THE UNIVERSITY
of EDINBURGH

Electronic voting in the classical and quantum settings: modelling, design and analysis

Nikolaos Lamprou

THE UNIVERSITY
of EDINBURGH

Thesis submitted in fulfilment of
the requirements for the degree of
Doctor of Philosophy
to the
University of Edinburgh — 2021

Declaration

I declare that this thesis has been composed solely by myself and that it has not been submitted, either in whole or in part, in any previous application for a degree. Except where otherwise acknowledged, the work presented is entirely my own.

Nikolaos Lamprou
August 2021

Abstract

This thesis explores the cryptographic field of electronic voting both in the classical and quantum regime. In the classical setting, we look at the problem of self-tallying elections, while in the quantum setting we initiate the formal study of quantum voting according to the principles of modern cryptography.

The concept of a self-tallying election (STE) scheme was first introduced by Kiayias and Yung [PKC 2002] and captures electronic voting schemes in which the tallying authorities are the voters of the election themselves. This type of electronic voting is particularly compatible with and suitable for (but not only) Blockchain governance, where governance is expected to be maintained in a fully distributed manner. In this thesis, we formalize the requirements for secure STE schemes in the Universal Composability (UC) framework. Our model captures the standard voting properties of eligibility, fairness, vote-privacy, and one voter-one vote. We present E-CCLESIA, a new family of STE schemes, and prove that it securely UC realizes the STE functionality. We propose E-CCLESIA 1.0 , the first concrete instantiation of E-CCLESIA using RSA accumulators in combination with a novel time-lock encryption scheme, name *Astrolabous*, that surpasses the limitations of previous ones. In addition, we provide a concrete security definition of TLE schemes and we formally abstract the concept of TLE into an ideal functionality following the real/ideal paradigm introduced by Canetti [IEEE FOCS 2001]. On top of that, we show that a protocol that uses a pair of TLE algorithms that satisfy these properties UC realises our ideal TLE functionality. Finally, we provide a novel TLE construction and we show that it satisfies our security definition making our whole argumentation of a fully-fledged E-CCLESIA protocol sound.

All practical e-voting constructions rely on computational assumption to satisfy various properties such as privacy and verifiability.

A milestone work published by Shor [IEEE SFCS 1994] indicates that well known mathematical problems can be solved efficiently if we have at our disposal a quantum computer. Recent advances indicate that quantum computers will soon be a reality. Motivated by this ever more realistic threat for existing classical cryptographic protocols, researchers have developed several schemes to resist quantum attacks. In particular, several e-voting schemes relying on the properties of quantum mechanics have been proposed for electronic voting. However, each of these proposals comes with a different and often not well-articulated corruption

model, has different objectives, and is accompanied by security claims that are never formalized and justified only against specific attacks. To address this, we propose the first formal security definitions for quantum e-voting protocols.

With these at hand, we systematize and evaluate the security of previously proposed quantum e-voting protocols; we examine the claims of these works concerning privacy, correctness and verifiability, and if they are correctly attributed to the proposed protocols. In all non-trivial cases, we identify specific quantum attacks that violate these properties. We argue that the cause of these failures lies in the absence of formal security models and references to the existing cryptographic literature.

Acknowledgements

I would like to genuinely thank my Supervisor Dr Myrto Arapinis for her continuous support and patience with me, giving me the chance to further continue my studies and evolve as a cryptographer. Except our technical discussions, I also sincerely thank her for teaching me not only how to be a technically sound scientist, but also one with moral principles. Her ethics and values will be a life lesson and inspiration for me.

I would also like to thank Dr Thomas Zacharias for our wonderful collaboration and for introducing me to the fascinating world of UC. His unique attitude and solid cryptographic knowledge made each of our discussions interesting and profitable in terms of knowledge.

Also, I would like to thank my examiners Prof Dominique Unruh and Dr Markulf Kohlweiss for their productive and insightful comments. My examination was not the easiest one but in the end, the quality and completeness of my thesis improved drastically.

I would like to thank all my friends from "Cinderella" for their support and the good times we have together. There were tough years and they were always there for me. Especially, I would like to thank my beloved friend (and upcoming doctor!) Giannis Stasinopoulos. He is one of the very rare people in terms of kindness, wisdom and character ethic.

Moreover, I would like to thank my professors back in Athens for introducing me to the magnificent world of cryptography.

In addition, I would like to thank my colleague Dr Aikaterini-Panagiota Stouka for all this journey in academia. Her morality, values and understanding will be always with me.

Last but not least, I would like to thank my family for all of their support all of these years, without them nothing of these would be possible.

Contents

Chapter 1

Introduction

Voting has been a fundamental component of democratic societies for over 2500 years. We can derive the term democracy from the Greek words "Demos" which means "people", and "Kratos" which means "government". Democracy is vital because a wider spectrum of people based on their social background, e.g. minorities, can affect decisions that hold the future of their state in a representative way. There are two main types of democratic societies, the first one is called *direct democracy* Tangian (2020b); where the eligible citizens vote for a decision, and the second one is called *representative democracy* Tangian (2020a); where the eligible citizens vote for a representative to decide for them for future matters related to the state. A combination of both is called *liquid democracy* Ramos (2015); whereby citizens vote for a representative but if they do not agree with the way he acts they can remove their vote from him at any time, in contrast with representative democracy where voters must wait until the service of the representative ends.

With the technological advancements of the computer era electronic voting, known as *e-voting* has been introduced. E-voting refers to any voting procedure that involves computer systems to support one (or more) stages, e.g. casting and/or tallying. The aspiration being that electronic voting systems could offer elections with higher voter participation and better accuracy, while also providing enhanced security guarantees such as privacy and verifiability, even in the face of dishonest election authorities compared to previous manual procedures. As a result, a direct benefit from the transition from paper ballot elections to e-voting is that voters will be more confident that their vote has been counted correctly while maintaining their privacy regarding their voting choice. Keeping that in mind, a potential other benefit is the increase of the participation rate in the elections; because enhanced security facilitates integrity and thus confidence in taking part in the elections. Another advantage would be the reduction of costs in comparison with paper ballot elections, making e-voting a more profitable choice in the long run Roberts (2007). With this drive, electronic voting has interested the research community and governments all over the globe for the last three decades.

Several cryptographic protocols have been proposed, implemented, and deployed for electronic voting Adida (2008); Cortier et al. (2016); Juels et al. (2005); Kiayias et al. (2015); Ryan and Schneider (2006); Okamoto (1998); Kiayias and Yung (2002); Szepieniec and Preneel (2015); Groth (2004); Chaum et al. (2009); Cortier et al. (2019). For example, Estonia deployed its own online e-voting system for national elections and other decision-making aspects Wikipedia (2020), albeit several security issues were raised Arthur (2014). Estonia's case was one of our motivations for defining and analysing a new e-voting protocol that tries to fill the gap in terms of security and efficiency concerning all the previous decentralized proposals. The existing electronic voting schemes vary and can be classified per their underlying trust assumptions and cryptographic tools. For example, *zero knowledge proofs* (ZKP) is a key cryptographic primitive that is used widely in e-voting. For example, with ZKP we can prove to the *auditors*, external entities of the protocol, that a ballot generated by a voter is well-formed without revealing the disclosed voting choice of the voter. Another key cryptographic primitive are *mix-nets* Sako and Kilian (1995), that are used to shuffle the contents of the *bulletin board* Kiayias et al. (2018), a distributed database where everyone can both read the contents and write in it but none can delete or tamper with information. The shuffling is important because it facilitates the unlinkability of the ballots with the sender's identity and thus preserves the anonymity of voters when the election result is announced. Another important cryptographic primitive is *homomorphic encryption* Yi et al. (2014), which allows a user to re-encrypt data without corrupting the initial plaintext. A common use of this primitive is in mix-nets, where the tallying authorities re-encrypt the ballots before the shuffling occurs so that the anonymity of the voters can be preserved.

On one end, one finds *fully centralized* schemes which are often vulnerable to large-scale attacks on robustness or privacy because of their centralized nature Bannet et al. (2004); Wolchok et al. (2012); Estehghari and Desmedt (2010). More relaxed variations of such systems Adida (2008) ground their security by trusting a server responsible for the collection of the ballots and the production of the election outcome. Of course, this procedure is publicly verifiable by external entities, which are called *auditors*.

On the other end, schemes that are fully decentralized except maybe from setting up the election, where the (tallying) authorities are the voters of the election themselves, are called *self-tallying* Okamoto (1998); Kiayias and Yung (2002); Szepieniec and Preneel (2015); Groth (2004). In between the two ends, and to avoid the difficulties of *self-tallying elections* (STE) and dangers of centralized elections, a range of e-voting schemes that distribute the trust among a small set of authorities have also been proposed. Such systems achieve security as long as a subset of those authorities is honest. For instance, in mix-net based schemes, at least one authority needs to be honest Adida (2008); Ryan and Schneider (2006), and in threshold encryption schemes, at most, a fraction ϵ of the authorities can be corrupt and collude Kiayias et al. (2015); Juels et al. (2005). Of course,

other things being equal, electronic voting schemes at the decentralized end of the spectrum are preferable as they can withstand more powerful adversaries. Self-tallying election schemes are a step forward in this direction and move us one step closer to real democracy. With the recent developments of Blockchain technologies and the appeal for on-chain distributed governance, such mechanisms are all the more topical. However, they present major challenges, which explains why full decentralization often needs to be abandoned in favour of achieving all the conflicting requirements that an electronic voting scheme should ensure. The main challenges of self-tallying elections (STE) are in guaranteeing that no one (or no coalition of) voter(s) can boycott the election, no intermediate results are being leaked during the casting phase (*fairness*), and no vote can be linked back to the voter that cast it (*vote-privacy*). Because of the mentioned challenges, this type of e-voting has drawn less attention than its predecessor, centralized e-voting. In this thesis, we are working towards this direction, modelling and defining security properties and analysing STE protocols against standard cryptographic approaches such as the *universal decomposable* (UC) framework Canetti (2001b). For that task, we define intermediate functionalities from scratch, like the *time-lock encryption* (TLE) functionality and security definitions in a *game-based* style that every TLE construction must satisfy to be proven UC secure. The necessity of using TLE in our protocol is important to solve the fairness issue all previous STE proposals in the literature were vulnerable. However, no TLE construction in the literature can be proven UC secure. The reason behind this limitation is related to the UC framework itself. Intuitively, to capture semantic security, no information should be leaked regarding the plaintext in the ideal world, on the other hand, all ciphertexts will eventually open to the correct plaintext, resulting in a trivial distinction of the real from the ideal setting. We go against this intuition and we define such a scheme in the random oracle model, showing that it satisfies our game-based definition and in turn, UC realizes our ideal functionality.

The second part of this thesis turn to the problem of e-voting against quantum adversaries. Ideally, the security of the most promising e-voting systems we mention above relies on computational assumptions such as the hardness of integer factorization and the discrete logarithm problem (see Chapter 2). But, these are easy to solve with quantum computers using Shor's algorithm Shor (1994). Although not yet available, recent technological advances indicate that quantum computers will soon be threatening existing cryptographic protocols Gibney (2020); IBM (2019). In this context, researchers have proposed to use quantum communication to implement primitives like key distribution Diffie and Hellman (2006), bit commitment Brassard et al. (1988) and oblivious transfer Rabin (2005). Unfortunately, perfect security without assumptions has proven to be challenging in the quantum setting Lo and Chau (1998); Mayers (1997), and the need to study different corruption models has emerged. This includes limiting the number of dishonest parties and introducing different non-colluding authorities. This is precisely what we set to address in the second part of

this thesis. We first give formal definitions for verifiability, integrity and vote privacy in the quantum setting considering adaptive corruption. Subsequently, we systematize and assess the security of existing e-voting protocols based on quantum technology. We specifically examine the claims of each of these solutions concerning the above-mentioned well-defined properties. Unfortunately, our analyses uncover vulnerabilities in all the proposed schemes. While some of them suffer from trivial attacks due to inconsistencies in the security definitions, one of the contributions of this thesis is to argue that sophisticated attacks can exist even in protocols that "seem secure" if the security is proven ad hoc, and not in a formal framework. We argue that the cause of these failures is the absence of an appropriate security framework in which to establish formal security proofs, which we have now introduced.

Therefore, this thesis follows previous works Barnum et al. (2002); Portmann (2017); Unruh (2010) in their effort to highlight the importance of formally defining and proving security in the relatively new field of quantum cryptography.

1.1 Contributions

We present our findings in the next Chapters.

Chapter 4: Our contributions are summarized as follows:

- We present a UC definition of secure TLE via an ideal functionality $\mathcal{F}_{\mathsf{TLE}}$ that captures naturally the concept of TLE as it provides the necessary security guarantees a TLE scheme should provide. Specifically, it captures *semantic security* as the encryption of a message is not correlated with the message itself. Instead, it is correlated only with the length of the message similar to the standard encryption functionality in Canetti (2001b). In addition, it captures *correctness* Goldreich (1999); Kościelny et al. (2013), i.e., if $\mathcal{F}_{\mathsf{TLE}}$ finds two different messages with the same ciphertext in its record, then it aborts. Finally, we note that in the literature, there are TLE constructions Liu et al. (2018) where the adversary holds an advantage in comparison with the other parties and which might allow him to decrypt a message earlier than the intended time. To cater for such constructions, we parameterise $\mathcal{F}_{\mathsf{TLE}}$ with a leakage function leak which specifies the exact advantage (in decryption time) of the adversary compared to the honest parties. Ideally, the leak function offers no advantage to the adversary. It is worth mentioning that TLE constructions in which the adversary holds an advantage in comparison with the honest parties in the decryption time are still useful to study in the UC framework because the computational burden for solving the puzzle can be transferred to external entities of the protocol (e.g., Bitcoin miners), making the decryption more client friendly Liu et al. (2018). Naturally, the encryption takes some time, depends on the actual

construction. For that task, we introduce a variable delay that captures the time encryption time of a message.

- We define a hybrid TLE protocol and a standalone basic security definition in a game-based fashion. We show that if the pair of TLE algorithms that our protocol uses satisfies our basic security definition then we have a UC realization of $\mathcal{F}_{\mathsf{TLE}}$.

  Our TLE protocol does not use the vanilla version of a TLE algorithm (e.g. a TLE algorithm as defined in Liu et al. (2018)). Instead, it relies on an extended one based on techniques introduced in Nielsen (2002); Camenisch et al. (2017) in the random oracle model. Our extension was necessary for the proof of UC realization. Specifically, in both the real and ideal world, all the messages eventually can be decrypted by any party. To avoid trivial distinctions[1], the simulator must be able to equivocate so that the ciphertext opens to the correct message. As a result, the simulator programs the random oracle so that the ciphertext opens to the target message, something that is not feasible with the vanilla version of a TLE scheme without the equivocation feature which our extension provides.

  In our hybrid protocol, we defined both a functionality wrapper $\mathcal{W}_q$ and an evaluation functionality $\mathcal{F}_{\mathsf{eval}}$, to model the computation that is necessary for solving the time-lock puzzle. In our case, this computation is a random oracle query, thus $\mathcal{F}_{\mathsf{eval}}$ is the random oracle. Like in Badertscher et al. (2017), the main function of the functionality wrapper is to restrict the access to $\mathcal{F}_{\mathsf{eval}}$ and thus to model the limited computational resources a party has at her disposal in each round. In our case, the limited amount of computation a party has to solve the time-lock puzzle through queries to $\mathcal{F}_{\mathsf{eval}}$.

  Our basic security definition of TLE schemes consists of two properties, named **Correctness** and **qSecurity**. The **Correctness** property states that the decryption of an encrypted message m leads to the message m again with high probability, similar to the definition of *correctness* in the standard encryption's case. We define the **qSecurity** property in a game-based style, between a challenger and an adversary where the latter tries to guess the *challenged message* with less than the required oracle queries. A TLE scheme satisfies the **qSecurity** property if the above happens with negligible probability, capturing the fact that a message can only be decrypted when "the time comes".

- We provide a novel construction, named *Astrolabous*, and we show that it satisfies our basic security definition, thus it supports the UC realisation of

[1] Recall that in the ideal world, to capture semantic security, ciphertexts do not contain any information about the actual message except its length

$\mathcal{F}_{\mathsf{TLE}}$ (in the random oracle model). Astrolabous combines ideas from both the constructions in Rivest et al. (1996) and in Mahmoody et al. (2011). Nevertheless, we did not use either of them for the following reasons. A critical drawback of Mahmoody et al. (2011) is that parts of the plaintext are revealed through the process of solving the time-lock puzzle, which is based on a hash evaluation, as the message is hidden in the puzzle itself. On the other hand, the construction in Rivest et al. (1996) encrypts a message with a symmetric encryption scheme Kościelny et al. (2013) and then hides the encryption key into the time-lock puzzle which is based on repeated squaring. The first problem with the latter construction was that the procedure for solving the puzzle is deterministic (repeated squaring) and thus a party can bypass the functionality wrapper and solve any time-lock puzzle in a single round, in contrast with the construction in Mahmoody et al. (2011) where the procedure for solving the puzzle is randomized (hash evaluation which is modelled as random oracle). The second problem with the construction in Rivest et al. (1996) was that even if a party provides the solution of the puzzle but the puzzle issuer does not provide the trapdoor information that is used by the time the time-lock puzzle was created (in this case, the factorization of a composite number N) then, to verify the validity of the provided solution, all the verifying parties must resolve the time-lock puzzle. Thus, the *optimal complexity* scenario is hard to achieve. In contrast, the time-lock puzzle in Mahmoody et al. (2011) is easily verifiable without the need for any trapdoor information from the puzzle issuer.

These were our motivations for defining Astrolabous that tackles all of the above-mentioned limitations. Specifically, Astrolabous uses an asymmetric key encryption scheme to hide the message like in Rivest et al. (1996) and then "hides" the symmetric key in a time-lock puzzle similar to the one in Mahmoody et al. (2011).

- We introduce an additional stronger game-based definition, named IND-CPA-TLE, to capture semantic security of TLE schemes in the spirit of IND-CPA security. Our stronger definition may serve as a standard for analysing TLE schemes in the standalone setting. To demonstrate the usefulness of our stronger definition and constructions, we prove that Astrolabous and an enhanced version of the construction in Mahmoody et al. (2011) achieve IND-CPA-TLE security.

Chapter 5: We formalize the concept of self-tallying elections by defining the ideal functionality $\mathcal{F}_{\mathsf{STE}}$. The functionality captures the standard e-voting properties such as one-voter-one-vote, eligibility and privacy. Specifically, if a voter votes more than once and her vote passes the verification test algorithm provided by the simulator in previous steps, the functionality outputs bottom, guarding the one-voter-one-vote property. Similarly works the eligibility check, if

the ballot of a non-eligible voter passes the verification test algorithm provided by the simulator then functionality outputs bottom. Last, privacy is defined in the same lines as $\mathcal{F}_{\mathsf{TLE}}$ captures semantic security.

Moreover, we provide a protocol, namely E-CCLESIA, that UC realizes $\mathcal{F}_{\mathsf{STE}}$. We separate two of the main functions of $\mathcal{F}_{\mathsf{STE}}$, which are the *vote management* and the *eligibility* check, into two sub-functionalities namely $\mathcal{F}_{\mathsf{vm}}$ and $\mathcal{F}_{\mathsf{elig}}$ respectively. Specifically, $\mathcal{F}_{\mathsf{vm}}$ handles the ballot generation and the ballot casting, capturing the privacy of the $\mathcal{F}_{\mathsf{STE}}$. Next, $\mathcal{F}_{\mathsf{elig}}$ handles the credential generation and the ballot authentication of the eligible voters, capturing the eligibility property of the $\mathcal{F}_{\mathsf{STE}}$. Take these two functionalities into account, we provide a proof of UC realization of $\mathcal{F}_{\mathsf{STE}}$. This modular approach allows modifications to our protocol, E-CCLESIA, without re-proving the whole construction making it more accessible for future development. Precisely, changing one cryptographic primitive affects only the proof of UC realization in that level, leaving intact the rest of the proof. This leads us to define a family of protocols that UC realize $\mathcal{F}_{\mathsf{STE}}$. Finally we provide a UC realization of $\mathcal{F}_{\mathsf{vm}}$ via $\mathcal{F}_{\mathsf{TLE}}$.

Chapter 6: We review the state-of-the-art of quantum e-voting protocols by systematizing them base on their special features. Unfortunately, we have discovered concrete attacks against all proposals. That means that quantum e-voting protocols fail to meet the cryptographic standards to be considered secure like innovative *quantum key distribution* (QKD) protocol Bennett and Brassard (1984).

Specifically, we have constructed an adversary that violates the claimed properties, making the protocols insecure for future deployment without first addressing the security issues we raised.

Chapter 2

Preliminaries

In this chapter we present the general background information so that the reader is comfortable enough to understand: i) the nature and the unique features of quantum information; ii) the general concept of e-voting; iii) the security requirements/definitions of an e-voting protocol that should satisfy if we are to argue about its security.

2.1 Basic mathematical background and security notions

Overview: In this section we give the necessary mathematical background and cryptographic notions that are required for the understanding of this work. We distinguish the background to mathematical (which is related closely to quantum information) and cryptographic definitions (which is related to both classical and quantum information) to help the reader to have a better understanding of which mathematical tools are necessary for each part of this work (although that some notions are common both in classical and quantum).

2.1.1 Mathematical background

Definition 2.1.1. We define the *field* $\mathbf{F}$ to be a non empty close set that satisfies all the field axioms regarding two operations, that we denote as addition and multiplication, specifically $\forall a, b, c \in \mathbf{F}$ it holds:

name	addition	multiplication
associativity	$(a+b)+c = a+(b+c)$	$(ab)c = a(bc)$
commutativity	$a+b = b+a$	$ab = ba$
distributivity	$a(b+c) = ab+ac$	$(a+b)c = ac+bc$
identity	$a+0 = a = 0+a$	$a \cdot 1 = a = 1 \cdot a$
inverses	$a+(-a) = 0 = (-a)+a$	$aa^{-1} = 0 = a^{-1}a$ for $a \neq 0$

where the 1 and 0 are the identity elements of addition and multiplication respectively.

