow [\mathcal{I}() \leftrightarrow \mathsf{Vote}_{\mathsf{CS}}(\mathsf{id})]$
if $\mathsf{Valid}(\mathsf{BB}_1, b') = 0$ **return** 0
$\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 || \mathsf{b}_0; \mathsf{BB}_1 \leftarrow \mathsf{BB}_1 || \mathsf{b}_1$
return $\mathsf{rcpt}_0, \mathsf{rcpt}_1, \mathsf{b}_0, \mathsf{b}_1$

Figure 5.2.: Vote deviation correctness experiment.

ensure that for any adversary's instruction, a voter is always able to cast their true voting intention through the deceiving algorithm. Hence, this experiment ensures that the ballots produced by the deceiving algorithm for a vote for v are counted in the same way as ballots produced by an honest $\mathsf{Vote}(\mathsf{v})$ algorithm.

This additional requirement is crucial. For example, a $\mathsf{Vote}_{\mathsf{V}}(\mathsf{id}, \mathsf{v})$ algorithm which would send a random tag t and v to CS and a $\mathsf{Vote}_{\mathsf{CS}}(\mathsf{id})$ algorithm which would append the hash of the tag and an NM-CPA encryption of v to the bulletin board can be similarly proven to be secure under this definition. However, if the Valid algorithm returns false if the first value bit of the vote is different from the first bit of the tag, then the candidates for which the voter can vote is limited and the voter does not fully escape receipt-freeness. This example is admittedly convoluted and we are not aware of any reasonable voting scheme with such a shortcoming, but this is still not captured by the definition.

Finally, we stress that if $\mathcal{I}$ outputs an invalid ballot, the deviation algorithm does not need to produce a valid ballot for the given message. This is simply because our receipt-free model does not consider forced-absention attacks, as discussed in Section 2.3.5.

5. Can we cast a ballot as intended and be Receipt-Free?

5.2.3. Verifiability against a malicious voting device

Regarding the verifiability, the interactivity of the voting model does not really impact the approach used so far. However, we now desire to offer protection against a corrupted voting device.

We are aware of only one formalization of this property as a game-based definition in the computational model of cryptography, which is proposed in [RRS21]. However, their definition does not model voting credentials and does not extend easily to a setting in which more parties are corrupted.

We thus extend Definition 18 to take into account the potential corruption of a voting device. In our definition, we assume first that all the official parties are honest and focus on the voting device, which will require more modifications. We will remove this assumption later.

Definition 30 (Verifiability against a malicious VD)**.** *A voting system $\mathcal{V}$ is verifiable against a malicous voting device if for all PPT $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the following is negligible in λ:*

$$\mathsf{Adv}^{\mathrm{vd-verif}}_{\mathcal{A},\mathcal{V}}(\lambda) = \Pr\left(\mathsf{Exp}^{\mathrm{vd-verif}}_{\mathcal{A},\mathcal{V}}(\lambda) = 1\right) \leq \mu(\lambda)$$

where $\mathsf{Exp}^{\mathrm{vd-verif}}_{\mathcal{A},\mathcal{V}}(\lambda)$ is defined in Figure 5.3.

We stress that in this scenario, the malicious voting device does not leak the voter's secret credentials to the adversary. We assume that the voter receives these credentials from another trusted device (or from a postal channel). However, some protocols might require the voter to give their credentials to the voting device. We thus propose a variant $\mathsf{Exp}^{\mathsf{cred},\mathsf{vd}-\mathrm{verif}}_{\mathcal{A},\mathcal{V}}(\lambda)$ in which the $\mathcal{O}\mathsf{register}$ oracle also returns usk to the adversary.

5.2.4. Verifiability against a malicious voting device and the casting server

We extend the previous definition to more corrupted parties. Especially, it is desirable to have verifiability even if the casting sever is corrupted and if the adversary has the secret key of the election. In such a low-trust setting, we only assume that the voter has trusted access to the bulletin board, via the auditing device for example and that the key of the election results from an honest generation of the Setup algorithm.

Definition 31 (Verifiability against a malicious VD and CS)**.** *A voting system $\mathcal{V}$ is verifiable against a malicious voting device and the casting*

$\mathsf{Exp}^{\mathrm{vd-verif}}_{\mathcal{A},\mathcal{V}}(\lambda)$

$\mathsf{pk}, \mathsf{sk} \leftarrow \mathsf{Setup}(1^\lambda)$;
$\mathsf{U}, \mathsf{CU} \leftarrow \{\}$;
$\mathsf{BB} \leftarrow \{\}$;
$\mathsf{st} \leftarrow \mathcal{A}_1^{\mathcal{O}\mathsf{register}}(\mathsf{pk})$;
$\mathsf{Voted} \leftarrow \{\}$;
$\mathcal{A}_2^{\mathcal{O}\mathsf{vote}}(\mathsf{st})$;
if $\forall \mathsf{id}, (\mathsf{id}, \star, \Gamma) \in \mathsf{Voted} \implies \mathsf{VerifyVote}(\mathsf{BB}, \Gamma) = 1$
then
  $r, \Pi \leftarrow \mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$;
  if $r \neq \perp$ and $\forall V_C$ s.t. $r \neq \rho(V_H \cup V_C)$ **then return** 1;
return 0;
  where $V_H = \{(\mathsf{v_{id}})\}$ for each $(\mathsf{id}, \star, \mathsf{v_{id}}) \in \mathsf{Voted}$

$\mathcal{O}\mathsf{register}(\mathsf{id})$

$\mathsf{upk_{id}}, \mathsf{usk_{id}} \leftarrow \mathcal{A}(\mathsf{id})$
$\mathsf{U} \leftarrow \mathsf{U} \cup \{(\mathsf{id}, \mathsf{upk_{id}}, \mathsf{usk_{id}})\}$
return $\mathsf{upk_{id}}, \mathsf{usk_{id}}$

$\mathcal{O}\mathsf{vote}(\mathsf{id}, \mathsf{v})$

if $(\mathsf{id}, \star, \star) \in \mathsf{U} \setminus \mathsf{CU}$
then
  $\Gamma, \mathsf{b} \leftarrow [(\mathsf{Vote_H}(\mathsf{v}) \leftrightarrow \mathcal{A}) \leftrightarrow \mathsf{Vote_{CS}}(\mathsf{id})]$
  $\mathsf{Voted} \leftarrow \mathsf{Voted}' \cup \{(\mathsf{id}, \mathsf{v}, \Gamma\}$;
  $\mathsf{BB} \leftarrow \mathsf{BB}||\mathsf{b}$;
  return b
  where Voted' is obtained
  by removing all instances of $(\mathsf{id}, \star, \star)$

$\mathcal{O}\mathsf{cast}(\mathsf{id})$

if $(\mathsf{id}, \star, \star) \in \mathsf{CU}$
then
  $\mathsf{rcpt}, \mathsf{b} \leftarrow [\mathcal{A} \leftrightarrow \mathsf{Vote_{CS}}(\mathsf{id})]$
  $\mathsf{BB} \leftarrow \mathsf{BB}||\mathsf{b}$;
  return b
  where Voted' is obtained
  by removing all instances of $(\mathsf{id}, \star, \star)$

Figure 5.3.: The verifiability against a malicious voting device experiment and its oracles. In the $\mathcal{O}\mathsf{vote}$ oracle, the adversary can freely choose the messages sent to the voter in the interactive voting protocol.

$\mathsf{Exp}^{\text{all-verif}}_{\mathcal{A},\mathcal{V}}(\lambda)$

$\mathsf{pk}, \mathsf{sk} \leftarrow \mathsf{Setup}(1^\lambda)$;
$\mathsf{U}, \mathsf{CU} \leftarrow \{\}$;
$\mathsf{st} \leftarrow \mathcal{A}_1^{\mathcal{O}\mathsf{register}}(\mathsf{pk}, \mathsf{sk})$;
$\mathsf{Voted} \leftarrow \{\}$;
$\mathsf{BB} \leftarrow \mathcal{A}_2^{\mathcal{O}\mathsf{vote}}(\mathsf{st})$;
if $\forall \mathsf{id}, (\mathsf{id}, \star, \Gamma) \in \mathsf{Voted} \implies \mathsf{VerifyVote}(\mathsf{BB}, \Gamma) = 1$
then
  $r, \Pi \leftarrow \mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$;
  if $r \neq \perp$ and $\forall V_C$ s.t. $r \neq \rho(V_H \cup V_C)$ **then return** 1;
return 0;
  where $V_H = \{(\mathsf{v_{id}})\}$ for each $(\mathsf{id}, \star, \mathsf{v_{id}}) \in \mathsf{Voted}$

$\mathcal{O}\mathsf{register}(\mathsf{id})$

$\mathsf{upk_{id}}, \mathsf{usk_{id}} \leftarrow \mathcal{A}(\mathsf{id})$
$\mathsf{U} \leftarrow \mathsf{U} \cup \{(\mathsf{id}, \mathsf{upk_{id}}, \mathsf{usk_{id}})\}$
return $\mathsf{upk_{id}}, \mathsf{usk_{id}}$

$\mathcal{O}\mathsf{vote}(\mathsf{id}, \mathsf{v})$

if $(\mathsf{id}, \star, \star) \in \mathsf{U} \setminus \mathsf{CU}$
then
  $\Gamma, \mathsf{b} \leftarrow [(\mathsf{Vote_H}(\mathsf{v}) \leftrightarrow \mathcal{A}) \leftrightarrow \mathsf{Vote_{CS}}(\mathsf{id})]$
  $\mathsf{Voted} \leftarrow \mathsf{Voted}' \cup \{(\mathsf{id}, \mathsf{v}, \Gamma\}$;
  $\mathsf{BB} \leftarrow \mathsf{BB}||\mathsf{b}$;
  return b
  where Voted' is obtained
  by removing all instances of $(\mathsf{id}, \star, \star)$

$\mathcal{O}\mathsf{cast}(\mathsf{id})$

if $(\mathsf{id}, \star, \star) \in \mathsf{CU}$
then
  $\mathsf{rcpt}, \mathsf{b} \leftarrow [\mathcal{A} \leftrightarrow \mathsf{Vote_{CS}}(\mathsf{id})]$
  $\mathsf{BB} \leftarrow \mathsf{BB}||\mathsf{b}$;
  return b
  where Voted' is obtained
  by removing all instances of $(\mathsf{id}, \star, \star)$

Figure 5.4.: The verifiability experiment against a malicious voting device and the casting server. This definition combines the features of Definition 18 and Definition 30.

server if for all PPT $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, the following is negligible in* λ*:*

$$\mathsf{Adv}^{\text{all-verif}}_{\mathcal{A},\mathcal{V}}(\lambda) = \Pr\left(\mathsf{Exp}^{\text{all-verif}}_{\mathcal{A},\mathcal{V}}(\lambda) = 1\right) \leq \mu(\lambda)$$

where $\mathsf{Exp}^{\text{all-verif}}_{\mathcal{A},\mathcal{V}}(\lambda)$ *is defined in Figure 5.4.*

5.2.5. Vote deviation verifiability

These definitions do not give any verifiability guarantee in case the voter is following the deceiving strategy instead of the normal voting procedure. In practice, we would want to preserve verifiability when deviating from a receipt-free adversary. As we did with the Vote algorithm, we can split $\mathcal{D}$ as an interactive protocol $[\mathcal{D}_\mathsf{H} \leftrightarrow \mathcal{D}_\mathsf{VD}]$ between the human voter and the voting device and adapt the verifiability definitions accordingly. More

precisely, we can add an $\mathcal{O}\mathsf{dev}(\mathsf{id}, \mathcal{I}, \mathsf{v})$ oracle to the original definition that executes $\mathcal{D}_{\mathsf{H}}(\mathsf{id}, \mathcal{I}, \mathsf{v})$ instead of $\mathsf{Vote}_{\mathsf{H}}(\mathsf{id}, \mathsf{v})$.

Given a verifiable protocol, one sufficient condition to obtain verifiability when following the deceiving strategy would be to have a $\mathcal{D}_{\mathsf{H}}$ algorithm indistinguishable from the $\mathsf{Vote}_{\mathsf{H}}$ algorithm from the view of the voting device, for any $\mathcal{I}$:

$$\mathcal{D}_{\mathsf{H}}(\mathsf{id}, \mathcal{I}, \mathsf{v}) \approx \mathsf{Vote}_{\mathsf{H}}(\mathsf{id}, \mathsf{v}) \quad \forall \mathsf{id}, \mathcal{I}, \mathsf{v}$$

Indeed, we can easily reduce to the original verifiability definition by a sequence of transitions that replaces executions of $\mathcal{D}_{\mathsf{H}}$ by $\mathsf{Vote}_{\mathsf{H}}$.

5.3. Impossibility results

Our definitions make an important departure from most of the traditional ones, that focus on a non-interactive voting protocol, also often called single-pass voting [Ber+11; Ber+15; Cor+16]. Here, we consider an interactive voting process, during which the voter and the casting server may have multiple rounds of interaction.

Since many voting protocols (e.g., Helios, Belenios, etc.) support a single-pass voting process, we may question the motivation for this additional complexity. The reason appears in the following impossibility result, which demonstrates that it is impossible to design a registration-free non-interactive voting protocol that has receipt-freeness and verifiability against a corrupted voting device. Helios [Adi+09], for instance, is registration-free, has a non-interactive voting protocol and is verifiable against a corrupted voting device, but is not receipt-free. On the other hand, BeleniosRF [Cha+16] brings receipt-freeness but requires a trusted voter registration process and does not support verifiability against a corrupted voting device. A consequence of our impossibility result is that it is not possible to make Helios receipt-free without adding either a trusted registration process or making the voting protocol interactive. In the next sections, we will show that both these options are possible.

The impossibility result proceeds by building three adversaries: one for the receipt-freeness game (Figure 5.1), one for the verifiability against a malicous voting device (Figure 5.3) and one for the vote deviation correctness (Figure 5.2). We then show that the sum of the advantages of these adversaries is at least 1, and thus that at least one them wins their games. Intuitively, we build the verifiability adversary such that it treats the human voter as a coercer and uses the deceiving algorithm to submit a ballot. Hence if the protocol is receipt-free, the human voter cannot distinguish between an honest voting device and a cheating device.

In the proof, we also rely on the fact that the family of result functions ρ that are not constant.[22] That is, for any integer τ and any family V of τ voters, there exists two families of τ votes S_1 and S_2 such that $\rho(S_1) \neq \rho(S_2)$.

Theorem 18. *There is no single-pass registration-free voting system that has receipt-freeness and verifiability against a corrupted voting device.*

More precisely, for any single-pass registration-free voting system $\mathcal{V}$ and any deviating strategy $\mathcal{D}$ there is an adversary $\mathcal{A}_V$ for the verifiability experiment $\mathsf{Exp}^{\mathsf{vd-verif}}_{\mathcal{A}_V,\mathcal{V}}(\lambda)$ of Definition 30, an adversary $\mathcal{A}_{RF}$ for the receipt-freeness experiment $\mathsf{Exp}^{\mathrm{rf}}_{\mathcal{A}_{RF},\mathcal{V}}(\lambda)$ and an adversary $\mathcal{A}_{\mathcal{D}}$ for the vote deviation correctness experiment $\mathsf{Exp}^{\mathrm{dev}}_{\mathcal{A}_{\mathcal{D}},\mathcal{V}}(\lambda)$, both of Definition 29, such that:

$$\mathsf{Adv}^{\mathsf{vd-verif}}_{\mathcal{A}_V,\mathcal{V}}(\lambda) + \mathsf{Adv}^{\mathrm{rf}}_{\mathcal{A}_{RF},\mathcal{V}}(\lambda) + \mathsf{Adv}^{\mathrm{dev}}_{\mathcal{A}_{\mathcal{D}},\mathcal{V}}(\lambda) \geq 1$$

Proof. Let us consider a voting system $\mathcal{V}$ with n voters and a deceiving strategy $\mathcal{D}$. We will build three adversaries $\mathcal{A}_V$, $\mathcal{A}_{RF}$ and $\mathcal{A}_{\mathcal{D}}$ such that at least one of them wins. From the assumption on the result function, there are two lists of votes S_0 and S_1 such that $\rho(\{\mathsf{v}_{0,i}\}) \neq \rho(\{\mathsf{v}_{1,i}\})$, where $\mathsf{v}_{0,i}$ and $\mathsf{v}_{1,i}$ are respectively the i-th element of S_0 and S_1. Since the voting system is registration free, the private credential of every voter is equal to $\perp$.

Also, we denote by $\mathcal{I}^{\mathsf{b}}$ an instruction that outputs b as the message to send to CS and $\perp$ the receipt. As the voting system is non-interactive, the instruction stops there. We also define $\mathcal{X}$ as the the distribution of $(\mathsf{BB}, \mathsf{RCPT})$ produced as follows, and parametrized by the security parameter:

$$\begin{aligned}
&\mathsf{msg}, \Gamma \leftarrow \mathsf{Vote}_{\mathsf{V}}(\mathsf{id}_i, \mathsf{v}_{1,i})\\
&\mathsf{rcpt}', \mathsf{b} \leftarrow\!\!\$\, [\mathcal{D}(\mathsf{id}_i, \mathcal{I}^{\mathsf{msg}}, \mathsf{v}_{0,i}) \leftrightarrow \mathsf{Vote}_{\mathsf{CS}}(\mathsf{id}_i)]\\
&\mathsf{BB} \leftarrow \mathsf{BB}||\mathsf{b}\\
&\mathsf{RCPT} \leftarrow \mathsf{RCPT}||\Gamma
\end{aligned}$$

for i in range(n).

We then proceed to build the adversaries for each experiment. All our three adversaries create a distribution in their respective experiments that we call $\mathcal{X}_V, \mathcal{X}_{RF}$ and $\mathcal{X}_{\mathcal{D}}$ that we want to be equal to $\mathcal{X}$.

First, we build $\mathcal{A}_V$ in the following way. For the i-th voter id, $\mathcal{A}_V$ runs the $\mathcal{O}\mathsf{register}(\mathsf{id})$, the $\mathcal{O}\mathsf{check}(\mathsf{id})$, and the $\mathcal{O}\mathsf{vote}(\mathsf{id}, \mathsf{v}_{1,i})$ oracles. During this call of the $\mathcal{O}\mathsf{vote}$ oracle, $\mathcal{A}_V$ interacts with an honest execution of the

[22] A result function which does not depend on the votes would be questionable anyway.

$\mathsf{Vote}_{\mathsf{H}}$ algorithm and simulates an execution of $\mathsf{Vote}_{\mathsf{VD}}$ up to the point of sending a ballot b to CS. Instead, it runs the algorithm $\mathcal{D}(\mathsf{id}, \mathcal{I}^{\mathsf{b}}, \mathsf{v}_{0,i})$ and sends to CS the resulting ballot. After all the oracle calls, the tally is computed on BB and every voter runs the $\mathsf{VerifyVote}$ algorithm. Finally, we define $\mathcal{X}_V$ as the pair of the bulletin board BB of the experiment and the list of receipts RCPT output by $\mathsf{Vote}_{\mathsf{H}}$ in the $\mathcal{O}\mathsf{vote}$ oracle call.

Second, we build $\mathcal{A}_{RF}$ in the following way. For the i-th voter id, $\mathcal{A}_{RF}$ runs the $\mathcal{O}\mathsf{register}(\mathsf{id})$ and the $\mathcal{O}\mathsf{corrupt}(\mathsf{id})$ oracles. Then, it runs the $\mathcal{O}\mathsf{receiptLR}(\mathsf{id}, \mathcal{I}^{\mathsf{b}}, \mathsf{v}_{0,i})$ oracle, where $\mathsf{b}, \Gamma \leftarrow\$ \mathsf{Vote}_{\mathsf{V}}(\mathsf{id}, \mathsf{v}_{1,i})$. Finally, $\mathcal{A}_{RF}$ calls the $\mathcal{O}\mathsf{tally}$ oracle and receives the result and a proof of validity of the tally. For each of the voters, it runs the $\mathsf{VerifyVote}$ algorithm on its view of the bulletin board and the receipt received from the $\mathcal{O}\mathsf{receiptLR}$ oracle. If any of these checks returns 0, $\mathcal{A}_{RF}$ outputs 1. Otherwise, it outputs 0. Finally, we define $\mathcal{X}_{RF}$ as the pair of the bulletin board BB_0 of the experiment and the list of receipts RCPT output by the $\mathsf{Vote}_{\mathsf{V}}$ algorithm used to create the instruction $\mathcal{I}^{\mathsf{b}}$ of each $\mathcal{O}\mathsf{receiptLR}$ oracle call.

Third, $\mathcal{A}_{\mathcal{D}}$ runs as follows. For the i-th voter id, $\mathcal{A}_{\mathcal{D}}$ runs the $\mathcal{O}\mathsf{register}(\mathsf{id})$ oracle and the $\mathcal{O}\mathsf{vote}(\mathsf{id}, \mathcal{I}^{\mathsf{b}}, \mathsf{v})$ oracle with a ballot b generated as $\mathsf{b}, \Gamma \leftarrow\$ \mathsf{Vote}_{\mathsf{V}}(\mathsf{id}, \mathsf{v}_{1,i})$. Finally, we define $\mathcal{X}_{\mathcal{D}}$ as the pair of the bulletin board BB_1 of the experiment and the list of receipts RCPT output by the $\mathsf{Vote}_{\mathsf{V}}$ algorithm used to create the instruction $\mathcal{I}^{\mathsf{b}}$ of each $\mathcal{O}\mathsf{vote}$ oracle call.

By construction, for all these three adversary, the ballots appended to the bulletin board and the receipts are computed exactly in the same way as $\mathcal{X}$ and these distributions are then equal: we have $\mathcal{X}_V = \mathcal{X}_{RF} = \mathcal{X}_{\mathcal{D}} = \mathcal{X}$. Let $\mathsf{Supp}(\mathcal{X}) = \{\mathsf{x} \,|\, \mathsf{Pr}_{\mathsf{y} \in \mathcal{X}}[\mathsf{y} = \mathsf{x}] \neq 0\}$ be the support of the distribution $\mathcal{X}$. For each $(\mathsf{BB}, \mathsf{RCPT}) \in \mathsf{Supp}(\mathcal{X})$, we now look at the following cases:

- $\mathsf{VerifyVote}(\Gamma) = 1$ for all $\Gamma \in \mathsf{RCPT}$: We write $r = \mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$, where the secret key sk of the election is equally distributed in all the experiments.
  - If $r = \rho(\{\mathsf{v}_{0,i}\})$: In that case, $\mathcal{A}_V$ wins its experiment as $\rho(\{\mathsf{v}_{0,i}\}) \neq \rho(\{\mathsf{v}_{1,i}\})$. We write $\mathcal{W}_V = \{\mathsf{x} \in \mathsf{Supp}(\mathcal{X}) \,|\, \mathsf{x}$ satisfies this case$\}$.
  - If $r \neq \rho(\{\mathsf{v}_{0,i}\})$: In that case, $\mathcal{A}_{\mathcal{D}}$ wins its experiment as the results computed in the experiments are such that $r_0 = \rho(\{\mathsf{v}_{0,i}\})$ and $r_1 = r$. We write $\mathcal{W}_{\mathcal{D}} = \{\mathsf{x} \in \mathsf{Supp}(\mathcal{X}) \,|\, \mathsf{x}$ satisfies this case$\}$.
- $\mathsf{VerifyVote}(\Gamma) = 0$ for some $\Gamma \in \mathsf{RCPT}$: In that case, $\mathcal{A}_{RF}$ wins

its experiment. Indeed, $\mathcal{A}_{RF}$ returns 1 if $\beta = 1$ because of the correctness of the voting protocol and returns 0 otherwise. We write $\mathcal{W}_{RF} = \{\mathsf{x} \in \mathsf{Supp}(\mathcal{X}) \,|\, \mathsf{x} \text{ satisfies this case}\}$.

Since $\forall \mathsf{x} \in \mathsf{Supp}(\mathcal{X})$ at least one of those three cases occurs, we have $\bigcup_{i \in \{V,RF,\mathcal{D}\}} \mathcal{W}_i = \mathsf{Supp}(\mathcal{X})$. Eventually,

$$\mathsf{Pr}_{\mathsf{y} \leftarrow\$\, \mathcal{X}_i}[\mathcal{A}_i \text{ wins} \,|\, \mathsf{x} = \mathsf{y}] \geq \mathsf{Pr}_{\mathsf{y} \leftarrow\$\, \mathcal{X}_i}[\mathsf{y} \in \mathcal{X}_i \,|\, x = y],$$

and, by the law of total probability, we conclude that

$$\begin{aligned}
&\sum_{i \in \{V,RF,\mathcal{D}\}} \mathsf{Adv}^{\mathsf{i}}_{\mathcal{A}_i}(\lambda) \\
&\quad = \sum_{\mathsf{x} \in \mathsf{Supp}(\mathcal{X})} \sum_{i} \mathsf{Pr}_{\mathsf{y} \leftarrow\$\, \mathcal{X}_i}[\mathcal{A}_i \text{ wins} \wedge \mathsf{x} = \mathsf{y}] \\
&\quad = \sum_{\mathsf{x}} \sum_{i} \mathsf{Pr}_{\mathsf{y} \leftarrow\$\, \mathcal{X}_i}[\mathcal{A}_i \text{ wins} \,|\, \mathsf{x} = \mathsf{y}] \mathsf{Pr}_{\mathsf{y} \leftarrow\$\, \mathcal{X}_i}[\mathsf{x} = \mathsf{y}] \\
&\quad \geq \sum_{\mathsf{x}} \sum_{i} \mathsf{Pr}_{\mathsf{y} \leftarrow\$\, \mathcal{X}_i}[\mathsf{y} \in \mathcal{X}_i \,|\, \mathsf{x} = \mathsf{y}] \mathsf{Pr}_{\mathsf{y} \leftarrow\$\, \mathcal{X}}[\mathsf{x} = \mathsf{y}] \\
&\quad = \sum_{\mathsf{x}} \mathsf{Pr}_{\mathsf{y} \leftarrow\$\, \mathcal{X}}[\mathsf{x} = \mathsf{y}] \sum_{i} \mathsf{Pr}_{\mathsf{y} \leftarrow\$\, \mathcal{X}_i}[\mathsf{y} \in \mathcal{W}_i \,|\, \mathsf{x} = \mathsf{y}] \\
&\quad \geq \sum_{\mathsf{x} \in \mathsf{Supp}(\mathcal{X})} \mathsf{Pr}_{\mathsf{y} \leftarrow\$\, \mathcal{X}}[\mathsf{x} = \mathsf{y}] \;=\; 1
\end{aligned}$$

Hence, even if we do not know which adversary wins with a non-negligible advantage, we get $\mathsf{Adv}^{\mathsf{vd-verif}}_{\mathcal{A}_V}(\lambda) + \mathsf{Adv}^{\mathrm{rf}}_{\mathcal{A}_{RF}}(\lambda) + \mathsf{Adv}^{\mathrm{dev}}_{\mathcal{A}_{\mathcal{D}}}(\lambda) \geq 1$. □

We stress that this impossibility result holds even if we considerably weaken the verifiability experiment by forbidding the adversary to corrupt voters. Indeed, the adversary for the verifiability experiment that we build in the proof behaves in this way.

Now if the protocol has a registration or an interactive voting protocol, then we cannot build the adversaries of our proof. Indeed, if there is a registration, then the deceiving strategy takes as input the private credentials of the voter to which the adversary for the verifiability experiment does not have access. Then, if the voting protocol is interactive, the adversary for the verifiability experiment cannot just send the output of the deceiving algorithm to CS. The instructions given as input to the deceiving algorithms are now interactions between the voter and CS that we cannot rewind if needed.

It is tempting to only use registration (like BeleniosRF). Indeed, if the voting device does not have access to the secret credentials, we will

give a protocol that has verifiability against a corrupted voting device and receipt-freeness. However, the secrecy of the private credentials is essential, as it is suggested by the following corollary.

Corollary 1. *There is no voting system with a non-interactive voting protocol that satisfies receipt-freeness and verifiability against a corrupted voting device with leaked credentials.*

Proof. We proceed similarly as in the previous proof but we add the private credential as input of the $\mathcal{I}$ and $\mathcal{D}$ algorithms. □

5.4. Protocols

We now propose three protocols that have receipt-freeness and verifiability against all parties. The first two protocols demonstrate how we can achieve these security properties without voter registration and by using an interactive voting process. The first protocol has a constant number of rounds but is quite demanding for the voter, while the second protocol is much simpler for the voter but comes with a number of rounds that is linear in the security parameter. Our third protocol achieves the same security properties with a non-interactive voting process but depends on a voter registration phase. We will prove the security of these protocols in Section 5.5.

For the sake of simplicity, we first focus on approval voting elections of one candidate. That is, $\mathbb{V} = \{0, 1\}$ and $\rho(\{\mathsf{v}_i\}) = \sum \mathsf{v}_i$. The protocols are easily extended to deal with the encryption of a larger number of bits.

5.4.1. A Constant-Round Interactive protocol

The first protocol relies on a pair of secret values α_0, α_1 that the voter generates outside of the voting device.

Instead of directly casting a vote v, the voter encrypts two multiplicative shares v_0 and v_1 in two ciphertexts c_0 and c_1 which are randomized and published on BB. We can homomorphically retrieve an encryption of the vote by computing $c = c_0 \cdot c_1$.

Then the voter will ask the voting device and the casting server to publish an opening of the ciphertext $c_0^{\alpha_0} \cdot c_1^{\alpha_1}$, which should contain $y = \mathsf{v}_0^{\alpha_0} \cdot \mathsf{v}_1^{\alpha_1}$ and does not reveal anything about the vote. Since neither the voting device nor the casting server knows α_0 and α_1 in advance, they cannot predict the value of this opening t when publishing the randomized ballot on the board. Hence, if they modify the value of c_0

or c_1, they will not be able to provide an opening consistent with the voter's ciphertexts, which will provide the cast-as-intended verifiability.

Crucially, the voter has to commit to these secret α_0, α_1 before giving them to the voting device and casting server, which is the form of interaction that is needed in this protocol. Otherwise, a voter would be unable to get around the instructions of a receipt-freeness adversary for the same reasons the voting device cannot modify the voter's ciphertexts.

Finally, we need to provide a proof that $c_0 \cdot c_1$ contains a plaintext that is a bit. To do so, we use a divertible proof executed with the casting server.

Here is a more precise description of our first interactive protocol Π_1. As a reminder, we respectively use H, VD and CS to designate the human voter, the voting device and the casting server.

$\mathsf{Setup}(1^\lambda)$: First, the election authorities generate the public parameters of the algorithms and two additional generators h_0 and h_1 of $\mathbb{G}$. Then, the talliers compute a distributed Elgamal key pair $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow\!\$\, \mathsf{TGen}(1^\lambda)$. The public key is computed as $\mathsf{pk} = (h_0, h_1, \mathsf{pk}_e)$ and the secret key as $\mathsf{sk} = \mathsf{sk}_e$.

$\mathsf{Vote}(\mathsf{id}, \mathsf{v})$: The voting procedure has 3 steps which are detailed in Fig 5.5.

In the first step, H samples two uniformly random values $\alpha_0, \alpha_1 \in \mathbb{Z}_q^*$ such that $\alpha_0 \neq \alpha_1$. Then, it computes a multi-Pedersen commitment $\mathsf{Com} = g^t h_0^{\alpha_0} h_1^{\alpha_1}$ and sends this commitment to CS.

In the second step, H encodes v in the voting device. VD chooses two random shares of v in $\mathbb{G}$ such that $\mathsf{v} = \mathsf{v}_0\mathsf{v}_1$. Then, it computes encryptions of these shares $c_0 = \mathsf{Enc}(\mathsf{pk}_e, \mathsf{v}_0; r_0)$ and $c_1 = \mathsf{Enc}(\mathsf{pk}_e, \mathsf{v}_1; r_1)$ and sends these values to CS. CS rerandomizes them into $\tilde{c}_0 = \mathsf{Rand}(\mathsf{pk}_e, c_0; s_0), \tilde{c}_1 = \mathsf{Rand}(\mathsf{pk}_e, c_1; s_1)$, and generates with VD a divertible proof π that $\tilde{c}_0\tilde{c}_1$ decrypts into a plaintext is in $\mathbb{V}$.