In this work, we consider $\mathbf{F}$ to be either the set of complex or real numbers (it will be clear from the context). A *finite field* is a field with finite number of elements. The most common example is the class of integers modulo n, where n is a big prime number. Moreover, a *group* is a finite field equipped only with multiplication that satisfies all of the field axioms except the *commutativity* one. Last, a *group generator* g is an element of the group $\mathbf{G}$ such that:

$$\forall y \in \mathbf{G}, \exists \alpha \in \mathbb{N} \text{ s.t } g^{\alpha} = y$$

This states that every element of the group can be created if we multiply the group generator with itself a certain amount of times.

Definition 2.1.2. We define the *vector space* $\mathbf{V}$ over a field $\mathbf{F}$ to be any set that is closed under addition and scalar multiplication, with scalar factor from the field $\mathbf{F}$.

For example, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is an element of $\mathbf{V} \subseteq \mathbb{R}^n$. In this work we consider that each element of the vector is either a complex or real number. We will use the vector, row, column representations interchangeably in this work.

Definition 2.1.3. The *inner product* between vectors $\mathbf{x}, \mathbf{y}$ in vector space $\mathbf{V}$ over field $\mathbf{F}$ is an additional structure of vector space $\mathbf{V}$ that associates $\mathbf{x}$ and $\mathbf{y}$ with an element of the field $\mathbf{F}$ denoted $< \mathbf{x}, \mathbf{y} >$. A vector space with such a structure is called *inner product space*.

In this work, we consider the *Euclidean inner product*, specifically for vectors $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$ we define:

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{j=1}^{n} x_j \bar{y_j} \in \mathbf{F}$$

Definition 2.1.4. The *outer product* between two vector spaces $\mathbf{V}_1$ and $\mathbf{V}_2$ over the same field $\mathbf{F}$ of dimensions n and m respectively[1] is a new vector space $\mathbf{V}_3$ over $\mathbf{F}$ with dimension nm.

For example the outer product of $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_m)$ is:

$$\mathbf{x} \otimes \mathbf{y} = (x_1\mathbf{y}, \ldots, x_n\mathbf{y})$$

[1]The dimension of a finite vector space $\mathbf{V}$ over a field $\mathbf{F}$ is an integer number that shows the number of elements of $\mathbf{V}$ that forms a *base* of $\mathbf{V}$.

Similarly, in quantum's case as we see, the outer product of $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$ and $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$ is:

$$\mathbf{x} \otimes \mathbf{y} = \mathbf{x}(\mathbf{y})^t = (y_1\mathbf{x}, \ldots, y_m\mathbf{x})$$

.

Definition 2.1.5. The *norm of a vector space* $\mathbf{V}$ over a field $\mathbf{F}$ is a function $\|\cdot\| : \mathbf{V} \to \mathbb{R}$ where $\mathbb{R}$ is the set of real numbers with the following properties: For every $\mathbf{x}, \mathbf{y} \in \mathbf{V}$ and $\alpha \in \mathbf{F}$ it holds that:

1. $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$
2. $\|\alpha\mathbf{x}\| = |\alpha|\|\mathbf{x}\|$
3. $\|\mathbf{x}\| = 0 \Rightarrow \mathbf{x} = 0$

where $|\cdot|$ is the absolute value if $\mathbf{F}$ is the real numbers or the norm of a complex number.

In this work we use only the *Euclidean norm* for complex vector spaces. For example, the norm of vector $\mathbf{x} = (x_1, \ldots, x_n)$ is:

$$\|\mathbf{x}\| = \sqrt{\sum_{j=1}^{n} x_j x_j^*}$$

where x_j^* is the complex conjugate of x_j.

2.1.2 Cryptographic notions

Negligible function Goldreich (1999): The definition of *negligible* function is widely used in the context of cryptography when we want to argue about the security of a protocol. More precisely, we will required the probability a malicious entity, called the *adversary*, to violate some of the desired properties to be very small, formalized as negligible. Formally:

Definition 2.1.6. A negligible function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ is function such that, for every $c > 0$ there exists $n_0 \in \mathbb{N}$ such that $\forall n \geq n_0$ it holds:

$$\mathsf{negl}(n) < 1/n^c$$

Intuitively, as the value n gets bigger, the negligible function negl becomes very small (in fact smaller than any polynomial function).

Security parameter Goldreich (1999): The *security parameter* of a protocol, usually symbolized by λ, is a number that defines the space and time complexity of the protocol and the level of security we want to achieve. Ideally, the time complexity of a protocol is polynomial to the security parameter; and the probability the adversary deviate from the protocol is negligible to the security parameter as well.

Hardness assumptions Goldreich (1999): In classical cryptography, in many cases the protocol's security relies on computational assumptions or in other words on the hardness of solving certain problems efficiently. A problem that is very useful in classical cryptography is the *discrete logarithm problem* which can be stated as follows:

Definition 2.1.7. Given a group $\mathbf{G}$, a group generator g and a random element of the group y, find the value x such that $g^x = y$.

Another established hard problem in the literature is the *factorization* of a big composite number. Specifically:

Definition 2.1.8. Given a composite number $n = pq$, where p, q are big primes, find the factorization of n.

The underline assumption for both of these problems is that no polynomial-time machine to the security parameter can find the value x or the factors p, q, except with negligible probability to the security parameter. Most fundamental cryptographic primitives rely for their security on these assumptions. For example commitments schemes Pedersen (1992), encryption systems ElGamal (1985) to name just a few.

Cryptographical hash function Goldreich (1999): A *hash function* is a very important building block in cryptography for many applications Nakamoto (2008); Nielsen (2002); Merkle (1979). It takes as input an arbitrary in size bit string and outputs a bit string polynomial in size to a security parameter. Specifically:

Definition 2.1.9. A hash function with respect to a security parameter λ is a function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{poly(\lambda)}$ where *poly* is a polynomial function.

Every hash function should satisfy the following security properties:

- **Pre-image resistance:** Given a random value $y \in \{0,1\}^{poly(\lambda)}$ the probability of finding a value $x \in \{0,1\}^*$ such that $\mathcal{H}(x) = y$ in polynomial time with respect to λ is $\mathsf{negl}(\lambda)$.
- **Second pre-image resistance:** Given a random value $x_1 \in \{0,1\}^*$, the probability of finding a value $x_2 \in \{0,1\}^*$ such that $\mathcal{H}(x_1) = \mathcal{H}(x_2)$ in polynomial time with respect to λ is $\mathsf{negl}(\lambda)$.

- **Collision resistance:** The probability of finding $x_1, x_2 \in \{0,1\}^*$ such that $\mathcal{H}(x_1) = \mathcal{H}(x_2)$ in polynomial time with respect to λ is $\mathsf{negl}(\lambda)$.

2.2 Quantum information

Overview: We use the term quantum bit or *qubit* Nielsen and Chuang (2011) to denote the simplest quantum mechanical object we will use. We say that a qubit is in a *pure* state if it can be expressed as a linear combination of other pure states:

$$|x\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,,\ \text{where}\ |0\rangle = \begin{bmatrix}1\\0\end{bmatrix}, |1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$$

where $|\alpha|^2 + |\beta|^2 = 1$ for some $\alpha, \beta \in \mathbb{C}$. The states $|0\rangle$ and $|1\rangle$ are called the *computational basis* vectors. Sometimes it is also helpful to think of a qubit as a vector in the two-dimensional *Hilbert space* $\mathcal{H}$.[2] A *mixed* state cannot be described as a linear combination of pure states. Instead, it can be described by a *density matrix*. The density matrix of a quantum state, usually denoted by ρ, is a linear combination of outer products between pure states where the squared amplitudes sum up to 1. For example, $\rho = 1/2\,|0\rangle\langle 0| + 1/2\,|1\rangle\langle 1|$. It is worth mentioning that with density matrices we can describe both pure and mixed states.

The generalization of a qubit to an m-dimensional quantum system is called *qudit*:

$$|y\rangle = \sum_{j=0}^{m-1} a_j\,|j\rangle\,,\ \text{where}\ \sum_{j=0}^{m-1} |a_j|^2 = 1$$

Let us now suppose that we have two qubits; we can write the state vector as:

$$|\psi\rangle = \sum_{i,j\in\{0,1\}} \alpha_{ij}\,|ij\rangle$$

where $\sum_{i,j\in\{0,1\}} |\alpha_{ij}|^2 = 1$. If the total state vector $|\psi\rangle$ cannot be written as a tensor product of two qubits (*i.e.* $|x_1\rangle \otimes |x_2\rangle$), then we say that qubits $|x_1\rangle$ and $|x_2\rangle$ are entangled. An example of two-qubit entangled states, are the four *Bell states*, which form a basis of the two-dimensional Hilbert space:

$$|\Phi^{\pm}\rangle = \tfrac{1}{\sqrt{2}}(|00\rangle \pm |11\rangle), |\Psi^{\pm}\rangle = \frac{1}{\sqrt{2}}(|01\rangle \pm |10\rangle)$$

A quantum system that is in one of the above states is also called an *EPR* (EinsteinPodolskyRosen) pair Einstein et al. (1935).

[2] A Hilbert space is an inner product space with norm that is associated with the inner product as $\|\,|\mathbf{x}\rangle\,\| = \sqrt{\langle \mathbf{x}|\,|\mathbf{x}\rangle}, \forall\,|\mathbf{x}\rangle \in \mathcal{H}$.

The no-cloning theorem: A very important difference between quantum and classical information, is that there is no mechanism to create a copy of an unknown quantum state Nielsen and Chuang (2011). This result, known as the *no-cloning theorem*, is one of the fundamental advantages and at the same time limitations of quantum information. Specifically, it states that there is no way to clone an unknown quantum state. It becomes extremely relevant for cryptography since *brute-force* types of attacks cannot be applied on quantum channels that carry unknown information.

In the following Figure we summarize the most standart Nielsen and Chuang (2011) quantum notations that we use.

Notation	Description
z^*	Complex conjugate of the complex number z, e.g. $(1+i)^* = 1-i$.
$\|\phi\rangle$	Vector, also known as `ket`.
$\langle\phi\|$	The dual vector of $\|\phi\rangle$, also known as `bra`.
$\langle\phi\|\psi\rangle$	The inner product between vectors $\|\phi\rangle$ and $\|\psi\rangle$.
$\|\phi\rangle \otimes \|\psi\rangle$	Tensor product of $\|\phi\rangle$ and $\|\psi\rangle$.
$\|\phi\rangle\|\psi\rangle$	Abbreviated notation for tensor product of $\|\phi\rangle$ and $\|\psi\rangle$.
A^*	Complex conjugate of matrix A.
A^T	Transpose of matrix A.
$A^\dagger$	Hermitian conjugate or adjoint of matrix A, $A^\dagger = (A^*)^\mathsf{T}$.
$\langle\phi\|A\|\psi\rangle$	Inner product between $\|\phi\rangle$ and $A\|\psi\rangle$.

2.2.1 Unitary operations

The evolution of a closed quantum system can be described by the application of a *unitary operator*. Unitary operators can be described by unitary matrices which are reversible and preserve the inner product[3]. Specifically, we say that a square matrix U is unitary if its *conjugate transpose* matrix $\mathsf{U}^\dagger$ (a matrix that can be obtained from U by taking the transpose and then the conjugate complex of each entry) is also its inverse matrix, specifically:

$$\mathsf{U}^\dagger\mathsf{U} = \mathsf{U}\mathsf{U}^\dagger = \mathbb{I}$$

where $\mathbb{I}$ is the identity matrix. Recall our first example, and let us say we would like to swap the amplitudes on state $|x\rangle$, then we can apply the operator X (known as *NOT-gate*):

$$X|x\rangle = \beta|0\rangle + \alpha|1\rangle\,, \quad \text{where} \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

The X-gate is one of the *Pauli* operators, which together with Z and Y, as well as the *identity operator* $\mathbb{I}$, form a basis for the vector space of 2×2 Hermitian matrices. The fact that these form a basis means that every possible computation

[3]This is true only for finite dimension spaces, which is the case in this thesis.

is achievable as long as we have at our disposal these operators. These operators are unitaries, and as such preserve the inner product. Specifically,

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, Y = \begin{bmatrix} 0 & i \\ -i & 0 \end{bmatrix}$$

As a result, the inner product of the quantum state after we applied the unitary operator, in our example X, with itself should be equal to one. In our example[4]

$$\langle x| X^\dagger X |x\rangle = \langle x| \mathbb{I} |x\rangle = |\alpha|^2 + |\beta|^2 = 1$$

2.2.2 Continuous & discrete measurement

The way we obtain information about a quantum system is by performing a *measurement* using a family of linear operators $\{M_j\}$ acting on the state space of the system, where j denotes the different outcomes of the measurement. It holds for the *discrete* and the *continuous* case respectively that:

$$\sum_j M_j^\dagger M_j = \int M_j^\dagger M_j dj = \mathbb{I}$$

where $M_j^\dagger$ is the conjugate transpose of matrix M_j, $\mathbb{I}$ the identity operator and the integral of the matrix $M_j^\dagger M_j$ is the matrix of the indefinite integral of each element of $M_j^\dagger M_j$. For qudit $|y\rangle$, the probability that the measurement outcome is w is: $\Pr(w) = \langle y| M_w^\dagger M_w |y\rangle$ and in the continuous case $\Pr(w \in [w_1, w_2]) = \int_{w_1}^{w_2} \langle y| M_j^\dagger M_j |y\rangle \, dj$.

For a single qubit $|x\rangle = \alpha |0\rangle + \beta |1\rangle$, measurement in the computational basis will give outcome zero with probability $|\alpha|^2$ and outcome one with probability $|\beta|^2$. If our state is entangled, a partial measurement (i.e. a measurement in one of the entangled qudits), not only reveals information about the measured qudit but possibly about the remaining state. For example, let us recall the Bell state $|\Phi^+\rangle$. A measurement of the first qubit in the computational basis will give measurement outcome 0 or 1 with equal probability and the remaining qubit will collapse to the state $|0\rangle$ or $|1\rangle$ respectively.

2.2.3 The cut-and-choose method

When verifying quantum resources (e.g. when we want to verify if a quantum state is sampled from a specific distribution), it is necessary to apply a *cut-and-choose* technique to test that the received quantum states are produced correctly. The quantum source would therefore need to send exponentially many copies of the quantum state Kashefi et al. (2017), for the verifier to measure most of them and

[4] This also shows that in whatever base we *measure* our state the probabilities of all possible outcomes will sum to one, as expected.

deduce that with high probability, the remaining ones are correct. Specifically, a cut-and-choose protocol is an interactive protocol between a source and a verifier. The source samples many quantum states $|x\rangle$ from the distribution $\mathbf{D}$ and sends them to the verifier. The verifier after an interaction with the source (which involves the exchange of either classical or quantum information) decides if the state $|x\rangle$ is sampled from $\mathbf{D}$ indeed or not. The first classical instance of a cut-and-choose protocol Rabin (1978) involves a source that tries to convince the verifier that the received number n is a product of two primes p and q. This can be done without the source sending many times the value n, in contrast with the quantum case where the state $|x\rangle$ must be sent multiple times. The reason behind this is that after the verifier measures the state $|x\rangle$, the state collapses in a specific eigenspace. As a result, the state cannot be reversed to its initial state and the verifier to repeat the verification procedure needs a new copy. On the other hand, the verifier cannot clone the state $|x\rangle$ because of the *no-cloning* theorem. The source needs to send exponentially many states to the protocol's security parameter for the following reasons: i) After each verification test the state is destroyed because usually, the verification procedure involves a measurement. So the verifying party needs another copy of the initial state if it wants to repeat the procedure on the initial state. ii) To achieve a negligible probability of cheating to the protocol's parameter, exponentially many states must be sent, else there is a non-negligible probability that the corrupted state remains unchecked.

2.3 E-voting

In this section, we present what an e-voting protocol is by describing the special roles of each party. Moreover, we outline the security requirements an e-voting protocol should satisfy to be considered secure according to today's cryptographic standards Cortier et al. (2016); Bernhard et al. (2015); Canetti and Krawczyk (2002); Canetti (2001a); Unruh (2010).

2.3.1 Protocol description

An e-voting protocol Adida (2008); Chevallier-Mames et al. (2010); Delaune et al. (2006); Okamoto (1998); Kiayias et al. (2015) is a protocol with the participating parties being the voters, denoted by V, the talliers, denoted by T, a setup authority, denoted by SA and a trusted bulletin board, denoted by BB. Any party can write and read on BB, but no data can be deleted on BB. In other words, BB is a read-write append-only structure. The role of SA is to set up the protocol parameters and give the participation token, usually called a *credential*, to the eligible voters[5]. The voters generate and cast their ballot and the talliers gather the ballots and produce the election outcome based on a tally function

[5]The eligible voters is a subset of the set of all voters

specified by the protocol (e.g. a majority function). So an e-voting protocol consists of the following phases:

1. `Setup` phase: The SA generates the protocol parameters and distributes them accordingly to the participating parties. This generation could require interaction with some of the other parties.

2. `Casting` phase: The eligible voters V complete locally their ballot and cast it over some channel to T directly or to the BB, depending on the specification of the protocol.

3. `Tally` phase: The Ts collect the ballots either from BB or directly from the voters and produce the election outcome.

Note that there are special types of e-voting protocols where the special role of the talliers is played by the voters themselves. This type of protocol is called self-tallying Kiayias and Yung (2002); Fujioka et al. (1993).

2.3.2 Security definitions

Every e-voting protocol in order to be considered secure by today's cryptographic standards Szepieniec and Preneel (2015); Cortier et al. (2016); Groth (2004); Bernhard et al. (2015); Canetti (2001a) should satisfy various properties. The most important are:

1. `Correctness`: Only eligible voters are allowed to vote (*eligibility*) and at most once (*one-voter-one-vote*) Arapinis et al. (2018). In addition, no voter can alter the votes of other voters.

2. `Privacy`: The vote of a voter remains confidential even after the end of the election procedure Bernhard et al. (2015).

3. `Coercion resistance/receipt freeness`: Voters cannot prove the way they voted to a passive coercer (*receipt freeness*) or an active one (*coercion resistance*) Delaune et al. (2006); Okamoto (1998); Juels et al. (2005).

4. `Verifiability`: A voter should be able to verify that their vote has been counted (*individual verifiability*), the election outcome is produced based on the collected ballots (*universal verifiability*), the counted ballots are originated only from eligible voters (*eligibility verifiability*) Kiayias et al. (2015); Cortier et al. (2016).

2.4 Protocol security

So far we mentioned which security requirements an e-voting protocol should satisfy but without formally defining them. In this section, we present two approaches towards defining security rigorously.

2.4.1 Game-based definitions

Overview: The first approach is known as the *game-based* approach. In this paradigm, a security definition is formalised as an experiment between a challenger Ch and an adversary $\mathcal{B}$ and sometimes an oracle $\mathcal{O}$. Both Ch and $\mathcal{B}$ are instantiated with the security parameter λ. Throughout the experiment, $\mathcal{B}$ can issue queries to both Ch and the oracle. At some point, $\mathcal{B}$ requests a challenge from Ch, to which Ch responds. $\mathcal{B}$ tries to solve the challenge and supplies his response to Ch. If the answer to the challenge is correct, the experiment outputs the bit 1, else it outputs 0. We say that the protocol satisfies the game-based definition if the adversary wins the game with negligible probability in λ (or probability equal to $1/2 + \mathsf{negl}(\lambda)$ if $\mathcal{B}$ has to choose between only two possible answers for the challenge).

In a nutshell, such an experiment tries to capture the interaction a malicious entity has with the real protocol but in terms of an experiment/game. Of course, if an experiment really captures the intended property is a subtle modelling task and an active research area Bernhard et al. (2015) where new definitions try to improve the older ones in terms of completeness and security. Specifically, the game-based Definition 1 in Bernhard et al. (2015) for privacy does not capture protocols where the voters vote with weights for the candidate of their choice (weighed tally function). That means that even if a protocol satisfies this game-based definition there might still be attacks that intuitively violate privacy and that the formal definition does not capture. We present Definition 1 of privacy as it appears in Bernhard et al. (2015) below:

Definition 2.4.1 (IND-BB)**.** Let I be the list of voters identity, $\mathbb{V}$ the candidate slate and $\mathsf{BB}_0, \mathsf{BB}_1$ are lists initialized at empty. The challenger starts by picking a random bit β

and the adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ is given access to the lists I and BB

. The challenger runs the setup algorithm and the keys (pk, sk) are created. The adversary $\mathcal{B}_1$ for voter $\{id, id_0, id_1\} \subseteq \mathbb{V}$ can repeatedly query the oracle $\mathcal{O}_{\mathsf{cast}}$ as follows:

- $\mathcal{O}_{\mathsf{cast}}(id, b)$: It runs $bb \rightarrow bb||b$ on ballot boxes BB_0 and BB_1. (The expression $bb||b$ appends b to bb.)

The adversary can also query once the oracle $\mathcal{O}_{\mathsf{VoteIND}}$ as follows:

- $\mathcal{O}_{\mathsf{VoteIND}}(id_0, id_1, v_0, v_1)$: If $v_\delta \in \mathbb{V}$ for $\delta = 0, 1$, it halts. Else, it runs $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0||\{\mathsf{Vote}(id_0, v_0), \mathsf{Vote}(id_1, v_1)|\}$ as well as $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1||\{\mathsf{Vote}(id_0, v_1), \mathsf{Vote}(id_1, v_0)\}$, where Vote is the ballot generation function that accepts as input the voters id and her voting choice.

At some point, the adversary $\mathcal{B}_1$ asks to see the result along with the proof of correct tallying. The challenger computes $(r, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}_\beta, sk)$, and handles the tally Π and the proof r to $\mathcal{B}_1$. Finally the IND-BB adversary $\mathcal{B}_2$ outputs β'

as the guess for β. Formally, we say that a voting scheme $\mathcal{V}$ is *IND-BB secure* if no *probabilistic polynomial time* (PPT) algorithm $\mathcal{B}$ can distinguish between the outputs in the experiment just described for $\beta = 0$ and $\beta = 1$, i.e. for any PPT adversary $\mathcal{B}$ it holds that:

$$|\Pr[EXP_{\mathcal{B}}^{indbb,0}(\lambda) = 1] - \Pr[EXP_{\mathcal{B}}^{indbb,1}(\lambda) = 1]| = \mathsf{negl}(\lambda)$$

where $EXP^{indbb,\beta}$ is the experiment defined above.

The problem that rises with the above definition is that for tally functions where the result does not define the set of votes in a unique way attacks can be missed. For instance, an adversary might not be able to distinguish the case $((V_1, v_1), (V_2, v_2))$ from $((V_1, v_2), (V_2, v_1))$ but she might be able to distinguish $((V_1, 3v_2), (V_2, v_2))$ from $((V_1, 2v_2), (V_2, 2v_2))$ where each voter can include a weight in their choice. The above definition does not capture this attack.

2.4.2 Universal composability

Overview: The second approach is called *Universal Composability* (UC) paradigm introduced by Canetti in Canetti (2001b), which is the state-of-the-art cryptographic model for arguing about the security of protocols when run under concurrent sessions. In the UC framework, the parties engage in a protocol session (labelled by a unique session ID, sid) modelled as *interactive Turing Machines* (ITMs) that communicate in the presence of an *adversary* ITM $\mathcal{A}$ that may control some of the parties. The protocol execution is scheduled by an *environment* ITM $\mathcal{Z}$ that provides parties with inputs and may interact arbitrarily with $\mathcal{A}$. The intuition here is that: i) $\mathcal{Z}$ captures the external "observer" that aims to break security by interacting with the protocol interface during session sid, while ii) $\mathcal{A}$ plays the role of the "insider" that helps $\mathcal{Z}$ via any possible information it can obtain by engaging in the session in the back-end of the current execution.

The UC security of a protocol Π follows the *real-world/ideal-world indistinguishability* approach. Namely, security is captured via a special *ideal protocol* that has the same interface as Π that $\mathcal{Z}$ interacts with, but now the parties are "dummy", in the sense that they only forward their inputs provided by $\mathcal{Z}$ to an *ideal functionality* $\mathcal{F}$, which is in the centre of the back-end (i.e., the ideal protocol has a star topology) and *does not interact* with $\mathcal{Z}$ directly. The ideal functionality $\mathcal{F}$ formalizes a trusted party carrying out the task that Π intends to realize (e.g. secure communication, key agreement, authentication, etc.). The functionality $\mathcal{F}$ interacts with the adversary present in the ideal protocol, usually called a *simulator* $\mathcal{S}$, and this interaction results in a "minimum leakage of information" that determines the ideal level of security that *any protocol* realizing said task should satisfy (not only Π). E.g. if $\mathcal{F}$ formalizes an ideal secure channel, then the minimum leakage could be the ciphertext length. In case that $\mathcal{Z}$ gives input to a corrupted party P in the ideal world, the functionality $\mathcal{F}$ passes that

message to $\mathcal{S}$ and returns to P whatever it receives from $\mathcal{S}$. In both executions, if a party has the token and halts, then by convention the token is passed to the environment. We say that the real-world protocol is UC-secure if no environment $\mathcal{Z}$ can distinguish its execution from one of the ideal protocols managed by $\mathcal{F}$.

More formally, let $\text{EXEC}^{\Pi}_{\mathcal{Z},\mathcal{A}}$ denote an execution of a real-world protocol Π in the presence of the adversary $\mathcal{A}$ scheduled by an environment $\mathcal{Z}$, and $\text{EXEC}^{\mathcal{F}}_{\mathcal{Z},\mathcal{S}}$ denote an execution of the ideal protocol managed by $\mathcal{F}$ in the presence of a simulator $\mathcal{S}$, again scheduled by $\mathcal{Z}$. The UC security of Π is defined as follows.

Definition 2.4.2 (UC realization Canetti (2001b))**.** The protocol Π is said to *UC-realize* the ideal functionality $\mathcal{F}$ if for any PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that for any PPT environment $\mathcal{Z}$, the random variables $\text{EXEC}^{\Pi}_{\mathcal{Z},\mathcal{A}}$ and $\text{EXEC}^{\mathcal{F}}_{\mathcal{Z},\mathcal{S}}$ are computationally indistinguishable.

Composition and modularity: Perhaps the most prominent feature of the UC paradigm is the preservation of the security of a protocol that runs concurrently with other protocol instances, or as a subroutine of another (often more complex) execution. In particular, assume a protocol Π that UC-realizes an ideal functionality $\mathcal{F}$ according to Definition 2.4.2, and is used as a subroutine of a "larger" protocol $\tilde{\Pi}$. Then, UC guarantees that if we replace any instance of Π with $\mathcal{F}$, we obtain a "hybrid" protocol, denoted by $\tilde{\Pi}^{\Pi\to\mathcal{F}}$, that enjoys the same security as $\tilde{\Pi}$. Namely, if $\tilde{\Pi}$ UC-realizes some ideal functionality $\tilde{\mathcal{F}}$, then so does $\tilde{\Pi}^{\Pi\to\mathcal{F}}$.

The power of composition facilitates the design and analysis of complex cryptographic schemes with a *high-degree of modularity*. Namely, the scheme's formal description can be over the composition of ideal modules that are concurrently executed as subroutines. When a protocol Π using the functionalities $\mathcal{F}_1,\ldots,\mathcal{F}_k$ UC-realizes a functionality $\mathcal{F}$, we say that it does so in the $\{\mathcal{F}_1,\ldots,\mathcal{F}_k\}$-*hybrid model* and we write $\Pi^{\mathcal{F}_1,\ldots,\mathcal{F}_k}$ to clearly denote the hybrid functionalities. For instance, an e-voting system Π_{vote} can be described using the ideal functionalities $\mathcal{F}_{\mathsf{sc}}$, $\mathcal{F}_{\mathsf{auth}}$ and $\mathcal{F}_{\mathsf{BB}}$ that formalize the notions of a secure channel, an authenticated channel, and a Bulletin Board, respectively. In this case, we say that Π_{vote} is UC-secure in the $\{\mathcal{F}_{\mathsf{sc}},\mathcal{F}_{\mathsf{auth}},\mathcal{F}_{\mathsf{BB}}\}$-*hybrid model* and we write $\Pi^{\mathcal{F}_{\mathsf{sc}},\mathcal{F}_{\mathsf{auth}},\mathcal{F}_{\mathsf{BB}}}_{\mathsf{vote}}$ to clearly denote the hybrid functionalities. Furthermore, composition allows us to extend the secure modular design into multiple ($\mathsf{poly}(\lambda)$ many) layers, since a protocol that uses a hybrid functionality as a subroutine may itself be the subroutine of another protocol of an "upper layer" until we reach the level of the root ideal protocol (in our example, an ideal e-voting functionality $\mathcal{F}_{\mathsf{vote}}$).

A major advantage of the modular design in UC is that we can describe the cryptographic scheme in a partially abstract black-box manner by simply using the hybrid functionalities instead of the associated real-world subroutines, and then prove security in the corresponding hybrid model. Henceforth, whenever the designer wishes to improve the scheme by implementing a more efficient version of some subroutine, they just need to prove the security of the new subroutine

w.r.t. the associated hybrid functionality, which will directly imply the overall security of the improved scheme.

Party corruption: In UC the environment informs the adversary which parties to corrupt by sending him the corruption vector. One problem addressed in the current version of Canetti (2000), is that the simulator corrupts all the parties, despite the instructions received from the environment, making the UC realisation always possible, even for an obviously insecure protocol. Canetti tackled this flaw by allowing the environment to gain some partial information about the corrupted parties. So, if the simulator decides to corrupt a different set of parties than the one instructed by the environment, then $\mathcal{Z}$ can trivially distinct the real from the ideal execution. In this thesis, we assume this model. We do not refer to it for simplicity and should be clear for the context. It is worth mentioning that the composition theorem does not preserve receipt freeness/incoercibility Alwen et al. (2015).

2.4.3 Setup functionalities

In the UC literature, hybrid functionalities do not only play the role of abstracting some UC-secure real-world subroutine (e.g. a secure channel), but also formalize possible setup assumptions that are required to prove security when this is not done (and in many cases even impossible to achieve) in the "standard model". For example, this type of setup functionalities may capture the concept of a trusted source of randomness, a clock, or a Public Key Infrastructure (*PKI*). Moreover, these setup functionalities can be *global*, i.e. they act as shared states across multiple protocol instances and they can be accessed by other functionalities and even the environment that is external to the current session (recall that standard ideal functionalities do not directly interact with the environment). The extension of the UC framework in the presence of global setups has been introduced by Canetti *et al.* in Canetti et al. (2007a). Below, we present the setup functionalities that we consider across this work.

- The *global clock* functionality $\mathcal{G}_{\mathsf{clock}}$ similar to Badertscher et al. (2017) (see Figure 2.1).
- The *random oracle* functionality $\mathcal{F}_{\mathsf{RO}}$, as defined in Nielsen (2002) (see Figure 2.2).
- The *common reference string* functionality $\mathcal{F}_{\mathsf{CRS}}$, as given by Canetti (2001b) (see Figure 2.3).
- The *anonymous broadcast* functionality $\mathcal{F}_{\mathsf{an.BC}}$. To guarantee the privacy of our STE scheme we assume that voters communicate via an *anonymous*

broadcast channel. This communication interface is formalized via the functionality $\mathcal{F}_{\mathsf{an.BC}}$ in Figure 2.5.

– The *certification* functionality $\mathcal{F}_{\mathsf{cert}}$, as defined in Canetti (2001b) (see Figure 2.6).

The global clock functionality $\mathcal{G}_{\mathsf{clock}}$: In Figure 2.1, we provide the definition of a *global clock* functionality $\mathcal{G}_{\mathsf{clock}}$ similar to Badertscher et al. (2017). Time advances only when the environment has allowed all involved parties to advance Katz et al. (2013); Badertscher et al. (2017).

The Global Clock functionality $\mathcal{G}_{\mathsf{clock}}(\mathbf{P}, \mathbf{F})$.
For each session sid, the functionality initializes the global clock variable $\mathsf{Cl} \leftarrow 0$ and the set of advanced parties per round as $L_{\mathsf{adv}} \leftarrow \emptyset$.
■ Upon receiving $(\mathsf{sid}, \text{ADVANCE\_CLOCK})$ from $P \in \mathbf{P}$, if $P \notin L_{\mathsf{adv}}$, then it adds P to L_{adv}. If $L_{\mathsf{adv}} = \mathbf{P} \cup \mathbf{F}$, then it updates as $\mathsf{Cl} \leftarrow \mathsf{Cl} + 1$ and resets $L_{\mathsf{adv}} \leftarrow \emptyset$. It notifies $\mathcal{A}$ by forwarding $(\mathsf{sid}, \text{ADVANCE\_CLOCK}, P)$.
■ Upon receiving $(\mathsf{sid}, \text{ADVANCE\_CLOCK})$ from $\mathcal{F} \in \mathbf{F}$, if $\mathcal{F} \notin L_{\mathsf{adv}}$, then it adds $\mathcal{F}$ to L_{adv} and sends the message $(\mathsf{sid}, \text{ADVANCE\_CLOCK})$ to $\mathcal{F}$. If $L_{\mathsf{adv}} = \mathbf{P} \cup \mathbf{F}$, then it updates as $\mathsf{Cl} \leftarrow \mathsf{Cl} + 1$ and resets $L_{\mathsf{adv}} \leftarrow \emptyset$.
■ Upon receiving $(\mathsf{sid}, \text{READ\_CLOCK})$ from $X \in \mathbf{P} \cup \mathbf{F} \cup \{\mathcal{Z}, \mathcal{A}\}$, then it sends $(\mathsf{sid}, \text{READ\_CLOCK}, \mathsf{Cl})$ to X.

Figure 2.1: The global clock functionality $\mathcal{G}_{\mathsf{clock}}(\mathbf{P}, \mathbf{F})$ interacting with the parties in $\mathbf{P}$, the functionalities in $\mathbf{F}$, the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$.

That is the standard way of capturing synchronicity in the UC model. Namely, $\mathcal{G}_{\mathsf{clock}}$ is publicly accessible by all entities, and time advances only when the environment has allowed all involved parties to advance. Intuitively, UC synchronicity suggests that the environment must respect the synchronization reference points, yet between consecutive points the protocol flow may be adversarially scheduled.

The Random Oracle functionality $\mathcal{F}_{\mathsf{RO}}(\mathbf{P}, A, B)$.

The functionality initializes a list $L_{\mathcal{H}} \leftarrow \emptyset$.

- Upon receiving $(\mathsf{sid}, \text{QUERY}, x)$ from $P \in \mathbf{P}$, if $x \in A$, then

  1. If there exists a pair $(x, h) \in L_{\mathcal{H}}$, it returns $(\mathsf{sid}, \text{RANDOM\_ORACLE}, x, h)$ to P.

  2. Else it picks $h \xleftarrow{\$} B$, and it inserts the pair to the list $L_{\mathcal{H}} \leftarrow (x, h)$. Then it returns $(\mathsf{sid}, \text{RANDOM\_ORACLE}, x, h)$ to P.

Figure 2.2: The random oracle functionality $\mathcal{F}_{\mathsf{RO}}$ w.r.t. domain A and range B interacting with the parties in $\mathbf{P}$.

The random oracle functionality $\mathcal{F}_{\mathsf{RO}}$: In Figure 2.2, we define a UC *random oracle* (RO) as in Nielsen (2002), a setup assumption widely used in the security analysis of efficient protocols. Like an RO, $\mathcal{F}_{\mathsf{RO}}$ behaves like a truly random function, by providing random yet consistent responses to evaluation queries (i.e. multiple queries for the same pre-image x from domain set A result in the same response h from the range set B).

The common reference string functionality $\mathcal{F}_{\mathsf{CRS}}$: Another setup assumption is the *common random string* model, where a single random string is drawn from a uniform distribution over strings. Figure 2.3 formally defines $\mathcal{F}_{\mathsf{CRS}}$ as given by Canetti (2001b). Note that $\mathcal{F}_{\mathsf{CRS}}$ requests permission from the simulator $\mathcal{S}$ before returning the value, disclosing the identity of the requesting party. This action is often called *(public) delayed output* in the literature. The *common reference string (CRS)* model generalizes to arbitrary distributions, which can be over parameters that need to be shared in some multi-party protocol. Realizing $\mathcal{F}_{\mathsf{CRS}}$ is by itself not a trivial task, so we do not choose a specific protocol. However, there have been efforts to relax the definition to allow for more practical implementations, e.g. Canetti et al. (2007b).

The anonymous broadcast functionality $\mathcal{F}_{\mathsf{an.BC}}$: To guarantee the privacy of our STE scheme we assume that the users communicate via an *anonymous broadcast channel*, i.e. every user transmits her messages to all other users without disclosing anything more than her eligibility to participate in the election and the validity of her ballot. This communication interface is formalized via the functionality $\mathcal{F}_{\mathsf{an.BC}}$ described in Figure 2.5. We stress that as in $\mathcal{F}_{\mathsf{CRS}}$, our formalization captures adversaries that can block communication at will via *private delayed output*, i.e. here $\mathcal{S}$ does not learn the identities of the parties

who send the messages. With this, we capture the control of the adversary over the network to block messages without learning the identity of the party. For example, the adversary might control some routing nodes, and thus be able to stop the transmission of any message. Providing a provably UC-secure realization of either $\mathcal{F}_{\mathsf{an.BC}}$ or $\mathcal{F}_{\mathsf{BC}}$ is out of the scope of this work. However, intuitively, one can instantiate an anonymous broadcast channel by deploying a bulletin board or a Blockchain (or any transaction ledger) where the users access the Blockchain via an anonymous communications channel such as Tor or any mix-network routing protocol.

The broadcast functionality $\mathcal{F}_{\mathsf{BC}}$: We make use of a broadcast channel to broadcast to the other parties the resulting ciphertext that includes the time-lock puzzle. The reason behind this design decision was that the parties need to start solving immediately the puzzle by the time of its creation. The communication interface is formalized via the functionality $\mathcal{F}_{\mathsf{BC}}$ described in Figure 2.5. We stress that our formalization captures adversaries that can block communication at will via *public delayed output*, i.e., the simulator $\mathcal{S}$ learns the identities of the parties who send the messages. Providing a provably UC-secure realization of $\mathcal{F}_{\mathsf{BC}}$ is out of the scope of this work. However, there are works that present constructions Khisti et al. (2008); Hirt and Zikas (2010) and provide simulation security Hirt and Zikas (2010) in the secure channel model.

The broadcast functionality is parameterized by a set of parties $\mathbf{P}$ and it is along the lines of Goldwasser and Lindell (2005).

The common reference string functionality $\mathcal{F}_{\mathsf{CRS}}(\mathbf{P}, D)$.

The functionality initializes a waiting list $L_{\mathsf{wait}} \leftarrow \emptyset$.

- Upon receiving $(\mathsf{sid}, \mathrm{CRS})$ from $P_i \in \mathbf{P}$ it does:
  1. If no value r is recorded then it picks $r \xleftarrow{\$} D$.
  2. Adds P_i to L_{wait} and sends $(\mathsf{sid}, \textsc{Allow}, P_i)$ to $\mathcal{S}$.
- Upon receiving $(\mathsf{sid}, \textsc{Allowed}, P_i)$ from $\mathcal{S}$, if $P_i \in L_{\mathsf{wait}}$, it sends $(\mathsf{sid}, \mathrm{CRS}, r)$ to P_i and $\mathcal{S}$ and removes P_i from L_{wait}.

Figure 2.3: The CRS functionality $\mathcal{F}_{\mathsf{CRS}}$ interacting with the parties in $\mathbf{P}$ and the simulator $\mathcal{S}$, parameterized by distribution D.

The Anonymous Broadcast functionality $\mathcal{F}_{\mathsf{an.BC}}(\mathbf{P})$.

The functionality initializes a list $L_{\mathsf{pend}} \leftarrow \emptyset$ of messages pending to be broadcast.

- Upon receiving $(\mathsf{sid}, \textsc{Broadcast}, M)$ from $P_i \in \mathbf{P}$, it adds (M, P_i) to L_{pend} and sends $(\mathsf{sid}, \textsc{Allow}, M)$ to $\mathcal{S}$.
- Upon receiving $(\mathsf{sid}, \textsc{Allowed}, M)$ from $\mathcal{S}$, if $(M, P_i) \in L_{\mathsf{pend}}$, then it sends $(\mathsf{sid}, \textsc{Broadcast}, M)$ to $P_1, \ldots, P_n$, and $\mathcal{S}$. Then, it removes (M, P_i) from L_{pend}.

Figure 2.4: The anonymous broadcast functionality $\mathcal{F}_{\mathsf{an.BC}}$ interacting with the parties in $\mathbf{P} = \{P_1, \ldots, P_n\}$ and the simulator $\mathcal{S}$.

The Broadcast functionality $\mathcal{F}_{\mathsf{BC}}(\mathbf{P})$.

- Upon receiving $(\mathsf{sid}, \textsc{Broadcast}, M)$ from $P \in \mathbf{P}$, it sends $(\mathsf{sid}, \textsc{Broadcast}, P, M)$ to $\mathcal{S}$.
- Upon receiving $(\mathsf{sid}, \textsc{Allow\_Broadcast}, P, M)$ from $\mathcal{S}$, it sends $(\mathsf{sid}, \textsc{Broadcast}, M)$ to all $P^* \in \mathbf{P} \setminus P$ and $\mathcal{S}$ and $(\mathsf{sid}, \textsc{Broadcasted}, M)$ to P.

Figure 2.5: The broadcast functionality $\mathcal{F}_{\mathsf{BC}}$ interacting with the parties in $\mathbf{P} = \{P_1, \ldots, P_n\}$.

The certification functionality $\mathcal{F}_{\mathsf{cert}}(P, \mathbf{V})$.

- Upon receiving $(\mathsf{sid}, \textsc{Corrupt}, \mathbf{V}_{\mathsf{corr}})$ from $\mathcal{S}$, if $\mathbf{V}_{\mathsf{corr}} \subseteq \mathbf{V} \cup P$, it fixes $\mathbf{V}_{\mathsf{corr}}$ as the set of corrupted parties.
- Upon receiving $(\mathsf{sid}, \textsc{Setup})$ from P, if $\mathsf{sid} = (P, \mathsf{sid}')$ for some sid', it sends $(\mathsf{sid}, \textsc{Setup})$ to $\mathcal{S}$.
- Upon receiving $(\mathsf{sid}, \textsc{Algorithms}, \texttt{Verify}, \texttt{Sign})$ from $\mathcal{S}$, it stores the algorithms and sends $(\mathsf{sid}, \textsc{Setup})$ to P.
- Upon receiving $(\mathsf{sid}, \textsc{Sign}, m)$ from P, it sets $\sigma \leftarrow \texttt{Sign}(m)$. If $\texttt{Verify}(m, \sigma) \neq 1$, it aborts. Otherwise it records (m, σ) and sends $(\mathsf{sid}, \textsc{Signature}, m, \sigma)$ to P.
- Upon receiving $(\mathsf{sid}, \textsc{Verify}, m, \sigma)$ from $V \in \mathbf{V}$, if $\texttt{Verify}(m, \sigma) = 1$, the signer is not corrupted and no entry (m, σ') for any σ' is recorded, it aborts. Otherwise it sends $(\mathsf{sid}, \textsc{Verified}, m, \texttt{Verify}(m, \sigma))$ to V.

Figure 2.6: The certification functionality $\mathcal{F}_{\mathsf{cert}}$ interacting with a prover P, a set of verifiers $\mathbf{V}$ and the simulator $\mathcal{S}$.

The certification functionality $\mathcal{F}_{\mathsf{cert}}$: The registration process for voting traditionally makes use of a channel through which the voters can identify

themselves to the election authority. It would be natural to utilize a form of public key infrastructure, but modelling it in UC is not straightforward, and turns out to not be necessary for the protocol that will be introduced in Subsection 5.4.3. Instead, we will use a certification scheme that provides signatures not bound to keys, but *identities*. $\mathcal{F}_{\mathsf{cert}}$, shown in Figure 2.6 as defined by Canetti (2001b) (2005), provides commands for signature generation and verification and is tied to a single party (so each party requires a separate instance). It can be realized as in Canetti (2003) by a EUF-CMA secure signature scheme combined with a party acting as a trusted certificate authority.

In this thesis, to make the notation simpler sometimes we refer to an ideal functionality without all of its inputs (e.g $\mathcal{F}_{\mathsf{test}}$ instead of $\mathcal{F}^{x}_{\mathsf{test}}(V)$). In the full description of the functionality, we refer to it with all of its input.

Chapter 3

Literature review

In this section, we present the state-of-the-art both for classical and quantum protocols. Specifically, in both classical and quantum e-voting we present solutions and we mention some of their limitations in terms of security and efficiency.