In the last step, H sends to the voting device an opening of (t, α_0, α_1) of Com. VD computes $x = \alpha_0 r_0 + \alpha_1 r_1$ and $y = \mathsf{v}_0^{\alpha_0}\mathsf{v}_1^{\alpha_1}$ and sends $(t, \alpha_0, \alpha_1, x, y)$ to CS. Then, CS updates x into $\tilde{x} = x + \alpha_0 s_0 + \alpha_1 s_1$ and returns it to the voter. It publishes on BB the ballot $\mathsf{b} = (\tilde{c}_0, \tilde{c}_1, \pi, \mathsf{Com}, t, \alpha_0, \alpha_1, \tilde{x}, y)$.

The receipt of the voter is $\Gamma = (\mathsf{v}, \mathsf{v}_0, \mathsf{v}_1, \tilde{c}_0, \tilde{c}_1, \alpha_0, \alpha_1)$.

$\mathsf{Valid}(\mathsf{b}, \mathsf{BB})$: Return 1 if b has the shape of $(\tilde{c}_0, \tilde{c}_1, \pi, \mathsf{Com}, t, \alpha_0, \alpha_1, \tilde{x}, y)$, if the proof π is valid, if $\alpha_0 \neq \alpha_1 \neq 0$, if (t, α_0, α_1) is an opening of Com, and if $\tilde{c}_0^{\alpha_0}\tilde{c}_1^{\alpha_1} = \mathsf{Enc}(\mathsf{pk}_e, y; \tilde{x})$.

$\mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$: At the tally phase, the talliers run the generic tally algorithm.

$\mathsf{VerifyVote}(\mathsf{BB}, \Gamma)$: returns 1 if Γ has the shape of $(\mathsf{v}, \mathsf{v}_0, \mathsf{v}_1, \tilde{c}_0, \tilde{c}_1, \alpha_0, \alpha_1)$, if there is a valid ballot b on BB with the same $\tilde{c}_0, \tilde{c}_1, \alpha_0, \alpha_1$ as in Γ, if $\alpha_0 \neq \alpha_1$, and if $\mathsf{v} = \mathsf{v}_0\mathsf{v}_1$. It returns 0 otherwise.

It is not necessary in the protocol to include y in the ballot, as the voter could without assistance check that $c_0^{\alpha_0} \cdot c_1^{\alpha_1} = \mathsf{Enc}(\mathsf{pk}_e, y; \tilde{x})$. However, it does not break the ballot privacy or the receipt-freeness and enables other parties to verify the validity of the ballot, which simplifies the voter's verification process.

We also propose the following deviating strategy $\mathcal{D}$ to cast a vote for v:

- Given some instruction $\mathcal{I}$, $\mathcal{D}$ first runs $\mathcal{I}$ once to get the commitment Com and sends it to CS.
- Then, $\mathcal{D}$ simulates an execution of $\mathsf{Vote}_{\mathsf{CS}}$ with $\mathcal{I}$. In the last step of the protocol, it receives an opening of t, α_0, α_1 of Com. If this is an invalid opening, $\mathcal{D}$ aborts.
- Using this opening, $\mathcal{D}$ finds the solution d_0, d_1 of this system[23]:

$$\begin{pmatrix} 1 & 1 \\ \alpha_0 & \alpha_1 \end{pmatrix} \begin{pmatrix} d_0 \\ d_1 \end{pmatrix} = \begin{pmatrix} \mathsf{Enc}(\mathsf{pk}_e, \mathsf{v}; r) \\ \alpha_0 c_0 + \alpha_1 c_1 \end{pmatrix}$$

  and sends d_0, d_1 to CS. It receives a randomized $\tilde{d}_0, \tilde{d}_1$ and runs the divertible proof with CS.
- $\mathcal{D}$ finally rewinds $\mathcal{I}$ to the beginning of the divertible proof and runs it with the randomized $\tilde{d}_0, \tilde{d}_1$ received from CS. Then if $\mathcal{I}$ outputs a valid opening of Com in the last round, $\mathcal{D}$ sends t, α_0, α_1 to CS. It aborts otherwise. Finally, $\mathcal{D}$ returns the output of $\mathcal{I}$ as the receipt.

On the number of rounds: With some optimizations, this protocol only needs three interactions between the voting device and the server. In the first interaction, VD sends Com, c_0, and c_1 and the commitments of the Σ-protocol for the divertible proof. It then receives the randomized ciphertext and the second message of the Σ-protocol in the second interaction. In the last interaction, VD sends the remaining values and the responses of the Σ-protocol.

[23] We use here the additive notation to have a clearer presentation.

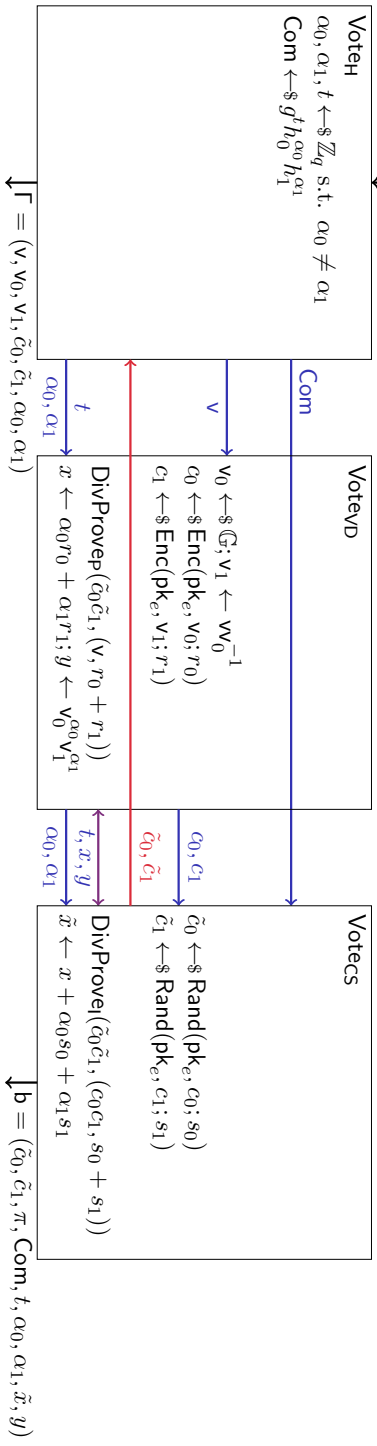


Figure 5.5.: Description of the Vote protocol of Π_1.

On the vote deviation verifiability: This protocol is also verifiable when using the deviating strategy, at the cost of more work for the voter. We write $\mathcal{D}$ as an interactive protocol $\mathcal{D}_{\mathsf{H}} \leftrightarrow \mathcal{D}_{\mathsf{VD}}$, where $\mathcal{D}_{\mathsf{H}}$ runs most of the steps described above. It only needs to send $\mathsf{Com}, d_0, d_1, t, \alpha_0, \alpha_1$ to VD which forwards them to CS and run the divertible proof with VD forwarding the messages. Similarly as we did, we can show that if a corrupted VD forwards a pair of ciphertexts d'_0, d'_1 that contains another vote, then the voter will notice it with overwhelming probability.

Scalability for more than one bit: To encrypt more than one bit in a ballot, one can run the second and third round of the protocol in parallel for each additional bit, using the same α_0 and α_1. The protocol has thus the same round complexity and the number of operations grows linearly with the number of bits.

Instantiation: We can instantiate this protocol using the ElGamal encryption scheme and the divertible proof described in the building blocks and a multi-Pedersen commitment scheme. This can thus be build in a (plain) DDH group.

While this protocol is technically secure, it is impractical to consider that a voter can compute the commitment of the α_0, α_1 values without the voting device. They could use another trusted device, or multiple devices in a MPC-way, but this may not be very practical from a usability perspective and this approach still relies on some trust (if all the devices are corrupted, the protocol is not verifiable anymore).

We thus present another protocol which, despite the use of more interaction, is more tractable for the voter.

5.4.2. A human-compatible interactive protocol

The second protocol relies on a different approach. Instead of asking the voting device and casting server to provide an opening of $c_0^{\alpha_0} c_1^{\alpha_1}$, we ask them to provide an opening of either c_0 or c_1, depending on the choice of the voter. If a corrupted party modifies c_0 or c_1, it will have a probability $\frac{1}{2}$ to get caught. By repeating this process $\mathcal{O}(\lambda)$ times, we can amplify this probability as much as we want. Here is a more precise description of our first interactive protocol Π_2:

1. $\mathsf{Setup}(1^\lambda)$: First, the election authorities generate the public parameters of the algorithms. Then, the talliers compute a distributed Elgamal key pair $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow\!\$\, \mathsf{TGen}(1^\lambda)$. The public key is computed as $\mathsf{pk} = \mathsf{pk}_e$ and the secret key as $\mathsf{sk} = \mathsf{sk}_e$.

2. $\mathsf{Vote}(\mathsf{id}, \mathsf{v}, \mathsf{BB})$: The voting procedure has $k = \mathcal{O}(\lambda)$ rounds and is detailed in Fig 5.6. In the first round, H sends a vote v to the voting

device, which computes $c = \mathsf{Enc}(\mathsf{pk}, \mathsf{v}; r)$ and sends the ciphertext to CS. Finally, CS rerandomizes it into $\tilde{c} = \mathsf{Rand}(\mathsf{pk}, c; s)$, and generates together with VD a divertible proof π that $\tilde{c}$ decrypts into a plaintext $\in \mathbb{V}$.

Then, in each of the following rounds, VD chooses again random shares of v in $\mathbb{G}$ such that $\mathsf{v} = \mathsf{v}_0\mathsf{v}_1$. It computes $c_0 = \mathsf{Enc}(\mathsf{pk}, \mathsf{v}_0; r_0)$ and $c_1 = \mathsf{Enc}(\mathsf{pk}, \mathsf{v}_1; r_1)$ such that $r_0 + r_1 = r$. It then sends the ciphertext to CS, which rerandomizes them into $\tilde{c}_0 = \mathsf{Rand}(\mathsf{pk}, c_0; s_0)$, $\tilde{c}_1 = \mathsf{Rand}(\mathsf{pk}, c_1; s_1)$ such that $s_0 + s_1 = s$ and sends them to H.

H can either choose to drop the ballot or to cast it. If H chooses to drop it, VD starts the round again from the beginning, chooses new shares of the vote, and proceeds as before. If H chooses to cast it, they also chooses a bit b and sends v_b, r_b and b to CS. In that case, CS publishes on BB the randomized ciphertexts, the bit b, v_b, and $z = r_b + s_b$. Anyone can check that $c_b = (g^z, \mathsf{v}_b\mathsf{pk}^z)$. If H is not trying to deviate from the instructions of a receipt-free adversary, they can always choose to cast the ballot.

The ballot published on the board has thus the shape of $(\tilde{c}, \pi, \{\tilde{c}_{0,i}, \tilde{c}_{1,i}, b_i, \mathsf{v}_i, z_i\}_{i=1}^k)$ and the receipt of the voter is $\Gamma = (\mathsf{v}, \tilde{c}_0, \tilde{c}_1, \{b_i, \mathsf{v}_{b_i}, \tilde{c}_{0,i}, \tilde{c}_{1,i}\}_{i=1}^k)$.

3. $\mathsf{Valid}(\mathsf{b}, \mathsf{BB})$: Return 1 if b has the shape of $(\tilde{c}, \pi, \{\tilde{c}_{0,i}, \tilde{c}_{1,i}, b_i, \mathsf{v}_i, z_i\}_{i=1}^k)$, if π is a valid proof, if $\tilde{c} = \tilde{c}_{0,i}\tilde{c}_{1,i}$ and if $\tilde{c}_{b_i,i} = \mathsf{Enc}(\mathsf{pk}, \mathsf{v}_i; z_i)$ for all i.

4. $\mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$: At the tally phase, the talliers first check for each ballot that it is valid. If this is not the case, the ballot is dropped. Finally, the tallier compute homomorphically the tally from the valid $\tilde{c}$ and publish a proof of correct decryption as well.

5. $\mathsf{VerifyVote}(\mathsf{BB}, \Gamma)$: In the verification phase, each voter returns 1 if the receipt $\Gamma = (\mathsf{v}, \tilde{c}, \{b_i, \mathsf{v}_{b_i}, \tilde{c}_{i,0}, \tilde{c}_{i,1}\}_{i=1}^k)$ and, if there is a ballot b on BB with the same ciphertext $\tilde{c}$ and the same tuples $(\tilde{c}_{i,0}, \tilde{c}_{i,1}, b_i, \mathsf{v}_{b_i})$ for each round as in Γ.

We also propose the following deviating strategy $\mathcal{D}$:

- Given some instruction $\mathcal{I}$, $\mathcal{D}$ first runs $\mathcal{I}$ once to get a ciphertext c' that will be ignored. Then, it computes its own $d = \mathsf{Enc}(\mathsf{pk}, \mathsf{v}; r)$ and sends it to CS. It gets the rerandomization $\tilde{d}$ and computes the divertible proof protocol π that the rerandomization of d is decrypted into a plaintext $\in \mathbb{V}$. Also, it simulates an execution of this protocol with $\mathcal{I}$ as in the deviation strategy of Π_1.

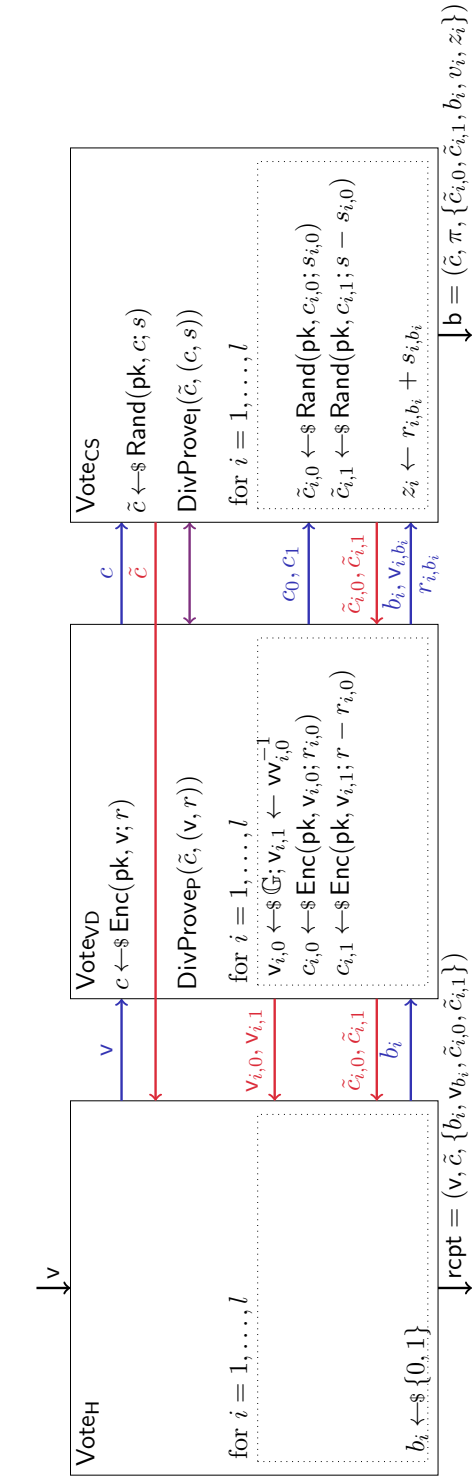


Figure 5.6.: Description of the Vote protocol of Π_2.

- Then in the i-th round, $\mathcal{D}$ runs $\mathcal{I}$ with the current state, which is the initial $(\tilde{d}, \pi)$ augmented by each of the part of the ballot that is cast. $\mathcal{I}$ outputs a tuple $(c_{i,0}, c_{i,1})$ and $\mathcal{D}$ chooses a bit b_i at random. It computes an encryption $d_i = \frac{d}{c_{1-b_i}}$ and it sends to CS either (d_i, c_1) if $b_i = 0$ or (c_0, d_i) if $b_i = 1$. $\mathcal{D}$ then receives a rerandomization of these ciphertexts from CS and it forwards these values to $\mathcal{I}$ which outputs a bit $b_{i,\mathcal{I}}$. If $b_{i,\mathcal{I}} = b_i$, then $\mathcal{D}$ sends a drop message to CS and starts again from the previous step. It also rewinds $\mathcal{I}$ to its state before the start of this previous step.

  If $b_{i,\mathcal{I}} \neq b_i$, it receives from $\mathcal{I}$ the values $v_{b_{i,\mathcal{I}}}$ and $r_{b_{i,\mathcal{I}}}$. $\mathcal{D}$ then casts the ballot with these values and repeats this step until k values have been posted on the board.

- $\mathcal{D}$'s receipt is the same as $\mathcal{I}$'s receipt.

On the number of rounds: It is possible to pack several rounds together. Instead of sending only one tuple (c_0, c_1), the voter can send l tuples in one round, and ask one opening for each of those tuples in the next round. However, the deviating strategy has now a probability $\frac{1}{2^l}$ to guess the value which will not reveal the deviating v to the adversary. To keep an expected polynomial time deviating strategy, we can thus batch at most $\mathcal{O}(\log^c \lambda)$ rounds together.

On the vote deviation verifiability: Unlike the previous protocol, this protocol does not have verifiability against a corrupted voting device if the voter uses the $\mathcal{D}$ algorithm. Indeed, for an instruction that always ask to reveal the bit $b = 0$ independently, the voting device can cheat on the d_1 ciphertext without being detected. It is however possible to slightly change the protocol to attain this notion of verifiability. Instead of choosing the random bit, the voter, runs a verifiable coin flipping algorithm with CS. This can be done by VD in the honest algorithm but must be done by the voter alone in the vote deviation algorithm.

Scalability for more than one bit: To encrypt more than one bit in a ballot, one can encrypt each bit independently in the setup round. Then in each of the following rounds, we compute two encryptions of the shares of each bit and ask CS to open the same encryption for all the bits. The deviation strategy proceeds similarly by always replacing the same share of the adversary by the desired value. The protocol has thus also the same round complexity and the number of operations also grows linearly with the number of bits.

Instantiation: We can instantiate this protocol using the ElGamal encryption scheme and the divertible proof described in the building blocks. Again, we can thus instantiate the protocol in a (plain) DDH

group.

5.4.3. Protocol with registration

We now show how to modify the first protocol Π_1 to replace the interactions with a registration process.

Getting rid of the commit-then-open mechanism

We make three changes in Π_1:

1. Instead of choosing α_0, α_1, the voter receives them from the registration authority as there is no interaction to prove the knowledge of them.

2. Since the protocol is-single pass, the voter cannot give the opening of y, x to CS in a later round. However the voter can send $y = \mathsf{v}_0^{\alpha_0}\mathsf{v}_1^{\alpha_1}$ to the voting device and still have the individual verifiability. Indeed, even if the voting device knows the target, it does not help to pass the test as (α_0, α_1) is independent of the view of the voting device.

3. As there is no opening of the $\tilde{c}_0^{\alpha_0}\tilde{c}_1^{\alpha_1}$ value on the bulletin board to test the authenticity of the ciphertext, we proceed differently. Instead, the talliers publish the result of a plaintext equivalence proof (PEP) that $\tilde{c}_0^{\alpha_0}\tilde{c}_1^{\alpha_1}$ and $\mathsf{Enc}(\mathsf{pk}, y; 0)$ contain the same value.

Non-interactive proof of a valid vote

We still need to make a non-interactive proof that $\tilde{c}_0\tilde{c}_1$ contains a valid vote. Instead of the divertible proof protocol, we could use a simulation sound randomizable proof. However, such a ballot would be replayable: an adversary could copy a ballot on the bulletin board and cast a rerandomization of it in order to break the receipt-freeness or even the ballot privacy. In order to address this problem, we turn to a traceable receipt-free encryption scheme (TREnc), which offers the TCCA security notion (Definition 22) that we need. Regarding this TREnc, we stress that we do not need it to be traceable or strongly randomizable: the TCCA security is enough.

We still need to be careful: in our TCCA reduction we can use only one challenge for each link key, and there are two ciphertexts with the same link key in a ballot. Fortunately, if $y = \mathsf{v}_0^{\alpha_0}\mathsf{v}_1^{\alpha_1}$, then we can define $\tilde{c}_1$ as $(\tilde{c}/\tilde{c}_0^{\alpha_0})^{\alpha_1^{-1}}$ where $\tilde{c}$ contains the same encryption of y for the ballots

output by $\mathcal{I}$ and the ballot output by $\mathcal{D}$. With this rewriting, we only need one TCCA reduction to switch the ballot output by $\mathcal{D}$ to the ballot output by $\mathcal{I}$.

However, we are not finished yet, since the ballot output by $\mathcal{I}$ does not verify $y = \mathsf{v}_0^{\alpha_0}\mathsf{v}_1^{\alpha_1}$ and will be dropped in the pre-tally. Hence, we need another TCCA reduction to switch the ballots to verify equation and to simulate the PEP proofs accordingly.

Non-interactive Protocol

We propose a non-interactive protocol Π_3 that makes use of the registration system:

1. $\mathsf{Setup}(1^\lambda)$: First, the election authorities generate a bilinear group $\mathcal{G} = (\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T) \leftarrow\!\$\ \mathsf{GrpGen}(1^\lambda)$ and the CRS crs of a malleable non-interactive proof system as described in section 2.2 in a public coin manner. Then, the talliers compute a distributed TREnc key pair $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow\!\$\ \mathsf{Gen}(1^\lambda)$, with non-interactive proofs of knowledge of their share of the key. They then return $\mathsf{pk} = (\mathcal{G}, \mathsf{crs}, \mathsf{pk}_e)$ and $\mathsf{sk} = \mathsf{sk}_e$.

2. $\mathsf{Register}(\mathsf{id}, 1^\lambda)$: We first generate random $S \leftarrow\!\$\ \mathbb{G}$ and set α_0, α_1 as the hash $H(S)$. Then, we generate the link key of a TREnc with the $\mathsf{lk} \leftarrow\!\$\ \mathsf{LGen}(\mathsf{pk})$ algorithm. Finally, we set $\mathsf{usk}_{\mathsf{id}}$ to $(\alpha_0, \alpha_1, \mathsf{lk})$ (which is sent to H) and $\mathsf{upk}_{\mathsf{id}}$ to $(\mathsf{Enc}(\mathsf{pk}, S), t)$ (which is sent to the bulletin board) where t is the trace of the TREnc linked to lk .

3. $\mathsf{Vote}(\mathsf{id}, \mathsf{v})$: The voting procedure is detailed in Fig 5.7.

   H first encodes v and lk in the voting device. Then VD chooses two shares $(\mathsf{v}_0, \mathsf{v}_1)$ of v in $\mathbb{G}$ such that $\mathsf{v} = \mathsf{v}_0\mathsf{v}_1$. From these shares, H computes the target value $y = \mathsf{v}_0^{\alpha_0}\mathsf{v}_1^{\alpha_1}$ and encodes it in the voting device.

   Then, the device computes two ciphertexts $c_0 = \mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, \mathsf{v}_0)$ and $c_1 = \mathsf{Enc}(\mathsf{pk}, \mathsf{lk}, \mathsf{v}_1)$ and a non-interactive divertible proof π that c_0c_1 decrypts into a plaintext $\in \mathbb{V}$. (c_0, c_1, π, y) is then sent to CS. CS then randomizes the ciphertext into $(\tilde{c}_0, \tilde{c}_1, \tilde{\pi}, y)$ and publishes it on the bulletin board next to the id of the voter. In case of a revote, the last ballot will be counted.

   We note $\mathsf{b} = (\tilde{c}_0, \tilde{c}_1, \tilde{\pi}, y)$ and $\Gamma = (\mathsf{v}, \mathsf{v}_0, \mathsf{v}_1, y, t, \alpha_0, \alpha_1)$.

4. $\mathsf{Valid}(\mathsf{b}, \mathsf{BB})$: Return 1 if b can be parsed as $(\mathsf{id}, \tilde{c}_0, \tilde{c}_1, \tilde{\pi})$, if the proof is valid and if $\mathsf{Trace}(\tilde{c}_0) = \mathsf{Trace}(\tilde{c}_1) = t$ where t is the trace of $\mathsf{upk}_{\mathsf{id}}$.

5. $\mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$: First the talliers decrypt together each of the $\mathsf{upk}_{\mathsf{id}}$ to get the α_0 and α_1 of the valid ballots. Then, they run the plaintext equality proof algorithm to check that each of the $c_0^{\alpha_0} c_1^{\alpha_1}$ is an encryption of the y value of the voter and publish the output π_{PEP} of the algorithm on the bulletin board. If the test is passed, they compute homomorphically the tally from the valid $\tilde{c}_0 \tilde{c}_1$ and publish a non-interactive zero-knowledge proof (using a standard Σ protocol) that the result is indeed a correct decryption.

6. $\mathsf{VerifyVote}(\mathsf{BB}, \Gamma)$: Return 1 if Γ has the shape of $(\mathsf{v}, \mathsf{v}_0, \mathsf{v}_1, y, t, \alpha_0, \alpha_1)$, if there is a valid ballot b on BB with trace t and the same y value as in Γ, if the decrypted α_0, α_1 associated to this ballot are the same as the values in the receipt, if the associated π_{PEP} is verified and if $\mathsf{v} = \mathsf{v}_0 \mathsf{v}_1$. Return 0 otherwise.

Since there is no interaction, the deviating strategy is simpler:

- Given some instruction $\mathcal{I}$, the voter first runs $\mathcal{I}$ once to get the ballot c_0, c_1, π, t to send to the server.
- To cast a vote v, the voter finds the solution d_0, d_1 of this system:

$$\begin{pmatrix} 1 & 1 \\ \alpha_0 & \alpha_1 \end{pmatrix} \begin{pmatrix} d_0 \\ d_1 \end{pmatrix} = \begin{pmatrix} \mathsf{Enc}(\mathsf{pk}, \mathsf{v}; r) \\ \alpha_0 c_0 + \alpha_1 c_1 \end{pmatrix} \tag{5.1}$$

  and computes the associated proof.
- The voter uses d_0 and d_1 for the CPA parts of two TREnc encryptions with lk, computes an independent proof, and sends the resulting ballot to CS.

On the vote deviation verifiability: This protocol has vote deviation verifiability for the same reason as Π_1.

Scalability for more than one bit: To encrypt more than one bit in a ballot, one can run the vote protocol for each bit independently, using the same link key lk. Thus the credentials keep a constant size and the number of operations also grows linearly with the number of bits.

Instantiation: We can instantiate this protocol using, for example, the TREnc of our first construction and the malleable proof system and the plaintext equality proof described in the building blocks. Those blocks are compatible, since the TREnc of contains an ElGamal CPA-secure part that can be used for the homomorphic operations and the plaintext equality test, and the proof of validity is homomorphic for the same trace. Hence, we can instantiate the protocol in a pairing-friendly SXDH group.

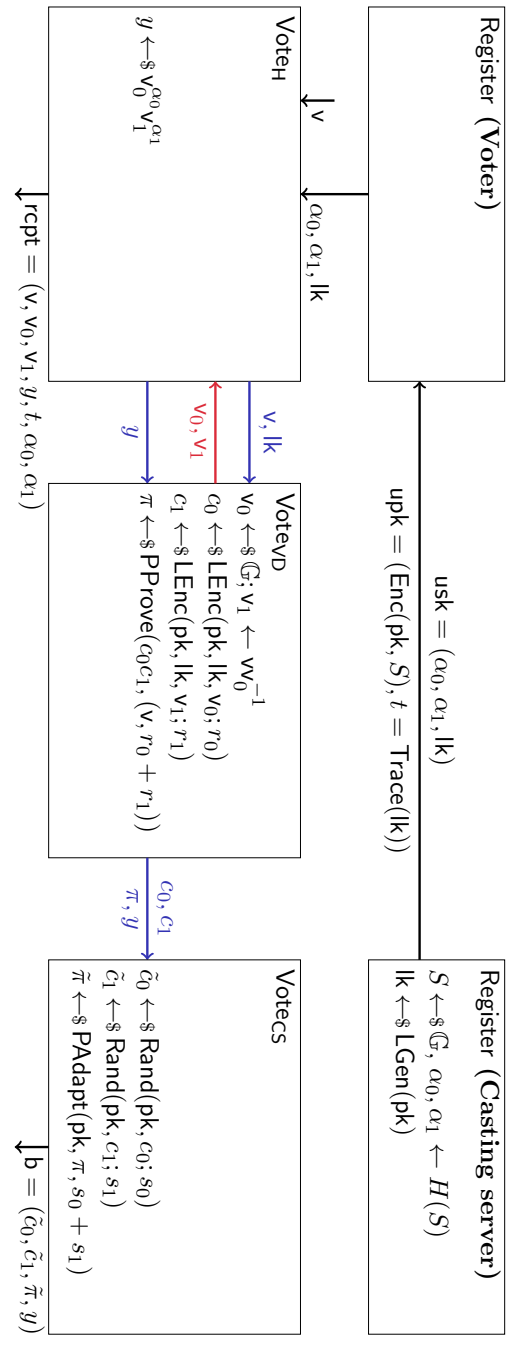


Figure 5.7.: Description of the Vote protocol of Π_3.

Even though this theorem does not give any guarantee about verifiability in the presence of malicious parties besides the voting device, it can be shown that the protocol is secure as long as neither the voting device nor CS has access to the private credentials of the voter.

However, Π_3 cannot be secure against a corrupted voting device if the credentials are leaked, as the voting device could follow the same deviating strategy. As hinted by Corollary 18, we need interactions in the voting procedure to achieve this.

5.5. Security of the protocols

Theorem 19. *Protocol Π_1 has receipt-freeness and is verifiable. Especially*

- Π_1 *has receipt-freeness if the encryption scheme is an ElGamal encryption scheme, and if the divertible proof has the zero-knowledge, special soundness and prover obliviousness properties.*

- Π_1 *is verifiable against a malicious voting device and the casting server if the encryption scheme is an ElGamal encryption scheme and if the hash function is modeled as a random oracle.*

Proof. **Receipt-freeness**

Let $\mathcal{A}$ be an adversary against the receipt-freeness of the voting scheme as in Definition 29. We give a succession of hybrid games, where the first one is the real RF game for $\beta = 0$. For each of those games, we denote by S_i the event that the i-th game outputs 1. When we say that the challenger aborts, we mean that the experiment outputs a random bit.

$\mathsf{Game}_0(\lambda)$**:** This it the original receipt-freeness game. By definition, $Pr[\mathsf{Exp}^{\mathsf{srf},0}_{\mathcal{A},\Pi_1}(\lambda) = 1] = Pr[S_0]$. As a reminder, the adversary in this game can ask a polynomially bounded number of $\mathcal{O}\mathsf{receiptLR}(\mathsf{b}_0, \mathsf{b}_1)$ and $\mathcal{O}\mathsf{cast}(\mathsf{b})$ requests and a final $\mathcal{O}\mathsf{tally}$ request.

$\mathsf{Game}_1(\lambda)$**:** We define $\mathsf{Game}_1(\lambda)$ as $\mathsf{Game}_0(\lambda)$ with the following modification. Instead of computing the proof of the tally honestly with TProve, we run the $\mathsf{TSimProve}$ algorithm and define $\mathsf{SimProve}$ accordingly. Also, we use the zero-knowledge simulator to simulate all the divertible proofs.

From the zero-knowledge property of the proof of the tally, this transition is indistinguishable. Also, from the zero-knowledge and the prover obliviousness properties of the divertible proof, this transition is indistinguishable.

$\mathsf{Game}_2(\lambda)$**:** In the next game, we compute the result of the election in another way. Instead of running the $\mathsf{TCompute}$ algorithm, we compute the result of the election differently. First, we use the simulation soundness of the divertible proof to extract the vote v of each valid ballot in an $\mathcal{O}\mathsf{cast}$ oracle. Together with the votes v of the $\mathcal{O}\mathsf{receiptLR}$ oracle, we compute directly the tally with the result function ρ applied on those votes.

From the correctness of the encryption scheme and the tallying procedure, and the special soudness of the divertible proof, $Pr[S_2] = Pr[S_1]$.

$\mathsf{Game}_3(\lambda)$**:** Now in $\mathsf{Game}_3(\lambda)$, each time we use the Rand algorithm in an $\mathcal{O}\mathsf{receiptLR}$ request to compute $\tilde{c}_0$ we instead compute a fresh encryption of the message.