3.1 Classical e-voting

In the next subsections, we present the two main types of e-voting and the necessary cryptographic tools that are essential for the design of these systems.

3.1.1 Centralized e-voting

In centralized e-voting, tasks such as the computation of tally and the generation of the protocol's parameters are assigned to the talliers and the election authority respectively. The approach of having a variety of parties except the voters facilitates efficiency and usually results in less complicated protocols. Moreover, centralized e-voting is the most well-studied type of e-voting in the literature which makes us draw safe enough conclusions regarding the expectations and the actual features it could provide. Centralized e-voting can be divided into two categories. The first one is called *On-line e-voting* which the majority of the procedure, if not all, happens remotely from a personal computer. The second one is called *Code-based e-voting* which combines elements from both the classical paper ballot and electronic elections.

On-line e-voting: In on-line e-voting systems usually the voters receive their login details in a trusted manner Adida (2008); Juels et al. (2005); Clarkson et al. (2008); Cortier et al. (2019), e.g. in person by an authorized entity. One of the greatest advantages of online e-voting is the fact that the voters can participate remotely without the need to physically be in a specific location. This is very important as it can increase the participation rate in the election

from sensitive groups, where the access otherwise would be restrictive, and for under-representative groups, as it is more convenient for them. In Adida (2008) the authors have introduced *Helios*, a fully-fledged on-line e-voting system based on well studied cryptographic primitives such as *mix-nets* Sako and Kilian (1995) and El-Gamal encryption ElGamal (1985). Although their construction is very efficient, security exploits have been demonstrated on the first version of Helios Cortier and Smyth (2013) such as *ballot stuffing* and *replay* attacks. In principle, a ballot stuffing attack means that non-eligible voters have voted and their votes have been included in the outcome of the election (violation of *eligibility verifiability*) or that eligible voters have voted more than once. Moreover, Helios is insecure in high coercion environments where a coercer actively coerces a voter Delaune et al. (2006), e.g. by being over the voter's shoulder throughout the whole election period. The authors in Cortier et al. (2019) develop a new voting system, namely *Belenios*, where it surpasses many of the previous limitations like the ballot stuffing attack. Still, Belenios is vulnerable in high coercion environments like Helios. Another online e-voting protocol is proposed in Juels et al. (2005); Clarkson et al. (2008) that tackles these limitations. The novelty of these systems is the use of the *plaintext equivalence test* (PET) method which allows a party to verify if the underline plaintext of two ciphertexts is the same without opening the ciphertexts. The authors argue about the security of their protocol by presenting two game-based definitions, one for the correctness and the other for the coercion resistance property. The authors show that their protocol satisfies both of them. The biggest drawback with their construction is the fact that it is inefficient for large scale elections as the time complexity depends on the number of voters in a more than linear way.

Code based e-voting: A code-based e-voting scheme Chaum et al. (2009); Ryan and Schneider (2006); Kiayias et al. (2015) is a semi-electronic e-voting setting in the sense that voters still need to use a voting booth to vote but with electronic means, such that optical scanners. Still, even if not clearly fully electronic it shares many of the advantages of pure digital systems such as low election costs, enhanced security guarantees compared to the paper ballot voting system and efficient tallying. In Chaum et al. (2009); Ryan and Schneider (2006) the authors use optical scanners Chaum et al. (2009) and invisible ink Ryan and Schneider (2006) so that: i) a voter can participate in the election procedure; ii) their construction satisfies essential security guarantees. Besides the security claims, none of these works come with rigorous security definitions and a well-articulated security model. The construction of Kiayias et al. (2015) is based on previous constructions Chaum (2001, 2004). One very important novelty of Kiayias et al. (2015) is the fact that their construction is in the *standard model*. This means that the security of their protocol does not rely on the *random oracle* (RO), which can be instantiated by a cryptographic hash function, or a *common reference sting* (CRS), where a special party provides a string from

a specific probability distribution in a trusted manner. Moreover, the authors establish their protocol's security against game-based definitions. Specifically, they introduced the notion of *end-to-end* (E2E) verifiability, a novel security definition for universal verifiability showing that their construction satisfies it. Despite that, their protocol is insecure in high coercion environments.

3.1.2 Self-tallying protocols

The first self-tallying protocol was from Fujioka et al. (1993) (although the authors do not state it explicitly as self-tallying). This protocol consists of an election authority EA, that is responsible only for providing a certificate to each voter V eligible to participate in the election procedure. In the `setup` phase each voter V interacts with the EA so that they can obtain their credential without EA knowing the link between the credential and the voter's identity. This is possible with blind signatures Chaum (1983). Specifically, V chooses a unique bit-string at random then blinds it and gives it to EA to sign it. After receiving the signed blind bit-string back from EA, V unblinds it and keeps it. In this way, EA does not know what she actually signs. As a result, the link between that string and the voter's identity is hidden from EA. Next, at the `casting` phase of the protocol, V commits via a commitment scheme Camenisch et al. (2016) their vote padded with their unique bit-string along with the signature of EA and broadcasts it via an anonymous channel to the other voters. Finally, during the `tally` phase each voter broadcasts their de-commit key to all other voters via an anonymous channel. As a result, each voter can open the committed values with the de-committed keys, check that the signature is issued from EA and then include that vote to the final tally given that is the first one related with that bit-string. One security issue with the previous protocol is the fact that intermediate results are leaked during the `tally` phase. As a result, malicious voters observe how the election result is created progressively and they may change their mind and don't cast their de-commit key so that they can favour one candidate over the other. So the fairness condition is not satisfied by the protocol. Another security issue is the fact that there is not any agreement between protocols phases. As a result, a voter does not know when the, e.g. `casting` phase ends.

In Kiayias and Yung (2002), which is the first to introduce the term self tallying protocol in the literature, the authors tried to address the violation of fairness condition in the self-tallying elections. This protocol consists of EA which provides the protocol parameters, a set of voters $\mathbf{V}$, a trusted bulletin board BB along with a trusted dummy party P_{D}. The presence of the dummy party is essential as it guarantees that the fairness condition is satisfied. In the `setup` phase of the protocol, the parameters are given to the voters by EA and each voter posts their public parameters to BB. Initially, all voters create a pre-voting matrix with the property that the multiplication of each row element is equal to the generator of the group which is given as a public parameter by EA. Next, in the `casting` phase each voter generates their ballot based on the pre-voting

matrix and posts it to the BB. The party P_{D} votes last so that the fairness condition be satisfied else intermediate results will be leaked to the other voters. Specifically, P_{D} can learn the election result without even post his empty ballot to BB, leading him to an obvious advantage over the other voters. However, by assumption P_{D} is a dummy party, meaning that he will not take advantage of that knowledge. Finally, in the `tally` phase the election outcome can be derived from the final matrix based on the property of each row we mentioned before. The security flaws here are i) to guarantee fairness the authors introduced the trusted party P_{D} which leads to a weaker threat model as it re-introduces a trusted party in the STE setting; ii) a single party can cause the protocol to abort. Specifically, if a party does not cast their ballot in the `casting` phase the tally cannot be produced. The authors mention that their protocol can handle such attacks if the protocol runs one extra round, as they call a *recovery* round. But if a new party deviates again during the recovery round, the protocol aborts completely making it impractical, especially in large scale elections where a new execution of the protocol is very costly in terms of time.

In Szepieniec and Preneel (2015); Groth (2004); Li et al. (2019) the authors suggest self tallying protocol based on Kiayias and Yung (2002) but with some of the limitations as mentioned above. One big difference in Li et al. (2019) is that there is not a dummy party to guarantee fairness, instead, in the `setup` phase the voters additionally commit their vote before generating and casting their ballot. If the last voter refuses to cast their ballot then similarly to the *recovery* round of Kiayias and Yung (2002) the other voters can reconstruct the vote based on the commitment from the `setup` phase, a feature that did not exist in Kiayias and Yung (2002). On the other hand, if another voter aborts then the protocol is terminated. Moreover, the authors proved their protocol's security against a new game based definition, namely *maximun ballot secrency* (MBS), which is pretty similar to Definition 2 in Bernhard et al. (2015). That means that it shares the same limitations as well; for instance, this definition does not capture vote privacy for protocols where the election result, produced by a tally function specified by the protocol, does not uniquely define the set of votes (e.g. weight elections). In Szepieniec and Preneel (2015), the authors define an ideal functionality for e-voting, name $\mathcal{F}_{\mathsf{VS}}$, that captures `correctness` and `privacy`. The authors provide a separate definition for *universal verifiability* as their functionality does not capture it. The main limitation in this work is that it allows a single voter to cause the whole election to abort. Similar, in Groth (2004) they define a voting ideal functionality and they provide a hybrid protocol that UC realises it. Both the ideal functionality and the hybrid protocol allow the adversary to decide if the voting parties will receive the election result or not.

3.1.3 Cryptographic primitives in e-voting

A common security flaw for every self-tallying protocol we mentioned is the fact that the fairness condition is not satisfied. One novelty of our work was to

search what cryptographic primitive we can use to surpass this difficulty without sacrificing the decentralized nature a self-tallying protocol provides to us. Another novelty of our work was to search for a cryptographic primitive that make our protocol satisfy `privacy` but at the same time to make it suitable for large scale elections.

For fairness: Although the commitment scheme in Okamoto (1998) guarantees that a voter cannot change their vote, after the de-commit phase a malicious voter can decide to not open its committed value and affect the final result of the election based on the intermediate results. For example, let us suppose that there are two candidates, `a`, `b` respectively, and a voting party which initially prefers the candidate `a` over `b`. During the `casting` phase, that party commits to the candidate `a`. During the `tally` phase, where each ballot is handled and opened in an individuality manner, the voter observes how the election result is formed. As a result, he might change his mind regarding his initial voting choice and refuse to open his ballot by providing his de-commitment key. As can be seen, commitment schemes guarantee that the vote of a voter remains committed to his initial choice but we can not guarantee that his ballot will eventually open. A solution for that task seems to be closer to encryption rather than commitments to achieve fairness. On the other hand, maintaining the decentralized character of the self-tallying approach is equally important, thus we need to seek cryptographic primitives in the literature so that we can handle the decryption of each ballot in a distributed manner.

Distributed key generation: Our first attempt was to use a *distributed key generation* (DKG) Gennaro et al. (2007) protocol so that the parties can generate the parameters of a threshold encryption scheme. Specifically, in a DKG protocol some parties, in our case, the voters, need to agree on a public key that is sampled from a specific distribution with integrity. Moreover, the secret key that is related to this public key should not appear explicitly anywhere. Instead, each party holds a share of it such that if m out of n parties (voters in our case) contribute their share, decryption of ciphertexts encrypted with the public key produced through a DKG session. The DKG protocol assumes that the communication between players is secure, this can be achieved with pairs of keys each voter is registered with to encrypt and sign messages. Finally, a commitment scheme is necessary to avoid trivial attacks. For example, if we do not use a commitment scheme the adversary has partial control over the newly created public key, thus the public key will be not generated uniformly at random but with some adversarial influence. So after such a public key is agreed between voters the violation of fairness can be countered by using a threshold encryption scheme Gennaro et al. (2007). Regarding the security of such a protocol, there are ideal functionalities in the literature and realizations as well. Specifically, there

are works which have proved UC security for DKG protocols based on discrete logarithm assumption, such as:

1. In Abe and Fehr (2004) the authors define the $\mathcal{F}_{\mathsf{DKG}}$ functionality and prove that their protocol *special inconsistent party* (SIP) UC realizes it. The SIP model is a little weaker than the general UC model as it considers that in all executions of the protocol a specific party remains uncorrupted. On the other hand, this party does not have any special role, it is sampled at random from the set of all parties at the beginning of the protocol. In Canetti et al. (1999) if this party is corrupted (both models consider an adaptive adversary) the simulator rewinds and picks another party. Essentially in Abe and Fehr (2004) the authors use the same proof technique of Canetti et al. (1999) but without using the rewinding technique as they assume that this special party remains uncorrupted and thus the simulator always succeeds. On the other hand, if we consider a static adversary the proof of Abe and Fehr (2004) holds for the standard UC as well.

2. Finally in Wikström (2005) the authors consider a static corruption. As a result, their protocol is more efficient. Is proven UC secure in the random string model (a specific instance of the CRS model).

In Abe and Fehr (2004) we can omit the SIP if we consider a static corruption. In their model, a simulator fails if the adversary sends a corruption message to the special party (SIP). This is happening because the simulator can not simulate that party's internal storage. So if we assume that the adversary corrupt parties at the beginning of the protocol, if the simulator picks a party outside of the corruption set defined by the adversary at the beginning of the protocol, then proof of UC realization is possible. Despite all this literature to back up our construction, we didn't choose this approach to solve the fairness problem for efficiency/practical reasons. For the voters to produce such a public key, the communication complexity is polynomial to the voters (at least quadratic). As a result, on large scale elections, this is computationally unrealistic. Moreover, an honest fraction of the voters would need to stay online until the end of the election procedure. The percentage of the fraction is a trade-off between practicality and security. If we require a "big" fraction to be needed to decrypt the ballots then this approach is secure (e.g the adversary needs to corrupt many voters to decrypt a ballot earlier and thus violate fairness) but it is not practical (e.g. many voters should remain on-line even after they cast their ballot). On the other hand, if we require a "small" fraction of voters to remain online, this approach will be practical but not secure (e.g. an adversary with a small percentage of voters under its possession can open the ballots earlier and thus violate fairness).

Time-lock encryption: All of these limitations lead us to our second approach for solving the fairness problem which is the *time-lock encryption* (TLE).

TLE is a cryptographic primitive that allows a ciphertext to be decrypted only after a specific time period has elapsed. This is possible by "hiding" the decryption key in a puzzle so that after a specific time period can be solved. The reward for solving the puzzle is the decryption key of the corresponding ciphertext. So the main purpose of the puzzle is to delay the party from opening the message before a specific amount of computation has been done. In some proposals, decryption can further be performed without requiring knowledge of any secret information Liu et al. (2018); Rivest et al. (1996). Previously proposed constructions are based either on witness encryption Garg et al. (2013) or symmetric encryption Kościelny et al. (2013). The authors of these works provide game-based definitions to argue about the security of their constructions. Unfortunately, game-based definitions do not capture the variety of adversarial behaviour the UC framework Canetti (2001b) does. Moreover, the task of transferring these definitions to the UC setting is quite challenging due to some incompatibilities between the two settings (concrete vs asymptotic adversary). More precisely, in Liu et al. (2018) the authors define a *computational reference clock* (CRC) which after specific time periods produces the secret key so that a message with the corresponding time labelling can be decrypted. In their construction, the authors instantiate the CRC with the Bitcoin ledger Badertscher et al. (2017); Garay et al. (2015); Nakamoto (2008). Specifically, they use a witness encryption scheme Liu et al. (2018) with the witness to be a length τ of Bitcoin's chain. So, every message that was encrypted with label τ can be decrypted with that part of the chain. So the CRC in that case is the Bitcoin ecosystem that is maintained by Bitcoin miners Nakamoto (2008). Moreover, they provide security arguments of their construction in a game based style in the sense that an adversary that executes t-steps cannot win the game except with small probability. If we want to argue in UC about the security of this scheme we have to consider adversaries that execute not a concrete number of steps (in this case t-steps) but instead asymptomatically polynomially many steps for arbitrary polynomials, which leads us to a completely new definition. Another TLE construction proposed in Rivest et al. (1996) which is based on a block cipher, e.g. *advanced encryption system* (AES) Daemen and Rijmen (2002), and the repeated squaring. Specifically, first, a party encrypts a message m by using AES and a secret key sampled from a keyspace uniformly at random. Then the party chooses the time that it will be needed for that key to be found and creates a "puzzle". The ciphertext is the encrypted message with the AES and the time puzzle. In Rivest et al. (1996) the authors claim that their construction is secure under the computational problem they have introduced (the repeated squaring). In their work, they have not provided any formal treatment of security. Another drawback with their construction is the fact that a party to solve the puzzle must be engaged in mathematical computations in contrast with the construction in Liu et al. (2018) where the solution of the puzzle is announced after a specific time. The only way that these computations could be avoided for the puzzles to be solved is for the issuer of the puzzle to announce

the solution (the solution is verifiable). This is the optimal case. A similar TLE construction is Mahmoody et al. (2011) where the time-lock puzzle is based on hash function evaluations. Specifically, the solver of the puzzle is engaged in serial hash evaluations until solving the puzzle. Similar to Rivest et al. (1996), if some party presents the solution of the puzzle any other party can verify it efficiently by doing all the hash evaluations in parallel. A drawback of this construction is that parts of the plaintext are revealed before the completion of the puzzle. There are TLE constructions Cheon et al. (2006); May (1993) where instead of relying on computational puzzles there is a *trusted third party* (TTP) who is responsible for announcing the witnesses. Most of these constructions are based on *public key infrastructure* (PKI). An obvious drawback with this type of TLE scheme is the fact that we ground a big part of security in the TTP, which in turn leads to weaker threat models. We distinguish these three categories of TLE schemes as *client free* Liu et al. (2018); where the computational burden of solving the time lock puzzle is not part of the description of the client, *client optimal free* Rivest et al. (1996); Mahmoody et al. (2011); where the computational burden of solving the time lock puzzle is part of the description of the client but if the solution is revealed then it can be verified efficiently, and *trusted agent* Cheon et al. (2006); May (1993); where a trusted party announces secret keys after some period of time. Last, the *client free* and *client optimal free* constructions are subject to malleability.

Although there are formal treatments of TLE in the literature Liu et al. (2018), these mainly provide standalone models of security while our work aims to provide a composable treatment of the TLE primitive. The only other such attempt to our knowledge is a recently published paper Baum et al. (2021). We present a detailed comparison in the next Subsection.

3.1.4 Comparison with Baum et al. (2021) and Baum et al. (2020)

A concurrent and independent work closely related to ours was very recently published at EUROCRYPT 2021 Baum et al. (2021), with a subsequent work seemingly in preparation Baum et al. (2020). In particular, Baum et al. (2021) proposes a composable treatment in the UC framework of time-lock puzzles whose security is captured by the ideal functionality $\mathcal{F}_{\mathsf{tlp}}$. It further proves how the scheme proposed by Rivest *et al.* in Rivest et al. (1996) can be used to UC realise $\mathcal{F}_{\mathsf{tlp}}$ in both the random oracle and generic group models. Their realisation, like ours, relies on techniques for equivocation borrowed from Nielsen (2002) and Camenisch et al. (2017). They further show that no time-lock puzzle is UC realizable outside the random oracle model. Finally, they show that time-lock puzzles can be used to ensure fairness in coin-flipping protocols.

The time-lock scheme proposed in Baum et al. (2021) is not verifiable. This is addressed in the subsequent pre-print Baum et al. (2020) where they adapt the

scheme to include the trapdoor information along the message to be time-lock encrypted, rendering it verifiable. There are some key differences between these two works and ours, rendering the proposed treatments of time-lock primitives orthogonal. The premises and assumptions are intrinsically different and capture different concepts and security notions. We discuss these differences here and argue why our formal treatment of time-lock encryption, and our proposed TLE scheme, namely Astrolabous, are preferable in some respects and more suited to many scenarios.

Apprehending time with computational puzzles

In Baum et al. (2021) and Baum et al. (2020), a resolutely different approach to ours is taken, when it comes to real-time. In particular, they introduce the global $\mathcal{G}_{\mathsf{ticker}}$ functionality to capture delays without referring to a global *"wall clock"*, and thus without referring to real-time.

We, on the other hand, insist on the importance of closely relating computational time and real-time and propose an alternative treatment in the global clock model ($\mathcal{G}_{\mathsf{clock}}$). Our approach is directly motivated by the seminal paper Rivest et al. (1996), in which R. L. Rivest, A. Shamir, and D. Wagner introduce the very concept of *time-release cryptography* to capture encryption schemes that ensure encrypted messages cannot be decrypted until a set amount of time has elapsed. The goal is to, as they put it, *"send information into the future [...] by making CPU time and real-time agree as closely as possible"*.

This is key to explaining why and how time-release cryptography is used in an increasing number of distributed applications, and in particular, schemes hinged on *computational puzzles*, *i.e.* puzzles that can only be solved if certain computations are performed continuously for at least a set amount of time. Indeed, the cryptographic protocols underlying these applications often rely on temporally disjoint phases. Time-release cryptographic primitives, as primitives apprehending real-time through computations, allow thus these temporally disjoint stages of the protocol to be enforced yet in an asynchronous manner.

This is reflected in our protocol realising the proposed ideal TLE functionality $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$. Parties only read the time from the global clock $\mathcal{G}_{\mathsf{clock}}$ to compute the amount of time the ciphertext needs to be protected for, and infer the corresponding puzzle difficulty. Decryption however requires continuous computations being performed until the set opening time is reached, and no read command being ever issued to $\mathcal{G}_{\mathsf{clock}}$. This protocol clearly demonstrates how time-lock puzzles apprehend real-time through computations.

In contrast, the protocol π_{tlp} realising the ideal time lock-puzzle functionality $\mathcal{F}_{\mathsf{tlp}}$ proposed in Baum et al. (2021) does not instruct parties to continuously work towards solving received puzzles (the scheduling of each step for solving a puzzle is left to the environment). So the treatment proposed in Baum et al. (2021) and Baum et al. (2020) leaves it to the protocol using π_{tlp} or $\mathcal{F}_{\mathsf{tlp}}$ as a subroutine to correctly takes care of appropriately enforcing relative delays between key events.

Ideal functionality and realisation

$\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ is more general than $\mathcal{F}_{\mathsf{tlp}}$. $\mathcal{F}_{\mathsf{tlp}}$ only captures constructions that rely on computational puzzles for "hiding" a message. In contrast, our time-lock encryption functionality $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ does not. As such it can cater for TLE schemes that do not rely on time-lock puzzles at all, such as the centralized solutions proposed in Cheon et al. (2006); May (1993) where a Trusted Third Party realises the solution in specific time-slots.

Moreover, some constructions such as Liu et al. (2018) allow the adversary an unavoidable advantage in solving TLE puzzles (*e.g.*, the adversary synchronizes faster than the honest parties in the Bitcoin network Garay et al. (2015); Badertscher et al. (2017)). $\mathcal{F}_{\mathsf{tlp}}$ does not capture such constructions. Our $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ functionality is parameterized with a leakage function, which specifies exactly the advantage of the adversary in each case.

Turning now to the realisations of UC secure time-lock primitives, the realisation of $\mathcal{F}_{\mathsf{tlp}}$ proposed in Baum et al. (2021) relies on stronger assumptions as it relies both on the random oracle model and the generic group model. In contrast, our realisation of $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ only relies on the random oracle model.

On public verifiability

While the time-lock encryption scheme proposed in Baum et al. (2020) is publicly verifiable in the sense that given a puzzle, the verifying party does not need to solve the puzzle for themselves to verify that an announced solution for that puzzle is valid. This is not enough in some scenarios. For instance, consider the scenario with a dedicated server to be the puzzle solver and all other parties to be "lite" verifiers. This is very realistic given the computational requirements for solving puzzles. For efficiency, one would let a server solve the puzzles and only check that the solutions it provided are valid ones. Now in such a scenario parties *i)* would not trust the server, *ii)* would not trust the issuer of the puzzle either, but *iii)* are also not willing to solve the puzzle themselves.

Now, in Baum et al. (2020) public verifiability is achieved because the issuer of the puzzle concatenates the message and the trapdoor information, which is the factorization of N. Given the trapdoor, one can efficiently verify that the announced solution to the puzzle is valid. However, the trapdoor announced (dishonest server) or the trapdoor included (dishonestly generated ciphertext) might not be valid for the puzzle. The only way to identify the dishonest party is to solve the puzzle for oneself and check it against the solution to the puzzle announced by the server. If they match, then the ciphertext was dishonestly generated, otherwise, the server is dishonest.

This is reflected in the public verifiability notion that $\mathcal{F}_{\mathsf{tlp}}$ captures that is one-sided: if an announced solution to a puzzle is valid, then the verification is successful. But if the verification fails, then some party has deviated from the protocol but it could either be the server or the issuer of the ciphertext.

In contrast, the solution of our puzzle is publicly verifiable as it does not rely on any trapdoor information from the puzzle issuer being included in the ciphertext for fast verification. So dishonestly generated ciphertexts are not meaningful anymore, and only dishonest servers need to be considered. Now if the server announces an invalid solution to a given puzzle, it gets detected.

Standalone security

Along with the composable definition of secure time lock encryption schemes provided by our ideal functionality $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$, we further provide two game-based definitions of security. A weaker one, capturing the one-way hardness of a TLE scheme; and a stronger one captures the semantic security of a TLE scheme, in the spirit of IND-CPA security. We show that a TLE scheme that satisfies the weaker definition suffices for UC realising the $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ functionality through our protocol π_{TLW}. The stronger game-based definition serves as a standard for the security analysis of TLE schemes in the stand-alone setting. To demonstrate the usefulness of our stronger definition, we show that Astrolabous and an enhanced version of Mahmoody *et al*'s construction Mahmoody et al. (2011) satisfy the said security standard. This result further validates our UC treatment and in particular our ideal functionality of time-lock encryption schemes.

For privacy/efficiency: As we have seen before, the self-tallying protocol in Okamoto (1998) for guaranteeing the eligibility of each voter and at the same time maintaining their privacy, uses blind signatures Chaum (1983). One drawback with this approach is the fact that these signatures are not reusable, which means that after the end of the election they cannot be used for future elections. Another novelty of our work was to search for alternative solutions that solve this problem. One way to achieve re-usability is to use signatures of knowledge Chase and Lysyanskaya (2006) for maintaining the privacy of each voter and dynamic accumulators Papamanthou et al. (2016); Fazio and Nicolosi (2002); Derler et al. (2015b); Zhang et al. (2017); Goodrich et al. (2002); Tremel (2013); Camenisch and Lysyanskaya (2002); Derler et al. (2015a); Baldimtsi et al. (2018), a very well studied cryptographic primitive, for efficiency. This idea was introduced in zerocoin Miers et al. (2013) but for a different purpose. Specifically, each party commits its locally generated serial number and announces the committed value to the other parties via the public ledger. This number shows that the party is the owner of a coin. All of the committed values are accumulated to just a single value via a dynamic accumulator. Next, when a party wants to "spend" their coin, they make a signature of knowledge over their serial number with the statement "I know the committed value of this serial number which is part of the accumulator" without revealing any further information. Then, the party sends the serial number along with the signature to the other parties via an anonymous channel Miers et al. (2013); Dingledine et al. (2004). In the voting case, the serial number is the voter's private part of the credential and the

commitment of that part is the public part of the credential. A drawback with this approach is the strong setup assumptions. Specifically, to set up the accumulator that is used in Miers et al. (2013) it requires an *RSA* number Goldreich (1999). The entity that knows the factorization of the RSA number can provide fake proof of membership and it can thus violate the eligibility property of the protocol. A potential solution is an *multi party computation* (MPC) type of protocol where this RSA number can be produced in a distributed manner without any party to know explicitly the factorization of that number. A protocol has been proposed for that task, calling the produced RSA numbers *RSA UFOS* Sander (1999). No security argumentation is provided in Sander (1999) in terms of a game-based Bernhard et al. (2015) or UC definition Canetti (2001b) for their scheme, making the protocol not secure based on today's cryptographic standards. A simpler construction than the one in Miers et al. (2013) was proposed in Groth and Kohlweiss (2015) without the existence of accumulators. Moreover, the authors provided better security argumentation for their scheme (they provided a game-based definition) but with worse time complexity, making their protocol hard to be used in large scale elections. Specifically, the verification of the *signature of knowledge* (SOK) is proportional (in fact more than linear) to the size of the voting parties. In contrast, the verification time of SOK in Miers et al. (2013) is constant.

3.2 Quantum e-voting

More than a decade of studies on quantum electronic voting has resulted in several protocols that use the properties of quantum mechanical systems. However, all these new protocols Huang et al. (2014); Wang et al. (2016); Vaccaro et al. (2007); Bonanome et al. (2011); Hillery et al. (2006); Li and Zeng (2008); Dolev et al. (2006); Okamoto et al. (2008); Zhou and Yang (2013); Thapliyal et al. (2017); Xue and Zhang (2017); Horoshko and Kilin (2011) are studied against different and not well-articulated corruption models and claim security using ad-hoc proofs that are not formalized and backed only against limited classes of quantum attacks. In particular, none of the proposed schemes provides rigorous definitions of `privacy` and `verifiability`, nor formal security proofs against specific, well-defined (quantum) attacker models. When it comes to electronic voting schemes, it is particularly hard to ensure that all the, somehow conflicting, properties hold Chevallier-Mames et al. (2010); it is therefore important that these new quantum protocols be rigorously and mathematically studied and the necessary assumptions and limitations formally established. Moreover, all of these protocols are not written in a common framework as they assume a different set of participating parties and a different protocol flow between the parties (e.g. protocol phases are not the same). As a result, it is hard for future researchers to study and improve these protocols both in terms of security and efficiency. On the other hand, there are classical e-voting protocols that claim security against

unbounded adversaries Broadbent and Tapp (2007), as well as ones based on problems believed to be hard even for quantum computers e.g. lattice-based Chillotti et al. (2016). Finally, we note that there exist protocols that consider elections with quantum input, see e.g. Bao and Yunger Halpern (2017). This type of protocol is more relevant to quantum game theory and less to election schemes with classical input/voting choice, and we have not considered them in this study.

3.2.1 Quantum e-voting definitions

As we mentioned, we can argue about the security of a protocol by either using a game-based definition and show that the protocol satisfies it or by defining an ideal functionality and prove that the protocol UC realizes it.

In the field of quantum cryptography, there are some works in which the authors provide a formal security treatment such as Barnum et al. (2002); Portmann (2017). In Barnum et al. (2002) the authors give a security definition and construction of a quantum authentication channel. In Portmann (2017) the authors present a more efficient construction of a quantum authentication channel in comparison with Barnum et al. (2002). Moreover, the authors argue about the security of their protocol by using abstract cryptography Maurer and Renner (2011), a relatively new framework in comparison with Canetti (2001b), that provides similar security guarantees as in Canetti (2001b)[1]. Specifically, they implement a quantum secure channel with the only set-up assumption to be a shared key among the communicating parties. Additionally, they capture the adversarial influence over the network by assuming that the communication channels are noisy.

In Moran and Naor (2006) the authors extend the ideal functionality of Canetti and Gennaro (1996) so that it can capture the `receipt-freeness` security property. It is worth mentioning that their functionality is more in the spirit of MPC rather than e-voting, as it does not capture the different protocol phases (e.g. `casting` phase, `tally` phase etc.) an e-voting protocol includes. Moreover, the corruption model is closely related to the one for MPC rather than e-voting (e.g. corruption of the tallier or the election authority) as it considers that the participating parties all play the same role, without any designated role among them. Despite these limitations to e-voting, the authors have shown a realization of a classical protocol of which the security holds against unbounded adversaries. As mentioned in Unruh (2013), UC realization of classical protocols against unbounded classical adversaries/environments can be extended to UC realization against unbound quantum adversaries/environments. In Unruh (2010) the authors present the first UC definition in the quantum setting for MPC.

[1]Further study needs to be conducted so that the security framework in Maurer and Renner (2011) be mature enough and can be used as widely as the ones in Canetti (2001b). This is something that the authors in Maurer and Renner (2011) mentioned as well.

A natural question is if we can extend every ideal functionality in classical e-voting to the quantum setting by using frameworks like the one in Unruh (2010). Unfortunately, the answer is not straightforward as the way the function handles the incoming information changes due to the fundamental principles of quantum mechanisms. For example, if we consider the public key encryption functionality appears in Camenisch et al. (2017), it cannot compare that two ciphertexts are the same so that it checks if the *correctness* property holds without measuring the ciphertexts, and thus destroying them. Another issue is that functionality cannot return the ciphertext to the requested party and at the same time hold a copy of that in its date base (no-cloning theorem).

As a result, there is no security definition in the literature to quantum electronic voting: neither in UC, where it can be illustrated via an ideal functionality nor in a game-based style, where it can be illustrated with an experiment/game between a quantum challenger and a quantum adversary.

Chapter 4

Time-lock encryption

We define one of our main building blocks, the $\mathcal{F}_{\mathsf{TLE}}$ functionality that captures the security properties of a *time-lock encryption* (TLE) scheme. In this work we focus on decentralized solutions which rely on computational puzzles rather than trusted third parties. Our ideal functionality is general to capture both settings but we focus here only on the decentralized solutions and present our work in these terms. The concept of TLE involves a party that initiates the encryption of a message that can only be decrypted only after a certain amount of time has elapsed. This can be achieved in two ways.

In the first approach Cheon et al. (2006); May (1993), a party, called the *manager*, releases the decryption keys on specific dates. For example, there are public and private keys for each day of the week. If a party wants to encrypt a message so that it will open on Wednesday 9/12/2020, she will use Wednesday's public key. The manager will on Wednesday announce Wednesday's private key, allowing the decryption of the message.

In the second approach Rivest et al. (1996); Mahmoody et al. (2011); Liu et al. (2018) a *computational puzzle*, which is a mathematical problem, needs be solved so that the message can be revealed. Let us consider again the example where a message needs to be opened on Wednesday 9/12/2020. The encryptor of the message creates a puzzle that once solved allows the message to be revealed. What the encryptor must be sure of is that the puzzle will be solved on Wednesday 9/12/2020, neither sooner nor later. The puzzle can differ from message to message, even if all messages are intended to open on Wednesday 9/12/2020. These *relativistic time* constructions Rivest et al. (1996); Mahmoody et al. (2011) are designed so that a puzzle can be solved only after a certain amount of computations have been performed. Such computations are enough so that the puzzle can be solved on 9/12/2020. Last, the puzzle can be the same for messages that are intended to open on 9/12/2020. These *absolute time* constructions Liu et al. (2018) are designed so that the solution of the puzzle can be delegated to external entities which try to solve the puzzle independently of the TLE protocol (e.g. Bitcoin miners in Liu et al. (2018)), giving an essence of absolute time. In either case, the message can be decrypted only after a *puzzle*

has been solved or its solution has been published. The solution of the puzzle is used as the secret key in the decryption algorithm so that the message can be revealed. The solution of the puzzle is used as the "secret" in the decryption algorithm so that the message can be revealed.

More precisely, a TLE scheme is a pair of algorithms (e, d) that a party P can use in order to encrypt a message m with time label τ_{dec} such that everyone can decrypt that message after the current time exceeds τ_{dec}. The decryption is possible because after the time τ_{dec} a witness $w_{\tau_{\mathsf{dec}}}$ is available (the solution to the puzzle), which acts as the secret key for algorithm d.

In the context of e-voting, specifically in self tallying elections, as we discussed all the proposed schemes suffer from fairness issues. This is where the TLE offer a viable solution. If a vote is encrypted by using a TLE scheme we are certain that it cannot be opened before the tallying phase, and thus the adversary changes his mind and vote for something else Kiayias and Yung (2002), or not open it at all Fujioka et al. (1993).

In Section 4.1, we provide a definition of the ideal TLE functionality $\mathcal{F}_{\mathsf{TLE}}$. In Section 4.2, we demonstrate a realization of $\mathcal{F}_{\mathsf{TLE}}$ via a new TLE scheme, which is called *Astrolabous*. Our TLE construction is based on Mahmoody et al. (2011) and Rivest et al. (1996).

4.1 Definition of $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$

We provide our UC treatment of TLE in the $\mathcal{G}_{\mathsf{clock}}$ model by defining the functionality $\mathcal{F}_{\mathsf{TLE}}$, following the approach of Canetti (2001b). The functionality is described in Figure 4.1, and at a high level operates as follows. The functionality is parameterized by a delay variable delay. This variable shows the time that a ciphertext needs to be created. There are settings where the ciphertetext generation needs some time, in some cases this time is very small or zero ($\mathsf{delay} = 0$) or noticeable ($\mathsf{delay} = 1$). The simulator $\mathcal{S}$ initially provides $\mathcal{F}_{\mathsf{TLE}}$ with the set of corrupted parties. Each time an encryption query issued by an honest party is handled to $\mathcal{F}_{\mathsf{TLE}}$, the functionality forwards the request to $\mathcal{S}$ without any information about the actual message except the size of the message and the party's identity. The simulator returns the token back to $\mathcal{F}_{\mathsf{TLE}}$ which replies with the message ENCRYPTING to the dummy party. This illustrates both the fact that the ciphertext does not contain any information about the message and that encryption might require some time to be completed. The environment can access the ciphertexts that this party has generated so far by issuing the command RETRIEVE, where $\mathcal{F}_{\mathsf{TLE}}$ returns all the ciphertexts that are created by that party back to it. It is worth mentioning, that the time labelling that is used in the encryption command refers to an absolute time rather than relative. On the other hand, the construction that we propose for realising $\mathcal{F}_{\mathsf{TLE}}$ is relative. That is why, as we see in detail in Section 4.2, the algorithm accepts the difference between the current time Cl and the time labelling τ as an input. In this way, the algorithm

computes the difficulty for the puzzle such that the message can be decrypted when time τ has been reached. In addition, $\mathcal{F}_{\mathsf{TLE}}$ handles the decryption queries in the usual way, unless it finds two messages recorded along the same ciphertext, in which case it outputs $\perp$. This enforces that the encryption/decryption algorithms used by $\mathcal{S}$ should satisfy Correctness. In addition, if $\mathcal{F}_{\mathsf{TLE}}$ finds the requested ciphertext in its database, the recorded time is smaller than the current one (which means that the ciphertext can be decrypted), but the party that requested the decryption of that ciphertext provided an invalid time labelling (labelling smaller than the one recorded in $\mathcal{F}_{\mathsf{TLE}}$'s database), it returns the message INVALID_TIME to that party. In the case where the encryption/decryption queries are issued by corrupted parties, $\mathcal{F}_{\mathsf{TLE}}$ responds according to the instructions of $\mathcal{S}$. When a party receives a decryption request from $\mathcal{Z}$, except from the ciphertext c, it receives as input a time labelling τ. Ideally, τ is the time when c can be decrypted. Of course, the labelling τ can also be different to then the decryption time of c. Nevertheless, this does not affect the soundness of $\mathcal{F}_{\mathsf{TLE}}$. Without the labelling, the $\mathcal{F}_{\mathsf{TLE}}$ or the engaging party in the real protocol would have to find the decryption time of c which is registered either in the functionality's database (ideal case) or in the party's list of received ciphertexts (real case) and then compare it with the current time Cl.

When a party P advances the $\mathcal{G}_{\mathsf{clock}}$, the simulator $\mathcal{S}$ is informed. Then, $\mathcal{S}$ can generate ciphertexts for each tag received from $\mathcal{F}_{\mathsf{TLE}}$ from P and send them to $\mathcal{F}_{\mathsf{TLE}}$ issuing the UPDATE. Later, $\mathcal{F}_{\mathsf{TLE}}$ will return these to P. This illustrates the fact that after some time ciphertexts are created. The specific delay is specified by $\mathcal{S}$. In TLE constructions where the encryption and decryption time is equal, $\mathcal{S}$ will force a delay on the ciphertext generation equal to the number of rounds that the ciphertext needs to be decrypted. Thus, the way we model $\mathcal{F}_{\mathsf{TLE}}$ allows us to capture a broader spectrum of TLE constructions (not necessary efficient) in the context of the Global Clock (GC) model.

Naturally, after some time, ciphertexts are eventually opened and every party, including $\mathcal{S}$, can retrieve the underlying plaintext. For that task, we include the command LEAKAGE. In the vanilla case, $\mathcal{S}$ can retrieve all the messages that can be opened by the current time Cl. However, there are cases where $\mathcal{S}$ can retrieve messages before their time comes. This advantage of $\mathcal{S}$ can be described by the function leak. This function accepts as input an integer (e.g., the current time Cl) and outputs a progressive integer (e.g., the time that the adversary can decrypt ciphertexts, which is the same or greater than Cl). For example, if $\mathsf{leak}(x) = x+1$, this means that at current time Cl the adversary $\mathcal{S}$ can retrieve messages that are supposed to be opened at time $\mathsf{Cl}+1$, meaning that the honest parties will gain access to these messages at the next clock advancement. Specifically, on demand, $\mathcal{F}_{\mathsf{TLE}}$ gives the record of all messages with encryption up to $\mathsf{leak}(\mathsf{Cl})$ to $\mathcal{S}$, where Cl is the current time provided by $\mathcal{G}_{\mathsf{clock}}$, and leak a leakage function that takes the current time as input and returns a later time. This function leak captures the fact that in some cases the adversary can decrypt messages before their opening

time has come. The ideal leak function with respect to security is the identity one, the one that gives no real advantage to the adversary in comparison to all other parties. There are however some time-lock encryption schemes which allow the adversary to decrypt a little bit earlier than the honest parties. For example, the Bitcoin based time-lock encryption scheme proposed in Liu et al. (2018). In this scheme, the adversary can locally compute some witness (e.g *selfish mining* Eyal and Sirer (2014)) without announcing them to the rest of the parties, providing him with an advantage with respect to decryption.

The time-lock encryption functionality $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$.

It initializes the list of recorded messages/ciphertexts L_{rec} as empty and defines the tag space TAG.

- Upon receiving $(\mathsf{sid}, \textsc{Corrupt}, \mathbf{P}_{\mathsf{corr}})$ from $\mathcal{S}$, it records the corrupted set $\mathbf{P}_{\mathsf{corr}}$.
- Upon receiving $(\mathsf{sid}, \textsc{Enc}, m, \tau)$ from $P \notin \mathbf{P}_{\mathsf{corr}}$, it reads the time Cl and does:
  1. If $\tau < 0$, it returns $(\mathsf{sid}, \textsc{Enc}, m, \tau, \perp)$ to P.
  2. It picks $\mathsf{tag} \xleftarrow{\$} \mathsf{TAG}$ and it inserts the tuple $(m, \mathsf{Null}, \tau, \mathsf{tag}, \mathsf{Cl}, P) \rightarrow L_{\mathsf{rec}}$.
  3. It sends $(\mathsf{sid}, \textsc{Enc}, \tau, \mathsf{tag}, \mathsf{Cl}, 0^{|m|}, P)$ to $\mathcal{S}$. Upon receiving the token back from $\mathcal{S}$ it returns $(\mathsf{sid}, \textsc{Encrypting})$ to P.
- Upon receiving $(\mathsf{sid}, \textsc{Update}, \{(c_j, \mathsf{tag}_j)\}_{j=1}^{p(\lambda)})$ from $\mathcal{S}$, for all $c_j \neq \mathsf{Null}$ it updates each tuple $(m_j, \mathsf{Null}, \tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, P)$ to $(m_j, c_j, \tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, P)$
- Upon receiving $(\mathsf{sid}, \textsc{Retrieve})$ from P, it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and it returns $(\mathsf{sid}, \textsc{Encrypted}, \{(m, c \neq \mathsf{Null}, \tau)\}_{\forall (m,c,\tau,\cdot,\mathsf{Cl}',P) \in L_{\mathsf{rec}} : \mathsf{Cl} - \mathsf{Cl}' \geq \mathsf{delay}})$ to P.
- Upon receiving $(\mathsf{sid}, \textsc{Dec}, c, \tau)$ from $P \notin \mathbf{P}_{\mathsf{corr}}$:
  1. If $\tau < 0$, it returns $(\mathsf{sid}, \textsc{Dec}, c, \tau, \perp)$ to P. Else, it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and:
     (a) If $\mathsf{Cl} < \tau$, it sends $(\mathsf{sid}, \textsc{Dec}, c, \tau, \textsc{More\_Time})$ to P.
     (b) If $\mathsf{Cl} \geq \tau$, then
     – If there are two tuples $(m_1, c, \tau_1, \cdot, \cdot, \cdot), (m_2, c, \tau_2, \cdot, \cdot, \cdot)$ in L_{rec} such that $m_1 \neq m_2$ and $c \neq \mathsf{Null}$ where $\tau \geq \mathsf{max}\{\tau_1, \tau_2\}$, it returns to P $(\mathsf{sid}, \textsc{Dec}, c, \tau, \perp)$.
     – If no tuple $(\cdot, c, \cdot, \cdot, \cdot, \cdot)$ is recorded in L_{rec}, it sends $(\mathsf{sid}, \textsc{Dec}, c, \tau)$ to $\mathcal{S}$ and returns to P whatever it receives from $\mathcal{S}$.
     – If there is a unique tuple $(m, c, \tau_{\mathsf{dec}}, \cdot, \cdot, \cdot)$ in L_{rec}, then if $\tau \geq \tau_{\mathsf{dec}}$, it returns (sid, Dec, c, τ, m) to P. Else, if $\mathsf{Cl} < \tau_{\mathsf{dec}}$, it returns (sid, Dec, c, τ, More_Time) to P. Else, if $\mathsf{Cl} \geq \tau_{\mathsf{dec}} > \tau$, it returns (sid, Dec, c, τ, Invalid_Time) to P.
- Upon receiving $(\mathsf{sid}, \textsc{Leakage})$ from $\mathcal{S}$, it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and returns $(\mathsf{sid}, \textsc{Leakage}, \{(m, c, \tau)\}_{\forall (m,c,\tau \leq \mathsf{leak}(\mathsf{Cl}),\cdot,\cdot,\cdot) \in L_{\mathsf{rec}}})$ to $\mathcal{S}$.
- Whatever message it receives from $P \in \mathbf{P}_{\mathsf{corr}}$, it forwards it to $\mathcal{S}$ and vice versa.