These games are indistinguishable because of the strong randomizability of the encryption scheme. It is non-trivial to retrieve the value v_0 of the ballot b output by $\mathcal{I}$ in the $\mathcal{O}\mathsf{receiptLR}$ oracle but with the values $\alpha_0, \alpha_1, y = \mathsf{v}_0^{\alpha_0}\mathsf{v}_1^{\alpha_1}$ (that we can obtain by rewinding the $\mathcal{I}$), and by extracting $\mathsf{v}_0 \cdot \mathsf{v}_1$ from the divertible proof as in the previous game, we can solve the linear system and find v_0.

$\mathsf{Game}_4(\lambda)$**:** In this game, we compute the $\tilde{c}'$ and $\tilde{c}_1$ values differently in the $\mathcal{O}\mathsf{cast}$ and $\mathcal{O}\mathsf{receiptLR}$ oracles. We first generate $\tilde{c}' = \mathsf{Enc}(\mathsf{v}_0^{\alpha_0}\mathsf{v}_1^{\alpha_1}; \tilde{x})$ for a random $\tilde{x} \leftarrow\!\!\$\, \mathbb{Z}_q$ and then we set $\tilde{c}_1 = (\tilde{c}'/\tilde{c}_0^{\alpha_0})^{\alpha_1^{-1}}$.

Since this transition is just a recomputation of the same values, $Pr[S_4] = Pr[S_3]$.

$\mathsf{Game}_5(\lambda)$**:** We at last set β to 1. That is, we return BB_1 instead of BB_0 in the $\mathcal{O}\mathsf{board}$ oracle and generate the simulated proof Π from BB_1 in the $\mathcal{O}\mathsf{tally}$ oracle.

We reduce to a multi-challenge version of the CPA game of the encryption scheme.

From an adversary $\mathcal{B}$ distinguishing between $\mathsf{Game}_4(\lambda)$ and $\mathsf{Game}_5(\lambda)$, we build the multi-challenge CPA adversary as follows. First, we define BB as an empty bulletin board and forward the public key of the CPA encryption scheme to $\mathcal{B}$ and each time we receive a $\mathcal{O}\mathsf{board}$ request, we send BB to the adversary.

Then, for each $\mathcal{O}\mathsf{receiptLR}$ request, we compute everything as in $\mathsf{Game}_4(\lambda)$, except $\tilde{x}_0$. Instead, we ask for a challenge with $(m_0, m_1) = (\mathsf{v}_0^{\mathcal{D}}, \mathsf{v}_0^{\mathcal{I}})$ to the CPA challenger, where $\mathsf{v}_0^{\mathcal{D}}$ and $\mathsf{v}_0^{\mathcal{I}}$ are respectively the first shares of the ballots output by $\mathcal{D}$ and $\mathcal{I}$. We then append the resulting ballot to BB and return the receipt output by $\mathcal{I}$.

Finally, when we receive the $\mathcal{O}\mathsf{tally}$ request, we directly compute the result of the election as in $\mathsf{Game}_4(\lambda)$, and use the $\mathsf{TSimProve}$ algorithm to prove the validity of the result. This adversary perfectly simulates $\mathsf{Game}_4(\lambda)$ or $\mathsf{Game}_5(\lambda)$ depending on the CPA challenge bit and hence, $|Pr[S_5] - Pr[S_4]| \leq \varepsilon_{CPA}$.

$\mathsf{Game}_6(\lambda)$**:** Finally in the last game $\mathsf{Game}_6(\lambda)$, we revert all the changes on the ballots in the reverse order. We use the Rand algorithm and the honest divertible proof again, and recompute the result from $\mathsf{TCompute}$ again. Again, from the strong randomizability of the encryption scheme, the zero-knowledge and prover obliviousness of the divertible proof and the correctness of the tallying algorithm, this transition is indistinguishable.

The last game is equivalent to the receipt-freeness experiment for $\beta = 1$, namely $\mathsf{Exp}^{\mathsf{irf},1}_{\mathcal{A},\Pi_1}(\lambda)$. Hence, we conclude that the probability that the adversary wins the receipt-freeness experiment is negligible.

Vote deviation correctness In each $\mathcal{O}\mathsf{receiptLR}(\mathsf{id}, \mathcal{I}, \mathsf{v})$ call, either $\mathcal{I}$ does not produce a valid opening of Com such that $\alpha_0 \neq \alpha_1$ (and in that case $\mathcal{I}$ forces the ballot to be dropped), or from this opening the linear system used in $\mathcal{D}$ has a solution. If so, the crafted ballot has the same distribution as an output of Vote_H.

Verifiability

Let $\mathcal{A}$ be an adversary against the verifiability of the voting scheme as in Definition 31. We give a succession of hybrid games, where the first one is the verifiability experiment $\mathsf{Exp}^{\mathrm{all-verif}}_{\mathcal{A},\Pi_1}(\lambda)$. For each of those games, we denote by S_i the event that the i-th game outputs 1. When we say that the challenger aborts, we mean that the experiment outputs a random bit.

$\mathsf{Game}_1(\lambda)$**:** In the first game, we create a trapdoor for the multi-Pedersen commitments. Namely, instead of picking h_0 and h_1 as random group elements, we generate $h_0 = g^{\gamma_0}$ and $h_1 = g^{\gamma_1}$.

This is statistically indistinguishable for the adversary.

$\mathsf{Game}_2(\lambda)$**:** In $\mathsf{Game}_2(\lambda)$, we change how we process the $\mathcal{O}\mathsf{Vote}$ requests.

Instead of computing a commitment for α_0 and α_1 as the first step of Vote_H, we commit to random α'_0 and α'_1. Using the trapdoor, we compute a consistent opening t, α_0, α_1.

This transition is statistically indistinguishable, as the commitment scheme is perfectly hiding.

5. Can we cast a ballot as intended and be Receipt-Free?

$\mathsf{Game}_3(\lambda)$**:** $\mathsf{Game}_3(\lambda)$ proceeds as $\mathsf{Game}_2(\lambda)$, but we compute the tally differently. Instead of computing the tally through the Tally algorithm, we decrypt each ballot and compute the results r from the result function ρ.

The correctness of the tally and the decryption guarantee that the result is computed identically if all ballots are in the range of $\mathsf{Vote}(\cdot)$.

$\mathsf{Game}_4(\lambda)$**:** Before defining $\mathsf{Game}_4(\lambda)$, we define some variables. We define v_{id} as the vote of an honest voter id and t_{id} as the t value computed in the execution of the $\mathsf{Vote}_{\mathsf{H}}$ algorithm in the $\mathcal{O}\mathsf{vote}$ oracle.

We also define $\tilde{\mathsf{v}}_{\mathsf{id}}$ as the vote contained in the ballot $\tilde{\mathsf{b}}_{\mathsf{id}}$ in BB with the same value t_{id} of the same voter (or $\perp$ if it does not exist). Finally, we define $\tilde{V}_H$ and $\tilde{V}_C$ as the set of $\{\tilde{\mathsf{v}}_{\mathsf{id}}\}$ and as the set of the decryption of the other valid ballots on BB not corresponding to the honest voters.

Now, we define $\mathsf{Game}_4(\lambda)$ as $\mathsf{Game}_3(\lambda)$, but we let the adversary win the game if all the $\mathsf{VerifyVote}$ checks go through and if for at least one honest voter, $\tilde{\mathsf{v}}_{\mathsf{id}} \neq \mathsf{v}_{\mathsf{id}}$.

This happens with a negligible probability. Indeed, we would have a receipt $\mathsf{rcpt} = (\mathsf{v}, \mathsf{v}_0, \mathsf{v}_1, \tilde{c}_0, \tilde{c}_1, \alpha_0, \alpha_1)$ such that $\mathsf{Dec}(\mathsf{sk}, \tilde{c}_0\tilde{c}_1) \neq \mathsf{v}$ but $\mathsf{VerifyVote}(\mathsf{BB}, \mathsf{rcpt}) = 1$. Then for $\mathsf{v}_{0,\mathcal{A}} = \mathsf{Dec}(\mathsf{sk}, \tilde{c}_0)$ and $\mathsf{v}_{1,\mathcal{A}} = \mathsf{Dec}(\mathsf{sk}, \tilde{c}_1)$, we have that either $\mathsf{v}_{0,\mathcal{A}} \neq \mathsf{v}_0$ or $\mathsf{v}_{1,\mathcal{A}} \neq \mathsf{v}_1$.

Since $\tilde{c}_0, \tilde{c}_1, \tilde{\pi}$ are computed before $\mathcal{A}$ has any information about (α_0, α_1), $\mathcal{A}$ has to guess the value of $\mathsf{v}_0^{\alpha_0}\mathsf{v}_1^{\alpha_1}$ at random, and the the probability that $\mathsf{v}_{0,\mathcal{A}}^{\alpha_0}\mathsf{v}_{1,\mathcal{A}}^{\alpha_1}$ is equal to $\mathsf{v}_0^{\alpha_0}\mathsf{v}_1^{\alpha_1}$ is bounded by $\frac{1}{q}$.

Finally, since $V_H = \tilde{V}_H$ and since the result is computed as $\rho(\tilde{V}_H \cup \tilde{V}_C)$, the probability that the adversary wins the verifiability experiment is negligible. □

Theorem 20. *Protocol* Π_2 *has receipt-freeness and is verifiable. Especially*

- Π_2 *has receipt-freeness if the encryption scheme is an ElGamal encryption scheme and if the divertible proof has the zero-knowledge, special soundness, and prover obliviousness properties.*
- Π_2 *is verifiable against a malicious voting device and the casting server if the encryption scheme is an ElGamal encryption scheme.*

Proof. **Receipt-freeness**: We can use the same succession of hybrids as in the proof of Π_1. Briefly, the modifications are as follows:

1. In $\mathsf{Game}_3(\lambda)$, we extract the value $\mathsf{v}^{\mathcal{I}}$ in the ballot computed by $\mathcal{I}$ in the $\mathcal{O}\mathsf{receiptLR}$ oracle (thanks to the special soundness of the divertible proof), and then ignore the remainder of the computation of this ballot, since it is independent of the view of the adversary. In all the other algorithms, we replace the executions of the Rand algorithm by fresh encryptions.

   Especially in $\mathcal{O}\mathsf{receiptLR}$:

   - During the execution of the $\mathcal{D}$ algorithm, in each round, $\mathcal{I}$ sends a message c_0^i, c_1^i. We run ourselves the next interaction of the algorithm and receive the message (b, v_b) from $\mathcal{I}$. Then we rewind and let $\mathcal{D}$ follow its execution. From this extraction, we compute $\mathsf{v}_{1-b} = \mathsf{v}/\mathsf{v}_b$.

     Then, instead of randomizing c_b and d/c_b, we encrypt v_b and v_{1-b}.
   - For the other ballot b_1, we set $\tilde{c}_b = \mathsf{Enc}(\mathsf{v}_b^i)$ and $\tilde{c}_{1-b} = \mathsf{Enc}(\mathsf{v}^{\mathcal{I}}/\mathsf{v}_b^i)$.

2. We do not need $\mathsf{Game}_4(\lambda)$.

3. In the reduction of $\mathsf{Game}_5(\lambda)$, we first use $(\mathsf{v}, \mathsf{v}^{\mathcal{I}})$ as a challenge to the CPA game, and then at each round for a guess bit b_i, we use the challenge $(\mathsf{v}_{1-b_i}) = (\mathsf{v}/\mathsf{v}_{b_i}^i, \mathsf{v}^{\mathcal{I}}/\mathsf{v}_{b_i}^i)$.

4. Finally we add a transition where we transform $\tilde{c}_{1-b_i} = \mathsf{Enc}(\mathsf{v}^{\mathcal{A}}/\mathsf{v}_{b_i}^i)$ into $\mathsf{Rand}(c_{1-b_i}^i)$, which have the same distribution.

Hence, we conclude that the probability that the adversary wins the receipt-freeness experiment is negligible.

Vote deviation correctness: In the first step of the protocol, $\mathcal{D}$ encrypts a vote and computes the divertible proof in the same way as the $\mathsf{Vote}_{\mathsf{V}}$ algorithm. Hence if the resulting ballot is valid, it will be counted similarly.

Then in each round, the deviating algorithm has one chance over two to choose the bit $b^{\mathcal{D}}$ that will not be chosen by $\mathcal{I}$ (that we call $b^{\mathcal{I}}$).

Indeed, from the DDH assumption, $\mathcal{I}$ cannot distinguish between a randomization of the ciphertext c_b and a randomization of the ciphertext of the deviating algorithm $d/c_{1-b^{\mathcal{D}}}$. Since the chosen bit $b^{\mathcal{D}}$ is then independent of $b_{\mathcal{I}}$, $\mathcal{D}$ will send a cast message with probability one over two. Hence for each round, the probability that it takes more than λ tries for the deviating strategy to send a "cast" message is bounded by $2^{-\lambda}$ (which is negligible) as the number of tries follows a geometric distribution

of parameter $\frac{1}{2}$. Thus, the probability that for all the rounds combined the deviating strategy needs more than $k\lambda$ tries is also negligible.

The expected number of rounds before the termination of the protocol is thus linear and the ballot posted on BB correctly encodes the voter's intention.

Verifiability: Let $\mathcal{A}$ be an adversary against the verifiability of the voting scheme as in Definition 31. We give a succession of hybrid games, where the first one is the verifiability experiment $\mathsf{Exp}^{\text{all-verif}}_{\mathcal{A},\Pi_1}(\lambda)$. For each of those games, we denote by S_i the event that the i-th game outputs 1. When we say that the challenger aborts, we mean that the experiment outputs a random bit.

$\mathsf{Game}_0(\lambda)$**:** We define $\mathsf{Game}_0(\lambda)$ as the original verifiability experiment. In this game, the adversary in this game can ask a polynomially bounded number of $\mathcal{O}\mathsf{vote}(\mathsf{id}, \mathsf{v})$ requests.

$\mathsf{Game}_1(\lambda)$**:** $\mathsf{Game}_1(\lambda)$ proceeds as $\mathsf{Game}_0(\lambda)$, but we compute the tally differently. Instead of computing the tally through the Tally algorithm, we decrypt each ballot and compute the results r from the result function ρ.

The correctness of the tally and the decryption guarantee that the result is computed identically if all ballots are in the range of $\mathsf{Vote}(\cdot)$.

$\mathsf{Game}_2(\lambda)$**:** Before defining $\mathsf{Game}_2(\lambda)$, we define some variables. We define v^{id} as the vote of an honest voter id and $\mathsf{v}^{\mathsf{id}}_{i,j}$ as the shares of the vote they generated in the i-th round of the $\mathsf{Vote}_{\mathsf{H}}$ algorithm in the $\mathcal{O}\mathsf{vote}$ oracle.

We also define $\tilde{\mathsf{v}}^{\mathsf{id}}$ as the vote in the ballot $\tilde{\mathsf{b}}_{\mathsf{id}}$ in BB associated to the voter, (or $\perp$ if it does not exist), and $\tilde{\mathsf{v}}^{\mathsf{id}}_{i,j}$ as the values inside the $\tilde{c}^{\mathsf{id}}_{i,j}$ of the ballot. Finally, we define $\tilde{V}_H$ and $\tilde{V}_C$ as the set of $\{\tilde{\mathsf{v}}^{\mathsf{id}}\}$ and as the set of the decryptions of the other valid ballots on BB not corresponding to the honest voters.

Now, we define $\mathsf{Game}_2(\lambda)$ as $\mathsf{Game}_1(\lambda)$, but we let the adversary win the game if all the $\mathsf{VerifyVote}$ checks go through and if for at least one honest voter, $\tilde{\mathsf{v}}^{\mathsf{id}} \neq \mathsf{v}^{\mathsf{id}}$.

This happens with a negligible probability. Indeed, if $\tilde{\mathsf{v}}^{\mathsf{id}} \neq \mathsf{v}^{\mathsf{id}}$, then for all i, $\tilde{\mathsf{v}}^{\mathsf{id}}_{i,0} \neq \mathsf{v}^{\mathsf{id}}_{i,0}$ or $\tilde{\mathsf{v}}^{\mathsf{id}}_{i,1} \neq \mathsf{v}^{\mathsf{id}}_{i,1}$. Also, for a ballot $\mathsf{b} = (\tilde{c}, \pi, \{\tilde{c}_{0,i}, \tilde{c}_{1,i}, b_i, \mathsf{v}_i, z_i\}_{i=1}^{k})$ to be valid, we need that: $\tilde{\mathsf{v}}^{\mathsf{id}}_{b_i,i} = \mathsf{v}^{\mathsf{id}}_{b_i,i}$ for all i. Since b_i is generated after the value $\tilde{\mathsf{v}}^{\mathsf{id}}_{b_i}$ and is uniformly random, the adversary has a probability $\frac{1}{2}$ to have modified the right $\tilde{\mathsf{v}}^{\mathsf{id}}_{j,i}$ which is not open. All

together, since there are $k = \mathcal{O}(\lambda)$ such tests, this event happens with a negligible probability $\frac{1}{2^k}$.

Finally, since $V_H = \tilde{V}_H$ and since the result is computed as $\rho(\tilde{V}_H \cup \tilde{V}_C)$, the probability that the adversary wins the verifiability experiment is negligible. □

Theorem 21. *Protocol* Π_3 *has receipt-freeness and is verifiable against all parties. Especially*

- Π_3 *has receipt-freeness if the encryption scheme is a homomorphic* TCCA*-secure and verifiable TREnc and if the malleable proofs, and if the plaintext equality proof are zero-knowledge.*
- Π_3 *is verifiable against a malicious voting device and the casting server if the encryption scheme is a homomorphic* TCCA*-secure TREnc and if the plaintext equality proof is sound.*

Proof. **Receipt-freeness**:
Let $\mathcal{A}$ be an adversary against the receipt-freeness of the voting scheme as in Definition 29. We give a succession of hybrid games, where the first one is the real RF game for $\beta = 0$. For each of those games, we denote by S_i the event that the i-th game outputs 1. When we say that the challenger aborts, we mean that the experiment outputs a random bit.

$\mathsf{Game}_0(\lambda)$**:** This it the original receipt-freeness game. By definition, $Pr[\mathsf{Exp}^{\mathsf{srf},0}_{\mathcal{A},\Pi_3}(\lambda) = 1] = Pr[S_0]$. As a reminder, the adversary in this game can ask a polynomially bounded number of $\mathcal{O}\mathsf{receiptLR}(\mathsf{b}_0, \mathsf{b}_1)$ requests and a final $\mathcal{O}\mathsf{tally}$ request.

$\mathsf{Game}_1(\lambda)$**:** We define $\mathsf{Game}_1(\lambda)$ as $\mathsf{Game}_0(\lambda)$ with the following modification. Instead of computing the proof of the tally honestly with TProve, we run the $\mathsf{TSimProve}$ algorithm and define $\mathsf{SimProve}$ accordingly. Also, we use the zero-knowledge simulator to simulate all the non-interactive proofs of validity instead of rerandomizing them.

From the zero-knowledge property of the proof of the tally and the malleable proof system, this transition is indistinguishable.

$\mathsf{Game}_2(\lambda)$**:** We introduce in this game the following failure event which happens if for at least one of the ballots of the $\mathcal{O}\mathsf{cast}$ requests, the ballot contains a ciphertext c_0 or c_1 which is valid but not in the range of $\mathsf{TEnc}(\mathsf{pk}, \cdot)$. If the event happens, we abort and return a random bit.

This event happens with negligible probability, since the TREnc is verifiable.

$\mathsf{Game}_3(\lambda)$**:** In the next game, we compute the result of the election in another way. Instead of running the $\mathsf{TCompute}$ algorithm, we compute the result of the election differently. First, we use the secret key to decrypt each ballot requested in the $\mathcal{O}\mathsf{cast}$ oracle calls. Together with the votes v of the $\mathcal{O}\mathsf{receiptLR}$, we compute directly the tally with the result function ρ applied on those votes.

From the correctness of the encryption scheme and the tallying algorithm, and the special soudness of the divertible proof, $Pr[S_3] = Pr[S_2]$.

$\mathsf{Game}_4(\lambda)$**:** In $\mathsf{Game}_4(\lambda)$, we compute the randomization of the ballots in the $\mathcal{O}\mathsf{receiptLR}$ requests in another way.

We first compute $c = c_0^{\alpha_0} c_1^{\alpha_1}$ where c_0 and c_1 are output by $\mathcal{I}$. Then for a $\mathsf{b} = (c_0, c_1, \pi, y)$ we compute, $\tilde{c}_0 = \mathsf{Rand}(c_0)$ as before but we compute $\tilde{c}_1 = \mathsf{Rand}(c)/\mathsf{Rand}(\tilde{c}_0^{\alpha_0})^{\alpha_1^{-1}})$.

Clearly, this value is computed as before and the games are identical.

$\mathsf{Game}_5(\lambda)$**:** Now, we simulate the plaintext equality proof. For each ballot (c_0, c_1, π, y) output by $\mathcal{I}$ in a $\mathcal{O}\mathsf{receiptLR}$ request, we set $y^* = \mathsf{Dec}(c_0^{\alpha} c_1^{\alpha_1})$. Then, when we compute the plaintext equality proof for this, we simulate a proof of equal plaintexts if $y^* = y$ or a proof of different plaintexts otherwise.

Using y^*, we know exactly when we need to simulate each mode and from the zero-knowledge property of the plaintext equality proof, those two games are indistinguishable.

$\mathsf{Game}_6(\lambda)$**:** $\mathsf{Game}_6(\lambda)$ differs from $\mathsf{Game}_5(\lambda)$ in the way we compute the $\mathcal{O}\mathsf{receiptLR}$ requests, again.

Instead of computing $c = c_0^{\alpha_0} c_1^{\alpha_1}$ as defined in the previous game, we compute $c = \mathsf{TEnc}(\mathsf{lk}, y)$. Also, in the $\mathcal{D}$ algorithm, we use the same c computed as an encryption of y in the second equation of the linear system that $\mathcal{D}$ solves.

We can reduce an adversary $\mathcal{B}$ distinguishing between $\mathsf{Game}_5(\lambda)$ and $\mathsf{Game}_6(\lambda)$ to a TCCA adversary $\mathcal{A}$ as follows.

First, we define BB as an empty bulletin board and forward the public key of the TCCA encryption scheme to $\mathcal{B}$. Each time we receive an $\mathcal{O}\mathsf{board}$ request, we send BB to the adversary.

For each $\mathcal{O}\mathsf{receiptLR}$ request, first request a decryption y^* of $c_0^{\alpha_0} c_1^{\alpha_1}$ output by $\mathcal{I}$. Then, we use a TCCA challenge to compute the randomization of the c or $\mathsf{TEnc}(\mathsf{lk}, y)$, depending on the challenge bit of the TCCA adversary.

Then for each $\mathcal{O}\mathsf{cast}(\mathsf{b})$ request, we ask the decryption oracle of the TCCA game to decrypt the content of the $\tilde{c}_0$ and $\tilde{c}_1$ to recover the ballot in order to count the result. For this ballot to be valid, it must have a trace distinct from the traces of the challenge ciphertexts and thus the decryption request is answered.

Finally, when we receive the $\mathcal{O}\mathsf{tally}$ request, we simulate the plaintext equality proof. Then, we compute the result of the election as in $\mathsf{Game}_5(\lambda)$ and use the $\mathsf{TSimProve}$ algorithm to prove the validity of the result. This adversary perfectly simulates $\mathsf{Game}_5(\lambda)$ or $\mathsf{Game}_6(\lambda)$ depending on the TCCA challenge bit and hence, $|Pr[S_6] - Pr[S_5]| \leq \varepsilon_{TCCA}$.

$\mathsf{Game}_7(\lambda)$**:** We at last set β to 1. That is, we return BB_1 instead of BB_0 in the $\mathcal{O}\mathsf{board}$ oracle and generate the simulated proof Π from BB_1 in the $\mathcal{O}\mathsf{tally}$ oracle. We also reduce to a multi-challenge version of the TCCA game of the TREnc, using almost the same reduction.

The only change is the challenges that we send to the TCCA request. For each $\mathcal{O}\mathsf{receiptLR}$ request, we now send a challenge $(c_0^{\mathcal{D}}, c_0^{\mathcal{I}})$, where $c_0^{\mathcal{D}}$ and $c_0^{\mathcal{I}}$ are respectively the c_0 components in the ballot output by $\mathcal{D}$ and $\mathcal{I}$. We compute the remainder of the ballots ourselves.

Depending on the bit of the TCCA challenger, we either compute the ballots of BB_0 or the ballots of BB_1, and this transition is indistinguishable.

$\mathsf{Game}_8(\lambda)$**:** Finally in the last game $\mathsf{Game}_8(\lambda)$, we reverse our modification of the ballots and recompute the result from $\mathsf{TCompute}$ again.

The last game is equivalent to the receipt-freeness experiment for $\beta = 1$, namely $\mathsf{Exp}^{\mathsf{irf},1}_{\mathcal{A},\Pi_3}(\lambda)$. Hence, we conclude that the probability that the adversary wins the receipt-freeness experiment is negligible.
Vote deviation correctness: With overwhelming probability, $\alpha_0 \neq \alpha_1$ and the linear system used in $\mathcal{D}$ has a solution. If so, the crafted ballot has the same distribution as an output of $\mathsf{Vote}_{\mathsf{H}}$.

Verifiability against a corrupted voting device:
We can use the same succession of hybrids as in the proof of Π_1. Briefly, the modifications are as follows.
We just need to add two transitions:

- In a first hybrid, $\mathsf{Game}_{0,1}(\lambda)$, we replace the encryptions of the private S of the voters. We use an hybrid in which we replace the encryption of the private S of the voters on BB with trivial encryptions of 0. In the tally phase, the talliers directly use the

$\mathsf{upk}_{\mathsf{id}}$ of each voter without decrypting the value of the bulletin board.

From the CPA-security of the encryption scheme, this change is indistinguishable by the adversary.

- In a second hybrid $\mathsf{Game}_{0,2}(\lambda)$, we decrypt for each ballot in BB the value $y^* = \mathsf{Dec}(\tilde{c}_0^{\alpha_0}\tilde{c}_1^{\alpha_1})$. If $y = y^*$ but the plaintext equality proof proves an inequality, we abort the experiment. Similarly, if $y \neq y^*$ but the plaintext equality proof proves an equality, we abort as well.

  This transition is indistinguishable because of the soundness of the plaintext equality proof.

Then, we can follow with the other transitions. □

5.6. Performance and comparisons

We provide a python implementation [Rfv] to evaluate the time to encrypt and craft a ballot of various size using our protocols.

We conducted our experiments on a laptop with a 1.8Ghz Intel i7 processor and 16GB of RAM running Ubuntu 20.04 LTS and Python 3.7.9. We used the Charm framework [Aki+13] with pbc-0.5.14 and OpenSSL-1.0.1. We used a secp256k1 elliptic curve for Π_1 and Π_2 and a BN254 pairing friendly curve for Π_3. For the protocol Π_2, we set the number of rounds to 16, which means that an adversary can change the vote of a voter without being detected with probability at most 2^{16}. All our results are averaged over 100 runs.

	lb = 1	2	4	8	32
Π_1	0.005	0.009	0.018	0.035	0.137
Π_2	0.026	0.047	0.097	0.195	0.763
Π_3	0.139	0.288	0.584	1.231	4.819

Table 5.1.: Time to encrypt the vote of a ballot containing $1, 2, 4, 8$, and 32 bits. The times reported are in seconds.

We see in Table 5.1 that even for a 32-choice race, the encryption time of the protocol with registration is only a couple of seconds. It is not surprising that this protocol is considerably slower than the other two protocols, as it uses a pairing friendly group. The first two protocols are faster, but only the computation time on the voter's device is computed.

This means that the communication time with the server is ignored, which would not be negligible. We stress that our implementation prototype was designed to offer an upper-bound on the running time of our protocols (in the absence of a deviation strategy) and to verify that this running time is not prohibitive for natural use cases. Faster implementations could be obtained easily if needed.

We also feel the computational overhead of our protocols, compared to traditional techniques, remains within reasonable bounds. For instance, for submitting an encrypted bit, our first protocol computes 14 exponentiations (using the suggested building blocks): 3 for the computation of the commitment, 4 for the computation of the encryptions, 2 for the computation of the value y, and 5 for the computation of the divertible proof. Compared to a standard ElGamal + zero-knowledge proof of validity, as implemented in ElectionGuard 2.0 for instance, this is 8 additional exponentiations. This computational overhead of course comes with considerably stronger security properties in terms of receipt-freeness.

The main properties and performances of our protocol are summarized in Figure 5.2.

5.7. Concluding remarks

In this chapter, we explored the limits of single-pass voting, and especially demonstrated that such voting protocols cannot achieve both receipt-freeness and cast-as-intended verifiability without a trusted registration service. This fundamental result questions the very motivation of TREncs. Since a single-pass voting protocol without a trusted registration is impossible (if we want to reconcile these two security notions), we could as well rely on the registration to help us build more efficient protocols.

Furthermore, our third protocol only need the TCCA property to achieve receipt-freeness, and regarding the individual verifiability, the traceability property is superseded by the verification through the linear combination of the shares of the vote. It is thus completely possible to improve the efficiency of the third protocol by using a TREnc which does not have the traceability property. Yet, I believe that TREncs and especially the TCCA security notion are still theoretically important for effectively capturing a notion that generically implies receipt-freeness.

Overall, the protocols that we proposed are arguably hardly practical. They mostly demonstrate the feasibility of achieving both receipt-freeness and cast-as-intended at the same time and using different assumptions. Mostly, it opens the way to build more practical, more efficient and more usable protocols in the future. Especially, our only protocol that can

	Π_1	Π_2	Π_3
# rounds	3	$\mathcal{O}(\log^c \lambda)$	1
Registration	no		yes
Receipt-Freeness	yes		
Verifiability	against all parties		against VD
Complexity of Vote_H	complex; need to compute a commitment	easy; only need to sample random bits	complex; need to compute $y = \mathsf{v}_0^{\alpha_0}\mathsf{v}_1^{\alpha_1}$
Security assumption	plain DDH	plain DDH	pairing-friendly SXDH
Time to create a ballot (for one bit, in seconds)	0.005	0.026 (k = 16)	0.139
Ballot size (for one bit)	6 elements in $\mathbb{G}$ 8 elements in $\mathbb{Z}_q$ $\approx$ 448 bytes	$5k + 2$ elements in $\mathbb{G}$ $k + 4$ elements in $\mathbb{Z}_q$ $\approx$ 3264 bytes (k = 16)	30 elements in $\mathbb{G}$ 14 elements in $\hat{\mathbb{G}}$ $\approx$ 1856 bytes
Scalability	Linear in the number of bits (for time and ballot size)		

Table 5.2.: Summary of the properties and performances of the protocols.

be deemed as computationally tractable for a human voter, in the sense that the voter only needs to sample some random numbers, has a linear number of interactions. On the other hand, our protocols that have more demanding assumptions on the capabilities of the human voter use an optimal number of interactions. Designing protocols that need a small number of interactions while requiring simple operations human-wise remains still an open question.

Part II.

Practical Voting Scheme

6. How to Verifiably Encrypt Many Bits for an Election?

In the previous chapters, we proposed new original techniques and protocols to achieve more secure and more efficient voting protocols. However, implementing correctly a complex system involving cutting-edge cryptographic primitives is a hard task, and most solutions used in practice still follow the general methodology of [CGS97] presented in Section 2.3.1. It is thus worth it to study this established protocol and investigate how we can improve it without relying on the use of heavy cryptographic machinery such as TREncs.

This methodology is tailored to approval voting, in which voters are given a list of candidates and express for each candidate either their approval or their disapproval. To achieve this, the voters first encrypt a bit for each candidate using an ElGamal encryption scheme, then compute a non-interactive zero-knowledge proof that the ElGamal ciphertexts contain a bit using a disjunctive Chaum-Pedersen [CDS94; CP92a] in conjunction with the Fiat-Shamir heuristic [FS86].

In this approach, the cost of verifiably encrypting a bit is largely dominated by 7 modular exponentiations: 2 for the ElGamal encryption, and 5 for the proof. This cost can quickly become limiting in practice. Let us consider, as a motivating example, the state-of-the-art ElectionGuard SDK developed by Microsoft, which has been used in various public elections since 2020 [Ele24]. When ElectionGuard was deployed for a Risk Limiting Audit in Inyo County, the encryption of a single ballot took around 6 seconds, the exponentiations being reported as the bottleneck [Ben+21]. In the context of such an audit, thousands and possibly millions of ballots have to be verifiably encrypted, which led to the deployment of a cluster, raising numerous practical challenges and requiring significant expertise [Wal20].

Hence, we explore in this chapter how to optimize these computations, through the use of classic techniques such as fixed base exponentiations but also new encryption and zero-knowledge proof mechanisms. While some of these techniques do not synergize well together, we also explore how we can balance them to achieve fast ballot generation.

Structure of the chapter At first, Section 6.1 describes the baseline algorithm that we will optimize. Then we discuss in Section 6.2 how we can take advantage of fixed based exponentiation, and we explore in Section 6.3 the use of the multi-ElGamal scheme. Regarding the computation of the proof of validity, Section 6.4 shows how we can dramatically decrease this computational cost with techniques that still have a linear complexity. In Section 6.5 we then discuss the use of more recent developments aiming at producing proofs with a sublinear size. Finally, Section 6.6 proposes benchmarks of our different optimizations and discuss their relative benefits.

6.1. The Baseline

In all the voting systems based on homomorphic tallying that we examined, including [Adi+09; SDW08; CGG19; Ele24], the verifiable encryption of bits is performed as in the original protocol of Cramer et al. [CGS97], which can be described as follows.

1. A group G is chosen as a subgroup of prime order q of a group $\mathbb{Z}_p^*$, with $|q| = 256$ and $2048 \leq |p| \leq 4096$ depending on the implementation. A generator g of G is also chosen. Then, an ElGamal public key $h \in G$ is selected, with the corresponding secret key $x : h = g^x$ being kept secret by a group of trustees. (A distributed key generation protocols are used for this.)

2. A bit v is encrypted as an exponential ElGamal pair $(g^r, g^v h^r)$ for a random $r \leftarrow \mathbb{Z}_q$.

3. A disjunctive Chaum-Pedersen proof [CP92b; CDS94] is computed in order to demonstrate that $v \in \{0, 1\}$. Given a ciphertext $(A, B) = (g^r, g^v h^r)$ as above, a commitment is computed as a pair of ciphertexts $(A'_0, B'_0) = (g^s, g^{vw} h^s)$ and $(A'_1, B'_1) = (g^t, g^{(1-v)w} h^t)$ where $s, t, w \leftarrow\$ \mathbb{Z}_q$. Then a random challenge $e \in \mathbb{Z}_q$ is obtained using the Fiat-Shamir transform. Eventually, the sub-challenges $e_0 = (1 - v)e - w$ and $e_1 = ve + w$ are computed, as well as the responses $f_0 = s + e_0 r$ and $f_1 = t + e_1 r$. The proof is made of $(A'_0, B'_0, A'_1, B'_1, e_0, e_1, f_0, f_1)$ and is verified by the equations

$$g^{f_0} = A'_0 A^{e_0} \qquad g^{f_1} = A'_1 A^{e_1}$$
$$h^{f_0} = B'_0 B^{e_0} \qquad h^{f_1} = B'_1 B^{e_1}$$

In real-world implementations, the choice of a subgroup of $\mathbb{Z}_p^*$ is preferred over a group on elliptic curves, motivated by the desire to

keep the implementation of a verifier as accessible as possible: the basics of modular arithmetic are part of any CS curriculum, which is not the case for elliptic curves. Furthermore, there are standard big number libraries available in essentially every porgramming language, while the choice of a suitable elliptic curve cryptography library may be more tricky. Exponential ElGamal is preferred over the Benaloh or the Paillier schemes [Ben94; Pai99], which is also additively homomorphic and has a more efficient decryption process, because ElGamal comes with efficient threshold key generation protocols. The disjunctive Chaum-Pedersen proof can be described in various ways. Here, we follow ElectionGuard, whose specification can be accessed for further details [Ele24]. Numerous other descriptions exist and lead to equally or less efficient implementations.

We observe that the verifiable encryption of a bit requires 2 exponentiations for the ciphertext and 5 more exponentiations for the proof. In terms of storage, the ciphertext takes 2 elements in $\mathbb{Z}_p^*$, and the proof takes 4 elements in $\mathbb{Z}_p^*$ and 4 elements in $\mathbb{Z}_q$. Using the ElectionGuard parameters with $|p| = 4096$, we see that each verifiably encrypted bit requires 25600 bits $\approx$ 3KB. However (and even though this is usually not the case in practice), the proof size can be much reduced by omitting the 4 elements in $\mathbb{Z}_p^*$, which can be recomputed from the other ones, taking the size down to about 1KB.

6.2. Fixed-base exponentiation

Existing implementations of homomorphic voting schemes (e.g., VoteBox, Helios, Belenios, ElectionGuard...) make use of the exponentiation function of standard Multi-Precision arithmetic libraries for computing modular exponentiations, including for example gmpy2 [Hor] in Python and jsbn [Wu09] in JavaScript. These libraries support the computation of modular exponentiations as a stateless operation.

Numerous techniques however exist that make it possible to compute multiple exponentiations w.r.t. a single base much faster than independently [Ber02; HMV03]. This is precisely our case here: we only use bases g and h.

The design of most fixed base exponentiation algorithms was however guided by constraints that are quite different of those of voting exponentiations: while these algorithms behave very well when a small amount of memory is available to store the result of precomputation, voting applications can typically dedicate several MB, and possibly even GB of memory to precomputation storage.

It is tempting to consider such an option, given that we may need to compute a lot of exponentiations: a single race with half a dozen candidates will already require a few dozens of exponentiation, and a full ballot, which can often contain hundreds of choices, can take a few thousands of exponentiations. Even more challenging is the encryption of all the ballots cast in an election, as needed for a publicly verifiable risk limiting audit, which may require millions of exponentiations. Based on these observations, we explore the use of a simple precomputation approach based on the standard right-to-left k-ary exponentiation algorithm, aiming at minimizing the number of multiplications needed per exponentiation. We will compare it to other traditional approaches below.

Precomputation Suppose that we are willing to compute a lot of exponentiations in base g, with exponents of at most ℓ bits. We select a parameter k, and precompute a table of $t = \lceil \ell/k \rceil$ lines and 2^k columns, in which $table[i][j] \leftarrow g^{2^i \cdot j}$. Such a table can be computed using $t \cdot (2^k - 1)$ multiplications as the first column of "1"'s, for $j = 0$, requires no computation. As an example, for $\ell = 6$ and $k = 2$, the table looks as follows:

1	g	g^2	g^3
1	g^4	g^8	g^{12}
1	g^{16}	g^{32}	g^{48}

Computation We can now compute g^e for $e = (e_{t-1} \dots e_0)_{2^k}$ with $e_i \in \{0, \dots, 2^k - 1\}$ immediately: we just need to pick the correct element on each line of the table, and multiply them together: $g^e = \prod_{i=0}^{t-1} table[i][e_i]$.

This algorithm requires $t-1$ multiplications, and makes use of the first column of the table in order to deal with the cases where some $e_i = 0$. An alternative would be to simply exclude these terms from the product but, since we intend to use relatively large values of k, this strategy would only save us a marginal amount of memory and computation, while adding a test on each e_i value.

How to choose k? We see that, for a fixed exponent size, the number of multiplications and the storage that are required for the precomputation grow like $2^k/k$, while the online computation decreases like $1/k$.

Obviously, if the precomputation time does not matter (because it can be performed well in advance), choosing a value of k as large as the memory can fit would lead to the fastest online exponentiations. If we would like to minimize the total computation time, then the right

balance needs to be found between the time spent on precomputation and the time spent on computation: for n exponentiations in base g, the total number of multiplications is $\lceil \ell/k \rceil (2^k - 1) + (\lceil \ell/k \rceil - 1)n$. When $n \gg 1$, this expression is minimum when $n \approx (\ln(2)k - 1)2^k$.

For the sake of concreteness, we explore these values in the group used in ElectionGuard, that is $\ell = |q| = 256$. Table 6.1 contains the maximum value of n until which various choices of k are optimal, based on the multiplication count made above. We may observe that $k = 15$ and $k = 17$ are never optimal choices.

k	3	4	5	6	7
n	16	54	121	332	692
TS (MB)	0.3	0.5	0.8	1.4	2.4
$t-1$ OM	85	63	51	42	36
k	8	9	10	11	12
n	2219	3926	11265	20481	36865
TS (MB)	4.2	7.6	14	25	46
$t-1$ OM	31	28	25	23	21
k	13	14	16	18	
n	147457	245761	2883585	3407873	
TS (MB)	84	159	537	2013	
$t-1$ OM	19	18	15	14	

Table 6.1.: The n line gives an estimation of the maximum value of n for which choices of k are optimal. For instance, $k = 8$ is the best choice for $n \in [693, 2219]$. We also give the precomputation storage volume as the Table Size (TS) and the Online Multiplication (OM) computation work for various parameters of k.

The storage that is required for the precomputation table grows relatively quickly if we ignore the first column that only contains "1", we need to store $t \cdot (2^k - 1)$ group elements. Table 6.1 also shows these volumes for various values of k. Even though this grows quickly, the volumes remain lower than 100MB for values of k up to 13, which should be within reach of any modern computer. The table also shows that the benefits of increasing the value of k for such values also starts plummeting: moving from $k = 7$ to $k = 10$ gives a speedup by a factor 36/25=1.44 for an extra 11.6MB of storage, while moving from $k = 10$ to $k = 13$ only gives an extra factor 25/19=1.31 for an extra 70MB of storage.

Overall, we observe a few "sweet spots" in this table: in a low-memory setting, we see that computational gains remain fairly high until we

reach $k = 8$ and $t - 1 = 31$, which still comes with a very small memory requirement of 4MB. The value $k = 13$ and $t - 1 = 19$ is the last that saves at least 2 online multiplications compared to the previous value of k, and keeps memory requirements below 100MB. As we will see in our benchmarks in Section 6.6.1, all these values offer important speed improvements over the standard exponentiation function of big integer libraries.

What about other fixed-base exponentiation methods? The method that is described above is demanding in terms of precomputation table size, compared to traditional solutions. Nevertheless, we see that, when aiming for very fast exponentiations, it is quite competitive, and remarkably simple. To offer some points of comparison, we consider the classical methods as described in [HMV03] for the parameters listed above.

- The fixed-base windowing method (Algo. 3.41) is expected to require $2^k + t - 3$ multiplications/exponentiation and the storage of t values. This is minimum for $k = 4$ in our case, and requires 77 multiplications per exponentiation and the storage of 63 group elements. So, our method leads to faster exponentiations as soon as $k > 4$.

- The fixed-base comb method (Algo. 3.44) is expected to require $2t - 2$ multiplications per exponentiation, and the storage of 2^k precomputed values. If we aim for the same number of multiplications or exponentiations (meaning that k needs to be approximately twice as big for the comb method compared to our method), our method is more efficient as soon as $k > 5$.

- The two-table fixed-base comb method (Algo 3.45) is expected to require $3t/2 - 2$ multiplications/exponentiation, and the storage of 2^{k+1} precomputed values. Here, our method is more efficient as soon as $k > 7$.

So, it seems that, apart from its extreme simplicity, the method we described also offers important speed-ups. Our estimates only focus on the number of multiplications, being the bulk of the work here. More sophisticated methods focusing on the efficient computation of short multiplication chains, e.g., Pippenger's [Pip76], may require a smaller number of multiplications. However, they also require more bit-by-bit inspection in the exponents, and create multiplication chains that combine all the exponents, which requires additional bookkeeping.

6.3. Multi-ElGamal

The implementation improvement discussed above does not touch the voting protocol itself, easing its integration in an existing system. Nevertheless, it is appealing to explore whether the use of other cryptographic mechanisms would reduce the efforts needed to verifiably encrypt a bit. We start by exploring the case of ElGamal encryption, and will turn to the ZK proofs in the next sections.

ElGamal encryption requires 2 exponentiations per bit. But ElGamal encryptions can be easily batched if we have multiple public keys $(h_1, \ldots, h_m) = (g^{x_1}, \ldots, g^{x_m})$ with each $x_i \leftarrow \mathbb{Z}_q$: we can encrypt m bits $(v_0, \ldots, v_m)$ as $(g^r, g^{v_1} h_1^r, \ldots, g^{v_m} h_m^r)$ – the security of this multi-ElGamal scheme can be reduced to the one of the original ElGamal encryption scheme [BBS03]. We can now encrypt m bits with $m+1$ exponentiations, compared to $2m$ with plain ElGamal, leading to a speed-up by a factor close to 2 even for relatively low values of m.

However, exponentiations are now computed w.r.t. $m+1$ bases instead of 2, which may require more efforts of precomputation if we want to use fixed-base exponentiation methods. If the precomputation is taken offline, and in the absence of memory concerns, multi-ElGamal will always be more efficient.

But if the precomputation needs to be made online, then what we gain on one side may be lost on the other side. We can estimate this by exploring a few values by multiplication counts. If $n = 1000$, the two exponentiations of plain ElGamal (i.e., $m = 1$) require an optimal effort of 78320 multiplications for $k = 8$ (including precomputation), and a storage of 8MB. The use of multi-ElGamal can offer some benefits: we reach 66045 multiplications for $m = 4$ and $k = 6$, and a slightly lower precomputation volume of 7MB. If $n = 100000$, plain ElGamal requires an optimal effort of 4.13 million multiplications for $k = 13$. If we switch to multi-ElGamal, we can for instance obtain 3.06 million multiplications for $m = 5$ and $k = 11$. The storage needed for the tables again slightly decreases from 168MB to 151MB. Overall, we observe that the benefits increase when we have more votes to encrypt. But they remain well below the factor ≈ 2 that was hoped for a large m.

The adoption of multi-ElGamal may also be complicated by extra validity requirements on ballots. It is for instance quite common to require that a maximum number of candidates are selected within a single race. This is typically handled by computing an encryption of the number of candidates selected within the race as the homomorphic sum of the ciphertexts computed for each choice, and proving that this sum is within the expected range. However, this homomorphic addition won't

work if the choices within a race are encrypted with a different public key h_i. Nevertheless, it remains an option to use multi-ElGamal in an election with multiple races, and to use one public key h_i per race.

6.4. Adapting the ZK 0-1 proofs – Linear Techniques

The disjunctive version of the Chaum-Pedersen protocol described in Section 6.1, requires 5 modular exponentiations: 3 in base g, and 2 in base h, and makes most of the computational effort.

We will first see how a simple change in the ElGamal encryption process makes it possible to save 1 exponentiation in base g, down to a total of 4 exponentiations. As a second step, we will turn to proofs for multi-ElGamal ciphertexts, and show how to compute the proof with $3m + 1$ exponentiations for an m-key multi-ElGamal ciphertext. Eventually, using batching techniques, we will show how to compute a proof for ℓ multi-ElGamal ciphertexts with $(\ell+1)\cdot(m+1)$ exponentiations, bringing the cost of the proof down to almost 1 exponentiation per encrypted bit.

6.4.1. From 5 to 4 Exponentiations

Looking back at the disjunctive Chaum-Pedersen proof as it is described in Section 6.1, we can observe that the computation of $(A'_0, B'_0) = (g^s, g^{vw}h^s)$ and $(A'_1, B'_1) = (g^t, g^{(1-v)w}h^t)$ requires a total of 3 exponentiations when v is 0 or 1.

We observe that exponential ElGamal encryption works just as well, and may be slightly more efficient by saving one multiplication, if bits are encrypted as a (g^r, h^{v+r}) pair. Now, the commitment of the proof can be computed with a pair of ciphertexts $(A'_0, B'_0) = (g^s, h^{vw+s})$ and $(A'_1, B'_1) = (g^t, h^{(1-v)w+t})$, and the rest of the proof can remain unchanged. This saves 1 exponentiation in base g, taking the cost of computing a proof from 5 to 4 exponentiations.

6.4.2. A 0-1 product proof

The adaptation of the disjunctive Chaum-Pedersen proof to multi-ElGamal ciphertexts does not offer any particular benefit, unfortunately: one basically needs to compute one full proof for each $(A, B_j) = (g^r, h_j^{v_j+r})$ pair, keeping a cost of 4 exponentiations per encrypted bit.

In Table 6.2, we describe another proof approach, that consists in proving that $v_j(1 - v_j) = 0$, adapting a classical approach described in [GK15] for instance, and see that it makes it possible to take the

cost of the 0-1 proof for a multi-ElGamal ciphertext down to $3m + 1$ exponentiations.

Commitment The prover computes:
$A' = g^s$, $B'_j = h_j^{w_j+s}$, $C_j = g^{t_j}$, $D_j = h_j{}^{w_j v_j + t_j}$,
where $s, w_j, t_j \leftarrow\$ \mathbb{Z}_q$ and $j \in [m]$.

Challenge The verifier sends a challenge $e \leftarrow\$ \mathbb{Z}_q$.

Response The prover computes the response, for $j \in [m]$:

$$f_r = s + er\,, \qquad f_{v_j} = w_j + ev_j\,, \qquad f_{u_j} = t_j + r(e - f_{v_j})\,.$$

Verification The verification proceeds by checking that, for every $j \in [m]$:

$$A^e A' = g^{f_r}\,, \qquad A^{e-f_{v_j}} C_j = g^{f_{u_j}}\,,$$
$$B_j^e \bar{B}'_j = h_j^{f_{v_j}+f_r}\,, \qquad B_j^{e-f_{v_j}} D_j = h_j^{f_{u_j}}\,.$$

Table 6.2.: Proof of 0-1 encryption for multi-ElGamal ciphertexts.

6.4.3. Batching the product proof

Another advantage of the product proof is that it becomes compatible with batching techniques. To compress the proof further and save the computation of some exponentiations, and even more when we have ℓ multi-ElGamal ciphertexts, we consider a batching process. More precisely, when we have ℓ ciphertexts of the form $(A_i, \{B_{ij}\}_{j=1}^m) = (g^{r_i}, \{h_j^{v_{ij}+r_i}\}_{j=1}^m)$, for $i \in [\ell]$, we seek to prove that $\sum_{ij} v_{ij}(1 - v_{ij}) \cdot \alpha^{(i-1)m+j-1} = 0$, where $\alpha \in \mathbb{Z}_q$ is a random value. As long as α is independent of the statements to be proven, the Schwartz-Zippel lemma [Sch80; Zip79] implies that the above equation (seen as the evaluation of a polynomial at the random point α) ensures $v_{ij} \in \{0, 1\}$, for all $i \in [\ell]$ and $j \in [m]$, with overwhelming probability $1 - \ell m/q$.

Our protocol is in Table 6.3. In the commit phase, we provide the A' and B' elements from which we can extract all the r_i and v_{ij} exponents, as in our previous protocol. This makes it possible to isolate all these exponents from each other before starting the batching.

Now, the batching proceeds by picking a random exponent α after all the ciphertexts have been chosen and compressing the C and D terms of our previous proof: using α, the ℓm C_{ij} terms can be compressed

into a single group element C_0 and the ℓm D_{ij} terms can be compressed into m group elements, one per h_j base. With respect to ℓ parallel executions of the first protocol, we would have the relation $C_0 = \prod_{i\in[\ell],j\in[m]} C_{ij}^{\alpha^{(i-1)m+j-1}}$ and $D_{0j} = \prod_{i\in[\ell]} D_{ij}^{\alpha^{(i-1)m+j-1}}$.

Statement Given $A_i = g^{r_i}$, and $\{B_{ij} = h_j^{v_{ij}+r_i}\}_{j=1}^m$, for $i \in [\ell]$, the verifier generates and sends $\alpha \leftarrow\$ \mathbb{Z}_q$ to the prover.

Commitment The prover computes $A'_i = g^{s_i}$, $B'_{ij} = h_j^{w_{ij}+s_i}$ for proving openings,

$$C_0 = g^{\sum_{i\in[\ell],j\in[m]} t_{ij}\cdot\alpha^{(i-1)m+j-1}},$$
$$D_{0j} = h_j^{\sum_{i\in[\ell]}(v_{ij}w_{ij}+t_{ij})\cdot\alpha^{(i-1)m+j-1}}$$

for proving the relations, where $s_i, w_{ij}, t_{ij} \leftarrow\$ \mathbb{Z}_q$, for all $i \in [n], j \in [m]$.

Challenge The verifier sends a challenge $e \leftarrow\$ \mathbb{Z}_q$.

Response The prover computes for all $i \in [\ell], j \in [m]$:

$$f_{r_i} = er_i + s_i, \quad f_{v_{ij}} = ev_{ij} + w_{ij}, \quad f_{u_{ij}} = r_i(e - f_{v_{ij}}) + t_{ij}.$$

Verification The verification proceeds by checking that, for all $i \in [n], j \in [m]$:

$$A_i^e A'_i = g^{f_{r_i}}, \qquad B_{ij}^e B'_{ij} = h_j^{f_{v_{ij}}+f_{r_i}},$$
$$C_0 \prod_{i\in[n],j\in[m]} A_i^{(e-f_{v_{ij}})\cdot\alpha^{(i-1)m+j-1}} = g^{\sum_{i\in[n],j\in[m]} f_{u_{ij}}\cdot\alpha^{(i-1)m+j-1}},$$
$$D_{0j} \prod_{i\in[n]} B_{ij}^{(e-f_{v_{ij}})\cdot\alpha^{(i-1)m+j-1}} = h_j^{\sum_{i\in[n]} f_{u_{ij}}\cdot\alpha^{(i-1)m+j-1}}.$$

Table 6.3.: Batch proof of 0-1 encryption for multi-ElGamal ciphertexts.

It would be tempting to further compress our D_{0j}'s into a single D_0 as we do for C_0, but the special-soundness would then have to rely on the hardness of computing the discrete logarithms of the h_j's in base g. While the security of the encryption implies this hardness, the authorities that know the secret key would have the possibility to cheat when colluding with corrupted users/provers. This is the reason why we keep our m values.

We prove the special-soundness of our protocol below – the other standard properties of Σ-protocols come by inspection.

Theorem 22. *The protocol in Table 6.3 is special sound.*

Proof. From any two valid transcripts with identical α and commitments $(A'_i, \{B'_{ij}\}_{j=1}^m)_{i=1}^\ell, C_0, \{D_{0j}\}_{j=1}^m$, distinct challenges e and e', and responses $(f_{r_i}, \{f_{v_{ij}}\}_{j=1}^m, \{f_{u_{ij}}\}_{j=1}^m)_{i=1}^\ell$ and $(f'_{r_i}, \{f'_{v_{ij}}\}_{j=1}^m, \{f'_{u_{ij}}\}_{j=1}^m)_{i=1}^\ell$, we can extract, for all $i \in [\ell]$ and $j \in [m]$,

$$r_i = \frac{f_{r_i} - f'_{r_i}}{e - e'}, \qquad v_{ij} = \frac{f_{v_{ij}} - f'_{v_{ij}}}{e - e'}, \qquad u_{ij} = \frac{f_{u_{ij}} - f'_{u_{ij}}}{e - e'},$$

where $(r_i, \{v_{ij}\}_{j=1}^m)_{i=1}^\ell$ are all the exponents of the ciphertexts, since dividing the first two verification equations yields

$$A_i^{e-e'} = g^{f_{r_i} - f'_{r_i}} \qquad B_{ij}^{e-e'} = h_j^{(f_{v_{ij}} - f'_{v_{ij}}) + (f_{r_i} - f'_{r_i})},$$

and the encryption scheme is perfectly binding.

It remains to show that $v_{ij} \in \{0, 1\}$, for $i \in [\ell], j \in [m]$. If we divide the remaining verification equations of the two transcripts by the corresponding equations, and raise them all to the power $(e - e')^{-1} \bmod q$, then take the discrete logarithms in their respective base, we get:

$$\begin{aligned}
&\sum\nolimits_{i\in[\ell], j\in[m]} r_i(1 - v_{ij}) \cdot \alpha^{(i-1)\ell + j - 1} \\
&\qquad = \sum\nolimits_{i\in[\ell], j\in[m]} u_{ij} \cdot \alpha^{(i-1)\ell + j - 1}, \\
&\qquad = \sum\nolimits_{j\in[m]} \sum\nolimits_{i\in[\ell]} u_{ij} \cdot \alpha^{(i-1)\ell + j - 1}, \\
&\sum\nolimits_{i\in[\ell]} (v_{ij} + r_i)(1 - v_{ij}) \cdot \alpha^{(i-1)\ell + j - 1} \\
&\qquad = \sum\nolimits_{i\in[\ell]} u_{ij} \cdot \alpha^{(i-1)\ell + j - 1},
\end{aligned}$$

for all $j \in [m]$. By injecting the values of the right-hand side member of the last equations for $j \in [m]$ into the one above and injecting the value u_{ij} that we extracted previously, we find

$$\begin{aligned}
&\sum_{i\in[\ell], j\in[m]} r_i(1 - v_{ij}) \cdot \alpha^{(i-1)\ell + j - 1} \\
&\qquad = \sum_{i\in[\ell], j\in[m]} (v_{ij} + r_i)(1 - v_{ij}) \cdot \alpha^{(i-1)\ell + j - 1},
\end{aligned}$$

where the constants in front of the powers of α are uniquely determined by the statement. Since α was generated after the verifier received the statement, we have $r_i(1 - v_{ij}) = (v_{ij} + r_i)(1 - v_{ij})$, for all $i \in [\ell], j \in [m]$, due to the Schwartz-Zippel lemma, which implies that $0 = v_{ij}(1 - v_{ij}) \bmod q$. □

Efficiency. Computing the proof in Table 6.3 requires $(\ell + 1)(m + 1)$ exponentiations. The number of multiplications in the exponents increases to $4\ell m + \ell$ multiplications, but this will remain negligible in the group we consider where $|p| = 16|q|$. We can make a few more observations:

- If we need to compute $n = \ell \cdot m$ 0-1 proofs and if there is no precomputation, then picking $m \approx \ell$ is the best choice. In this case, when n is large, the cost of the proof comes close to 1 exponentiation per encrypted bit.

- However, when precomputation is used, and since the number of exponentiations in each base is well-balanced, the remarks made for the choice of m in the multi-ElGamal encryption still apply: we can expect only marginal benefits when increasing m above 4, for most values of n.

- Especially for the case $m = 4$, we need $\frac{5}{4}n + 5$ exponentiations to compute the proof and $\frac{5}{4}n$ exponentiations to compute the multi-ElGamal ciphertext. Together, we arrive at a practical cost close to 2.5 exponentiations per bit.

- While our basic product proof did not offer any benefit for a regular ElGamal ciphertext ($m = 1$), the batching process helps quite a bit in that case: we move from $4n$ exponentiations down to $2n + 2$.

6.5. Adapting the ZK 0-1 proofs – Logarithmic batching

With the advent of sublinear proof (or argument) techniques [Gro09; Bün+18; Ben+18; Bit+12], it is tempting to apply these approaches to our own problem of optimizing the time to compute many bits. We still need to be careful though, as a trusted setup is concerning in the context of verifiable elections. Furthermore, these proofs systems are usually designed to optimize another metric: the size of the proof, which includes a commitment of the statement, and the amount of computation to verify it.

Hence, a solution like Bulletproof [Bün+18], which relies on a public coin CRS, still needs a linear number of exponentiations (of the group elements) to compute the proof that has a logarithmic size, and it does not help us so much.

In our election usecase, we need to encrypt each bit individually and cannot compress these bits in a constant-size multi-Pedersen commitment,[24] as we need to be able to compute the tally of the election. Hence, the combination of the proof and the statement cannot be sublinear in our case, if we include the ElGamal encryptions of the bits.

In the paper [DPP22a] from which this chapter is written, we propose a new proof inspired from the folding technique of [Gro09] that needs only a logarithmic number of exponentiations of group elements at the prover's side. To do so, we take advantage of the fact that we have one commitment (the ElGamal encryption) for each bit, and manage to reduce the number of exponentiations by doing most of the work in the exponent group instead. Crucially, we need the encrypted bits to use the same ElGamal public key to run the log proof. This technique still works well with a multi-ElGamal ciphertext, since we can simply run m times the proof algorithm, once for each base.

Interestingly, we need to do only $4\log n + 4$ exponentiations in $\mathbb{Z}_p$ (using non-fixed bases) but $n\log n + \mathcal{O}(n)$ multiplications in the exponent group $\mathbb{Z}_q$. So the proof is technically asymptotically worse than the linear batching proposed in the previous section, but the multiplications in $\mathbb{Z}_q$ are considerably cheaper compared to exponentiations in $\mathbb{Z}_p$ so it is still very worthwhile to use this logarithmic proof. Even more, we can split the ciphertexts in several chunks and run a logarithmic proof for each of those chunks before computation time of the multiplications in $\mathbb{Z}_q$ takes over.

For example, if we assume that an exponentiation in $\mathbb{Z}_p$ is roughly 16 times more expensive than a multiplication in $\mathbb{Z}_q$ (using the fixed-base exponentiation for $k = 16$) and that a multiplication in $\mathbb{Z}_p$ is roughly 50 times more expensive than a multiplication in $\mathbb{Z}_q$ (using empirical evidence) for the considered group ($|p| = 4096$ and $|q| = 256$), then we can evaluate the cost of our proof to be roughly equivalent to $t(n) = 16 \times 50 \times 2\log n + n\log n$ multiplications in $\mathbb{Z}_q$ for large enough n. It becomes more optimal to do two logarithmic proofs for the first value n such that $t(n) > 2t(n/2)$, which happens at approximately 2×10^4 bits, which is larger than any practical numbers of bits in a ballot.

For the linear batch proof on the other hand, we can count $1.25n$ exponentiations in $\mathbb{Z}_p$ to compute the proof with linear batching for the number of multi-ElGamal bases set to $m = 4$, which corresponds to roughly $t'(n) = 16 \times 50 \times 1.25n \approx 1000n$ multiplications in $\mathbb{Z}_q$. So even if the linear batch proof is asymptotically better than the logarithmic proof,

[24] We could actually compute a multi-Pedersen commitment from ElGamal encryptions of the votes if we encode each vote on a different base and use the Bulletproof algorithm, but we would still need to pay the linear number of exponentiations.

	8 bits	32 bits	256 bits	1024 bits
Speed ratio	1.66	4	17.2	40
Size ratio	0.088	0.21	1.06	3.39

Table 6.4.: Comparison of the linear batching and the logarithmic proof techniques. The speed (resp. size) ratio is defined as the ratio of the computation time (resp. size) of the linear proof over the computation time (resp. size) of the logarithmic proof. For a small number of bits, the speed ratio is approximately the same regardless of the size of the fixed-based exponentiation table and the group that is used, as the exponentiation of the logarithmic proof dominates the cost.

it only becomes faster at around the tantalizing number of 2^{1000} encrypted bits. For a more reasonable number of 1024 bits, the logarithmic proof is around 40 times faster than the linear batch proof, and at this point the computation of the proof is negligible compared to the computation of the encryption of the bits. The speed up ratio of the logarithmic proof is given for several number of bits in Table 6.4.

Regarding the size of the proof, the voter needs to send $4 \log n$ elements in $\mathbb{Z}_p$ and $4 \log n$ elements in $\mathbb{Z}_q$ to the bulletin board. In comparison, the linear batch proofs need to send $2.25n$ elements of $\mathbb{Z}_q$ again for $m = 4$ and while the logarithmic proof is asymptotically better, it is actually now worse for small values of n. For a small ballot of $n = 32$ encrypted bits, the linear proofs are around 4.5 times smaller and the logarithmic ballots start to get smaller at around 240 bits. So interestingly, while the motivation of these logarithmic proofs was first to optimize the size of the proof, they are actually worse in this regard for practical values!

Overall, the total cost of the encryption and the proof is upper-bounded by $\frac{5}{4}n + 2m \log(\frac{n}{m}) = \frac{5}{4}n + 8 \log n - 16$ exponentiations (for $m = 4$), which is about two times faster than the multi-ElGamal encryption and the linear batch proof. However, these techniques are certainly more technical and depart heavily form the baseline protocol.

6.6. Benchmarking

We implemented all the algorithms described above in Python, and executed them on an AMD 3990X processor with the turbo boost technology disabled, using the Python 3.8.10 interpreter, and gmpy2

2.1.0. All the running times listed below are in milliseconds. While a Python implementation is unlikely to be deployed in practice (in favour of a JavaScript or WebAssembly implementation), these benchmarks offer a methodology that can be used to determine what are the best parameters for a given use case.

In our benchmarks, we focus on the time taken to compute a ballot from the voter perspective. We omit the time taken to verify the ballots, as this is typically much less constrained. The verification can be run in parallel on many devices for several days if needed, while the computation of a ballot is done on a possibly old and slow device owned by a voter who expects the computation to be done quickly. However for large elections, optimizing the verification time and the size of the audit data would help and is left open as future work.