Figure 4.1: Functionality $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ parameterized by λ, a leakage function leak, a delay variable delay ,interacting with simulator $\mathcal{S}$, parties in $\mathbf{P}$, and global clock $\mathcal{G}_{\mathsf{clock}}$.

4.2 Realization of $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ via time-lock puzzles

In this section, we present the realization of $\mathcal{F}_{\mathsf{TLE}}$ via a protocol that uses a pair of encryption/decryption algorithms that satisfy a specific security notion that we formally define in Definition 4.2.1. We prove that our construction which is based on Mahmoody et al. (2011) and Rivest et al. (1996) is secure with respect to the required security notion.

The general idea of a time-lock puzzle scheme is that the parties have restricted access to a specific computation in any given period of time for solving a puzzle. In Rivest et al. (1996)'s case that computation is repeated squaring, and in Mahmoody et al. (2011) the computation is sequential hash evaluations. Of course, the underlying assumption here is that there is no "better" way to solve that puzzle except for sequentially applying the specific computation. Some of the most prominent proposed time-lock constructions are based on such assumption Rivest et al. (1996); Liu et al. (2018); Badertscher et al. (2017); Mahmoody et al. (2011).

In the UC framework, to construct a time-lock protocol we need to abstract such computations through an oracle $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$. The reasoning behind this modelling is simple. In the UC framework, all the parties are allowed to run polynomial time with respect to the protocol's parameter. As a result, it is impossible to impose on a party the restriction that in a specific period of time they can only execute a constant number of computations. This is why we abstract such computations as a functionality/oracle and wrap the oracle with a *functionality wrapper* that restricts the access to the oracle. The approach is similar to the one proposed in Badertscher et al. (2017), for modelling proof-of-work in the Bitcoin protocol.

In the following paragraphs, we present the evaluation oracle $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, the functionality wrapper $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ and the protocol Π_{TLE}. We provide a security definition that captures both *correctness* and *one-wayness* of TLE constructions. The latter is illustrated via an experiment in a game-based style described in Figure 4.5. We prove that Π_{TLE} UC realises $\mathcal{F}_{\mathsf{TLE}}$ given that the underlying TLE construction satisfies our security definition. Having at hand a UC realisation and given that our ideal functionality $\mathcal{F}_{\mathsf{TLE}}$ captures accurately the concept of what we expect from a TLE scheme, this validates the definition of security of TLE algorithms.

In the following section, we propose a new TLE construction and prove it satisfies our security definition, completing our construction argument. Finally, we provide a stand-alone security definition in the same spirit as *IND-CPA* security, named *IND-CPA-TLE*, which is captured via an experiment. We prove that Astrolabous satisfies this as well.

Our security definition that captures the one-wayness of a TLE construction was enough for having a UC realization. Although one-wayness as a property is very weak when arguing about the security of an encryption scheme, in our case was enough as we do not use the actual construction but we extend it in

the random oracle model. On the other hand, such definition in the stand alone model is weak. That was the reason of why we introduced IND-CPA-TLE.

The evaluation functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ The evaluation functionality captures the computation that is needed for a time-lock puzzle to be solved by the designated parties. An explanatory example can be found bellow.

Initially, the functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, as described in Figure 4.2, creates the list L_{eval} for keeping a record of the queries received so far. Then, upon receiving a query from a party in $\mathbf{P}$, $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ checks if this query has been issued before. If this is the case, it returns the recorded pair. If not, then for the query x it samples the value y from the distribution $\mathbf{D}_x$ and returns to that party the pair (x, y).

The distribution $\mathbf{D}_x$ in cases such as in Liu et al. (2018); Badertscher et al. (2017); Mahmoody et al. (2011) is a random value over a specific domain. Thus, $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ is the random oracle in these cases. More precisely, $\mathbf{D}_x = \mathcal{U}\{0, 2^n - 1\}$ where $\mathcal{U}$ is the uniform distribution and $[0, 2^n - 1]$ is its domain, in our example the domain of the random oracle. In that case, the parametrization of $\mathbf{D}$ with x is unnecessary. On the other hand, if we study other time-lock puzzles such as the one in Rivest et al. (1996), where the computation to solve a puzzle is the repeated squaring, the parametrization of $\mathbf{D}$ with x becomes necessary. An example of distribution $\mathbf{D}$ follows.

Example 4.2.1. In our modelling approach, for a random value $b \in \mathbb{Z}_n$, where n is a composite number, the k-repeated squaring of b is the value b^{2^k}. In that case, the oracle queries are of the form $x = b^{2^k}$ and the oracle response is $y = b^{2^{k+1}}$. Thus, the distribution $\mathbf{D}_{b^{2^k}}$ is equal to the constant distribution $\mathcal{C}\{b^{2^{k+1}}\}$ where the probability to sample the value $b^{2^{k+1}}$ is equal to 1.

If $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ is instantiated by the random oracle, then the distribution $\mathbf{D}_x$ is the uniform distribution for every x over the domain $(0, 2^n - 1)$. Similarly in Rivest et al. (1996), the distribution is constant as argued above (accepts as input x and returns b^{2^x}).

Example 4.2.2. Adapting the relative time-lock puzzle of Mahmoody et al. (2011) to our modelling approach, the evaluation functionality is instantiated by the random oracle. Let us consider that the solution of the puzzle is the value r. The creator of the puzzle P chooses the desired difficulty of the puzzle, τ. Then, P splits the puzzle r into $q\tau$ equal pieces $r_0, \ldots, r_{q\tau}$ such that $r = r_0 || \ldots || r_{q\tau}$. Here, q is the maximum number of evaluation queries that the party can make to the oracle in one round. Remember that the essence of round can be defined with respect to the functionality $\mathcal{G}_{\mathsf{clock}}$. Next, P makes one call to the random oracle functionality with the values $(r_0, \ldots, r_{q\tau-1})$ and receives back $(y_{r_0}, \ldots, y_{r_{q\tau-1}})$. Note that this call is counted as one. Finally, P creates the puzzle $(r_0, y_0 \oplus r_1, \ldots, y_{r_{q\tau-1}} \oplus r_{q\tau})$ for the secret r. Now, if some party P^* wants to solve the puzzle, it needs to send the query r_0 to the random oracle functionality. Upon receiving the value y_0 back from the random oracle functionality, P^* computes

$r_1 = y_0 \oplus (y_0 \oplus r_1)$. Next, it repeats the procedure with the value r_1. Note that, the maximum number of evaluation queries to the functionality oracle in one round is q and thus the puzzle to be solved needs τ rounds. It is worth mentioning that for capturing the limited access to the functionality in the UC framework, a functionality wrapper needs to be defined as it is described in a dedicated Paragraph below.

The evaluation functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}(\mathcal{D}, \mathbf{P})$.
Initializes an empty evaluation query list L_{eval}.
■ Upon receiving $(\mathsf{sid}, \textsc{Evaluate}, x)$ from a party $P \in \mathbf{P}$, it does:

1. It checks if $(x, y) \in L_{\mathsf{eval}}$ for some y. If no such entry exists, it samples y from the distribution $\mathbf{D}_x$ and inserts the pair (x, y) to L_{eval}. Then, it returns $(\mathsf{sid}, \textsc{Evaluated}, x, y)$ to P. Else, it returns the recorded pair.

Figure 4.2: Functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ parameterized by λ, a family of distributions $\mathcal{D} = \{\mathbf{D}_x | x \in \mathbf{X}\}$ and a set of parties $\mathbf{P}$.

The functionality wrapper $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ Our wrapper is defined along the lines of Badertscher et al. (2017). The functionality wrapper is an ideal functionality parameterized by another ideal functionality, mediating the access to the latter functionality only possible through the wrapper. Moreover, the wrapper restricts the access to the parameter functionality allowing parties to access it only a certain number of times per round. Here, the notion of round is defined with respect to the $\mathcal{G}_{\mathsf{clock}}$ functionality defined in Figure 2.1. In a nutshell, the wrapper models in the UC setting the limited resources a party has at their disposal for solving the underlying puzzle. Because in UC every party is a PPT ITM, the same holds for the adversary. So, the adversary can interact with any functionality polynomially many times in each round. There are several protocols that hinge their security on the limited computational capabilities of the participants. For example, the whole security argument for the Bitcoin protocol Nakamoto (2008) goes as follows: if the adversary does not maintain more than 50% of the network's hashing power, then some desired properties hold. Modelling this in the UC framework would mean that the parties try to extend the ledger by engaging in a series of hash evaluations Garay et al. (2015). If the parties and the adversary have unlimited access to the random oracle functionality (the modelling of the hash function in UC) that would mean that an adversary with less than 50% of hashing power can violate the *common prefix* property in Garay et al. (2015). For that reason, we need to restrict the access to the random oracle functionality, as in Badertscher et al. (2017). The same holds for our case. We need to restrict the access each party has to $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, else the time-lock puzzle can be solved in just one round, making the whole modelling of TLE in UC defective. Next, follows the description of $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$.

The functionality wrapper as described in Figure 4.3 is parameterized by the evaluation oracle $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ as described in Figure 4.2, the global clock $\mathcal{G}_{\mathsf{clock}}$, a set of parties $\mathbf{P}$, and the function $\mathsf{f}_{\mathsf{state}}$.

When $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ receives an evaluation query from a party P, it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. If this is the first time that this party issued a query, then it creates the list L^P to keep track of how many queries that party does in one round. Else, $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ checks if the number of queries the party issued that round does not exceed q, modelling in this way the limited computational resources a party has in every round. Last, if the party was activated in previous rounds, then the counter of issued oracle queries resets to 1, modelling that unused queries in previous rounds are lost if not made.

When $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ receives the answer from the functionality oracle $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, it returns the oracle's answer to party P.

Adversary can issue q queries in total: The wrapper handles independently queries issued by corrupted parties. Specifically, it allows q queries in total for all corrupted parties instead of q for every corrupted party. With that, we model that the adversary does not possess any advantage for sequential computation despite the fact how many parties are corrupt, in comparison with each party individually. Specifically, in reality, a computation can be either parallelized or not. Each computational task is carried away from a single CPU core at a time. For example, a 10-core CPU can parallelize a 10-step computation at once. On the other hand, in the UC framework, all parties can parallelize any arbitrary polynomial-step computation, assuming that the number of CPU cores they possess is arbitrary polynomial many. When the computation cannot be parallelized, despite how many cores a party has at their disposal, they can process one computation at a time, leaving aside any advantage of the number of CPU cores they possess. For example, if the adversary corrupts two parties or ten with 10-CPU each (twenty and one hundred in each case respectively) it is the same for sequential computation. This is exactly what we illustrate in our functionality wrapper, and thus giving to the adversary q queries in total despite the number of the corrupted parties.

In some settings, the interaction with the oracle is necessary even for the creation of the time-lock puzzle and not just for solving it. In reality, the creator of the puzzle can parallelize this computation, and this is what happens here. In a single oracle query, the party can both ask the oracle queries for puzzle creation and puzzle solving. Recall the example: 4.2.2. In order to create a puzzle for time labelling τ_{dec}, the party needs to engage with the oracle just a single time with $q\tau_{\mathsf{dec}}$ values in total to create the puzzle $c = (r_0, y_0 \oplus r_1, \ldots, y_{r_{q\tau_{\mathsf{dec}}-1}} \oplus r_{q\tau_{\mathsf{dec}}})$, where the secret is the value $r_0 || \ldots || r_{q\tau_{\mathsf{dec}}}$. Note that the solver of the puzzle cannot parallelize the computation for solving the puzzle, because she does not know the secret.

The functionality wrapper is parameterized by q the number of oracle queries

per round that are allowed from each party, the oracle $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, which evaluates these queries, the global clock $\mathcal{G}_{\mathsf{clock}}$ and a set of parties $\mathbf{P}$ that are allowed to engage with the oracle. The total number of queries q per round captures the fact that the parties have limited resources per round. In addition, we allow multiple value evaluation in a single query. With that we illustrate the fact that the computation can be parallelized (e.g. hash evaluation is parallelizable if the queries are stateless). The number of evaluations in a single oracle query is upper bounded by an arbitrary polynomial (like a UC execution). This in turn means that we assume that the parties have access to an arbitrary polynomial number of CPU cores that can handle independent computations. In the rest of this work we use the abbreviation $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ instead of $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}, \mathcal{G}_{\mathsf{clock}}, \mathbf{P})$ when it is obvious from the context.

Functionality wrapper $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}, \mathcal{G}_{\mathsf{clock}}, \mathbf{P})$.

- Upon receiving $(\mathsf{sid}, \textsc{Corrupt}, \mathbf{P}_{\mathsf{corr}})$ from $\mathcal{S}$, it records the corrupted set $\mathbf{P}_{\mathsf{corr}}$.
- Upon receiving $(\mathsf{sid}, \textsc{Evaluate}, (x_1, \ldots, x_j))$ from $P \in \mathbf{P} \setminus \mathbf{P}_{\mathsf{corr}}$ it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and does:

1. If there is not a list L^P it creates one, initially as empty. Then it does:
   (a) For every k in $\{1, \ldots, j\}$, it forwards the message $(\mathsf{sid}, \textsc{Evaluate}, x_k)$ to $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$.
   (b) When it receives back all oracle queries, it inserts the tuple-$(\mathsf{Cl}, 1) \in L^P$.
   (c) It returns $(\mathsf{sid}, \textsc{Evaluate}, ((x_1, y_1), \ldots, (x_j, y_j)))$ to P.
2. Else if there is a tuple-$(\mathsf{Cl}, j_{\mathsf{c}}) \in L^P$ with $j_{\mathsf{c}} < q$, then it changes the tuple to $(\mathsf{Cl}, j_{\mathsf{c}} + 1)$, and repeats the above steps 1a,1c.
3. Else if there is a tuple-$(\mathsf{Cl}^*, j_{\mathsf{c}}) \in L^P$ such that $\mathsf{Cl}^* < \mathsf{Cl}$, it updates the tuple as $(\mathsf{Cl}, 1)$, and repeats the above steps 1a,1c.

- Upon receiving $(\mathsf{sid}, \textsc{Evaluate}, (x_1, \ldots, x_j))$ from $P \in \mathbf{P}_{\mathsf{corr}}$ it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and repeats steps 1,3 except that it keeps the same list, named L^{corr}, for all the corrupted parties.

Figure 4.3: The Functionality wrapper $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ parameterized by λ, a number of queries q, functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, $\mathcal{G}_{\mathsf{clock}}$ and parties in $\mathbf{P}$.

The protocol Π_{TLE}: We are now ready to present the protocol Π_{TLE} which is proved in later Sections that it UC realises the $\mathcal{F}_{\mathsf{TLE}}$ functionality. The protocol consists of the functionality wrapper $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ as described in Figure 4.3, the global clock $\mathcal{G}_{\mathsf{clock}}$, the random oracle $\mathcal{F}_{\mathsf{RO}}$, the broadcast functionality $\mathcal{F}_{\mathsf{BC}}$ and a set of parties $\mathbf{P}$.

Example 4.2.3. Recall Example 4.2.2 and assume the time-lock puzzle $c = (r_0, y_0 \oplus r_1, \ldots, y_{r_{q\tau_{\mathsf{dec}}-1}} \oplus r_{q\tau_{\mathsf{dec}}})$. If the function $\mathsf{wit\_con}$ is given less than $q\tau_{\mathsf{dec}}$ oracle responses (e.g. $(y_0, \ldots, y_{q\tau_{\mathsf{dec}}-3})$) for the puzzle c, it returns $\perp$ else it returns $w_{\mathsf{dec}} = (r_0, y_0, \ldots, y_{r_{q\tau_{\mathsf{dec}}-1}}, c)$. Note that here, the ciphertext and the puzzle coincide as there is no actual encryption of a message. Thus, $\mathsf{f}_{\mathsf{puzzle}}$ is simply the identity function.

Necessity of extending the TLE algorithms: In order to realise $\mathcal{F}_{\mathsf{TLE}}$ with some TLE construction we need to extend a given TLE algorithm in the random oracle model. Recall that in $\mathcal{F}_{\mathsf{TLE}}$ all the ciphertexts eventually open. To capture semantic security, the ciphertext contains no information about the actual message, in contrast to the real protocol that contains the encryption of the actual message. So, for the simulator to simulate this difference when the messages are opened, $\mathcal{S}$ must be able to *equivocate* the opening of the ciphertext, else the environment $\mathcal{Z}$ can trivially distinguish the real from the ideal execution of the protocol. When we say that $\mathcal{S}$ equivocates the opening of the ciphertext, it means that $\mathcal{S}$ can open a ciphertext to whatever plaintext message needs to be opened. Equivocation is also used for other cryptographic primitives, such as bit commitments, where the simulator can equivocate because it knows the trapdoor information related to the *common reference string* (CRS) Lindell (2011). Our extension, that can be applied in any TLE construction, offers the feature of equivocation but at the expense of assuming the random oracle model. For example, consider a TLE scheme $(e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}, d_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}})$ with respect to oracle $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$. In the ideal world, when a party wants to encrypt a message m with time labelling τ, the functionality $\mathcal{F}_{\mathsf{TLE}}$ informs $\mathcal{S}$ about this request without revealing the identity of the party and the message m. The simulator creates a ciphertext c without knowing the message m and returns it back to $\mathcal{F}_{\mathsf{TLE}}$. After the current time Cl exceeds τ, $\mathcal{Z}$ can compute the underlying message to the ciphertext c, which in the ideal world does not contain any information about the message m. This allows $\mathcal{Z}$ to distinguish the real from the ideal execution of the protocol. For that reason, we extend the $(e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}, d_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}})$ to $(e^*_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}, d^*_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}})$ by borrowing techniques from Nielsen (2002); Camenisch et al. (2017) as follows: the ciphertext for a message m and time τ is the tuple $e^*_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}(m, \tau) = (c_1, c_2, c_3)$, where c_1 results from the encryption of a random string r, i.e., $c_1 = e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}(r, \tau)$; c_2 is the XOR between the message m and the random oracle call $\mathcal{H}(r)$ on r, i.e., $c_2 = m \oplus \mathcal{H}(r)$; and c_3 is a random oracle call on the concatenation $r||m$, i.e., $c_3 = \mathcal{H}(r||m)$. The c_3 makes the encryption scheme non-malleable Nielsen (2002). This extension allows $\mathcal{S}$ to equivocate when needed. We informally explain why this holds and formalize this in the proof of Theorem 1.

When $\mathcal{S}$ receives an encryption request from $\mathcal{F}_{\mathsf{TLE}}$ for time labelling τ, he returns the ciphertext $c = (e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}(r_1, \tau), r_2, r_3)$ where r_1, r_2, r_3 are random values. Observe that, in the ideal world, neither the evaluation functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ nor the random oracle $\mathcal{F}_{\mathcal{O}_{\mathsf{RO}}}$ exist. Instead, both of them are emulated by $\mathcal{S}$. As a result,

when the time Cl exceeds τ, $\mathcal{Z}$ can retrieve r_1 but in order to retrieve the message m he must issue a random oracle query on r_1 through a corrupted party. In that case, $\mathcal{S}$ can retrieve the message m from $\mathcal{F}_{\mathsf{TLE}}$, because the time Cl exceeded τ, programme the $\mathcal{F}_{\mathsf{RO}}$ so that $\mathcal{H}(r_1) = m \oplus r_2$ (equivocate) and return the answer back to $\mathcal{Z}$.

Table 4.1: Functions and list each party holds in Π_{TLE}.

Functions/Lists	Description
$\mathbf{P}, \mathbb{N}, \mathbf{Q}, \mathbf{R}, \mathbf{C}, \mathbf{M}, \mathbf{W}$	The space of time-lock puzzles, integers, oracle queries and responses to/from $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, ciphertexts, plaintexts and witnesses.
$e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}} : \mathbf{M} \times \mathbb{N} \times \mathbf{Q}/\mathbf{R} \to \mathbf{C}$	The encryption algorithm takes as input the plaintext, the puzzle difficulty and the pair of oracle queries/responses so that the puzzle can be created.
$d_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}} : \mathbf{C} \times \mathbf{W} \to \mathbf{M}$	The decryption algorithm takes as input the ciphertext and the secret key.
$\mathsf{f}_{\mathsf{state}} : \mathbf{P} \times \mathbb{N} \times \mathbf{Q}/\mathbf{R} \to \mathbf{Q}$	It prepares the next oracle query to $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$. Specifically, it accepts a puzzle, the number of query that needs to be prepared and all the previous queries and responses from the oracle.
$\mathsf{f}_{\mathsf{puzzle}} : \mathbf{C} \to \mathbf{P}$	It extracts the time-puzzle from a ciphertext.
$\mathsf{puz\_cr} : \mathbf{M} \times \mathbb{N} \to \mathbf{Q}$	The puzzle creation function takes as input the plaintext and the desired difficulty and creates the oracle queries so that a puzzle for that plaintext of that difficulty can be created.
$\mathsf{wit\_con} : \mathbf{Q}/\mathbf{R} \times \mathbb{N} \times \mathbf{P} \to \mathbf{W}$	The witness construction function that returns the solution of the puzzle or the witness if that is possible.
L_{rec}^{P}	The list of the generated ciphertexts.
$(z, \tau, \{(\mathsf{state}_k^z, y_k)\}_{k=0}^{j_t}, j_c, j_t)$	The tuple contains a puzzle z, the difficulty of the puzzle τ, the pairs of oracle queries/responses to solve puzzle z, the current number j_c of oracle queries in that round and the total number of oracle queries j_t.