6.6.1. Precomputation and exponentiations

We first tested the time needed to precompute and compute 1000 exponentiations in the ElectionGuard default group ($|p| = 4096$, $|q| = 256$).

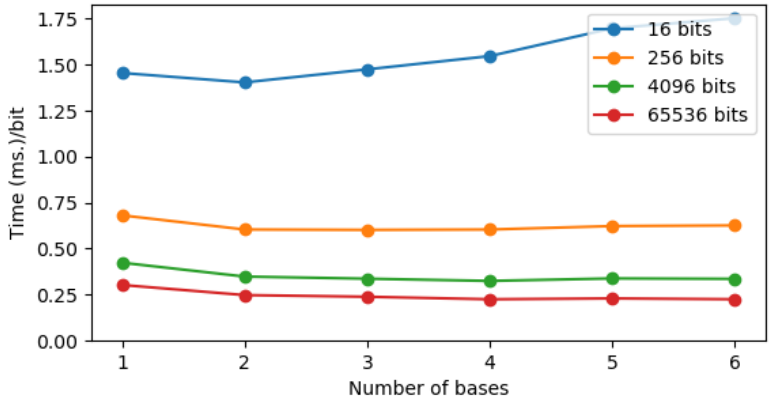
As a baseline, computing 1000 modular exponentiations with the library gmpy2.powmod takes 1560 ms. Figure 6.1.(a) shows the precomputation and computation time for various values of the precomputation parameter k. Already, the speed-up factors of the exponentiation, compared to gmpy2.powmod are quite important: they range from a factor 4 when $k = 4$ to a factor 12 when $k = 13$.

Depending on the application context, the precomputation time may need to be taken into account. This question is particularly interesting in the case of multi-ElGamal encryption where, for a given number of bits to verifiably encrypt, we may wonder which (k, m) pair leads to an optimal running time, as discussed in Section 6.3.

This is explored in Figure 6.1.(b), in which we show, for a number n of bits to be encrypted with $n \in \{2^4, 2^8, 2^{12}, 2^{16}\}$, the computation time/bit for $m \in [1, 6]$, selecting the optimal k every time. The lines are surprisingly flat: what is gained by multi-ElGamal is lost in the fixed-base exponentiations. The maximum benefits of multi-ElGamal are around 25%, much lower than the factor almost 2 that was hoped for, and this gain is reached around $m = 4$ for our 3 highest values of n. So, picking $m = 4$ independently of n might be a reasonable choice, should multi-ElGamal encryption be adopted.

k	4	8	13
Precomputation	6.1	51	1042
1000 exponentiations	387.6	204.2	131

(a)



(b)

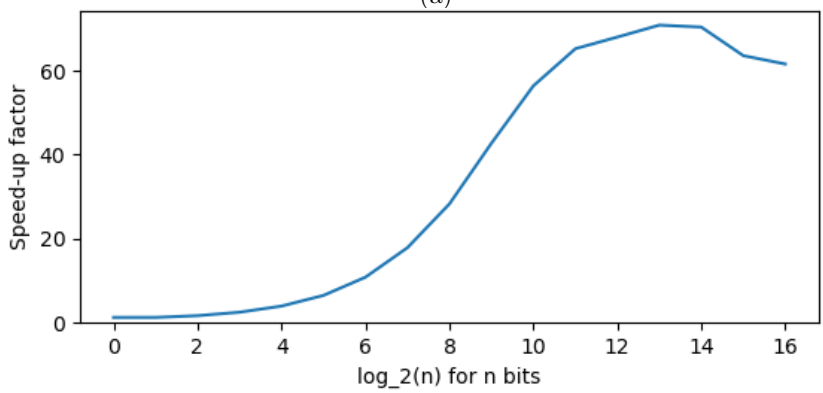
Figure 6.1.: (a) Precomputation time and time to compute 1000 exponentiations (in milliseconds per bit). (b): evolution of the time needed to encrypt votes with optimal choice of k and various choices of the number of multi-ElGamal bases.

6.6.2. Verifiable bit encryption using linear techniques

As a second step, we explore in Figure 6.2(a) the efficiency of our proof techniques with linear complexity. For the same values of n as above, a choice of $m = 4$ for the multi-ElGamal encryption, and optimal values of k, we explore the time needed for performing the precomputation, encryption, product proof (from Table 6.2), and batch proof (from Table 6.3). We can make a few observations from this figure: (i) As expected, the optimal k grows with n, and so does the time spent in precomputation. But the proportion of the time spent in precomputation decreases when n increases, which illustrates the decreasing returns of increasing k. (ii) While n is multiplied by a factor 16 from line to line, the computation time is only multiplied by a factor around 10, thanks to the amortization coming from an increased amount of precomputation. (iii) The cost of our batch proof is essentially equal to the cost of the multi-ElGamal encryption.

n	k	Pre.	Multi-ElGamal	Prod. Proof	Batch Proof
16	2	12	13	33	16
256	5	51	103	270	107
4096	7	150	1214	3187	1256
65536	11	1573	13022	34353	13605

(a)



(b)

Figure 6.2.: (a) Running time with linear techniques. (b) Speed-up given by the log proof for $n \in \{2^i | i \in [16]\}$.

6.6.3. Verifiable bit encryption using logarithmic proof techniques

Eventually, Figure 6.2(b) shows the speed-ups obtained in an implementation of our logarithmic proof of Section 6.5, compared to the baseline Chaum-Pedersen disjunctive proof (the batched proof is faster by a factor ≈ 2 when n is big enough). Both proofs take advantage of the fixed base precomputation, which is needed for the ElGamal encryption.

Here, and despite a higher asymptotic complexity due to the $n \log(n)$ multiplications needed in $\mathbb{Z}_q$, the log proof provides dramatic speed improvements, by a factor up to 70 when around 8000 proofs need to be computed, making the cost of the proof computation negligible compared to the one of encryption. The gains are already there for a small number of proofs: we obtain a speedup by a factor 2.5 for 8 proofs and a factor 18 for 128 proofs, about the size of a ballot. If one needs to compute a large number of 0-1 proofs, it may be more convenient and efficient to compute many logarithmic proofs by batches of a size between 2^{10} and 2^{14} for instance.

6.7. Concluding remarks

In this chapter, we proposed various techniques that can be used to increase the speed of verifiably encrypting bits, both at the arithmetic level (modular exponentiations) and at the protocol level (encryption and ZK proofs), compared to the usual implementation of the protocol of Cramer et al. [CGS97].

From all the techniques we proposed, fixed-based exponentiation is probably the most impactful and showed dramatic speed improvements. Switching to a multi-ElGamal encryption scheme is also a simple modification of the classic protocol which is potentially desirable, but the benefits diminish rapidly as the number of bases increases and it does not synergize well with the fixed-based exponentiations.

Regarding the zero-knowledge proofs, we managed to decrease the computational cost by using a product proof instead of a disjunctive proof and by taking advantage of the of the multi-ElGamal structure of the ciphertext. Combined with linear batching techniques, we managed to go down to about 2.5 exponentiations per bit to compute an encryption and the proof of validity. This is for 4 multi-ElGamal bases, which we empirically found to be a good choice.

It is also possible to take inspiration from the very recent development of fast and short proof systems, such as Bulletproof [Bün+18]. While these techniques aim to optimize the verification time and the length of the proof, we can adapt them to build proofs that need a sublinear number of exponentiations. While fast, these techniques are more complex, making implementation and audit more challenging. It is still good to know that such techniques exist, in case we need a particularly efficient voting tool.

Finally, there are still open directions in the optimization of the protocol of Cramer et al. If the efficiency of ballot casting is not the bottleneck anymore, we could look at optimizing the verification time (using the aforementioned fast proof techniques) or aim to reduce the size of the bulletin board. Indeed in large elections, bulletin boards can be excessively large and deter auditors from downloading the data, which might not even fit in the RAM of the auditor's device. Designing a more compact bulletin board would thus be of great interest.

7. Secure Postal Voting

So far in this thesis, we have considered the use of electronic voting protocols. Thanks to the computational nature of these protocols, it is feasible to perform complex cryptographic operations that would not be realizable by a human voter by hand. However, the voter often does not have convincing guarantees that their ballot is correctly computed and that a malicious voting device did alter their vote. This notion is called the cast-as-intended property and is notoriously hard to achieve.

In contrast to electronic voting, postal voting (or vote-by-mail) is another approach to let remote voters participate in an election. Even more, this approach is gaining in popularity: it is offered to all the voters of several European countries, including Germany where its adoption grew from 29% in 2017 to 47% in the Bundestag Election of 2021,[25] or Switzerland where more than 90% of the ballots are mailed according to Swiss Post,[26] and is also increasingly used in the rest of the world, including by 43% of the voters during the 2020 U.S. presidential election.[27] This adoption clearly dwarfs the use of Internet voting, which is often considered as out-of-reach of current security technologies and has only been consistently offered to the general public in Estonia.

One of the main advantages of postal voting is its simplicity. To cast a ballot, a voter can simply hand-mark it, which solves the problem of cast-as-intended verifiability. On the other hand, it is more difficult to ensure that a ballot has been correctly received untampered by the election officials. Indeed, using a public bulletin board is arguably more difficult since the ballot is not encrypted. Another common issue with postal voting is the vote secrecy, since anyone holding the ballot can open the envelope and read its contents. This is additionally problematic as there is often a link to the voter (such as a signature) in the ballot for eligibility reasons. Making matters even worse, those two issues are amplified by the fact that the ballots are handled by postal services

[25] https://www.bundeswahlleiterin.de/en/info/presse/mitteilungen/bundestagswahl-2021/53_21_briefwahlbeteiligung.html

[26] https://www.post.ch/fr/notre-profil/actualites/2023/comment-la-suisse-est-devenue-le-pays-du-vote-postal

[27] https://nap.nationalacademies.org/catalog/25120/securing-the-vote-protecting-american-democracy

(including foreign ones when voting abroad) for a long period of time (usually several days).

Despite this current popularity and its associated risks, the existing postal voting protocols remain quite simple. Very few proposals have been made to offer verifiable vote-by-mail solutions, and none of them come with a systematic security analysis.

In this chapter, we make a first step to fill this gap. We propose a new verifiable postal voting protocol, and formally analyze its security in a new model instantiated in a variant of the Universally Composable framework. Notably, our protocol was primarily designed to be compatible with Belgian elections and originated from a study requested by the federal election office in Belgium.[28]

Structure of the chapter The chapter starts with a brief review of the already existing postal voting protocols in Section 7.1. Then, Section 7.2 introduces the structure of our postal voting protocol. Section 7.3 offers a high-level description of our voting protocol, followed by a full description in Section 7.4. We finally introduce our postal voting ideal functionality and prove that our protocol emulates that functionality in Section 7.5.

7.1. Related works

A large proportion of the (often partially) verifiable voting systems that use the mail are not actually vote-by-mail: voters receive a ballot by mail, which typically contains various codes, and the voters use these codes to cast their ballot through the Internet. These systems include the original code voting proposal by Chaum [Cha01] and many follow-up works, including some that have been actually deployed in government elections, including Remotegrity [Zag+13], the Norwegian voting system [Gjø11] and the Swiss one [Swi23].

In 2013, Benaloh et al. proposed a first verifiable postal voting protocol [BRT13]. In their protocol, voters access and complete their ballot on their computer, which then prints a human readable version of the vote together with an encrypted and signed version of it. That printed ballot is then submitted by mail. Because of the presence of the signature, anyone opening an envelope may be able to violate the ballot privacy. It also does not offer protection against a malicious device and a malicious voting channel: the voting device could print a human-readable vote that matches the voter intent and encrypt and sign a different vote intent.

[28] `https://elections.fgov.be/informations-generales/etude-sur-la-possibilite-dintroduire-le-vote-internet-en-belgique/`

A mail adversary can then change the human-readable part and keep the signed encrypted record, so that no auditing step would make it possible to detect the inconsistency. A protocol that shares these same features was proposed recently proposed by McMurtry et al.[McM+21], which offers a higher probability of detecting vote manipulations and slightly better receipt-freeness. By comparison, our protocol supports the submission of ballots that do not contain any identity, guarantees privacy if the voting device is corrupted and can still guarantee integrity when both the voting device and the postal channel are corrupted.

In a different approach, STROBE [Ben21] offers en elegant solution to the vote-by-mail problem, based on the same cryptographic primitives that we are using. A central benefit of STROBE is its compatibility with an E2E verifiable voting process that could be used by voters voting in-person in the same election for instance. An important constraint of STROBE is its requirement that authorities produce two ciphertexts for each voter encrypting and proving the validity of every possible voter choice, which may quickly become prohibitive depending on the ballot style. A malicious authority is also only detected with probability at most one half, while our design can easily make it much larger, typically above $1 - 2^{-10}$. The STROBE design has been further refined in RemoteVote and SafeVote [CRH22]. These protocols support a much broader class of ballot styles, but also bring additional complexities: all the voters need to interact with a server as part of the voting process in order to decide how a spoiling process needs to be executed. Besides, as in STROBE, the verification of spoiled ballots requires the use of an honest device to perform heavy (public) cryptographic operations. In contrast, all the voter verification steps of our protocol can mainly be performed "in the head": they are just string comparisons, except for a single hash value of a random token printed on the empty ballot. This printed token can be only 20-character long with 6-bit encoding characters and is independent of the votes.

7.2. Postal Voting Model

7.2.1. Overview

We consider a remote election in a vote-by-mail setting. The election is overseen by a party called the elections authorities. This party sets the parameters of the election p, which consists among other things of the list of valid voters, the list of candidates for which one can vote, and a list of talliers. These parameters are given to each participant at the beginning of the election.

Ballots are generated by an independent server. When a voter asks the server for a ballot, the voter authenticates (with an electronic ID for example), and receives a blank ballot if they have not received any ballot yet.

The voter can then print the ballot, mark it by hand, and send it by mail to a postal voting election office. There might be additional materials associated to the ballot that should not be sent. That includes resources that could be used in a vote verification procedure. Alternatively, the voter can also fill the ballot electronically before printing it, for usability reasons for example. However, marking the ballot by hand is better privacy-wise, as the voter does not need to trust the computing and printing devices, in a context where personal printers in homes are becoming incresingly rare.

During the tallying phase, the election office interacts with the talliers to compute the result of the election that can be published once every ballot has been counted. Moreover, the election office interacts with a verification server that can be used by voters to verify that their votes have been correctly counted as intended without any loss of vote privacy.

7.2.2. Parties

The proposed election system includes the following independent parties:

EA: The election authorities set up and manage the execution of the election. Other parties report to the election authorities if they notice an irregular behavior in the protocol.

BI: The ballots issuer server generates fresh ballots and issues them to voters after proper identification.

ID: The voters participate in the election with identities $\mathsf{ID} = \{\mathsf{id}_1, \ldots, \mathsf{id}_l\}$. Each voter can request a fresh ballot from BI, ballot that they can print, mark with their vote, and send by mail to the election office. They can later check that their ballots have been correctly recorded by interacting with the verification server VS.

CB: The ballot collecting box collects the envelopes of the ballots sent by mail. It shuffles these envelopes, making them anonymous, before delivering them to the election office EO.

EO: The election office counts the votes of the election and computes the result of the election in collaboration with the talliers. It also interacts with the verification server VS in order to make the auditing data available.

T: The talliers $\mathsf{T} = \{\mathsf{T}_1, \ldots, \mathsf{T}_n\}$ compute the result of the election, jointly with EO.

VS: The verification server provides the voters with the data they need to verify that their votes have been correctly recorded and tallied.

All the parties but the voters are together called the *official parties*.

7.2.3. Postal Voting Protocol

Definition 32 (Postal Voting Protocol)**.** *A* Postal Voting Protocol $\mathcal{V} = (\mathsf{Setup}, \mathsf{Vote}, \mathsf{Tally}, \mathsf{VerifyVote})$ *is a tuple of protocols:*

1. Setup is a distributed protocol initiated by EA and jointly executed by the official parties, with as input the election parameters p that contain the list of voters, candidates, and talliers.

2. Vote is a protocol executed by the voters. Given a blank ballot b°, the auxiliary information aux, and the vote intention of the voter, it returns an interpretable ballot $\mathsf{b}^\bullet$ encoding the vote intention and a receipt rcpt used in the verification process.

3. Tally is a protocol executed by the election office EO and the talliers after having received the ballots. At the end of this protocol, each tallier and the election office sends to EA the result of the election r and sends the proofs of individual verifiability Π to the verification server.

4. VerifyVote is a protocol executed by the verification server VS and voters willing to verify their votes. At the end of the protocol, the voters output honest if they are convinced that their votes have been correctly counted and cheat otherwise. In that case, they report the inconsistency to the election authorities.

We do not discuss formally what we expect from those algorithms yet. This will be captured in the security analysis through an ideal functionality.

During the execution of the protocol, parties can notice inconsistencies that should not occur during an honest execution of the protocol. In that case, they can report a complaint by sending a message cheat to the election authorities EA.

Voters send their completed ballots $\mathsf{b}^\bullet$ by mail. This postal channel is unreliable and asynchronous, hence ballots could be modified or never arrive at CB. Since CB sends the ballots to EO at some point, ballots

received after the tally has started are discarded without even being ever opened. We consider that all the other communication channels between the parties are secure, that is they are confidential and authentic. In particular the set of identities $\mathsf{ID} = \{\mathsf{id}_1, \ldots, \mathsf{id}_l\}$ is known by all the official parties, and communications related to any voter's identity id cannot be misidentified. If the interface that shows b° to voters is corrupt, for simplicity we assume that BI is corrupt, and the same for VS in VerifyVote.

7.3. Protocol overview

To help get some intuition about our protocol, we first give an overview of the different parts of the process. The protocol is designed for approval voting elections. By approval voting, we mean that given a list of candidates or questions, a voter can choose to approve any subset of these candidates or questions.

7.3.1. Design choice

Our choices within a relatively large design space are motivated by various practical considerations that were developed as part of a study requested by the federal election office in Belgium.

Hand-marked paper ballots We wish to keep intact the benefit of cast-as-intended verifiability offered by vote-by-mail. As such, we seek for a design in which voters can hand-mark their ballot and submit them in a way that is similar to traditional vote-by-mail experiences. Our scheme remains compatible with the use of ballot marking devices, but this is not a requirement. This allows side-stepping the pitfalls related to the (lack of) auditability of ballot marking devices [AWA20; Ber+20]. This is also beneficial for privacy as no potentially corrupted device linked to a voter can learn the content of a vote, which may become more important as private printers are also becoming less common.

Internet ballot delivery and verification Even if the vote goes by mail, there are steps that can take advantage of the Internet. In our case, voters download blank ballots from the internet and also use the internet to verify that their votes have been recorded without being altered. Internet ballot delivery has the advantage over mail delivery that it saves the physical ballots half of their trip – only the return is needed. The delays incurred by postal services often make it impossible for voters to receive and return their ballot before the closing of the polls, especially for voters living abroad and on different continents[Fau+18]. This choice may also support stronger forms of voter authentication in

places where online authentication mechanisms are available – which may be safer than accessing a physical mailbox as a means of authentication, assuming that voters have one. Still, our system is also compatible with ballot delivery by mail if it happens to be a good option for external reasons.

Distributed trust for integrity We do not trust the postal service for integrity. Of course, we cannot guarantee that it will not alter or suppress ballots. But our system makes it observable if a ballot is not delivered or if it is altered.

Furthermore, even in collusion with the postal service, no single entity can break the integrity of the election. Still, integrity can be broken by a coalition of other actors.

This integrity model may appear to be weaker than what is demanded by some definitions of end-to-end verifiability (see discussions in [Ben21]), which require that no collusion of parties should be able to violate the integrity of the election result without a voter (or an observer) being able to detect it. Here, we would argue that such a requirement cannot be satisfied in most practical cases and that, as such, our integrity model does not substantially weaken the best form of verifiability that we can hope for. Indeed, in most government elections, lists of voters are not public,[29][30] [31] which makes it impossible to detect integrity violations resulting from ballot stuffing – election officials and observers with specific access rights need to be trusted here. Similarly, in large scale paper elections, voters and observers must trust the ballot box chain-of-custody. It may still be a design goal to make the system as end-to-end verifiable as possible. But, as we will see below, by accepting that trust needs to be partially delegated, our approach leads to a system that is notably simpler than other solutions that offer a better compatibility with E2E-verifiability, which may be a crucial quality for a correct deployment.

Distributed trust for privacy, except for the post Apart from the postal service, no single party can break ballot privacy alone. The trust in the postal service for privacy seems unavoidable when ballots are mailed: depending on the context, the postal service may be able to identify who sends a ballot (even if our ballots do not contain any identifying mark, the post office may see who deposits an envelope), and the postal service would likely be able to carefully open an envelope, read its content, then close it (or place the ballot in a new envelope).

[29] https://www.service-public.fr/particuliers/vosdroits/F1367?lang=en

[30] https://www.bundeswahlleiterin.de/en/service/glossar/w/waehlerverzeichnis.html

[31] https://www.elections.ca/content.aspx?dir=reg%2Fdes&document=index&lang=e&section=vot

However, such a link is unlikely to be available for a return office that receives anonymous envelopes.

Limited coercion resistance Remote voting has inherent limitations regarding coercion: if a coercer can obtain a voter's ballot, then it can just submit it on its own. Still, a voter who follows our voting instructions gets no useful receipt. And, the verification process is also designed in such a way that, if the voter is instructed by an external coercer to send information after submitting a vote, the voter can do so in such a way that all the verification steps will be consistent with the coercer requirements. In other words, the verifiability of our system does not worsen its limited coercion resistance compared to traditional vote by mail – even though the coercion resistance strategy may require a bit more work. This differs from various Internet voting systems where a voter who keeps the random coins used to encrypt a ballot is then able to prove how they voted to a third party thanks to the election bulletin board.

Classical cryptographic primitives Verifiable voting protocols may quickly become quite sophisticated and gather a variety of complex cryptographic primitives that have never been standardized and for which well tested and maintained implementations do not exist. This has been the source of numerous failures in the deployment of Internet voting systems in the past. As such, we aimed at designing our protocol in such a way that it only requires relatively well-known primitives for which various tested implementations exist: on top of TLS, all our primitives are implemented in systems like Helios [Adi+09], Belenios [CGG19], and in the ElectionGuard open source SDK [Ele24].

7.3.2. Ballot structure

During the setup of the election, the ballot issuer BI crafts all the ballots. These ballots consist in three different components that can be printed on paper.

The first component is a simple list of the candidates of the race with a box for each candidate that can be checked to show the approval (see Figure 7.1). This sheet will be marked by the voter with their voting intentions and sent to CB. This sheet is also designed to be easily scanned to tally the result electronically by EO.

Moreover, there is an unique voting token tk included at the bottom of the selection sheet, which is a relatively short sequence of characters. This voting token is used to identify the ballot and to ensure that the ballot is tied to a real voter who authenticated to the ballot issuer server.

The second component is the code sheet. This component is similar to

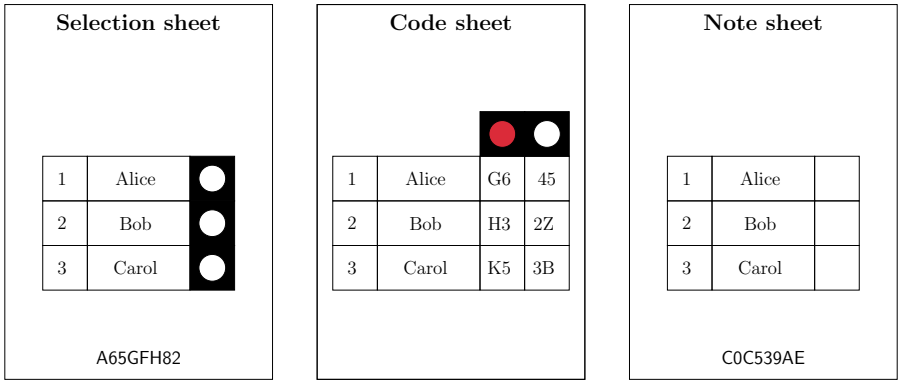


Figure 7.1.: Example of a blank ballot.

the first component with the following modification. The box that can be checked next to each candidate is replaced by two codes corresponding to the two possible choices. These codes can be printed on the ballot as a sequence of alphanumeric symbols.

The third and final component is the note sheet. This is a blank list without the names of the candidates. It contains $H(\mathsf{tk})$ at the bottom of the sheet. During the casting process, the voter is expected to write the codes associated to their choice in this note sheet. The hash of the voting token, is also written on this note sheet.

7.3.3. Voting procedure

The voting procedure of a voter is quite intuitive. Given a paper ballot as described above, a voter selects the candidates they approve by checking the corresponding boxes and copies the chosen codes of the code sheet in the note sheet. The voter can check that tk and $H(\mathsf{tk})$ match, for instance, by taking a picture of tk with a dedicated mobile application that returns hash values, or any other means. They then destroy the code sheet and send the selection sheet by mail to the ballot collecting box. At a later stage, it will be possible for the voter to verify that their ballot has been correctly counted using these codes.

An interesting and desirable feature of this paper voting protocol is that the ballots are interpretable. That is, a voter can recover their vote from the ballot without any other information and does not have to trust a mechanism hiding their vote, like in most onsite elections.

7.3.4. Tallying procedure

After delivery by postal service, the ballot collecting box receives all the envelopes. These envelopes are open and shuffled, then the election office EO proceeds to the tally. Since the ballots are interpretable, the election office can directly compute the result.

To prove to the voter that their vote has been counted correctly, the election office EO and the talliers $\mathsf{T}_1, \ldots, \mathsf{T}_n$ proceed to compute the codes of the voter's selection. These codes have been encrypted by the ballot issuer using a threshold encryption scheme at the ballot generation.

The election office then sends to the talliers the selections of the ballot for each voting token. These talliers, who possess the shares of the private key of the threshold encryption scheme, then compute together the decryption of the code corresponding to the voter's choice. To compute these codes, they all have to agree on the choice of the voter and then on the result of the election. These codes are finally sent to the verification server which will interact with the voters.

7.3.5. Verification procedure

The procedure is also quite intuitive: The voter sends the hash of their voting token to the verification server and receives a code for each option. If these codes match the codes they have written on the note sheet, then the voter has good evidence that their vote has been correctly counted.

A malicious election office could always try and guess the codes of a voter while modifying their vote. We can control the probability of success of this adversary with the length of these codes: a longer code will be less likely to be guessed, at the cost of usability. 16 bits seems to be a good trade-off between security and usability.

7.4. Protocol

In this section, we present the formal description of the postal voting protocols. We use a Python-like notation for sets and dictionaries. We also define $\mathsf{list}(\mathsf{n})$ as a list of n entries and $\mathsf{list}(\mathsf{n}, \mathsf{m})$ as a list of n lists, each of them containing m entries. Those entries are initialized with the value $\perp$. We extract the i-th bit of an integer x as $\mathsf{x}.\mathsf{bits}(\mathsf{i})$. We recall that all the channels are confidential and authentic, except the postal channel, and that the voters are identified by $\mathsf{ID} = \{\mathsf{id}_1, \ldots, \mathsf{id}_l\}$.

7.4.1. Building blocks

Before describing the protocols, we introduce the building blocks that we are going to use. We designed the protocol to use as simple cryptographic primitives as possible. Hence, we only need

- An hash function H modelled as a random oracle.
- A threshold encrypted scheme (as defined in Definition 2).

Regarding the threshold encryption scheme, we replace the centralized algorithm TGen by a distributed protocol Π_{TGen} realizing TGen in a generic way (through a MPC protocol). At the end of this protocol, each party T_i derives their private key sk_i. Since the encryption and the decryption simulator algorithms are non-interactive, all the above notions still hold in our system when a malicious adversary participates in the key generation. Finally, we stress that the encryption algorithm does not have to come with any additional verifiability notions. Therefore, we can use a plain ElGamal encryption scheme whose decryption can be computed by a threshold following the TDH constructions [SG02]. This gives the decryption simulatability, but we keep the CPA encryption algorithm (CCA-security is useless here). This centralized threshold encryption scheme can be decentralized by following [Gen+07] (based on [Ped91]), for instance. We note that decryption share validity is enforced in [SG02] thanks to a non-interactive Chaum-Perdersen zero-knowledge proof of language membership (in the random oracle model). Such proofs allow determining which decryption shares to use in $\mathsf{TCombine}$.

We stress that even if the encryption and the decryption are very efficient (only Π_{TGen} is costly, but it is run prior to the election), voters will never have to compute any of them. Given a blank ballot, the only cryptographic function that they could verify in our voting system is the computation of a single hash value.

7.4.2. Setup procedure

In our protocol, the setup procedure serves two purposes: the election key generation procedure and the ballots generations. More formally, Setup is a pair of algorithms $(\mathsf{KeyGen}, \mathsf{GenerateBallot})$ that takes as input the parameters of the election p. p consists in $(1^\lambda, m, n, t, \mathsf{ID}, \mathsf{p_limit})$, where λ is the security parameter, m and n are respectively the number of candidates (or questions) and talliers and t is the minimum number of talliers that have to be honest for the protocol to be verifiable. Regarding $\mathsf{p_limit}$, a voter in our protocol might believe that their vote is correctly counted while this is not the case. This is because the protocol is based

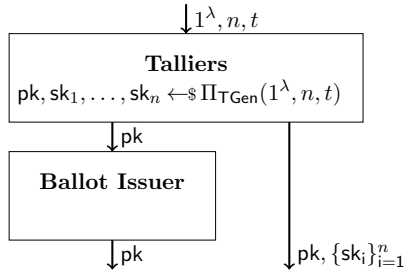


Figure 7.2.: Key Generation procedure.

on a return code which gives confidence to the voter that their vote has been cast correctly if they receive in return the right code. An adversary could always try to guess this code and convince the voter in this way. We thus also include an upper bound on the probability of such a failure in the verification process, defined as $\mathsf{p_limit}$.

At the end of the setup phase, the ballot issuer BI has generated for each voter the blank ballot and the auxiliary information needed to vote. The trustees have the secret key material that they will keep to make sure that their contribution is necessary for the ballot verification process to succeed. The election office EO and the verification server VS both also receive auxiliary information for the ballot verification process.

Election key generation (Figure 7.2)

The election key generation protocol $\mathsf{KeyGen}(1^\lambda, n, t)$ is run by the talliers under the supervision of EA. It takes as input the number n of talliers and threshold number t of talliers. It outputs the public key pk of the election (and all the validity proofs) to all the involved parties, and EA sends it to BI. Moreover, this protocol also results in each tallier T_i creating and keeping their partial private key sk_i.

The protocol KeyGen is described in Figure 7.2. Essentially, the talliers together run the secure distributed protocol Π_{TGen} of the threshold encryption scheme.

Ballots generation (Figure 7.3)

We define the ballot generation algorithm $\mathsf{GenerateBallot}(\lambda, \mathsf{pk}, m, \mathsf{ID}, \mathsf{p_limit})$ run by the ballot issuer BI. It takes as input the election key pk computed by KeyGen, the number of candidates m, and the $\mathsf{p_limit}$ parameter. The algorithm is described in Figure 7.3 in the box of the ballot issuer. It outputs a blank ballot $\mathsf{b}^\circ_{\mathsf{id}}$ and auxiliary information $\mathsf{aux}_{\mathsf{id}}$ for each $\mathsf{id} \in \mathsf{ID}$. It also outputs to every tallier, the election office,

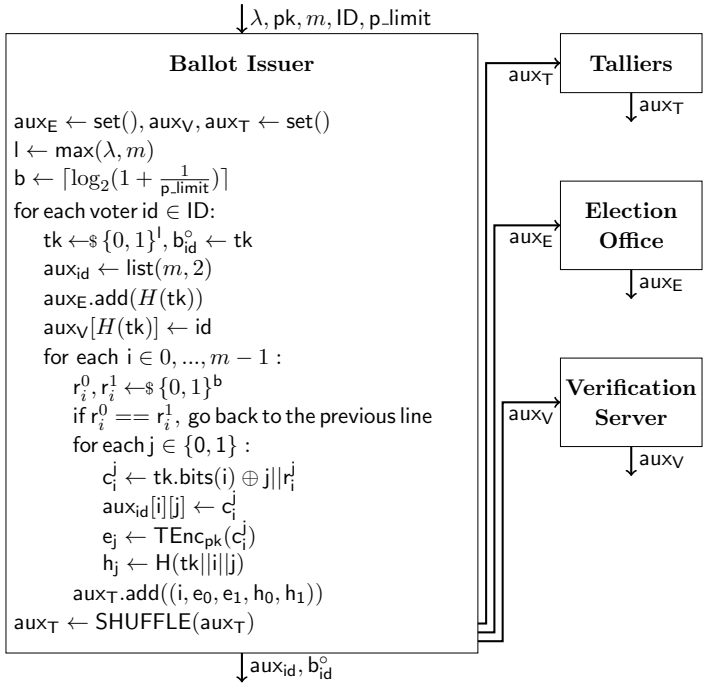


Figure 7.3.: Ballot Generation procedure.

and the verification server respectively the values $\mathsf{aux_T}$, $\mathsf{aux_E}$ and $\mathsf{aux_V}$ that will be defined next.

Essentially, BI generates for each voter a random voting token tk which constitutes b°. Voters have different voting tokens with overwhelming probability. Hashes of those tokens are sent to the election offices EO, which constitutes $\mathsf{aux_E}$. EO will later check if a voting token is valid by checking if its hash is in $\mathsf{aux_E}$.

BI also generates additional information $\mathsf{b}^\circ_{\mathsf{id}}$ for the voter with identity $\mathsf{id} \in \mathsf{ID}$. For each possible choice of the candidates, BI generates two different codes c^0 and c^1. Those codes are such that their first bits are distinct and their remaining trailing bit-strings are distinct as well. The first bit of the code c^j is actually computed as follows: $\mathsf{tk}[\mathsf{i}] \oplus \mathsf{j}$. This link between the voting token and the codes is a protection against a corrupt BI. Those codes form the auxiliary information of the voter $\mathsf{aux_{id}}$ and are only needed for the verification procedure.

The length of trailing bits of the codes is $\mathsf{b} = \lceil \log_2(1 + \frac{1}{\mathsf{p_limit}}) \rceil$. Given one code, we want the probability for an adversary to guess the other code to be bounded by $\mathsf{p_limit}$. This is because if the adversary modifies the choice of one voter, it will receive the code of this other choice and

will have to guess the other code to convince the voter that their vote has been correctly counted, even if it is false. Since the trailing bits r^0 and r^1 of the codes are independent up to their inequality, we have that:

$$\mathsf{Pr}[\mathsf{r}^0 = \mathsf{r}] = \begin{cases} 0 & \mathsf{r}^1 = \mathsf{r} \\ \frac{1}{2^{\mathsf{b}}-1} & \mathsf{r}^1 \neq \mathsf{r} \end{cases}$$

for any string r of b bits. Our choice of b lead to this bound of $\mathsf{p_limit}$. Moreover, as the codes are independent for different choices (i.e., candidates), a cheating adversary who guesses k codes will not be caught with a probability bounded by $\mathsf{p_limit}^{\mathsf{k}}$.

Furthermore, BI generates a way for the talliers to retrieve the codes of the voter's choice. To do so, for each ballot it generates an encryption $\mathsf{e}_\mathsf{i}^\mathsf{j}$ of the code $\mathsf{c}_\mathsf{i}^\mathsf{j}$ using the public key pk of the threshold encryption scheme and creates the tuple $(\mathsf{i}, \mathsf{e}_\mathsf{i}^0, \mathsf{e}_\mathsf{i}^1, \mathsf{H}(\mathsf{tk}||\mathsf{i}||0), \mathsf{H}(\mathsf{tk}||\mathsf{i}||1))$. Instead of including tk directly in the tuple, we use $\mathsf{H}(\mathsf{tk}||\mathsf{i}||\mathsf{j})$ to hide voting patterns. In this way, the talliers cannot link two votes from the same ballot. Moreover they will be able to detect if EO will ask them to decrypt both codes on a tuple, which is forbidden. Those tuples are shuffled and define the auxiliary information of the talliers aux_T.

Finally, VS receives from BI the link between the voters identity id and the hash $H(\mathsf{tk}_\mathsf{id})$ of their voting token, which constitutes aux_V. (Jumping ahead, VS will later receive from EO the decrypted codes associated with the voter's choice. EO will get these codes from the talliers and know the choices on each counted ballot, but not the link to the voter identities. In contrast, VS knows the voter identities linked to $H(\mathsf{tk}_\mathsf{id})$ and will get the codes for verification, but not the choices.)

7.4.3. Voting Procedure (Figure 7.4)

In the voting procedure, a voter first obtains a ballot by authenticating and proving their identity to BI. They then run the $\mathsf{Vote}(\mathsf{v}, \mathsf{b}^\circ, \mathsf{aux})$ algorithm, which takes as input the voting intention v, a blank ballot b° and some auxiliary information aux. In our approval voting case, v is a list of m bits. This algorithm outputs a completed ballot $\mathsf{b}^\bullet$ and a receipt rcpt. To cast a vote, the voter simply sends $\mathsf{b}^\bullet$ to the collecting box.

The Vote algorithm is described in Figure 7.4 in the box of the voter. Essentially, it proceeds as follows. b° is first parsed as a voting token tk and auxiliary information aux which consists in the codes $\{\mathsf{c}_\mathsf{i}^\mathsf{j}\}_{\mathsf{i}\in[m]}^{\mathsf{j}\in\{0,1\}}$ for each choice i and each choice j. Given those codes, the voter verifies their validity. That is, the voter should check that the first bit of $\mathsf{c}_\mathsf{i}^\mathsf{j}$ is

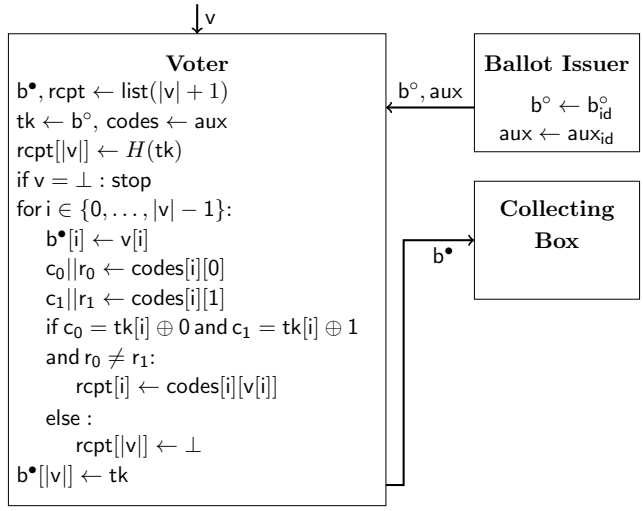


Figure 7.4.: Voting procedure.

$\mathsf{tk}[\mathsf{i}] \oplus \mathsf{j}$ and that for the same candidate, the trailing bits of c_i^0 and c_i^1 are different strings. If this verification fails, the voter does not have any guarantee of verifiability and should report to the election office. It also checks tk upon $H(\mathsf{tk})$.

$\mathsf{b}^\bullet$ is then computed as the concatenation of the voting token tk and the list of choices v, where v_i is equal to 1 if the voter approves candidate/question i and 0 otherwise. The voter also keeps a receipt rcpt consisting of $\mathsf{H}(\mathsf{tk})$ (pre-computed on the selection sheet and the note sheet as depicted in Figure 7.1, but this can be verified) and the codes $[\mathsf{c}_0^{\mathsf{v}_0}, \ldots, \mathsf{c}_{m-1}^{\mathsf{v}_{m-1}}]$ of the selected choices (that can be written on the note sheet). The voter then destroys the unused codes (by destroying the code sheet illustrated in Figure 7.1) and erases any local records of tk. $\mathsf{b}^\bullet$ is sent to CB through the postal channel (without any additional information about the sender).

7.4.4. Tallying procedure (Figure 7.5)

After the ballot collecting deadline, CB shuffles the received envelopes, opens them and transfers their contents to EO. Now that all the involved parties have their inputs, the actual tally protocol can start.

The election office EO and the talliers $\mathsf{T}_1, \ldots, \mathsf{T}_n$ engage in the protocol $\mathsf{Tally}(\mathsf{ballots}, \mathsf{aux}_\mathsf{E}, \mathsf{aux}_\mathsf{T}, \mathsf{pk}, \{\mathsf{sk}_\mathsf{i}\}_{\mathsf{i}=1}^n, m, t)$. EO takes as inputs the sets of ballots $\mathsf{ballots}$ received from CB, the election key pk, the auxiliary information aux_E received from the ballot issuer BI, the number of candidates or questions m, and the threshold number of talliers t. Each

tallier $\mathsf{T_j}$ takes as input the auxiliary information $\mathsf{aux_T}$ received from BI, the election public key pk, and their own private key $\mathsf{sk_j}$. At the end of the protocol, EO outputs the result r of the election as well as the individual proofs of verifiability Π that contain the verified decrypted codes. $\mathsf{T_j}$ outputs the result $\mathsf{r_j}$ of the election.

The full tally procedure is described in Figure 7.5. The boxes of the tallier $\mathsf{T_j}$ and EO, and the interactions between them consist of the Tally protocol that we comment on in the next paragraphs. The results of the election $\mathsf{r}, \mathsf{r_1}, \ldots, \mathsf{r}_n$ respectively output by $\mathsf{EO}, \mathsf{T_1}, \ldots, \mathsf{T}_n$ are sent to EA through their own secure channels. EA verifies whether at least t of these $\mathsf{r_j}$ are consistent with r, and, if it is case, outputs r, and $\perp$ otherwise. Finally, EO sends the individual proofs of verifiability Π to VS that will be useful for the verification procedure.

Now, we turn back to the Tally protocol and provide additional explanation about how it works. First, EO computes the result of the election from all the ballots $\mathsf{b}^\bullet$ in $\mathsf{ballots}$. If the token of a ballot appears on another ballot or if the hash of the (printed) token in $\mathsf{b}^\bullet$ does not correspond to any entry of $\mathsf{aux_E}$, then the ballot is discarded. Note that this can be done electronically by scanning the ballots.

Afterward, EO interacts with the talliers to produce the proofs for the voters that their votes have been correctly counted. To do so, EO computes for each (non-discarded) ballot m values $\{\mathsf{H}(\mathsf{tk}||\mathsf{i}||\mathsf{v_i})\}_1^\mathsf{m}$ from the voting token tk, each choice i, and the vote $\mathsf{v_i} \in \{0, 1\}$ for this choice written on this ballot. All the elements of all these lists of values are then shuffled and sent to the talliers. As discussed before, this shuffling helps to hide the voting patterns to the talliers. Before computing the result, the talliers verify with each other that they have received the same list from the election office. Otherwise, they abort the election, and each $\mathsf{T_j}$ outputs $\mathsf{r_j} = \perp$.

For each of those $\mathsf{h} = \mathsf{H}(\mathsf{tk}||\mathsf{i}||\mathsf{v_i})$, each tallier checks that they received a tuple $(\mathsf{i}, \mathsf{e}^0, \mathsf{e}^1, \mathsf{h}^0, \mathsf{h}^1))$ in $\mathsf{aux_T}$ such that $\mathsf{h} = \mathsf{h}^{\mathsf{v_i}}$. If this is not the case, the tuple is discarded. Otherwise, the tallier can recover the vote $(\mathsf{i}, \mathsf{v_i})$ associated to this value and add it to its own tally. Moreover, the tallier also computes the decryption factor of each ciphertext e in the tuple using its partial private key $\mathsf{sk_j}$. $\mathsf{T_j}$ sends all these decryption factors $\mathsf{DF_j}$ to EO. EO can recover the code associated to each $\mathsf{H}(\mathsf{tk}||\mathsf{i}||\mathsf{v_i})$ entry (since $\mathsf{DF_j}$ keeps the same order as the list D of shuffled tuples sent by EO and) by running $\mathsf{TCombine}$ (using the first t valid decryption factors it receives for example), but without ever knowing whether this code is the right one (because BI could have encrypted bad codes, for instance). Nevertheless, we assume that at least t lists of decryption factors have been computed honestly (and are detectable thanks to the validity decryption proofs

they contain).

Given the list of codes computed by TCombine, EO can re-order them and gather the selection codes $\mathsf{c}_\mathsf{i}^\mathsf{j}$ according to the structure of each ballot. Then, using the corresponding voting token tk, EO checks that the first bit of each code is correctly computed as $\mathsf{tk}[\mathsf{i}] \oplus \mathsf{j}$. If this is not the case, EO will not forward these codes to VS and the voter will ultimately report an error if the verification procedure is run. The main intuition behind this first-bit check is to avoid a code-swapping attack. That is, a corrupt BI could have swapped the encrypted codes sent to the talliers. Then, if the vote is swapped to the other value before the tally, the voter would not see anything since the two swaps cancel each other.

7.4.5. Verification procedure (Figure 7.6)

After the elections, voters can verify that their votes have been counted correctly. To do so, they run the VerifyVote protocol, which is run with the verification server VS. Each voter takes as input a receipt rcpt and the number of candidates m while the verification server takes as input the auxiliary information aux_V received from the ballot issuer and the proofs Π. At the end of the protocol, the voter outputs the string "cheat" if the verification fails or "honest" otherwise.

The protocol is described in Figure 7.6. The voter authenticates to VS as id and sends the hash of its voting token that was stored on rcpt. VS then verifies that this hash corresponds to the hash received from BI in aux_V for this id. In that case, it sends to the voter the tuple of the codes received from EO for that $\mathsf{H}(\mathsf{tk})$. Otherwise, it sends an error message verification_fails to the voter.

This check ensures that the voter received the right voting token from the ballot issuer. Indeed, aux_V contains the link between the voters identities and their voting token. This should be consistent with the voter's token received from the ballot issuer.

The voter then checks if the list of codes corresponds to the codes written on rcpt and outputs honest or cheat accordingly. It reports to the election authorities EA if this is not the case.

7.5. Security

7.5.1. Model

We introduce a simulation-based security definition in the simplified universally composable security model (SUC) [CCL15] for the security of postal voting protocols. We assume basic familliary with this model (or

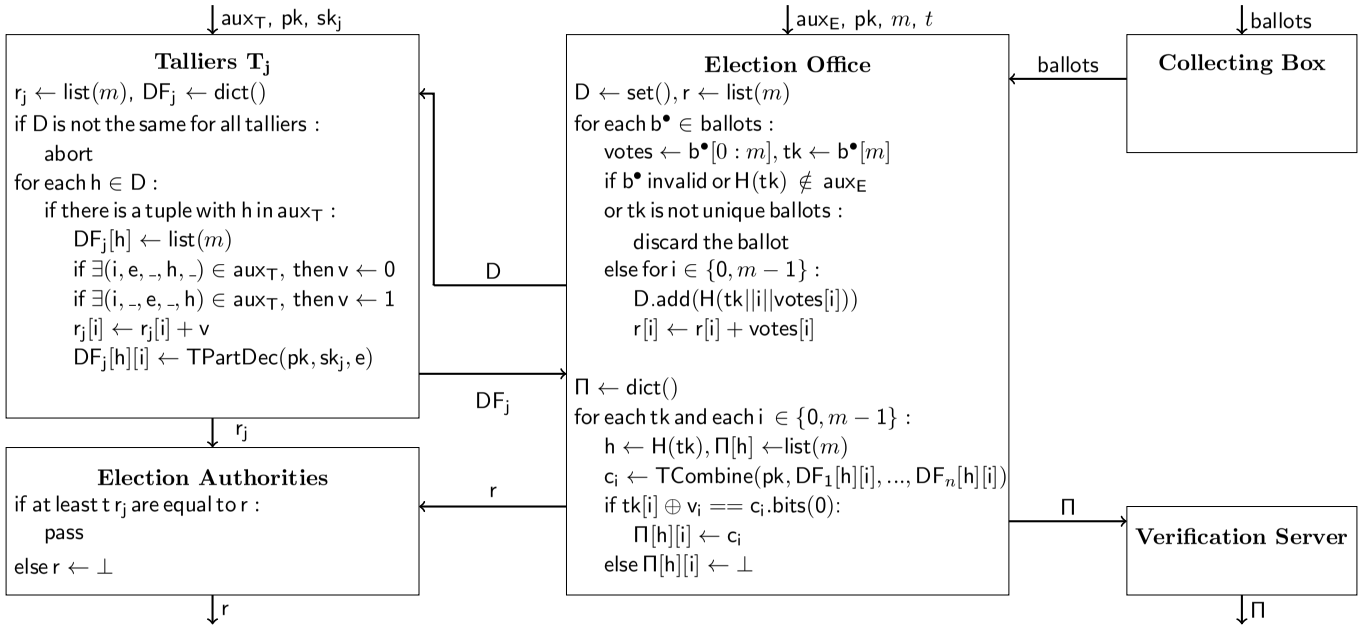


Figure 7.5.: Tallying procedure. The t decryption factors used in the TCombine algorithms are the first t decryption factors that have been received.

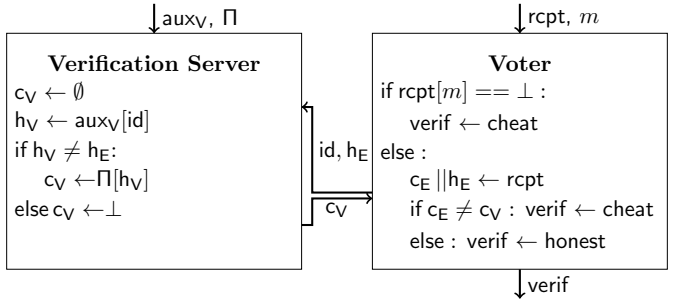


Figure 7.6.: Verification procedure

even with the UC model) and refer the reader to the former reference for further details.

The SUC model is particularly suitable in our setting: it has been designed for protocols executed by a fixed set of parties, as it is our case, and avoids the numerous technical challenges that come in the UC framework due to the dynamic machine creation process.

As usual in this framework, we define a trusted third party which would give natural security guarantees if it happened to exist. This incorruptible trusted third party implements an ideal functionality (which is described below). This functionality helps the parties to achieve some secure computations while limiting the view of the adversary.

In this model, there is also an external environment $\mathcal{E}$ that controls whatever is external to the protocol. This includes the scheduling of the communications between the parties and the handling of the inputs and outputs. In our case, $\mathcal{E}$ chooses the settings of the election (as EA's input) and the votes of the voters (as their input). $\mathcal{E}$ also receives the output of each party, specifically the result of the election and the verification output of the voters. Finally, $\mathcal{E}$ interacts arbitrarily with the adversary during the execution of the protocol.

As a reminder, a protocol is secure in this model if for any adversary $\mathcal{A}$ against the protocol, there exists an adversary $\mathcal{S}$ (also called the simulator) such that the environment $\mathcal{E}$ cannot distinguish between a real execution of the protocol with the adversary $\mathcal{A}$ and an ideal execution of the protocol through the trusted third party with $\mathcal{S}$.

We assume that all the communication channels between the parties are secure such that the only thing that the adversary learns about the content of a message is its size.[32]. Even though communication channels are not secure per se in the SUC model, we can compose our protocol

[32]Since the lengths of the messages in our protocol are bounded by a constant, we assume that the messages are padded so that the adversary does not learn anything.

with an instantiation of the Secure Message Transmission functionality proposed in [Can01] In practice, we assume that all the communications happen through TLS channels (or equivalent), with properly certified keys.

To model the potential lack of security of the postal channel, we allow the adversary to corrupt the postal endpoint CB. It can thus read, drop, and modify any ballot before the tally if the postal channel is corrupted.

Regarding the scheduling of the executions of the different entities and the communication between them, we use the same model as the SUC model and refer the reader to Section 2 of [CCL15].

The execution of the protocol in the real world follows the description given in Section 7.4. We fully describe here the instantiation of the protocol in the SUC model.

1. At the beginning of the execution, all parties and the environment start with the security parameter 1^λ written on their security parameter tapes. $\mathcal{E}$ is then activated and writes an input on EA's tape to start the protocol. If this input cannot be parsed as $(m, t, \mathsf{p_limit})$ such that m is a non-zero integer, t is an integer between 0 and n, and $\mathsf{p_limit}$ is a floating-point number between 0 and 1,[33] EA aborts the protocol.

   Otherwise, the official parties run the Setup protocol with input $\mathsf{p} = (1^\lambda, m, n, t, \mathsf{ID}, \mathsf{p_limit})$.[34] More precisely:

   a) EA forwards p to each participating party.

   b) The talliers run the KeyGen protocol together and obtain the public key pk and the secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{T}$. pk is sent to BI.

   c) BI runs the $\mathsf{GenerateBallot}(\lambda, \mathsf{pk}, m, \mathsf{ID}, \mathsf{p_limit})$ algorithm and obtains $\{\mathsf{b}^\circ_\mathsf{id}, \mathsf{aux}_\mathsf{id}\}_{\mathsf{id}\in\mathsf{ID}}, \mathsf{aux}_\mathsf{E}, \mathsf{aux}_\mathsf{T}, \mathsf{aux}_\mathsf{V}$. The last three values are respectively sent to EO, VS and the talliers.

   d) These parties check the consistency of their auxiliary values. Precisely:

      i. EO aborts if aux_E cannot be parsed as $|\mathsf{ID}|$ distinct outputs of the hash function.

      ii. VS aborts if aux_V cannot be parsed as a mapping of outputs of the hash function into ID.

[33] The size of this number is bounded by a polynomial number of bits as the environment runs in polynomial time.

[34] ID and n are known by EA because every party knows the set of identities of all participating parties.

When an official party has finished its execution of the setup procedure, it sends a confirmation message to EA. EA starts the voting phase when it has received a message from all official parties.

2. At the beginning of the voting phase, EA sends to every voter in ID a message voting_starts. When a voter id receives this message:
   - Either their input tape contains a voting intention $\mathsf{v} \in \{0,1\}^m$ and
     a) Sends a message retrieve_ballot to BI.
     b) Upon receiving a ballot $\mathsf{b}^\circ, \mathsf{aux}$ from BI, runs the vote algorithm $\mathsf{b}^\bullet, \mathsf{rcpt} \leftarrow \mathsf{Vote}(\mathsf{v}, \mathsf{b}^\circ, \mathsf{aux})$
     c) Sends $\mathsf{b}^\bullet$ to CB and keep rcpt in memory.
   - Or their input tape is empty or does not encode a valid vote and the voter does nothing.

   Similarly to the ideal execution, the adversary chooses when the voting phase finishes by sending a stop_vote command to CB.[35]

3. When the voting phase is finished, CB shuffles the ballots it received during the voting phase and sends them to EO.

   Together with the talliers and EA, EO runs the $\mathsf{Tally}(\mathsf{ballots}, \mathsf{aux}_\mathsf{E}, \mathsf{aux}_\mathsf{T}, \mathsf{pk}, \{\mathsf{sk}_\mathsf{i}\}_{\mathsf{i}=1}^n)$ procedure. At the end of the algorithm, EA outputs the result r and sends it to every voter, which also outputs it. If the result has been voided (that is, $\mathsf{r} = \perp$), then every party aborts the protocol. Otherwise, EO outputs the proof of individual verifiability Π, which is sent to VS.

4. At the beginning of the verification phase, EA sends to every voter in ID a message verification_starts. When a voter id receives this message, they run with VS the $\mathsf{VerifyVote}(\mathsf{rcpt}, \mathsf{aux}_\mathsf{V}, \Pi)$. The voter then outputs to $\mathcal{E}$ the result of the protocol (either cheat or honest).

5. Finally, the execution of the protocol ends when $\mathcal{E}$ outputs a bit.

7.5.2. Ideal Functionality for postal voting

In the definition of the ideal execution of the protocol, honest voters and the election authorities are replaced by so-called dummy voters and a

[35] In the SUC model, the adversary cannot directly communicate with the honest parties. We can solve this technical detail by introducing a new party (that we call the election scheduler) which is always corrupted and sends a message to CB when the voting phase finishes.

dummy EA. When a dummy voter id receives a voting_starts(sid) message from the functionality, they send their input v to the functionality or they do nothing if they have not received any input from the environment. Also, when they receive the result of the election or the result of the verification procedure, they output this result to the environment (if the voter receives a message nothing_received during the verification procedure, they either output honest if they have not sent anything or cheat otherwise).

Likewise, when the dummy EA is first activated, it sends a command Setup(sid, p) with the elections parameters $\mathsf{p} = (1^\lambda, m, n, t, \mathsf{ID}, \mathsf{p_limit})$ to the functionality, where $(m, t, \mathsf{p_limit})$ is the content of its input tape. It also returns to the environment the message abort if it receives this message from the functionality. After sending a message abort or after the last verification message, the functionality stops its execution.

The adversary $\mathcal{S}$ (which we also call the simulator in the ideal execution) interacts with the functionality and simulates the talliers, BI, CB, EO, and VS. Given an adversary $\mathcal{A}$ in the real execution, $\mathcal{S}$ tries to simulate the behavior of $\mathcal{A}$ such that $\mathcal{E}$ cannot distinguish them.

For ease of readability, the functionality has been broken into several chunks corresponding to the phases of the protocol.

Postal Voting functionality - Setup phase

- Upon receiving a command Setup(sid, p) from EA:
  - Parse p as $(1^\lambda, m, n, t, \mathsf{ID}, \mathsf{p_limit})$.
  - Send the command Setup(sid, p) to every voter and to the simulator.
  - Generate for each voter id in ID a random distinct string handle in $\{0,1\}^\star$ and record (sid, id, handle).
  - For each each voter id in ID, set the variables cast[id] $\leftarrow []$ and counted[id] $\leftarrow []$.
  - Send to the simulator the records (sid, id, handle) for each $\mathsf{id} \in \mathcal{C}$.
  - If the postal channel is corrupted, send all the records (sid, id, handle) for each $\mathsf{id} \in \mathsf{ID}$.
- Upon receiving a command abort(sid), send abort to EA.
- Upon receiving a command continue(sid) from the simulator, enter the Voting phase.

Every voter has a handle hiding their identity. The functionality will later know the vote of each handle but the mapping between the handles and the voters' id*s* will remain secret, hence preserving the privacy (except for the corrupted voters). The variables cast[id] and counted[id]

respectively represent the voter's ballots that are cast by the voter and counted in the tallying procedure. These might be different because of the adversarial behavior.

Secure Postal Voting functionality - Voting phase

- When entering the Voting phase, send a voting_starts(sid) command to every voter.
- Upon receiving a Vote(sid, id, v) command from the voter with identity id:
  - If $v \in \{0,1\}^m$, append v to cast[id] and counted[id].
  - If the postal channel is corrupted, forward the command to the simulator.
- Upon receiving a Vote(sid, id, v) command from the simulator, append v to counted[id] if $v \in \{0,1\}^m$.
- Upon receiving a command continue(sid) from the simulator, enter the Tallying phase.

In the real world, CB stops collecting the ballots at a given date and time. Even though time is not modeled here, it is equivalent to give to the adversary the power to stop the voting phase since it controls all the asynchronicity of the ballots.

Secure Postal Voting functionality - Tallying phase

- When entering the tallying phase:
  - Generate a list votes in the following way. For each $id \in ID$:
    - If $|counted[id]| = 1$, append (handle, counted[id]) to votes.
    - If $|counted[id]| \neq 1$, append $\perp$ to votes.
  - Shuffle the list votes.
  - Send to the simulator a command tally(sid, votes).
- Upon receiving a command modify_ballot(sid, handle, new_v) from the simulator:
  - If there exists some recorded (sid, id, handle) update counted[id] into new_v.
- Upon receiving a command abort(sid) from the simulator, send a message abort to EA.
- Upon receiving a command continue(sid) from the simulator:
  - Compute the result r of the election as follows:
    1. Set $r = list(m)$

2. For each $\mathsf{id} \in \mathsf{ID}$ and $i \in \{0, m\}$: if $|\mathsf{counted}[\mathsf{id}]| = 1$, set $\mathsf{r}[i]$ to $\mathsf{r}[i] + \mathsf{v}[i]$ where v is the unique vote in $\mathsf{counted}[\mathsf{id}]$.

– Send to the dummy EA and to every voter the result of the election.

– Enter the Verification phase.

In the tallying phase, the ideal functionality computes the result of the election as the election office would do in our protocol. Even though the adversary learns the values of the ballots, it does not know to which voter each ballot is assigned, except if the postal channel is non-confidential. If several ballots are recorded for one voter, the simulator does not learn any of them. This presents attacks in which the simulators sends a distinct number of ballots for each voter in order to learn their voting intention.

Secure Postal Voting functionality - Verification phase

- When entering the verification phase, wait until receiving a command identification_type(sid, *type*) from the simulator.
- Then, upon receiving a command verify(sid, voter, verification_type) from the simulator, set
  - id to voter if identification_type = identification_by_id
  - id to the record corresponding to the handle voter if identification_type = identification_by_handle

  and send to the id one of the following answer:
  - If verification_type = ballot_based, send:
    - nothing_received if cast[id] = counted[id] = []
    - cheat if cast[id] ≠ counted[id]
    - honest otherwise
  - If verification_type = conditional_cheat(fail_vote), where fail_vote is a set of pair (candidate, choice), send:
    - cheat if for some (i, j) ∈ fail_vote, (i, j) ∈ counted[id] as well
    - follow the ballot_counted command otherwise
  - If verification_type = verification_fail, send cheat.
  - If verification_type = conditional_success(p_success), ignore the command if p_success > p_limit. Otherwise, send honest with probability p_success or cheat with probability 1 − p_success.

In the verification phase, the adversary can choose the behavior of the ideal functionality for each voter. It can set the voter to return cheat (verification_fails request) or to output honest if and only if the ballot cast by the voter has been correctly counted (ballot_based request), which is the desired behavior.

There is also a conditional_cheat request that makes the voter output cheat if their vote is in some given set. This is needed, because a malicious ballot issuer could encrypt different codes than the ones given to the voter. Depending on their choices, the voter would see or not see the wrong codes which do not correspond to their auxiliary data.

Finally, the functionality can also set the voter to output honest with a probability at most p_limit, where this p_limit is a parameter of the election (with the verification_success request). This essentially means that a malicious adversary can only modify the vote of a voter without being detected with a controlled probability. Hence if a voter outputs honest, they know that with probability at least $1 - \mathsf{p_limit}$, their vote has been correctly counted. However the converse is not true: if the voter outputs cheat, there is no information about whether their vote has been correctly counted or not.

This tolerance of 'true-negative' attacks is actually needed because of the nature of the return code protocol. Indeed, an adversary could always cheat and try to guess the correct code of a voter. Indeed when the guess is correct, the voter cannot detect the vote manipulation. However for a small p_limit, the adversary is much more likely to get caught if it tries to modify many votes. In our protocol, p_limit can be controlled by decreasing or increasing the length b of the codes.

7.5.3. Secure Postal Voting

From this description of the ideal world, we can finally state our security definition that our protocol realizes.

Definition 33 (Secure postal voting)**.** *A Postal Voting Protocol $\mathcal{V}$ is secure if for any PPT adversary $\mathcal{A}$, there exists an ideal PPT adversary $\mathcal{S}$ such that for any PPT environment $\mathcal{E}$, the probability that the environment distinguishes between the execution of the real protocol and an interaction with the ideal postal voting functionality is negligible. In this game, at most one of these sets of official parties is corrupted:* {EO *and less than* t *talliers*}, {BI}, *and* {VS}. *Any number of voters and the postal channel can be corrupted.*

Theorem 23. *The postal voting protocol π described in Section 7.4 is secure in the sense of Definition 33.*

To prove this result, we first start by reminding readers of some technicalities of the SUC model. We then give the different simulators that realize the theorem, for each scenario of corruption, and finally the proofs of indistinguishability between the ideal and the real worlds.

The participating parties are the parties from Section 7.2.2. Each of them is modeled as a Turing machine and has a unique id. We consider a single protocol session, and define a single session identifier *sid* that contains the list of id's of all the protocol participants.[36] Some of these parties are corrupted following a static setting. That is, the set of corrupted parties is fixed by the adversary before the beginning of the protocol and is known by everyone.

Parties communicate together and with the functionality through a router, which is controlled by the adversary. When a party or the functionality wants to send a message to another party, the message is stored in the router and the adversary is notified about the identity of the sender and the receiver of the message. Then, the adversary can notify the router to deliver the message at any time and the message is written on some communication tape of the receiver. To fully show indistinguishability, the simulator needs to simulate this communication router as well.