Description of protocol Π_{TLE}: Each party $P \in \mathbf{P}$ is parameterized/maintains the following:

- She maintains the list of recorded messages/ciphertexts L_{rec}^{P}, in which the requested messages for encryption by $\mathcal{Z}$ are stored along with the ciphertext of that message (initially stored as Null), a random identifier of the message tag, the time τ that the message should open, the time Cl that is recorded for the first time and a flag which shows if that message has been broadcast or not to the other parties. Broadcast is necessary, as the construction that UC realises our $\mathcal{F}_{\mathsf{TLE}}$ is relativistic. More precisely, it is based on a time-lock puzzle. So, for that message to be opened by any honest party when the

time comes, that party should start to solve the puzzle as soon as it can. So the transmission of the ciphertext (and thus the puzzle) is necessary.

- She parameterized by The tag space TAG and a pair of TLE algorithms $(e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}, d_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}})$ are hard-coded in each party.
- A function $\mathsf{f}_{\mathsf{state}}$ which prepares the next oracle query to $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ for a puzzle to be solved.
- A function $\mathsf{f}_{\mathsf{puzzle}}$ which extracts from a TLE ciphertext c_* the underlying time-lock puzzle.
- A function $\mathsf{puz\_cr}$ which generates the oracle queries to $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ so that a puzzle of the desired difficulty can be created.
- A function $\mathsf{wit\_con}$ which computes witnesses by performing the necessary sequential computations. More precisely, given oracle queries/responses to/from the functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, time labelling τ and the time-lock puzzle, it returns a witness w_τ or $\perp$ if the computation fails.

When a party receives ENCRYPTION from $\mathcal{Z}$ for a message m with difficulty τ, it picks a random tag for future reference of that message, reads the current time Cl from $\mathcal{G}_{\mathsf{clock}}$, and stores the tuple $(m, \mathsf{Null}, \tau, \mathsf{tag}_m, \mathsf{Cl}, 0)$ to L^P_{rec}. Then, it returns the message $\mathsf{Encrypting}$ to $\mathcal{Z}$. That means that the encryption is going to take some time, in our case one turn. When the party receives from $\mathcal{Z}$ ADVANCE_CLOCK, she reads time Cl from $\mathcal{G}_{\mathsf{clock}}$ and checks if a decryption command has been issued in this turn. If this is the case, that means that the party, before attempting to decrypt, she depletes all her oracle queries for both solving puzzles and creating puzzles for encrypting a message in this turn by executing the procedure *Puzzle*. This is necessary, as the party attempts to decrypt after her witness is updated for that turn and this is possible only by querying the oracle $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$. If no decryption command has been issued in this turn, the party executes both the procedures *Puzzle*, for puzzle solving and puzzle creation, and *Encrypt*, for encrypting the messages issiued by $\mathcal{Z}$ in the current round. Then she broadcasts the ciphertexts that correspond to messages received by $\mathcal{Z}$ in this round (after the end of the turn encryption ends). Finally, the party changes the flags from 0 to 1 in the tuples that the broadcast ciphertexts are stored in L^P_{rec} and informs the global clock that she was activated in that round by sending a clock advancement command.

Broadcast: The broadcast is necessary because the TLE constructions we study are *relativistic* Rivest et al. (1996); Mahmoody et al. (2011), and thus the message can be opened only when a certain amount of computations has been spent by the parties to solve the puzzle. In contrast, with *absolute* time-lock constructions such as in Liu et al. (2018), the broadcast of the ciphertext is unnecessary because the message will be opened once the current time reaches the decryption time of the time puzzle. That is why we require that the ciphertext must be sent to the designated parties upon its creation. In this work, we realise

$\mathcal{F}_{\mathsf{TLE}}$ only with relativistic based constructions. When the party receives the broadcast ciphertexts, she creates a tuple that contains the time-lock puzzle of the ciphertext, the difficulty, the queries for solving the puzzle and the responses and two counters that show how many oracle queries she has issued both this round and in total. The time-lock puzzle can be extracted from a ciphertext with the help of the function f_{puzzle}.

Flag that distinguishes broadcast from non-broadcast messages: When a message is created but is not allowed to be broadcast by $\mathcal{A}$, it means that the other parties will not receive it. Thus, they cannot solve the underlying puzzle so it can be opened when the time comes. In a nutshell, it is like that message did not exist. So, when the environment issues a RETRIEVE command to retrieve the ciphertexts created in this turn, the non-broadcast ciphertexts are not returned. The only ciphertexts returned to $\mathcal{Z}$ are the ones that will eventually be opened by all parties, which is the ones that are broadcast. If we allow the non-broadcast ciphertexts to be returned to $\mathcal{Z}$ then we will have a trivial distinction between the ideal and the real setting for the reasons explained above.

When a party receives a decryption command from $\mathcal{Z}$ for a ciphertext c it uses the function $\mathsf{wit\_con}$ to construct the decryption key. The input of $\mathsf{wit\_con}$ is the collection of states that the party received so far from the $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ through the functionality wrapper $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$. Next, the party returns to $\mathcal{Z}$ either the message m, if the decryption was successful, or $\perp$, otherwise. Note that, as in the construction of Nielsen (2002); Camenisch et al. (2017), the third argument in the ciphertext renders the scheme non-malleable Dwork (2011). In trivial cases where the difference $\tau_{\mathsf{dec}} - \mathsf{Cl}$ is negative or zero, decryption can occur instantly.

Observe that the witness an honest party uses for decrypting a message before receiving a clock advancement command is not the most updated one. Specifically, the most updated witness in the current round is the one obtained after the honest party uses all of her q oracle queries. A natural question is, can the party use all of her queries when receiving a decryption command? If the party uses all of her oracle queries when receiving a decryption command from $\mathcal{Z}$ for updating her witness then, there will not be any queries left for encrypting the messages of the current round. This is why the honest parties both generate and solve puzzles at the end of the round (e.g. when receiving a clock advancement command). On the contrary, the adversary can use all of his q queries whenever he wants. Thus, he possesses an advantage in the decryption time by one round compared to the honest parties. This is why in our proof of realization we assume that the leakage function leak is equal with $x + 1$ where x is the time Cl of the clock.

As mentioned, there are two procedures, named *Puzzle* and *Encrypt* that each party executes one or both either when she receives a clock advancement or decryption command from $\mathcal{Z}$. Specifically, in *Puzzle* the party issues q oracle queries in total both for puzzle creation and for puzzle solving. This is achievable since the computation can be parallelized. Specifically, the last oracle query to $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ contains both the queries that are needed for the creations of the

ciphertexts and the queries for solving the puzzles. When chose the last one to illustrate that puzzle creation needs some time and thus is the last computation for that round. The next procedure called *Encrypt*, and uses the puzzles created from *Puzzle* to create the new ciphertexts. Specifically, it uses the puzzles to encrypt a random string by using the TLE scheme (*Time-lock encryption*). Then, encrypts the actual message by XORed the message with the random oracle response of the random string (*Extended encryption*). Thus, the first argument of the ciphertext is the TLE encryption of the random string, the second argument is the XORed message with the random oracle responce on the random string, and the third and final argument is the random oracle response on the concatenation of the message and the random string.

$\Pi_{\mathsf{TLE}}(\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}), e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}, d_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}, \mathsf{f}_{\mathsf{state}}, \mathsf{wit\_con}, \mathsf{f}_{\mathsf{puzzle}}, \mathsf{puz\_cr}, \mathcal{G}_{\mathsf{clock}}, \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{BC}}, \mathbf{P})$.

Each party maintains the list of recorded messages/ciphertexts L_{rec}^P, initially as empty, a tag space TAG and the algorithms $(e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}, d_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}})$. Moreover, she follows the procedure described below:

Puzzle:

1. *Preparing queries for puzzle creation:* She collects all tuples $\{(m_j, \mathsf{Null}, \tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, 0) \in L_{\mathsf{rec}}^P\}_{j=1}^{p_1(\lambda)}$ for $\mathsf{Cl}_j = \mathsf{Cl}$. She picks $\{r_1^j \xleftarrow{\$} \{0,1\}^{p^*(\lambda)}\}_{j=1}^{p_1(\lambda)}$. For each j she computes $\mathsf{puz\_cr}(r_1^j, \tau_j - (\mathsf{Cl}+1)) \rightarrow \{x_k\}_{k=1}^{p_2(\lambda)}$.

2. *Puzzle solving:* For ($j_l = 0$, $j_l < q$, j_l++) she collects all $\{\mathsf{state}_{j_t}^{z_n}\}_{n=1}^{p_3(\lambda)}$, such that $(z_n, \tau_{\mathsf{dec}}, \{(\mathsf{state}_k^{z_n}, y_k)\}_{k=0}^{j_t}, \mathsf{Cl}, 0, j_t)$ is recorded.

   (a) *Parallelize puzzle creation queries and puzzle solve:* If $j_l = q-1$, she sends $(\mathsf{sid}, \textsc{Evaluate}, \{\mathsf{state}_{j_t}^{z_n}\}_{n=1}^{p_3(\lambda)} \cup \{x_k\}_{k=1}^{p_2(\lambda)})$ to $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ and receives back $(\mathsf{sid}, \textsc{Evaluate}, \{(\mathsf{state}_{j_t}^{z_n}, y_{j_t}^*)\}_{n=1}^{p_3(\lambda)} \cup \{(x_k, y_k)\}_{k=1}^{p_2(\lambda)})$. Else she sends $(\mathsf{sid}, \textsc{Evaluate}, \{\mathsf{state}_{j_t}^{z_n}\}_{n=1}^{p_3(\lambda)})$ to $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$.

   (b) *Update the record:* In each case, she updates each tuple as $(z_n, \tau_{\mathsf{dec}}, \{(\mathsf{state}_k^{z_n}, y_k)\}_{k=0}^{j_t+1}, \mathsf{Cl}, j_l + +, j_t + +)$ where $\mathsf{state}_{j_t+1}^{z_n} = \mathsf{f}_{\mathsf{state}}(z_n, j_t, \{(\mathsf{state}_k^{z_n}, y_k)\}_{k=0}^{j_t})$, $y_{j_t+1} = \mathsf{Null}$ and $y_{j_t} \leftarrow y_{j_t}^*$. In case that $j_l = q$, she changes the Cl in the tuple to $\mathsf{Cl}+1$ and $j_l = 0$.

Encryption:

1. *Time-lock encryption:* She computes $\{c_1^j \leftarrow e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}(r_1^j, \{(x_k, y_k)\}_{k=1}^{p_2(\lambda)}, \tau_j - (\mathsf{Cl}+1))\}_{j=1}^{p_1(\lambda)}$.

2. *Extended encryption:* For each r_1^j, she sends $(\mathsf{sid}, \textsc{Query}, r_1^j)$ to $\mathcal{F}_{\mathsf{RO}}$. Upon receiving $(\mathsf{sid}, \textsc{Random\_Oracle}, r_1^j, h^j)$ from $\mathcal{F}_{\mathsf{RO}}$, P sends $(\mathsf{sid}, \textsc{Query}, r_1^j || m_j)$ to $\mathcal{F}_{\mathsf{RO}}$. Upon receiving $(\mathsf{sid}, \textsc{Random\_Oracle}, r_1^j || m_j, c_3^j)$ from $\mathcal{F}_{\mathsf{RO}}$, she computes $c_j \leftarrow (c_1^j, h \oplus m, c_3^j)$ and updates the tuple $(m_j, c_j, \tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, 0) \rightarrow L_{\mathsf{rec}}^P$.

■ Upon receiving $(\mathsf{sid}, \textsc{Enc}, m, \tau)$ from $\mathcal{Z}$, P reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and if $\tau < 0$ she returns $(\mathsf{sid}, \textsc{Enc}, m, \tau, \perp)$ to $\mathcal{Z}$. Else, it does:

1. She picks $\mathsf{tag} \xleftarrow{\$} \mathsf{TAG}$ and she inserts the tuple $(m, \mathsf{Null}, \tau, \mathsf{tag}, \mathsf{Cl}, 0) \rightarrow L_{\mathsf{rec}}^P$.

2. She returns $(\mathsf{sid}, \textsc{Encrypting})$ to $\mathcal{Z}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Advance\_Clock})$ from $\mathcal{Z}$, P reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. She executes both *Puzzle* and *Encryption* procedure. Then, she sends $(\mathsf{sid}, \textsc{Broadcast}, \{(c_j, \tau_j)\}_{j=1}^{p_1(\lambda)})$ to $\mathcal{F}_{\mathsf{BC}}$. Upon receiving $(\mathsf{sid}, \textsc{Broadcasted}, \{(c_j, \tau_j)\}_{j=1}^{p_1(\lambda)})$ from $\mathcal{F}_{\mathsf{BC}}$, for each j she updates each tuple $(m_j, \mathsf{Null}, \tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, 1)$ to $(m_j, c_j, \tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, 1)$ and sends $(\mathsf{sid}, \textsc{Advance\_Clock})$ to $\mathcal{G}_{\mathsf{clock}}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Retrieve})$ from $\mathcal{Z}$, P reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and returns $(\mathsf{sid}, \textsc{Encrypted}, \{(m_j, c_j, \tau_j) : (m_j, c_j, \tau_j, \cdot, \mathsf{Cl}_j, 1) \in L_{\mathsf{rec}}^P : \mathsf{Cl} - \mathsf{Cl}_j \geq 1\})$ to $\mathcal{Z}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Broadcast}, \{(c_j, \tau_j)\}_{j=1}^{p_1(\lambda)})$ from $\mathcal{F}_{\mathsf{BC}}$ where $c_j = (c_1^j, c_2^j, c_3^j)$, P reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and does for every j:

1. She computes $\mathsf{state}_0^{\mathsf{f_{puzzle}}(c_1^j)} \leftarrow \mathsf{f_{state}}(\mathsf{f_{puzzle}}(c_1^j), 0, \mathsf{Null})$.
2. She records the tuple-$(\mathsf{f_{puzzle}}(c_1^j), \tau_{\mathsf{dec}}, \{(\mathsf{state}_0^{\mathsf{f_{puzzle}}(c_1^j)}, \mathsf{Null})\}, \mathsf{Cl}, 0, 0)$.

■ Upon receiving $(\mathsf{sid}, \textsc{Dec}, c := (c_1, c_2, c_3), \tau_{\mathsf{dec}})$ from $\mathcal{Z}$, P reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. Then she does:

1. If $\tau_{\mathsf{dec}} < 0$, she returns $(\mathsf{sid}, \textsc{Dec}, c, \tau_{\mathsf{dec}}, \perp)$ to $\mathcal{Z}$.
2. If $\mathsf{Cl} < \tau_{\mathsf{dec}}$, she returns $(\mathsf{sid}, \textsc{Dec}, c, \tau_{\mathsf{dec}}, \textsc{More\_Time})$.
3. She searches for a tuple $(\mathsf{f_{puzzle}}(c_1), \tau, \{(\mathsf{state}_k^{\mathsf{f_{puzzle}}(c_1)}, y_k)\}_{k=0}^{j_t}, \mathsf{Cl}, q, j_t)$. If $\tau_{\mathsf{dec}} < \tau \leq \mathsf{Cl}$ then she returns $(\mathsf{sid}, \textsc{Dec}, c, \tau_{\mathsf{dec}}, \textsc{Invalid\_time})$ to $\mathcal{Z}$.
4. She computes $w_{\tau_{\mathsf{dec}}} \leftarrow \mathsf{wit\_con}(\{(\mathsf{state}_k^{\mathsf{f_{puzzle}}(c_1)}, y_k)\}_{k=0}^{j_t}, \tau_{\mathsf{dec}}, \mathsf{f_{puzzle}}(c_1))$.
5. She runs $x \leftarrow d_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}(c_1, w_{\tau_{\mathsf{dec}}})$ and she sends $(\mathsf{sid}, \textsc{Query}, x)$ to $\mathcal{F}_{\mathsf{RO}}$. Upon receiving $(\mathsf{sid}, \textsc{Random\_Oracle}, x, h)$ from $\mathcal{F}_{\mathsf{RO}}$, she computes $m \leftarrow h \oplus c_2$. She sends $(\mathsf{sid}, \textsc{Query}, x||m)$ to $\mathcal{F}_{\mathsf{RO}}$. Upon receiving $(\mathsf{sid}, \textsc{Random\_Oracle}, x||m, c_3^*)$ from $\mathcal{F}_{\mathsf{RO}}$: If $c_3 \neq c_3^*$, she returns to $\mathcal{Z}$ $(\mathsf{sid}, \textsc{Dec}, c, \tau_{\mathsf{dec}}, \perp)$. Else, she returns to $\mathcal{Z}$ $(\mathsf{sid}, \textsc{Dec}, c, \tau_{\mathsf{dec}}, m)$.
6. If such tuple does not exist then she returns $(\mathsf{sid}, \textsc{Dec}, c, \tau_{\mathsf{dec}}, \perp)$ to $\mathcal{Z}$.

Figure 4.4: The Protocol Π_{TLE} in the presence of a functionality wrapper $\mathcal{W}_q$, an evaluation functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, a random oracle $\mathcal{F}_{\mathsf{RO}}$, a broadcast functionality $\mathcal{F}_{\mathsf{BC}}$, a global clock $\mathcal{G}_{\mathsf{clock}}$, where $e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}, d_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}$, $\mathsf{f_{state}}$, $\mathsf{wit\_con}$ and $\mathsf{f_{puzzle}}$ are hard-coded in each party in $\mathbf{P}$.

4.2.1 Security definitions of time-lock puzzles

In this Subsection, we provide security definitions that a TLE scheme (e.g a pair $((e_{\mathcal{O}}, d_{\mathcal{O}}))$ must satisfy to provide a UC realization of our $\mathcal{F}_{\mathsf{TLE}}$ functionality. Our security definition captures two properties, namely Correctness and qSecurity. A TLE scheme that satisfies both properties is considered *one-way secure* based on Definition 4.2.1.

Intuitively, the Correctness property states that the decryption of the ciphertext with underlying plaintext m results in the message m itself with high probability provided that the underlying time-lock puzzle has been solved. The qSecurity property is described in a game-based style via the experiment in Figure 4.5 and states that an adversary can win the experiment only with a very small probability. Specifically, the experiment captures the one-way security of a TLE

scheme as in the concept of *one-way functions* security **?**Goldreich (1999). Although indistinguishability, like in *IND-CPA* security Goldreich (1999); Kościelny et al. (2013), is stronger than the hardness to reverse a function, for our purpose of achieving UC realization (Theorem 1) it is enough. This is possible because we extend our TLE construction into a bigger one in the random oracle model and we rely on the hardness of inverting the underlying TLE construction. Because of that, in Subsection 4.3.3, we provide an indistinguishability game-based definition, similar to IND-CPA but in the context of TLE so that we can argue about the security of a TLE construction even in the standalone model.

In Figure 4.5, we present the experiment $\mathbf{EXP}_{\mathsf{TLE}}$ in the presence of a challenger Ch and an adversary $\mathcal{B}$. This experiment illustrates the security of a TLE scheme in the sense that no adversary can open a message before a certain number of computations has been performed. Specifically, we allow access to the adversary to the evaluation oracle $\mathcal{O}_{\mathsf{eval}}$. It is worth mentioning that the time τ in encryption requests refers to a *relativistic* notion of time (the time that the puzzle needs to be solved) rather than an *absolute* one (the time that the puzzle will eventually be decrypted). If the adversary queries the oracle q times for a ciphertext c, the challenger, which maintains a counter for that ciphertext, increases that counter by one, allowing him to keep track of the number of queries the adversary made for that particular ciphertext. With this, we model the essence of the round and the limited resources the adversary has at his disposal but in a game-based style (without $\mathcal{G}_{\mathsf{clock}}$ and $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$). The oracle queries are formed with the help of the state function $\mathsf{f}_{\mathsf{state}}$ and puzzle function $\mathsf{f}_{\mathsf{puzzle}}$, as described in Table 4.1 and in a dedicated paragraph on page 52, with the initial query for ciphertext c being $\mathsf{f}_{\mathsf{state}}(\mathsf{f}_{\mathsf{puzzle}}(c), 0, \mathsf{Null})$. Again, the state function $\mathsf{f}_{\mathsf{state}}$ takes as an input the time puzzle of the ciphertext c, the number oracle query issued so far in the current round and the previous response of the oracle (e.g., for the initial query it is Null). The state function $\mathsf{f}_{\mathsf{state}}$ illustrates the sequential oracle queries a party does in order to solve the time-lock puzzle. Moreover, $\mathsf{f}_{\mathsf{state}}$ gives a precise description (and enforcement) of how each oracle query must be formed before being issued to the oracle. In that way, we "enforce" the property that the time-lock puzzle cannot be parallelized. Although the adversary can issue encryption and decryption queries on his own because he knows the description of the encryption and decryption algorithms, the challenger only records the encryption and decryption requests that are issued through him. The reason behind this modelling choice is that we only care only to keep track of legitimate encryption and decryption queries, similar to 4.1. In other words, we cannot guarantee that the adversary uses the correct algorithm to encrypt a message and thus we cannot argue about the security of these ciphertexts. Moreover, a valid witness w_{t} for time label τ with $\tau \leq t$, can be constructed from the responses of the oracle $\mathcal{O}_{\mathsf{eval}}$ with the help of the function $\mathsf{wit\_con}$ as described in Table 4.1 and in a dedicated paragraph on page 52. Again, the function $\mathsf{wit\_con}$ takes as input

oracle queries, a time labelling and a time puzzle and outputs either a witness, if it can be constructed from the provided oracle queries, or $\perp$ otherwise. Upon request, the adversary receives a challenge ciphertext from the challenger. If the adversary can guess correctly the underlying plaintext with less than the expected computations, then he wins the game. For example, if the challenge queried by the adversary is formed with time label τ (e.g. following experiment's glossary, he sends $(\text{CHALLENGE}, \tau)$ to the challenger) but the adversary manages to retrieve the message with less then $q\tau$ oracle queries, then he wins the game. In this game the description of the oracle $\mathcal{O}_{\mathsf{eval}}$ in Figure 4.5, is exactly that of the ideal functionality in Figure 4.2 without the UC interface.

The experiment $\mathbf{EXP}_{\mathsf{TLE}}(\mathcal{B}, \mathcal{O}_{\mathsf{eval}}, e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}}, \mathsf{f}_{\mathsf{state}}, \mathsf{f}_{\mathsf{puzzle}}, q)$

Initialization Phase.