The adversary has access to the communication tapes of the corrupted parties and has thus access every message they receive. The adversary can also write on their communication tapes in order to send messages on their behalves. As usual, we consider a dummy adversary controlled by the environment. The dummy adversary executes every message sent by the environment and forwards its view each time it is activated.

To prove the theorem, we have to show that $\mathcal{E}$'s view in the real world is indistinguishable from $\mathcal{E}$'s view in the ideal world. We respectively write $\mathsf{view}_{\mathcal{R}}$ and $\mathsf{view}_{\mathcal{I}}$ $\mathcal{E}$'s view in the real or ideal world.

More precisely, the view of the environment consists in the following: the communications of the honest parties (view_{COM}), the view of the corrupted voters (view_V), and the input and output of EA (io_{EA}) and of each of the honest voters (io_V). We will detail how each part of this view is computed identically in both worlds.

Simulators

Generic case: $\mathcal{S}$. Regardless of which official party is corrupted, we first build a generic simulator $\mathcal{S}$. That is, we assume first that only voters may be corrupted and thus that BI, EO, CB, VS, and the talliers are honest. This simulator will later be modified depending on which party is corrupted. We build the simulator $\mathcal{S}$ in the following way:

1. $\mathcal{S}$ invokes and simulates $\mathcal{A}, \mathsf{BI}, \mathsf{EO}, \mathsf{CB}, \mathsf{VS}$, and each of the talliers. $\mathcal{S}$ also creates a simulated router interacting with $\mathcal{A}$ that will be

[36] In the SUC model, the set of parties is known to all.

used to handle the communications between the participating and the simulated parties. $\mathcal{S}$ notifies $\mathcal{A}$ each time a party wants to communicate with another and delivers the message according to $\mathcal{A}$'s instructions. Also, $\mathcal{S}$ forwards messages between $\mathcal{E}$ and $\mathcal{A}$.

2. In the setup phase, $\mathcal{S}$ receives from the functionality the parameters of the election p and the handles corresponding to the corrupted voters. It simulates an execution of the Setup protocol and sends a command $\mathsf{continue}$ to the functionality.

3. During the voting phase, $\mathcal{S}$ simulates the view of the corrupted voters:

   - When a malicious voter authenticates to the simulated BI, $\mathcal{S}$ sends to $\mathcal{A}$ the ballot data $(\mathsf{b}^{\circ}_{\mathsf{id}}, \mathsf{aux}_{\mathsf{id}})$ of the corresponding malicious voter.
   - When $\mathcal{A}$ delivers a voting instruction to CB from a malicious voter that has the form of a valid voting ballot $\mathsf{b}^{\bullet} = \mathsf{v}|\mathsf{tk}_{\mathsf{id}}$ with a voting token $\mathsf{tk}_{\mathsf{id}}$ associated to an honest voter, then $\mathcal{S}$ sends the command $\mathsf{vote}(\mathsf{id}, \mathsf{v})$ to the functionality.
   - When the adversary sends a $\mathsf{continue}$ command to CB, $\mathcal{S}$ forwards the message to the functionality as well.

4. At the beginning of the tallying phase, $\mathcal{S}$ receives from the functionality the list of pairs of handles and votes. It randomly assigns to each honest handle the id of a different voter (abstentionist voters are associated to an handle without vote).

   Then it computes for each vote v of each handle the ballot $\mathsf{b}^{\bullet} = \mathsf{Vote}(\mathsf{b}^{\circ}_{\mathsf{id}}, \mathsf{v}, \mathsf{aux}_{\mathsf{id}})$ where $\mathsf{b}^{\circ}_{\mathsf{id}}$ and $\mathsf{aux}_{\mathsf{id}}$ are the values generated for the id assigned to the given handle. Using this list of ballots, $\mathcal{S}$ simulates an execution of the Tally protocol and obtains the result of the election r and the proofs of individual verification Π. At the end of the Tally protocol, $\mathcal{S}$ sends a $\mathsf{continue}$ command.

5. Finally in the verification phase, $\mathcal{S}$ sends to the functionality an $\mathsf{identification_type}(\mathsf{sid}, \mathsf{identification_by_id})$ command and simulates the messages of the $\mathsf{VerifyVote}$ algorithm for every voter.

   - When the last message of the $\mathsf{VerifyVote}$ interaction is delivered to a voter id, $\mathcal{S}$ sends a command $\mathsf{verify}(\mathsf{sid}, \mathsf{id}, \mathsf{ballot_based})$ to the functionality.
   - When interacting with a malicious voter, $\mathcal{S}$ executes the $\mathsf{VerifyVote}$ protocol with $\mathsf{aux}_{\mathsf{V}}$ and Π as input.

Corrupted postal channel corrupted case: $\mathcal{S}^{\mathsf{PC}}$. If the postal channel is corrupted, the corrupted CB is expected to receive the ballots in the same order as they are delivered. This include the well-formed ballots, but also the malformed ballots sent by the adversary. So in step 3 of the generic simulator, we add the following bullet point:

- When $\mathcal{A}$ delivers a vote command of the simulated router from a voter to CB, $\mathcal{S}^{\mathsf{PC}}$ delivers the corresponding command to the functionality and receives the content of the command $\mathsf{vote}(\mathsf{sid}, \mathsf{id}, \mathsf{v})$. Given this vote v and the public voting token $\mathsf{tk}_{\mathsf{id}}$ of this voter, $\mathcal{S}^{\mathsf{PC}}$ then computes the corresponding ballot with $\mathsf{Vote}(\mathsf{v}, \mathsf{tk}_{\mathsf{id}}, \mathsf{aux}_{\mathsf{id}})$ and forwards it to the corrupted CB on behalf of id.

In step 4, $\mathcal{S}^{\mathsf{PC}}$ receives the mapping between the handles and the ids. So, it uses it in step 4 instead of the random mapping of the generic simulator. Also, in step 4, when receiving a message with the ballots from the corrupted CB:

- Instead of using the votes output by the simulators, $\mathcal{S}^{\mathsf{PC}}$ uses this list as an input of the Tally protocol.
- For each ballot $\mathsf{b}^\bullet$ on this list, $\mathcal{S}^{\mathsf{PC}}$ checks if the ballot has a valid voting token $\mathsf{tk}_{\mathsf{id}}$ associated to a voter id. In that case, it recovers the vote and adds it to a list $\mathsf{counted}^{\mathcal{S}}[\mathsf{id}]$. Otherwise, it drops the ballot.
- $\mathcal{S}^{\mathsf{PC}}$ makes the functionality's $\mathsf{counted}^{\mathcal{S}}[\mathsf{id}]$ consistent with the corrupted CB's $\mathsf{counted}[\mathsf{id}]$ list with appropriate $\mathsf{modify_ballot}$ commands.

All the other messages from CB are ignored.

Corrupted verification server case: $\mathcal{S}^{\mathsf{VS}}$. If VS is corrupted, $\mathcal{A}$'s view is augmented with the values $\mathsf{aux}_{\mathsf{V}}$ in the setup phase and the verification data Π at the end of the tallying procedure. Thus, $\mathcal{S}^{\mathsf{VS}}$ sends these values (on behalf of BI and EO) respectively in step 2 during the $\mathsf{GenerateBallot}$ execution and in step 4 at the end of the Tally execution.

In the verification phase, the corrupted VS also interacts with the honest voters through an execution of the $\mathsf{VerifyVote}$ algorithm. So in step 5, we modify $\mathcal{S}$ as such:

- When simulating the $\mathsf{VerifyVote}$ protocol of honest voters, send $H(\mathsf{tk}_{\mathsf{id}})$ to the corrupted VS on behalf of id.
- When $\mathcal{A}$ delivers a message m from VS to an honest voter id:

- If m is equal to the codes $\Pi[\mathsf{H}(\mathsf{tk}_{\mathsf{id}})]$ computed by the Tally algorithm, send a $\mathsf{verify}(\mathsf{sid}, \mathsf{id}, \mathsf{ballot_based})$ command to the functionality (as $\mathcal{S}$ would do).
- Otherwise, if the postal channel is honest, use $\mathsf{verification_fails}$ as the $\mathsf{verification_type}$.

In the other cases, $\mathcal{S}^{VS}$ knows the $\mathsf{cast}[\mathsf{id}]$ and $\mathsf{counted}[\mathsf{id}]$ values associated to id. Let c_i^{true} and $c_i^{\mathcal{A}}$ be respectively the i-th code of $\Pi[\mathsf{H}(\mathsf{tk}_{\mathsf{id}})]$ and m. The $\mathsf{verification_type}$ is chosen as follows:

- If for some candidate i the choice is the same in $\mathsf{cast}[\mathsf{id}]$ and $\mathsf{counted}[\mathsf{id}]$ but $c_i^{true} \neq c_i^{\mathcal{A}}$, then $\mathcal{S}^{\mathsf{VS}}$ uses $\mathsf{verification_fails}$.
- Otherwise, if for some candidate i the choice is different in $\mathsf{cast}[\mathsf{id}]$ and $\mathsf{counted}[\mathsf{id}]$ but $c_i^{true} = c_i^{\mathcal{A}}$, then $\mathcal{S}^{\mathsf{VS}}$ uses $\mathsf{verification_fails}$.
- Otherwise, if $c_i^{true} \neq c_i^{\mathcal{A}}$ but both codes share either the same first bit or the same trailing bits, $\mathcal{S}^{\mathsf{VS}}$ uses $\mathsf{verification_fails}$.
- Otherwise, $\mathcal{S}^{\mathsf{VS}}$ finally uses $\mathsf{verification_success}(\mathsf{p}^{\mathsf{k}})$ where k is the number of different entries in $\mathsf{cast}[\mathsf{id}]$ and $\mathsf{counted}[\mathsf{id}]$ and p is equal to $\frac{1}{2^{\mathsf{b}}-1}$.

Finally if at any time $\mathcal{A}$ aborts the election on behalf of VS, $\mathcal{S}^{\mathsf{VS}}$ sends an abort message to the functionality as well. The other messages are ignored.

Corrupted election office and less than t corrupted talliers case: $\mathcal{S}^{\mathsf{EO}}$. If EO and less than t talliers are corrupted by the adversary, then $\mathcal{A}$'s view is augmented with the values $\mathsf{aux}_{\mathsf{E}}$ and $\mathsf{aux}_{\mathsf{T}}$ in the setup phase. Thus, $\mathcal{S}^{\mathsf{EO}}$ sends these values (on behalf of BI) in step 2 during the $\mathsf{GenerateBallot}$ execution.

At the end of the voting phase, the corrupted EO expects to receive from CB a shuffled list of ballots (if the postal channel is not corrupted). Similarly as before, $\mathcal{S}^{\mathsf{EO}}$ uses the voting token $\mathsf{tk}_{\mathsf{id}}$ to craft ballots of the votes list received from the functionality. Additionally, $\mathcal{S}^{\mathsf{EO}}$ includes each malformed ballots sent by the adversary that were dropped in step 2, shuffles the list, and sends it to EO.

In the tallying phase, the corrupted EO interacts with the honest talliers and sends them a list of values D to be tallied. When such a list has been sent to each honest tallier, they run a consensus protocol (with the corrupted talliers) as well to check that this value is the same for all of them. When the consensus algorithm terminates for one tallier, $\mathcal{S}^{\mathsf{EO}}$ executes the tallier's Tally algorithm on D and receives as output

a decryption factor that is sent to EO and a result r_j that is kept in memory.

Also, for each value $\mathsf{h}_{\mathsf{tk},i,\mathsf{v}_i} \in D$, the simulator computes the unique $(\mathsf{tk}, \mathsf{i}, \mathsf{v}_\mathsf{i})$ such that $\mathsf{H}(\mathsf{tk}||\mathsf{i}||\mathsf{v}_\mathsf{i}) = \mathsf{h}_{\mathsf{tk},\mathsf{i},\mathsf{v}_\mathsf{i}}$. If this tuple is not unique, $\mathcal{S}$ aborts. From this vote v and the handle associated with the voting token tk, $\mathcal{S}$ sends a $\mathsf{modify_ballot}$ command to the functionality to ensure that for this id, $\mathsf{counted}[\mathsf{handle}] = \mathsf{v}_\mathsf{i}$.

To conclude the tallying phase, if the malicious and the honest parties do not agree on the election result, $\mathcal{S}^{\mathsf{EO}}$ aborts the election. Otherwise, $\mathcal{S}^{\mathsf{EO}}$ sends the $\mathsf{continue}$ command to the functionality.

Finally, when the corrupted EO sends the codes Π^{EO} to VS, $\mathcal{S}$ launches the verification procedure with a command $\mathsf{identification_type}(\mathsf{sid}, \mathsf{identification_by_handle})$. Similarly to the corrupted VS case, $\mathcal{S}^{\mathsf{EO}}$ sends different $\mathsf{verify}(\mathsf{sid}, \mathsf{id}, \mathsf{verification_type})$ requests depending on the values of the codes Π^{EO}.

For a given handle and its token tk, we define v^{true} either as the vote in votes associated to handle if the postal channel is not corrupted or as the vote cast by the voter associated to handle which can be tracked otherwise. Then, we define c_i^{true} as the code associated to i, v^{true} and tk in the ballot generation mechanism. Similarly, we define $\mathsf{v}^{\mathcal{A}}$ as the vote associated with tk in the list D sent to the honest talliers (or $\perp$ otherwise) and $c_i^{\mathcal{A}}$ as the i-th value of $\Pi^{\mathsf{EO}}[H(\mathsf{tk})]$.

With these values in mind, $\mathcal{S}^{\mathsf{EO}}$ proceeds as follows in step 5:

- If $\mathsf{v}^{true} = \mathsf{v}^{\mathcal{A}}$ and for every candidate i, $c_i^{true} = c_i^{\mathcal{A}}$, then send a $\mathsf{verify}(\mathsf{sid}, \mathsf{handle}, \mathsf{ballot_based})$ command to the functionality.
- If for some candidate i the choice is the same in v^{true} and $\mathsf{v}^{\mathcal{A}}$ but the corrupted $c_i^{true} \neq c_i^{\mathcal{A}}$, use $\mathsf{verification_fails}$ as the $\mathsf{verification_type}$.
- Otherwise, if for some candidate the choice is different in v^{true} and $\mathsf{v}^{\mathcal{A}}$ but $c_i^{true} = c_i^{\mathcal{A}}$, use $\mathsf{verification_fails}$ as the $\mathsf{verification_type}$.
- Otherwise, if $c_i^{true} \neq c_i^{\mathcal{A}}$ but both codes share either the same first bit or the same trailing bits , then $\mathcal{S}^{\mathsf{VS}}$ uses $\mathsf{verification_fails}$ as the $\mathsf{verification_type}$.
- Otherwise, use $\mathsf{verification_success}(\mathsf{p}^{\mathsf{k}})$ as the $\mathsf{verification_type}$ where k is the number of different entries in the arrays $\mathsf{votes}[\mathsf{handle}]$ and $\mathsf{counted}[\mathsf{handle}]$, and p is equal to $\frac{1}{2^{\mathsf{b}}-1}$ if $\mathsf{v}^{\mathcal{A}} \neq \perp$ or $\frac{1}{2^{\mathsf{b}}}$ otherwise.

Finally if at any time $\mathcal{A}$ aborts the election on behalf of EO or a corrupted tallier, $\mathcal{S}^{\mathsf{EO}}$ sends an abort message to the functionality as well. The other messages are ignored.

Corrupted ballot issuer case: $\mathcal{S}^{\mathsf{BI}}$. If BI is corrupted, the simulator does not compute the ballots and the auxiliary data anymore. Instead, it receives from $\mathcal{A}$ the sets $\mathsf{aux_T}, \mathsf{aux_E}$, and $\mathsf{aux_V}$ in the setup phase.

In the voting phase, $\mathcal{S}^{\mathsf{BI}}$ sends to the adversary a $\mathsf{retrieve_ballot}$ message for each honest voter id and receives in return an empty ballot $\mathsf{b}^{\circ}_{\mathsf{id}}$ and the auxiliary data $\mathsf{aux_{id}}$.

Then, in the tally phase, some ballots might have to be dropped because the voting token is not in $\mathsf{aux_E}$. More precisely, for every id such that $H(\mathsf{b}^{\circ}_{\mathsf{id}}) \notin \mathsf{aux_E}$, $\mathcal{S}^{\mathsf{BI}}$ sends a command $\mathsf{modify_ballot}(\mathsf{sid}, \mathsf{id}, \perp)$.

Finally in the verification phase, BI might have computed the data sent to the honest voters and the data sent to the other parties maliciously. So for every voter id:

- If $(\mathsf{b}^{\circ}, \mathsf{aux})$ cannot be parsed as tk and 2 codes for each candidate i such that the first bit of each code is set by $\mathsf{tk}[\mathsf{i}]$, then $\mathcal{S}^{\mathsf{BI}}$ uses $\mathsf{verification_fails}$ as the $\mathsf{verification_type}$.

- Otherwise, if $(\mathsf{id}, H(\mathsf{b}^{\circ}_{\mathsf{id}})) \notin \mathsf{aux_V}$, then $\mathcal{S}^{\mathsf{BI}}$ uses $\mathsf{verification_fails}$ as the $\mathsf{verification_type}$.

- Otherwise, if for candidate i and choice j we have that the code $\mathsf{c}^{\mathsf{j}}_{\mathsf{i}}$ in $\mathsf{aux_{id}}$ is different from the decryption of the code corresponding to $\mathsf{H}(\mathsf{tk}||\mathsf{i}||\mathsf{j})$ in $\mathsf{aux_T}$, then $\mathcal{S}^{\mathsf{BI}}$ adds (i, j) to a set $\mathsf{fail_vote}$. It then uses a $\mathsf{conditional_cheat}(\mathsf{fail_vote})$ as the $\mathsf{verification_type}$.

- Otherwise, $\mathcal{S}$ uses $\mathsf{ballot_based}$ as the $\mathsf{verification_type}$ as in the generic simulator.

Again, all the other messages from BI are ignored.

Indistinguishability

Generic case:

Let F be the event that for two different tuples $(\mathsf{id}, \mathsf{i}, \mathsf{j})$ and $(\mathsf{id}', \mathsf{i}', \mathsf{j}')$ we have $\mathsf{H}(\mathsf{tk_{id}}||\mathsf{i}||\mathsf{j}) = \mathsf{H}(\mathsf{tk_{id'}}||\mathsf{i}'||\mathsf{j}')$ or $\mathsf{H}(\mathsf{tk_{id}}) = \mathsf{H}(\mathsf{tk_{id'}})$ (in the real world or in the simulated protocol). We will show that 1) F happens with negligible probability 2) If F does not happen, then the view of the environment is identical in both worlds.

Proof that F happens with negligible probability: With overwhelming probability, all the voting tokens are different since they are uniformly sampled from $\{0,1\}^{\lambda}$. Also, as H is modeled as a random oracle, there is a collision on the random values of the function with negligible probability.

Proof that $\mathsf{view}_{\mathcal{R}} = \mathsf{view}_{\mathcal{I}}$ if F does not happen: Regarding the view_{COM} part, $\mathcal{S}$ perfectly simulates the execution flow of the real protocol. For the view_V part, the ballot data $(\mathsf{b}^\circ, \mathsf{aux})$ of each malicious voter is computed identically as in the real world by the simulator. Also, the ballots cast and counted for each malicious voter is correctly tracked by the functionality. If the voter sent only one vote, the tally algorithm will correctly recover the corresponding codes. Otherwise, no code will be retrieved, as in the real world. This discussion regarding view_{COM} and view_V will also hold for the other simulators when more parties are corrupted, thus this will not be discussed further.

We then look at the io_{EA} part of $\mathcal{E}$'s view, especially EA's output. In both the real world and the ideal world, there is no abort since the official parties are all honest. Also, the functionality correctly records the set of ballots in the counted variable and the result of the election is computed similarly because the tallying procedure followed by the functionality is the same as what EO is doing in the real world (discarding ballots if the voting token appears more than once). Moreover, as every tallier is honest, they will agree with EO on the result they send to EA.

Let us finish with the output of the voters io_V. In the real world, an honest voter returns honest if and only if:

- On the ballot they receive, the first bit of each code is consistent with the voting token.
- On the verification procedure, they send the value h that the verification server has received from BI in aux_V.
- On the verification procedure, they receive from VS the codes associated with their selection in aux.

As all the official parties are honest, the first two bullet points are always verified. For the last point, the codes received from VS correspond to the decrypted codes of aux_T for the choices counted by EO. These decrypted codes are the same as the codes sent by BI in the aux data. This holds because again, all the official parties are honest.

Hence an honest voter returns honest if and only if the vote they sent is the same as the vote counted by EO, which is also the case in the ideal world because the functionality consistently tracks the ballots cast and counted for every voter. Thus the io_V part of $\mathcal{E}$'s view is identical again in both worlds.

We now see how we need to modify the simulator if more parties are corrupted.

Postal channel corrupted

The event F happens with the same probability because it is independent of the view and actions of the postal channel. Note that this will be true for any corrupted party except the ballot issuer.

Proof that $\mathsf{view}_{\mathcal{R}} = \mathsf{view}_{\mathcal{I}}$ if F does not happen:

$\mathcal{E}$'s view is now augmented with the view of the corrupted CB, which is perfectly simulated thanks to the functionality's forwarding of all the vote commands.

Moreover, the result of the election is consistent in both worlds because of the $\mathsf{modify_ballot}$ commands. If the adversary tampers with the list of ballots output by the corrupted CB, then the simulator updates the $\mathsf{counted}$ variable of the voter accordingly. Thus the $\mathsf{io}_{\mathsf{EA}}$ and io_{V} view will be the same in both worlds.

VS corrupted

First, the result of the election is computed similarly and independently of the action of VS. If the corrupted VS aborts the election in the real world, the simulator also aborts the election in the ideal world. Hence $\mathsf{io}_{\mathsf{EA}}$ is also the same in both worlds.

Then, $\mathcal{E}$'s view is now augmented with the view of the corrupted VS $\mathsf{view}_{\mathsf{VS}}$, which consist of $(\mathsf{aux}_{\mathsf{V}}, \Pi, \{\mathsf{h}_{\mathsf{E},\mathsf{id}}\}_{\mathsf{id}\in\mathcal{V}})$.

More precisely, $\mathcal{E}$'s view for each honest voter id is $(\mathsf{id}, \mathsf{v}_{\mathsf{id}}, \mathsf{H}(\mathsf{tk}_{\mathsf{id}}), \mathsf{codes}_{\mathsf{id}}^{\mathsf{received}}, \mathsf{codes}_{\mathsf{id}}^{\mathsf{sent}}, \mathsf{h}_{\mathsf{E},\mathsf{id}}, \mathsf{out}_{\mathsf{id}})$ where $\mathsf{codes}_{\mathsf{id}}^{\mathsf{received}}$ and $\mathsf{codes}_{\mathsf{id}}^{\mathsf{sent}}$ are respectively the codes received from EO and sent to the voters and $\mathsf{out}_{\mathsf{id}}$ is the status (cheat or honest) of id (from io_{V}). If the postal channel is corrupted, this view also includes $\mathsf{b}_{\mathsf{id}}^{\bullet}$.

In both words, $\mathsf{id}, \mathsf{v}_{\mathsf{id}}$, and $\mathsf{codes}^{\mathsf{sent}}$ are identical as they are input by the environment. Also, the values $\mathsf{H}(\mathsf{tk}_{\mathsf{id}})$ and $\mathsf{h}_{\mathsf{E},\mathsf{id}}$ are identically distributed in both worlds since they come from the honest parties BI and voter id. If the postal channel is corrupted $\mathsf{b}_{\mathsf{id}}^{\bullet}$ is also identical in both worlds, as the simulator knows the mapping between ids and handles.

It remains to look at $\mathsf{codes}_{\mathsf{id}}^{\mathsf{sent}}$ and the output of the honest voters $\mathsf{out}_{\mathsf{id}}$.

There are two case:

- *The postal channel is honest*:

  Let Q_{RO} be the set of values queried to the random oracle by the adversary in the voting phase and F_{RO} the event that for at least one voting token tk associated with some honest voter, $\mathsf{tk} \in Q_{RO}$.

We argue that this event happens with negligible probability. If there are n honest voters and c corrupted voters, the first query hits a voting token with probability at most $\frac{\mathsf{n}}{2^{\mathsf{l}}-\mathsf{c}}$. Similarly, if the first query does not hit a voting token, the second does with probability bounded by $\frac{\mathsf{n}}{2^{\mathsf{l}}-\mathsf{c}-1}$ and so on. Hence, the probability that F_{RO} happens is bounded by $\frac{\mathsf{n}|\mathsf{Q_{RO}}|}{2^{\mathsf{l}}-\mathsf{c}-|\mathsf{Q_{RO}}|}$, which is negligible. As a consequence, the event F_{tk} that at some point in the voting phase $\mathcal{A}$ sends a ballot $\mathsf{b}^\bullet$ with the voting token tk of an honest voter happens with negligible probability.

If F_{tk} does not happen, we have for every honest voter that $\mathsf{counted}[\mathsf{id}] = \mathsf{cast}[\mathsf{id}]$ and EO decrypts with the talliers the codes $\mathsf{codes}^{\mathsf{received}}$ expected by the voter in a successful verification procedure. If, the corrupted VS sends $\mathsf{codes}^{\mathsf{sent}} \neq \mathsf{codes}^{\mathsf{received}}$ to an honest voter, the voter will then output $\mathsf{out}_{\mathsf{id}} = \mathsf{cheat}$ in both worlds. Otherwise, the voter will output $\mathsf{out}_{\mathsf{id}} = \mathsf{honest}$.

- *The postal channel is corrupted*: In that case, $\mathsf{codes}^{\mathsf{received}}$ is computed identically in both worlds since the map between $\mathsf{handles}$ and ids is known. Also, the new simulator $\mathcal{S}^{\mathsf{VS}}$ correctly tracks if the corrupted VS sends a code that will result in a cheat output by the voter in the real world. Otherwise, it is also possible that the corrupted VS correctly guesses all the right codes of the voter's choices for a modified ballot. Since $\mathcal{A}$ knows the voting token of each voter through the corrupted postal channels, it knows as well the first bit of the code. Moreover, it knows that the trailing bits of the code are different from the trailing bits of the code of the other choice received from EO, hence it has a probability equal to $\frac{1}{2^{\mathsf{b}}-1}$ of guessing it correctly.

  Finally, VS guesses all the right codes with probability $\mathsf{p}^{\mathsf{k}} \leq \mathsf{p_limit}$, where k is the number of codes to guess. From the description of the $\mathsf{verification_success}$ command, the dummy voter's output is indistinguishable from the same voter's output in the real world.

EO and talliers corrupted

The view of the adversary is perfectly simulated, as the $\mathsf{aux}_{\mathsf{E}}$ values, the shuffling of the ballots, and the decryption factors of an honest tallier are computed identically in both worlds. Again, the result of the election is identical in both worlds, since the $\mathsf{modify_ballot}$ requests make sure that the $\mathsf{counted}$ values are consistent with the count of the honest tallier. It remains to look at io_{V}.

Let us look first at voters who did not vote. Arguably, the probability that EO sends a value $\mathsf{h}_{\mathsf{tk},\mathsf{i},\mathsf{v}_\mathsf{i}}$ to the honest talliers for a token associated with such a voter is negligible, using the same argument as the F_{RO} event of the corrupted VS case. Hence, the honest tallier will not count a vote for this voter and the adversary will not receive any code for this voter. If the corrupted EO sends codes for this voter to the verification server, they will output cheat in both worlds.

For the other voters, we argue that for a given honest voter and a given choice for a candidate, the election office can guess the code with probability negligibly close to $p = \frac{1}{2^b-1}$ if the adversary sends the corresponding value $h_{\mathsf{tk},i,\mathsf{v}_\mathsf{i}}$ to the talliers or $\frac{1}{2^b}$ otherwise.

We proceed by a sequence of indistinguishable games (in both the real and ideal worlds) from the security of the threshold encryption scheme.

In the first game, we use the simulator TSim of the threshold encryption scheme to compute the decryption factor of the honest party. So instead of computing $\mathsf{TPartDec}(\mathsf{sk}_t, c)$ for each encryption c of the i-th code, we use $\mathsf{TSim}(\mathsf{pk}, \mathcal{H}, \mathsf{c}, \mathsf{m})$ where m is equal to $\mathsf{c}_\mathsf{i}^{\mathsf{v}_\mathsf{i}}$. This transition is indistinguishable from the decryption simulatability property.

In the second game, we replace during the ballot generation the encryption of c_i^0 by a random value. This transition is indistinguishable from the CPA security of the scheme. In the third game, we proceed similarly and replace the encryption of c_i^1 by a random value.

As the code $\mathsf{c}_\mathsf{i}^{1-\mathsf{v}_\mathsf{i}}$ is now independent of the adversary view (up to the fact that the first bit and the leading bits of the code are different from the code of the other choice), it has a probability $\frac{1}{2^b-1}$ or $\frac{1}{2^b}$ to guess the right code like in the corrupted VS case.

BI corrupted

Compared to the generic simulator, the adversary does not learn anything from the view of the ballot issuer.

Regarding $\mathsf{io}_{\mathsf{EA}}$, the simulator still consistently tracks valid ballots and discards ballots which would be discarded as well in the real protocol. Moreover, an honest tallier aborts the election if it receives a value inconsistent with the value aux_T received from the corrupted BI both in the real and ideal world.

For io_V, either the honest voter outputs cheat if the codes are not consistent with tk or if the code vote of their vote is different from the encrypted value in aux_T, or else the output is the same as in the generic simulator.

7.5.4. Limit of the security analysis

Theorem 23 guarantees that if at most one party is corrupted, then the protocol attains the security definition formalized by the ideal functionality. However, it does not give any guarantee if more than one of those parties are corrupted, and it is thus natural to ask if the protocol can still be considered secure in this case.

Unfortunately, the answer is rather pessimistic. Any pair formed by the ballot issuer and either the election office or the verification server can intuitively break the verifiability of the protocol. Indeed, the ballot issuer knows all the codes and together with the verification server or the election office, they can make the voters believe anything. There is still no hope to build a simulator in this case, even if the postal channel is honest.

Furthermore, the privacy of all the voters can be broken even if two of those three parties are semi-honest. That is, two colluding parties can recover the vote of every voter without deviating from an honest execution of the protocol, from their view only. Indeed, the election office views all the votes and together with the verification server or the ballot issuer, it can map voting tokens to voters identities and hence break the privacy. In a similar fashion, the ballot issuer knows the mapping between the codes and the choices for each voter, and can thus also break the privacy with the verification server.

In conclusion, Theorem 23 is tight and captures exactly the scenarios for which we can prove the security of the protocol in our model.

7.6. Concluding remarks

While being extensively used worldwide, postal voting has received so far very little interest from the secure voting academic community, unlike electronic voting. Hence, this chapter is a first try at formalizing a security model for postal voting elections and proposing a protocol that is provably secure in this regard.

Compared to the existing models and protocols of electronic voting, the security level of our protocol may seem substandard. Indeed, we prove the security of the protocol if no official party is colluding. However, in any election system, it is impossible to obtain unconditional verifiability. For instance, we need to trust the entity providing the list of voters, which could falsify the election using fake identities and ballot stuffing. Moreover, security against a malicious voting device is often overlooked, and there is no satisfying solution which is robust against human behavior. Hence, we can see our distributed trust for integrity as a compromise

to obtain cast-as-intended and a simple voting process in a challenging setting.

Furthermore, we deliberately focused our design on interpretable hand-marked ballots and a simple verification process. Indeed, a contrived voting process makes it less likely that a voter correctly follows the protocol. This can lead to invalid ballots, the loss of verifiability or privacy guarantees, or distrust in the system. Besides, most of our protocol uses simple standard cryptographic primitives. This design choice eases the implementation process , hence reducing the likelihood of vulnerabilities due to mistakes, and makes our protocol compatible with post-quantum constructions. There is still an important caveat in our protocol though, which is the need for a trusted device to compute a hash and check the correctness of the ballot, and it would be highly desirable to avoid this requirement.

The security model we propose captures good properties we would want from a voting protocol. While the functionality might look convoluted, especially regarding the verification steps, it highlights a limit of protocols based on return codes. If the codes do not have a length linear in the security parameter, it is always possible for an adversary to guess the codes to return, and thus a voter can never be convinced with an overwhelming probability. In a remote setting as we considered, it is also difficult to ensure receipt-freeness, which was not formalized in our security model, but still considered in the design of the protocol. Finally, our protocol does not offer much accountability, and if a voter or an official party is acting maliciously, there is no convincing way to determine who is behaving wrongly. Such considerations are important for real-world elections and deserve more attention. We do not claim to propose a perfect security model, but merely to give the first steps in this direction.

Regarding receipt-freeness, our protocol has been designed to offer some protection against vote-selling. If the receipt-freeness coercer is not present at the time a voter receives their voting token, the voter can generate independently a fake voting token and swap the codes to reflect the coercer's will. This strategy only works if the coercer gives their voting instructions to the voter before asking for the voting token, which is weaker than the notion captured in Definition 16 and explored in the first part of this thesis. Moreover this notion is not formalized in our security analysis, which would need to be extended to some formalization of receipt-freeness in the UC model [Alw+15; UM10].

Finally, while our protocol is perfectly suitable for approval voting elections, building intuitive mechanisms for more complex ballots such as ranked choice voting remains an open direction for future work.

8. Conclusion

In this thesis, we investigated the design of secure voting protocols from several perspectives.

First, we introduced a new primitive, Traceable Receipt-Free Encryption as a tool to design receipt-free protocols in the rerandomized ballots approach. This primitive is supplemented by a new security notion, TCCA security, which naturally considers a pair of challenge ciphertexts to randomize instead of a pair of plaintexts to encrypt as in more classical CCA-like notions. We also proposed a generic voting protocol built from TREncs, and we exposed how TCCA security is particularly well-suited to prove receipt-freeness.

Then, we proposed several constructions of Traceable Receipt-Free Encryption schemes from bilinear groups based solely on the SXDH assumption and in the standard model. Specifically, we designed efficient constructions with a public coin CRS that can be compatible with homomorphic-based tallying or mix networks. To build these constructions, we leveraged linearly homomorphic signature schemes and adapted simulation soundness techniques.

Concluding our investigations on receipt-freeness, we explored if this property can be combined with a notion of cast-as-intended verifiability. On the one hand, we found out that it is impossible to simultaneously attain those two notions when the voting device is potentially dishonest if the protocol is single-pass and if we do not use a trusted registration service. On the other hand, we propose three different protocols that can achieve these two properties, either through an interactive ballot-casting process or through a trusted registration service. We also proposed a new definition of receipt-freeness that is well-suited for interactive voting protocols and is of independent interest.

On a more applied side, we studied a standard blueprint used in many protocol designs and researched how to optimize it on the voter's side. We considered fixed-based exponentiation, simple yet effective alterations of the encryption mechanism and several flavors of proof mechanisms. Eventually, we successfully demonstrated dramatic speed improvements by a theoretical analysis and benchmarks implemented in Python. Some of our improvements have been integrated in the ElectionGuard SDK, which has been deployed in various public elections

in the US and elsewhere.

Finally, we shifted from studying electronic voting protocols to considering postal voting protocols. We proposed the very first protocol formally proven in a security model instantiated in a variant of the universally composable framework. Notably, our protocol offers an intuitive ballot casting via hand-marked ballots and the overarching system using simple cryptographic blocks.

To summarize, if one thing can be learned from this thesis, it is that even 40 years after the seminal protocol of Benaloh [CF85], building secure voting protocols remains a challenging task. In particular, receipt-freeness and cast-as-intended properties are two security properties that are infamously hard to guarantee. This thesis tried to provide a better understanding of these notions and what we can expect from voting protocols. Our study of TREncs refined the design of single-pass protocols to achieve receipt-freeness in a more modular way, but we also exposed the need of interactivity to obtain cast-as-intended as well. Regarding cast-as-intended, we exhibited two radically different ways to achieve this property: on the one hand by sharing the vote and revealing a linear combination of these shares, and on the other hand, through the use of paper ballots in a postal voting setting.

There is still much to explore. This work engendered many new open problems and directions, discussed in the concluding remarks of each chapter. Still, I would like to emphasize some of them as the final words of this thesis.

Regarding TREncs, TCCA security is likely a stronger notion than what we really need to generically obtain receipt-freeness. In order to achieve more efficient constructions, it would be helpful to capture the exact security definition that we need in order to achieve this property. Relaxing the strong randomization property and exploring other ballot processing techniques besides just rerandomization could also help the design of better TREncs.

However, I believe that the techniques employed in Chapter 5 are a more promising way to build receipt-free protocols. First, they can help achieve cast-as-intended verifiability as well and do not need a bilinear group (at least in our interactive variants), but mostly they are conceptually simpler. Our protocols are somewhat naïve and were presented to show the feasibility of this approach, but it would be worth investigating more thoughtful designs.

Furthermore, the last chapter about postal voting opened the way for building secure postal voting protocols with a detailed security analysis. We only focused on a protocol suited for Belgian elections, but there are still many other design choices to explore.

And finally, besides the exciting process of designing new innovative solutions for secure voting, it is highly valuable to study well-established protocols used in the real world from a different perspective and ponder how we can improve them. This final remark is particularly relevant beyond the field of secure voting.

List of Notations

This list describes several symbols and notations that will be later used within the body of the thesis.

Algorithms

$\mathcal{A}, \mathcal{B}$	Adversary
$\mathcal{S}$	Simulator
$\mathcal{E}$	Environment
$\mathcal{D}$	Deceiving algorithm
$\mathcal{I}$	Receipt-freeness instruction
$\mathcal{F}$	Functionality
$\mathcal{H}$	Hash function
PPT	Probabilistic polynomial time
μ, ε	Negligible function
Pr	Probability
$y \leftarrow P(x; r)$	Execution of algorithm P with input x, randomness r and output y
$[P_1(x_1) \leftrightarrow P_2(x_2)]$	Interactive algorithm
st	State

Arithmetics

$\mathbb{Z}_q$	Set of integers modulo q
$\mathbb{G}$	Group
$(\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T)$	Bilinear group
e	Bilinear map
$g, h, \hat{g}, \hat{h}$	Generators

Security notions

λ	Security parameter
DDH	Decisional Diffie-Hellman
SXDH	Symmetric external Diffie-Hellman
Adv	Advantage
Exp	Experiment
IND	Indistinguishable
NM	Non-malleable
CPA	Chosen plaintext attack
CCA	Chosen ciphertext attack

TCCA	Traceable chosen ciphertext attack
wCCA	Weak chosen ciphertext attack
RCCA	Replayable chosen ciphertext attack
pdRCCA	Publicly detectable replayable chosen ciphertext attack

Abbreviations

ZK	Zero-knowledge
NIZK	Non-interactive zero-knowledge
qaNIZK	Quasi-adaptative non-interactive zero-knowledge
CRS	Common reference string
UC	Universally Composable
SUC	Simple Universally Composable
SRC	Signature on randomizable ciphertexts
TBE	Tag-based encryption
LHSP	Linearly homomorphic structure-preserving
TREnc	Traceable Receipt-Free encryption

Keys and output

pp	Public parameters
pk, PK	Public key
sk, SK	Secret key
tk, TK	Trapdoor key
ek, dk	Encryption/Decryption key (SRC context)
sk, vk	Signing/Verification key (SRC context)
$\mathsf{osk}, \mathsf{ovk}$	One time Signing/Verification key
ct, CT	Ciphertext
σ	Signature
π	Proof

Voting protocols

$\Pi, \mathcal{V}$	Voting protocol
$\mathbb{R}$	Result space
$\mathbb{V}$	Vote space
ρ	Result function
EA	Election authorities
BI	Ballot issuer
CB	Collecting box
EO	Election office
CS	Casting server
VS	Verification server
VD	Voting device
AD	Audit device

H	Human voter
BB	Bulletin board
v	Vote
tk	Voting token
b	Ballot
Γ	Verification data
rcpt	Receipt
r	Election result
Π	Proof of tally correctness

Bibliography

[Adi08] Ben Adida. "Helios: Web-based Open-Audit Voting". In: *USENIX Security 2008 Proceedings*. USENIX Association, 2008, pp. 335–348.

[Adi+09] 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". In: *EVT/WOTE 2009*. USENIX Association, 2009.

[Aki+13] Joseph A Akinyele, Christina Garman, Ian Miers, Matthew W Pagano, Michael Rushanan, Matthew Green, and Aviel D Rubin. "Charm: a framework for rapidly prototyping cryptosystems". In: *Journal of Cryptographic Engineering* 3.2 (2013).

[Alw+15] Joël Alwen, Rafail Ostrovsky, Hong-Sheng Zhou, and Vassilis Zikas. "Incoercible Multi-party Computation and Universally Composable Receipt-Free Voting". In: *Advances in Cryptology - CRYPTO 2015 Proceedings, Part II*. Vol. 9216. LNCS. Springer, 2015, pp. 763–780.

[AWA20] Philip B Stark Andrew W Appel Richard A DeMillo. "Ballot-marking devices cannot ensure the will of the voters". In: *Election Law Journal: Rules, Politics, and Policy* 19.3 (2020), pp. 432–450.

[BBS03] Mihir Bellare, Alexandra Boldyreva, and Jessica Staddon. "Randomness Re-use in Multi-recipient Encryption Schemeas". In: *Public Key Cryptography - PKC 2003 Proceedings*. Vol. 2567. Lecture Notes in Computer Science. Springer, 2003, pp. 85–99.

[Ben13] Josh Benaloh. "Rethinking Voter Coercion: The Realities Imposed by Technology". In: *EVT/WOTE 2013*. USENIX Association, 2013.

[Ben+13] Josh Benaloh, Michael D. Byrne, Bryce Eakin, Philip T. Kortum, Neal McBurnett, Olivier Pereira, Philip B. Stark, Dan S. Wallach, Gail Fisher, Julian Montoya, Michelle Parker, and Michael Winn. "STAR-Vote: A Secure, Transparent, Auditable, and Reliable Voting System". In: *EVT/WOTE 2013*. USENIX Association, 2013.

[Ben+18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. "Scalable, transparent, and post-quantum secure computational integrity". In: *IACR Cryptol. ePrint Arch.* (2018), p. 46.

[Ben21] Josh Benaloh. "STROBE-Voting: Send Two, Receive One Ballot Encoding". In: *E-Vote-ID 2021 Proceedings*. Vol. 12900. LNCS. Springer, 2021, pp. 33–46.

[Ben+21] Josh Benaloh, Kammi Foote, Philip B. Stark, Vanessa Teague, and Dan S. Wallach. "VAULT-Style Risk-Limiting Audits and the Inyo County Pilot". In: *IEEE Secur. Priv.* 19.4 (2021), pp. 8–18.

[Ben94] Josh Benaloh. "Dense probabilistic encryption". In: *Proceedings of the workshop on selected areas of cryptography*. 1994, pp. 120–128.

[Ber02] Daniel J. Bernstein. *Pippenger's exponentiation algorithm*. `https://cr.yp.to/papers/pippenger.pdf`. Jan. 2002.

[Ber+11] David Bernhard, Véronique Cortier, Olivier Pereira, Ben Smyth, and Bogdan Warinschi. "Adapting Helios for Provable Ballot Privacy". In: *ESORICS 2011 Proceedings*. Vol. 6879. LNCS. Springer, 2011, pp. 335–354.

[Ber+15] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. "SoK: A Comprehensive Analysis of Game-Based Ballot Privacy Definitions". In: *IEEE SP 2015*. IEEE Computer Society, 2015, pp. 499–516.

[Ber+20] Matthew Bernhard, Allison McDonald, Henry Meng, Jensen Hwa, Nakul Bajaj, Kevin Chang, and J. Alex Halderman. "Can Voters Detect Malicious Manipulation of Ballot Marking Devices?" In: *IEEE SP 2020*. IEEE, 2020, pp. 679–694.

[BF20] Balthazar Bauer and Georg Fuchsbauer. "Efficient Signatures on Randomizable Ciphertexts". In: *SCN 2020 Proceedings*. Vol. 12238. LNCS. Springer, 2020, pp. 359–381.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. "Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract)". In: *STOC 1988 Proceedings*. ACM, 1988, pp. 103–112.

[Bit+12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again". In: *ITCS 2012*. ACM, 2012, pp. 326–349.

[Bla+11] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. "Signatures on Randomizable Ciphertexts". In: *PKC 2011 Proceedings*. Vol. 6571. LNCS. Springer, 2011, pp. 403–422.

[Bls] *bls12_381 Rust crate. Available at* `https://github.com/zkcrypto/bls12_381`. 2023.

[Bou+22] Mikael Bougon, Hervé Chabanne, Véronique Cortier, Alexandre Debant, Emmanuelle Dottax, Jannik Dreier, Pierrick Gaudry, and Mathieu Turuani. "Themis: An On-Site Voting System with Systematic Cast-as-intended Verification and Partial Accountability". In: *CCS ACM 2022 Proceedings*. ACM, 2022, pp. 397–410.

[BPW12] David Bernhard, Olivier Pereira, and Bogdan Warinschi. “On Necessary and Sufficient Conditions for Private Ballot Submission”. In: *IACR Cryptol. ePrint Arch.* (2012), p. 236.

[Brf] *BeleniosRF javascript implementation. Available at* `https://gist.github.com/pyrros/4fddd7d49ae7c9c935f5d6a9a27d14c3#file-belenios-booth-js-L539`. 2016.

[BRT13] Josh Benaloh, Peter Y. A. Ryan, and Vanessa Teague. “Verifiable Postal Voting”. In: *Security Protocols XXI, Revised Selected Papers*. Vol. 8263. LNCS. Springer, 2013, pp. 54–65.

[BST19] Josh Benaloh, Philip B Stark, and Vanessa Teague. “VAULT: Verifiable audits using limited transparency”. In: *E-Vote ID Proceedings* (2019).

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. “Functional Encryption: Definitions and Challenges”. In: *TCC 2011 Proceedings*. Vol. 6597. LNCS. Springer, 2011, pp. 253–273.

[BT94] Josh Cohen Benaloh and Dwight Tuinstra. “Receipt-free secret-ballot elections (extended abstract)”. In: *STOC 1994 Proceedings*. ACM, 1994, pp. 544–553.

[Bün+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. “Bulletproofs: Short Proofs for Confidential Transactions and More”. In: *IEEE SP 2018, Proceedings*. IEEE Computer Society, 2018, pp. 315–334.

[Can01] Ran Canetti. “Universally Composable Security: A New Paradigm for Cryptographic Protocols”. In: *FOCS 2001*. IEEE Computer Society, 2001, pp. 136–145.

[CCL15] Ran Canetti, Asaf Cohen, and Yehuda Lindell. “A Simpler Variant of Universally Composable Security for Standard Multiparty Computation”. In: *Advances in Cryptology - CRYPTO 2015 Proceedings, Part II*. Vol. 9216. LNCS. Springer, 2015, pp. 3–22.

[CCM08] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. “Civitas: Toward a Secure Voting System”. In: *IEEE SP 2008*. IEEE Computer Society, 2008, pp. 354–368.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. “Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols”. In: *Advances in Cryptology - CRYPTO 1994, Proceedings*. Vol. 839. LNCS. Springer, 1994, pp. 174–187.

[CF85] Josh D. Cohen and Michael J. Fischer. “A Robust and Verifiable Cryptographically Secure Election Scheme (Extended Abstract)”. In: *FOCS 1985*. IEEE Computer Society, 1985, pp. 372–382.

[CGG19] Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. "Belenios: A Simple Private and Verifiable Electronic Voting System". In: *Foundations of Security, Protocols, and Equational Reasoning - Essays Dedicated to Catherine A. Meadows*. Vol. 11565. LNCS. Springer, 2019, pp. 214–238.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. "A secure and optimally efficient multi-authority election scheme". In: *Eur. Trans. Telecommun.* 8.5 (1997), pp. 481–490.

[CH20] Geoffroy Couteau and Dominik Hartmann. "Shorter Non-interactive Zero-Knowledge Arguments and ZAPs for Algebraic Languages". In: *Advances in Cryptology - CRYPTO 2020, Proceedings, Part III*. Vol. 12172. LNCS. Springer, 2020, pp. 768–798.

[Cha01] David Chaum. "SureVote: Technical Overview". In: *WOTE 2001 Proceedings*. 2001.

[Cha+12] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. "Malleable Proof Systems and Applications". In: *Advances in Cryptology - EUROCRYPT 2012 Proceedings*. Vol. 7237. LNCS. Springer, 2012, pp. 281–300.

[Cha+16] Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. "BeleniosRF: A Non-interactive Receipt-Free Electronic Voting Scheme". In: *CCS ACM 2016*. ACM, 2016, pp. 1614–1625.

[Cha81] David Chaum. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms". In: *Commun. ACM* 24.2 (1981), pp. 84–88.

[Che+10] Benoît Chevallier-Mames, Pierre-Alain Fouque, David Pointcheval, Julien Stern, and Jacques Traoré. "On Some Incompatible Properties of Voting Schemes". In: *Towards Trustworthy Elections, New Directions in Electronic Voting*. Vol. 6000. LNCS. Springer, 2010, pp. 191–199.

[CHK04] Ran Canetti, Shai Halevi, and Jonathan Katz. "Chosen-Ciphertext Security from Identity-Based Encryption". In: *Advances in Cryptology - EUROCRYPT 2004 Proceedings*. Vol. 3027. Lecture Notes in Computer Science. Springer, 2004, pp. 207–222.

[CKN03] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. "Relaxing Chosen-Ciphertext Security". In: *Advances in Cryptology - CRYPTO 2003, Proceedings*. Vol. 2729. LNCS. Springer, 2003, pp. 565–582.

[CL18] Véronique Cortier and Joseph Lallemand. "Voting: You Can't Have Privacy without Individual Verifiability". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. ACM, 2018, pp. 53–66.

[Cor+13] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. “Distributed ElGamal à la Pedersen: Application to Helios”. In: *WPES 2013 Proceedings*. ACM, 2013, pp. 131–142.

[Cor+14] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. “Election Verifiability for Helios under Weaker Trust Assumptions”. In: *ESORICS 2014, Proceedings, Part II*. Vol. 8713. LNCS. Springer, 2014, pp. 327–344.

[Cor+16] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Müller, and Tomasz Truderung. “SoK: Verifiability Notions for E-Voting Protocols”. In: *IEEE SP 2016*. IEEE Computer Society, 2016, pp. 779–798.

[CP92a] David Chaum and Torben P. Pedersen. “Wallet Databases with Observers”. In: *Advances in Cryptology - CRYPTO 1992, Proceedings*. Vol. 740. LNCS. Springer, 1992, pp. 89–105.

[CP92b] David Chaum and Torben P. Pedersen. “Wallet Databases with Observers”. In: *Advances in Cryptology - CRYPTO 1992, Proceedings*. Vol. 740. LNCS. Springer, 1992, pp. 89–105.

[CRH22] Braden L. Crimmins, Marshall Rhea, and J. Alex Halderman. “RemoteVote and SAFE Vote: Towards Usable End-to-End Verification for Vote-by-Mail”. In: *FC 2022, Revised Selected Papers*. Vol. 13412. LNCS. Springer, 2022, pp. 391–406.

[CS13] Véronique Cortier and Ben Smyth. “Attacking and fixing Helios: An analysis of ballot secrecy”. In: *J. Comput. Secur.* 21.1 (2013), pp. 89–148.

[Dev+24] Henri Devillez, Olivier Pereira, Thomas Peters, and Quentin Yang. “Can we cast a ballot as intended and be receipt free?” In: *IEEE SP 2024*. IEEE, 2024, pp. 3440–3457.

[DJ01] Ivan Damgård and Mads Jurik. “A Generalisation, a Simplification and Some Applications of Paillier’s Probabilistic Public-Key System”. In: *PKC 2001 Proceedings*. Vol. 1992. LNCS. Springer, 2001, pp. 119–136.

[DPP22a] Henri Devillez, Olivier Pereira, and Thomas Peters. “How to Verifiably Encrypt Many Bits for an Election?” In: *ESORICS 2022 Proceedings, Part II*. Vol. 13555. LNCS. Springer, 2022, pp. 653–671.

[DPP22b] Henri Devillez, Olivier Pereira, and Thomas Peters. “Traceable Receipt-Free Encryption”. In: *Advances in Cryptology - ASIACRYPT 2022 Proceedings, Part III*. Vol. 13793. LNCS. Springer, 2022, pp. 273–303.

[DPP24a] Henri Devillez, Olivier Pereira, and Thomas Peters. “Practical Traceable Receipt-Free Encryption”. In: *SCN 2024 Proceedings, Part I*. Vol. 14973. LNCS. Springer, 2024, pp. 367–387.

[DPP24b] Henri Devillez, Olivier Pereira, and Thomas Peters. "Verifiable and Private Vote-by-Mail". In: *IACR Cryptol. ePrint Arch.* (2024).

[Ele24] ElectionGuard. `https://www.electionguard.vote/`. 2024.

[Fau+18] Clara Faulí, Katherine Stewart, Federica Porcu, Jirka Taylor, Alexandra Theben, Ben Baruch, Frans Folkvord, Fook Nederveen, Axelle Devaux, and Francisco Lupiáñez-Villanueva. *Study on the benefits and drawbacks of remote voting.* `https://commission.europa.eu/strategy-and-policy/policies/justice-and-fundamental-rights/eu-citizenship/democracy-and-electoral-rights/studies/study-benefits-and-drawbacks-remote-voting_en`. 2018.

[FPS00] Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. "Sharing Decryption in the Context of Voting or Lotteries". In: *FC 2000.* Vol. 1962. LNCS. Springer, 2000, pp. 90–104.

[FS86] Amos Fiat and Adi Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *Advances in Cryptology - CRYPTO 1986, Proceedings.* Vol. 263. LNCS. Springer, 1986, pp. 186–194.

[Gen+07] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems". In: *J. Cryptol.* 20.1 (2007), pp. 51–83.

[Gjø11] Kristian Gjøsteen. "The Norwegian Internet Voting Protocol". In: *VoteID 2011, Revised Selected Papers.* Vol. 7187. LNCS. Springer, 2011, pp. 1–18.

[GK15] Jens Groth and Markulf Kohlweiss. "One-Out-of-Many Proofs: Or How to Leak a Secret and Spend a Coin". In: *Advances in Cryptology - EUROCRYPT 2015, Proceedings, Part II.* Vol. 9057. LNCS. Springer, 2015, pp. 253–280.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof Systems". In: *SIAM J. Comput.* 18.1 (1989), pp. 186–208.

[Gro09] Jens Groth. "Linear Algebra with Sub-linear Zero-Knowledge Arguments". In: *Advances in Cryptology - CRYPTO 2009 Proceedings.* Vol. 5677. LNCS. Springer, 2009, pp. 192–208.

[GS08] Jens Groth and Amit Sahai. "Efficient Non-interactive Proof Systems for Bilinear Groups". In: *Advances in Cryptology - EUROCRYPT 2008 Proceedings.* Vol. 4965. Lecture Notes in Computer Science. Springer, 2008, pp. 415–432.

[Hir01] Martin Hirt. "Multi party computation: efficient protocols, general adversaries, and voting". PhD thesis. ETH Zurich, Zürich, Switzerland, 2001. ISBN: 978-3-89649-747-5.

[HK08] Dennis Hofheinz and Eike Kiltz. "Programmable Hash Functions and Their Applications". In: *Advances in Cryptology - CRYPTO 2008 Proceedings*. Vol. 5157. Lecture Notes in Computer Science. Springer, 2008, pp. 21–38.

[HL02] Jeremy Horwitz and Ben Lynn. "Toward Hierarchical Identity-Based Encryption". In: *Advances in Cryptology - EUROCRYPT 2002 Proceedings*. Vol. 2332. LNCS. Springer, 2002, pp. 466–481.

[HMV03] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Berlin, Heidelberg: Springer-Verlag, 2003. ISBN: 038795273X.

[Hor] Case Van Horsen. *gmpy2 module*. `https://pypi.org/project/gmpy2/`.

[HS00] Martin Hirt and Kazue Sako. "Efficient Receipt-Free Voting Based on Homomorphic Encryption". In: *Advances in Cryptology - EUROCRYPT 2000, Proceedings, Part II*. Vol. 9057. LNCS. Springer, 2000, pp. 468–498.

[JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. "Coercion-resistant electronic elections". In: *WPES 2005, Proceedings*. ACM, 2005, pp. 61–70.

[Kil06] Eike Kiltz. "Chosen-Ciphertext Security from Tag-Based Encryption". In: *TCC 2006 Proceedings*. Vol. 3876. LNCS. Springer, 2006, pp. 581–600.

[KTV10] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. "A Game-Based Definition of Coercion-Resistance and Its Applications". In: *CSF 2010*. IEEE Computer Society, 2010, pp. 122–136.

[KW15] Eike Kiltz and Hoeteck Wee. "Quasi-Adaptive NIZK for Linear Subspaces Revisited". In: *Advances in Cryptology - EUROCRYPT 2015 Proceedings, Part II*. Vol. 9057. LNCS. Springer, 2015, pp. 101–128.

[Lib+13] Benoît Libert, Thomas Peters, Marc Joye, and Moti Yung. "Linearly Homomorphic Structure-Preserving Signatures and Their Applications". In: *Advances in Cryptology - CRYPTO 2013, Proceedings, Part II*. Vol. 8043. LNCS. Springer, 2013, pp. 289–307.

[Lib+14] Benoît Libert, Thomas Peters, Marc Joye, and Moti Yung. "Non-malleability from Malleability: Simulation-Sound Quasi-Adaptive NIZK Proofs and CCA2-Secure Encryption from Homomorphic Signatures". In: *Advances in Cryptology - EUROCRYPT 2014 Proceedings*. Vol. 8441. LNCS. Springer, 2014, pp. 514–532.

[LQT20] Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. "VoteAgain: A scalable coercion-resistant voting system". In: *USENIX Security 2020*. USENIX Association, 2020, pp. 1553–1570.

[McM+21] Eleanor McMurtry, Xavier Boyen, Chris Culnane, Kristian Gjøsteen, Thomas Haines, and Vanessa Teague. "Towards Verifiable Remote Voting with Paper Assurance". In: *CoRR* abs/2111.04210 (2021). arXiv: 2111.04210.

[MN06] Tal Moran and Moni Naor. "Receipt-Free Universally-Verifiable Voting with Everlasting Privacy". In: *Advances in Cryptology - CRYPTO 2006 Proceedings*. Vol. 4117. LNCS. Springer, 2006, pp. 373–392.

[MPT20] Eleanor McMurtry, Olivier Pereira, and Vanessa Teague. "When Is a Test Not a Proof?" In: *ESORICS 2020 - Proceedings, Part II*. Vol. 12309. LNCS. Springer, 2020, pp. 23–41.

[MRY04] Philip D. MacKenzie, Michael K. Reiter, and Ke Yang. "Alternatives to Non-malleability: Definitions, Constructions, and Applications (Extended Abstract)". In: *TCC 2004 Proceedings*. Vol. 2951. LNCS. Springer, 2004, pp. 171–190.

[OO89] Tatsuaki Okamoto and Kazuo Ohta. "Divertible Zero Knowledge Interactive Proofs and Commutative Random Self-Reducibility". In: *Advances in Cryptology - EUROCRYPT 1989 Proceedings*. Vol. 434. Lecture Notes in Computer Science. Springer, 1989, pp. 134–148.

[Pai99] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *Advances in Cryptology - EUROCRYPT '99 Proceedings*. Vol. 1592. Lecture Notes in Computer Science. Springer, 1999, pp. 223–238.

[Ped91] Torben P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *Advances in Cryptology - CRYPTO 1991 Proceedings*. Vol. 576. LNCS. Springer, 1991, pp. 129–140.

[Pip76] Nicholas Pippenger. "On the Evaluation of Powers and Related Problems (Preliminary Version)". In: *FOCS 1976*. IEEE Computer Society, 1976, pp. 258–263.

[PR08] Manoj Prabhakaran and Mike Rosulek. "Homomorphic Encryption with CCA Security". In: *ICALP 2008 Proceedings, Part II*. Vol. 5126. LNCS. Springer, 2008, pp. 667–678.

[Ràf15] Carla Ràfols. "Stretching Groth-Sahai: NIZK Proofs of Partial Satisfiability". In: *TCC 2015, Proceedings, Part II*. Vol. 9015. LNCS. Springer, 2015, pp. 247–276.

[Rfv] *Receipt-free verifiable protocols benchmarking. Available at* https://github.com/VerifiableRF/verifiable_rf_benchmarks. 2023.

[RRS21] Peter B. Rønne, Peter Y. A. Ryan, and Ben Smyth. "Cast-as-Intended: A Formal Definition and Case Studies". In: *FC 2021 Revised Selected Papers*. Vol. 12676. LNCS. Springer, 2021, pp. 251–262.

[Sch80] Jacob T. Schwartz. “Fast Probabilistic Algorithms for Verification of Polynomial Identities”. In: *J. ACM* 27.4 (1980), pp. 701–717.

[SDW08] Daniel Sandler, Kyle Derr, and Dan S. Wallach. “VoteBox: A Tamper-evident, Verifiable Electronic Voting System”. In: *USENIX Security 2008, Proceedings*. USENIX Association, 2008, pp. 349–364.

[SG02] Victor Shoup and Rosario Gennaro. “Securing Threshold Cryptosystems against Chosen Ciphertext Attack”. In: *J. Cryptol.* 15.2 (2002), pp. 75–96.

[Sha84] Adi Shamir. “Identity-Based Cryptosystems and Signature Schemes”. In: *Advances in Cryptology, CRYPTO 1984 Proceedings*. Vol. 196. LNCS. Springer, 1984, pp. 47–53.

[SHD10] Oliver Spycher, Rolf Haenni, and Eric Dubuis. “Coercion-Resistant Hybrid Voting Systems”. In: *EVOTE 2010*. Vol. P-167. LNI. GI, 2010, pp. 269–282.

[SK95] Kazue Sako and Joe Kilian. “Receipt-Free Mix-Type Voting Scheme - A Practical Solution to the Implementation of a Voting Booth”. In: *Advances in Cryptology - EUROCRYPT 1995 Proceeding*. Vol. 921. LNCS. Springer, 1995, pp. 393–403.

[SW05] Amit Sahai and Brent Waters. “Fuzzy Identity-Based Encryption”. In: *Advances in Cryptology - EUROCRYPT 2005 Proceedings*. Vol. 3494. LNCS. Springer, 2005, pp. 457–473.

[Swi23] Swiss Post. *The Swiss Post e-voting system*. `https://gitlab.com/swisspost-evoting/`. 2023.

[Tre] *Tight TREnc benchmarking. Available at* `https://github.com/uclcrypto/tight-trenc`. 2024.

[UM10] Dominique Unruh and Jörn Müller-Quade. “Universally Composable Incoercibility”. In: *Advances in Cryptology - CRYPTO 2010 Proceedings*. Vol. 6223. LNCS. Springer, 2010, pp. 411–428.

[Wal20] Dan Wallach. *Anyscale Connect: Encrypting and Tabulating Big Elections*. `https://www.youtube.com/watch?v=m7r33EuN6Zw`. Dec. 2020.

[Wat05] Brent Waters. “Efficient Identity-Based Encryption Without Random Oracles”. In: *Advances in Cryptology, EUROCRYPT 2005 Proceedings*. Vol. 3494. Lecture Notes in Computer Science. Springer, 2005, pp. 114–127.

[Wik08] Douglas Wikström. “Simplified Submission of Inputs to Protocols”. In: *SCN 2008 Proceedings*. Vol. 5229. LNCS. Springer, 2008, pp. 293–308.

[Wik09] Douglas Wikström. “A Commitment-Consistent Proof of a Shuffle”. In: *ACISP 2009 Proceedings*. Vol. 5594. LNCS. Springer, 2009, pp. 407–421.

[Wu09] Tom Wu. *RSA and ECC in JavaScript*. `http://www-cs-students.stanford.edu/~tjw/jsbn/`. 2009.

[Zag+13] Filip Zagórski, Richard Carback, David Chaum, Jeremy Clark, Aleksander Essex, and Poorvi L. Vora. "Remotegrity: Design and Use of an End-to-End Verifiable Remote Voting System". In: *ACNS 2013 Proceedings*. Vol. 7954. LNCS. Springer, 2013, pp. 441–457.

[Zip79] Richard Zippel. "Probabilistic algorithms for sparse polynomials". In: *EUROSAM '79, Proceedings*. Vol. 72. Lecture Notes in Computer Science. Springer, 1979, pp. 216–226.