- Ch is initialized with $e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}}$ and sends them to $\mathcal{B}$. In addition it creates a local time counter $\mathsf{Cl}_{\mathsf{exp}}$.

Learning Phase.

- When $\mathcal{B}$ issues the query $(\text{EVALUATE}, (x_1, \ldots, x_j))$ to $\mathcal{O}_{\mathsf{eval}}$ through the Ch, he gets back $(\text{EVALUATE}, ((x_1, y_1), \ldots, (x_j, y_j)))$.
- $\mathcal{B}$ can request the encryption of a message $m \in \mathbf{M}_\lambda$ with time label τ_{dec} by sending $(\text{ENC}, m, \tau_{\mathsf{dec}})$ to Ch.
- When Ch receives a $(\text{ENC}, m, \tau_{\mathsf{dec}})$ request from $\mathcal{B}$, it runs the algorithm $e_{\mathcal{O}_{\mathsf{eval}}}(m, \tau_{\mathsf{dec}}) \rightarrow c$ and returns c to $\mathcal{B}$.
- Ch increases $\mathsf{Cl}_{\mathsf{exp}}$ by 1 for every q queries $\mathcal{B}$ issues to $\mathcal{O}_{\mathsf{eval}}$.
- $\mathcal{B}$ can request the decryption of a ciphertext c by sending (DEC, c, w_τ) to Ch. Then, Ch just runs the algorithm $d_{\mathcal{O}_{\mathsf{eval}}}(c, w_\tau) \rightarrow y \in \{m, \perp\}$ and returns to $\mathcal{B}$ $(\text{DEC}, c, w_\tau, y)$.

Challenge Phase.

- $\mathcal{B}$ can request for a single time a challenge from Ch by sending $(\text{CHALLENGE}, \tau)$. Then, Ch picks a value $r \xleftarrow{\$} \mathbf{M}_\lambda$ and sends $(\text{CHALLENGE}, \tau, c_r \leftarrow e_{\mathcal{O}_{\mathsf{eval}}}(r, \tau - \mathsf{Cl}_{\mathsf{exp}}))$ to $\mathcal{B}$. Then, $\mathcal{B}$ is free to repeat the *Learning Phase*.
- $\mathcal{B}$ sends as the answer of the challenge the message $(\text{CHALLENGE}, \tau, c_r, r^*)$ to Ch.
- If $(r^* = r) \wedge (\tau > \mathsf{Cl}_{\mathsf{exp}})$ (i.e. $\mathcal{B}$ manages to decrypt c_r before the decryption time comes) then $\mathbf{EXP}_{\mathsf{TLE}}$ outputs 1. Else, $\mathbf{EXP}_{\mathsf{TLE}}$ outputs 0.

Figure 4.5: Experiment $\mathbf{EXP}_{\mathsf{TLE}}$ for a number of queries q, function $\mathsf{f}_{\mathsf{state}}$, message domain $\mathbf{M}_\lambda$, algorithms $e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}}$ in the presence of an adversary $\mathcal{B}$, oracle $\mathcal{O}_{\mathsf{eval}}$ and a challenger Ch all parameterized by 1^λ.

Definition 4.2.1. A one-way secure *time-lock encryption scheme* with respect to an evaluation oracle $\mathcal{O}_{\mathsf{eval}}$, a relation $\mathsf{R}_{\mathcal{O}_{\mathsf{eval}}}$, a state function $\mathsf{f}_{\mathsf{state}}$, puzzle function $\mathsf{f}_{\mathsf{puzzle}}$ and a witness construction function $\mathsf{wit\_con}$ for message space $\mathbf{M}$ and a security parameter λ is a pair of PPT algorithms $(e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}})$ such that:

- $e_{\mathcal{O}_{\mathsf{eval}}}(m, \tau_{\mathsf{dec}})$: The encryption algorithm takes as input message a $m \in \mathbf{M}$, an integer $\tau_{\mathsf{dec}} \in \mathbb{N}$ and outputs a ciphertext c.
- $d_{\mathcal{O}_{\mathsf{eval}}}(c, w_{\tau_{\mathsf{dec}}})$: The decryption algorithm takes as input $w_{\tau_{\mathsf{dec}}} \in \{0,1\}^*$ and a ciphertext c, and outputs a message $m \in \mathbf{M}$ or $\perp$.

The pair $(e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}})$ satisfies the following properties:

1. **Correctness:** For every $\lambda, \tau_{\mathsf{dec}} \in \mathbb{N}, m \in \mathbf{M}$ and $w_{\tau_{\mathsf{dec}}}$, it holds that

$$\Pr\left[m' \leftarrow d_{\mathcal{O}_{\mathsf{eval}}}(e_{\mathcal{O}_{\mathsf{eval}}}(m, \tau_{\mathsf{dec}}), w_{\tau_{\mathsf{dec}}}) \wedge \mathsf{R}_{\mathcal{O}_{\mathsf{eval}}}(w_{\tau_{\mathsf{dec}}}, (\mathsf{f}_{\mathsf{puzzle}}(c), \tau_{\mathsf{dec}})) : m' = m\right] > 1 - \mathsf{negl}(\lambda)$$

   where $w_{\tau_{\mathsf{dec}}}$ can be constructed from the received responses of $\mathcal{O}_{\mathsf{eval}}$ and function wit_con as it is described in both Table 4.1 and Figure 4.4.

2. **qSecurity:** For every PPT adversary $\mathcal{B}$ with access to oracle $\mathcal{O}_{\mathsf{eval}}$, the probability to win the experiment $\mathbf{EXP}_{\mathsf{TLE}}$ and thus output 1 in Figure 4.5 is $\mathsf{negl}(\lambda)$.

4.2.2 Proof of UC realizing $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$

In this Subsection we show that if the TLE scheme used in protocol Π_{TLE} in Figure 4.4 is a secure time-lock encryption scheme according to Definition 4.2.1 then the protocol Π_{TLE} UC realizes $\mathcal{F}_{\mathsf{TLE}}$.

Theorem 1. *Let $(e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}})$ be a pair of* encryption/decryption *algorithms that satisfies Definition 4.2.1. Then, the protocol Π_{TLE} in Figure 4.4 UC-realizes functionality $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ in the $(\mathcal{W}_q(\mathcal{F}_{\mathsf{RO}}^*), \mathcal{G}_{\mathsf{clock}}, \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{BC}})$-hybrid model with leakage function $\mathsf{leak}(x) = x + 1$, $\mathsf{delay} = 1$, where $\mathcal{F}_{\mathsf{RO}}$ and $\mathcal{F}_{\mathsf{RO}}^*$ are two distinct random oracles.*

Proof. Let us suppose that protocol Π_{TLE} does not UC-realize $\mathcal{F}_{\mathsf{TLE}}$. Then, by Definition 2.4.2, there is an adversary $\mathcal{A}$ s.t. for every simulator $\mathcal{S}$ there is an environment $\mathcal{Z}$ s.t.:

$$|\Pr[\mathrm{EXEC}_{\mathcal{Z},\mathcal{A}}^{\Pi_{\mathsf{TLE}}} = 0] - \Pr[\mathrm{EXEC}_{\mathcal{Z},\mathcal{S}}^{\mathcal{F}_{\mathsf{TLE}}} = 0]| > \alpha(\lambda) \tag{4.1}$$

where $\alpha()$ is a non negligible function.

Now consider the specific simulator $\mathcal{S}$ below: At the beginning, $\mathcal{S}$ receives the corruption vector from $\mathcal{Z}$ and informs $\mathcal{A}$ as if it was $\mathcal{Z}$. When $\mathcal{S}$ gets the token back from $\mathcal{A}$, he sends the corruption vector to $\mathcal{F}_{\mathsf{TLE}}$. Moreover, $\mathcal{S}$ registers the encryption/decryption algorithms $(e_{\mathcal{S}}, d_{\mathcal{S}})$, which are the same as in protocol Π_{TLE}, namely $(e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}, d_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}})$. However, the *Extended encryption* is not the same, specifically the created cipher texts c_2, c_3 are equal to a random value. Observe that still the distribution of both (c_2, c_3) in both executions are still the same as both c_2, c_3 in the real protocol are random. If $\mathcal{S}$ receives an encryption request $(\mathsf{sid}, \mathrm{ENC}, \tau, \mathsf{tag}, \mathsf{Cl}, 0^{|m|}, P)$ from $\mathcal{F}_{\mathsf{TLE}}$ on behalf of an honest party P, he stores

the tuple $(\tau_{\mathsf{dec}}, \mathsf{tag}_m, \mathsf{Cl}, 0^{|m^*|}, c, \mathsf{nobroadcast}, P)$, where c is the encryption of $0^{|m^*|}$ by using the algorithm $e_{\mathcal{S}}$, he updates his list, named $L^{\mathcal{S}}_{\mathsf{RO}^*}$ (initially empty), for the generation of that ciphertext. Moreover, he updates his list for the second and the third argument of the encryption as if it was $\mathcal{F}_{\mathsf{RO}}$ (e.g c_2 and c_3). Then, he returns the token back to $\mathcal{F}_{\mathsf{TLE}}$.

Upon receiving $(\mathsf{sid}, \textsc{Advance\_Clock}, P)$ from $\mathcal{G}_{\mathsf{clock}}$ from an honest party P, $\mathcal{S}$ reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. Then, for every stored tuple $(\tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, 0^{|m_j|}, c_j, \mathsf{broadcast}, \cdot)$, he updates his list, named $L^{\mathcal{S}}_{\mathsf{RO}^*}$, with q evaluation queries for solving the ciphertexts issued by honest parties on previous rounds, as if it was $\mathcal{F}^*_{\mathsf{RO}}$ in the real protocol. Then, he seeks the permission for broadcasting the ciphertetext created for P in this round from $\mathcal{A}$ as if it was $\mathcal{F}_{\mathsf{BC}}$. If $\mathcal{A}$ allows the broadcast, he updates the tuples $(\tau_{\mathsf{dec}}, \mathsf{tag}_m, \mathsf{Cl}, 0^{|m^*|}, c, \mathsf{nobroadcast}, P)$ to $(\tau_{\mathsf{dec}}, \mathsf{tag}_m, \mathsf{Cl}, 0^{|m^*|}, c, \mathsf{broadcast}, P)$ and returns back to $\mathcal{F}_{\mathsf{TLE}}$ the resulting ciphertexts along with their difficulty issued by P in this round. When $\mathcal{S}$ receives an encryption request from $\mathcal{F}_{\mathsf{TLE}}$ on behalf of a corrupted party he reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$, he forwards the message to $\mathcal{A}$ as if it was from that party and keeps record both corrupted party's identity, message and the current time Cl (e.g. (P, m, Cl)). Then, $\mathcal{S}$ returns whatever he receives from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{TLE}}$ after updating his record with that response. In any of these cases, $\mathcal{S}$ keeps the randomness that he used for that task. In case $\mathcal{S}$ receives a decryption request from $\mathcal{F}_{\mathsf{TLE}}$ with ciphertext c and time label τ on behalf of an honest party, he does: If c was recorded as a ciphertext of a corrupted party as above, then $\mathcal{S}$ generates the witness $w_{\tau_{\mathsf{dec}}}$ similar to protocol Π_{TLE} as if it was an honest party and updates his list $L^{\mathcal{S}}_{\mathsf{RO}^*}$ exactly as $\mathcal{F}^*_{\mathsf{RO}}$ in protocol Π_{TLE} for consistency between the witness and the oracle queries. Specifically, $\mathcal{S}$ reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and records to $L^{\mathcal{S}}_{\mathsf{RO}^*}$ as many queries as the honest party in Π_{TLE} should do between the time that c was recorded from $\mathcal{S}$ and the current time Cl. Next, $\mathcal{S}$ generates the witness based on these queries exactly as in the real protocol. Then, $\mathcal{S}$ returns to $\mathcal{F}_{\mathsf{TLE}}$ the message $\{m, \perp\} \leftarrow d_{\mathcal{S}}(c, w_{\tau_{\mathsf{dec}}})$. The only way for $\mathcal{S}$ to be asked the opening of such a ciphertext is that the ciphertext is not legitimate (e.g. not issued through $\mathcal{F}_{\mathsf{TLE}}$). This can be easily observed by the $\mathcal{F}_{\mathsf{TLE}}$'s command interface. The $\perp$ occurs in the case that the algorithm detects no knowledge over the plaintext (recall the check $c_3 = \mathcal{H}(r_1||m)$ in Figure 4.4). If $\mathcal{S}$ receives a decryption request for a ciphertext c with time label τ from $\mathcal{F}_{\mathsf{TLE}}$ on behalf of a corrupted party, he forwards the message to $\mathcal{A}$ as if it was from that party. $\mathcal{S}$ returns whatever he receives from $\mathcal{A}$ as if it was the corrupted party back to $\mathcal{F}_{\mathsf{TLE}}$. In case $\mathcal{S}$ receives a random oracle query request ($\mathcal{F}_{\mathsf{RO}}$) from $\mathcal{F}_{\mathsf{TLE}}$ on behalf of a corrupted party, he forwards the message to $\mathcal{A}$ as if it was from that party. When $\mathcal{S}$ receives this request from $\mathcal{A}$ playing the role of $\mathcal{F}_{\mathsf{RO}}$, he sends the command $\textsc{Leakage}$ to $\mathcal{F}_{\mathsf{TLE}}$. Then $\mathcal{S}$ checks if the received record from $\mathcal{F}_{\mathsf{TLE}}$ contains any relation between a message m and the random oracle query that $\mathcal{S}$ received initially from the corrupted party. If $\mathcal{S}$ finds such relation, he programs the oracle so that ciphertext can be opened to message m. Then, he responds

to $\mathcal{A}$ as if it was the $\mathcal{F}_{\mathsf{RO}}$. For example, let us suppose that the oracle query is the value r_1. Remember that $\mathcal{S}$ issues all the ciphertexts, so he knows the randomness that it was used in each one of them. As a result, he can check if r_1 used for the production of a ciphertext. In case that he founds that r_1 was used for the production ciphertext c, he sends the command LEAKAGE to $\mathcal{F}_{\mathsf{TLE}}$. In the fortunate scenario where he finds in the received list a tuple that contains a message m and the ciphertext $c = (c_1, c_2, c_3)$, he registers and returns as if it was $\mathcal{F}_{\mathsf{RO}}$ the response $\mathcal{H}(r_1) = c_2 \oplus m$ to $\mathcal{A}$ (equivocation).

In the case $\mathcal{S}$ founds the oracle query but the list does not contain the message, he outputs "$\perp$" (meaning that the adversary was lucky enough to guess a plaintext before the time comes, or the adversary "broke" the security of the encryption scheme). Specifically, when $\mathcal{S}$ receives $(\mathsf{sid}, \text{QUERY}, x)$ from $\mathcal{F}_{\mathsf{TLE}}$ on behalf of a corrupted party he forwards the message to $\mathcal{A}$ as if it was from that party. When $\mathcal{S}$ receives the same message from $\mathcal{A}$ as if it was $\mathcal{F}_{\mathsf{RO}}$, he sends $(\mathsf{sid}, \text{LEAKAGE})$ to $\mathcal{F}_{\mathsf{TLE}}$. Upon receiving $(\mathsf{sid}, \text{LEAKAGE}, \{(m, c, \tau_{\mathsf{dec}}) \in L_{\mathsf{rec}}\}_{\tau_{\mathsf{dec}}:\tau_{\mathsf{dec}} \leq \mathsf{leak}(\mathsf{Cl})})$ from $\mathcal{F}_{\mathsf{TLE}}$, $\mathcal{S}$ searches into his database ($\mathcal{S}$ generates all the ciphertexts so he knows the randomness of each) for a ciphertext c_1 on message x. If such ciphertext does not exist, he behaves exactly like the $\mathcal{F}_{\mathsf{RO}}$. If it does, he searches the set $\{(m, c, \tau_{\mathsf{dec}}) \in L_{\mathsf{rec}}\}_{\tau_{\mathsf{dec}}:\tau_{\mathsf{dec}} \leq \mathsf{leak}(\mathsf{Cl})}$ to find a c such that $c[1] = c_1$. If $\mathcal{S}$ does not find such ciphertext, he outputs $\perp$, else he retrieves the corresponding message m and returns as the answer to the random oracle query the message $(\mathsf{sid}, \text{QUERY}, x, y = c[2] \oplus m)$ to $\mathcal{A}$ as if it was from $\mathcal{F}_{\mathsf{RO}}$. In any other case he behaves just like a random oracle. Finally, when $\mathcal{S}$ receives the command EVALUATE from $\mathcal{F}_{\mathsf{TLE}}$ on behalf of a corrupted party, he forwards the message to $\mathcal{A}$ as if it was that party. When $\mathcal{S}$ receives the EVALUATE command from $\mathcal{A}$ on behalf of the corrupted party as if it was $\mathcal{W}_q(\mathcal{F}_{\mathsf{RO}}^*)$, he behaves exactly as $\mathcal{W}_q(\mathcal{F}_{\mathsf{RO}}^*)$ in protocol Π_{TLE}.

By the assumption of $\mathcal{A}$ for $\mathcal{S}$ defined above there is an $\mathcal{Z}_{\mathcal{S}}$ such that Equation 4.1 holds. There are two possible ways for $\mathcal{Z}$ to distinguish the real from the ideal execution of the protocol based on the syntax of $\mathcal{F}_{\mathsf{TLE}}$.

Distinction when $\mathcal{F}_{\mathsf{TLE}}$ outputs $\perp$: The first way for $\mathcal{Z}$ to distinguish the two executions is when $\mathcal{F}_{\mathsf{TLE}}$ outputs the special $\perp$ symbol. This happens when $\mathcal{F}_{\mathsf{TLE}}$ detects the same ciphertext for two different messages, meaning that the Correctness property has been violated. In all other cases when $\mathcal{F}_{\mathsf{TLE}}$ returns $\perp$ the same occurs in the real execution, thus the $\mathcal{Z}$ can not distinct the two execution in such cases.

Distinction when leak is not "enough": Last, $\mathcal{Z}$ can distinct the two executions when $\mathcal{S}$ cannot retrieve the message m via the command LEAKAGE and $\mathcal{Z}$ managed to solve the puzzle that correspond to that message. Note that the puzzle is created by $\mathcal{S}$. As a result, $\mathcal{S}$ cannot equivocate the message correctly and $\mathcal{Z}$ can distinguish the real from the ideal execution. For example, if we have

a protocol that uses a TLE scheme such that it is not necessary for a party to ask all the oracle queries so that she can solve the puzzle at the desired time, instead she can solve it much faster (broken by design). In such cases, $\mathcal{F}_{\mathsf{TLE}}$ is not realizable.

Lets us suppose that the pair $(e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}})$ satisfies the Correctness property. We construct an adversary $\mathcal{B}$ that can break the qSecurity with probability at least $\tilde{\alpha}(\lambda)$, where $\tilde{\alpha}()$ a non negligible function.

The only way for $\mathcal{Z}_{\mathcal{S}}$ to distinguish the real from the ideal execution with non-negligible probability based on the argumentation of Paragraphs 4.2.2 and 4.2.2 is to decrypt/solve the first argument of a ciphertext/puzzle, namely c_1, generated by an honest party before the time comes and issues a random oracle query on it so that $\mathcal{Z}$ retrieves the message. This is possible if $\mathcal{Z}_{\mathcal{S}}$ is able to construct a witness $w_{\tau_{\mathsf{dec}}}$ for an honest generated ciphertext c_1 via the queries issued by a corrupted party to $\mathcal{W}_{\mathcal{F}^*_{\mathsf{RO}}}$ in the real execution of the protocol or in $\mathcal{S}$ in the ideal execution given that the global time Cl provided by $\mathcal{G}_{\mathsf{clock}}$ is strictly smaller than τ_{dec}. Next, $\mathcal{Z}_{\mathcal{S}}$ will request a random oracle query from a corrupted party with the query value to be the plaintext of the ciphertext c_1. Next, $\mathcal{S}$ in order to equivocate correctly, he needs the corresponding message. But if the time of that message has not come yet (e.g. $\mathsf{Cl} < \tau_{\mathsf{dec}}$), the recorded table that $\mathcal{S}$ will request from $\mathcal{F}_{\mathsf{TLE}}$ via the Leakage command, it will not contain that message. As a result, $\mathcal{S}$ will fail to equivocate correctly and $\mathcal{Z}_{\mathcal{S}}$ can distinguish the two executions. Now $\mathcal{B}$ takes advantage of that environment, and uses it in order to win the experiment $\mathbf{EXP}_{\mathsf{TLE}}$ with non negligible probability in the following way: $\mathcal{B}$ simulates the interface to the environment as in the ideal execution of the protocol in the presence of the global clock. Specifically, $\mathcal{B}$ runs every procedure locally simulating every role in the ideal execution, without engaging Ch at all. Every time $\mathcal{B}$ receives q queries $(\mathsf{sid}, \mathsf{Evaluate}, \{x_j\}_{j=0}^{p_l(\lambda)})$ where p_l a polynomial function, from $\mathcal{Z}$ as if it was a corrupted party, he increases by 1 the local counter Cl, (similar to the one Ch has) and forwards $(\mathsf{sid}, \mathsf{Evaluate}, \{x_j\}_{j=0}^{p_l(\lambda)})$ to the oracle $\mathcal{O}_{\mathsf{eval}}$ through the challenger in $\mathbf{EXP}_{\mathsf{TLE}}$. Then returns to $\mathcal{Z}$ whatever it receives. After that point if $\mathcal{Z}$ does not send a clock advancement command, $\mathcal{B}$ does not allow $\mathcal{Z}$ to issue more queries. Now, $\mathcal{B}$ knows that the environment will make at most $p_{\mathcal{H}}(\lambda), p_{\mathsf{enc}}(\lambda)$ random oracle and encryption queries respectively, where $p_{\mathcal{H}}(), p_{\mathsf{enc}}()$ are polynomial functions. At least one of these random oracle queries made by $\mathcal{Z}_{\mathcal{S}}$, from the observation at the beginning of the Paragraph, will contain the plaintext (namely the value r_1 as described in Figure 4.4) of one of the $p_{\mathsf{enc}}(\lambda)$ ciphertexts that has been decrypted by $\mathcal{Z}_{\mathcal{S}}$ before its decryption time with non negligible probability $\alpha(\lambda)$. Therefore, $\mathcal{B}$ picks $j_1 \xleftarrow{\$} \{1, \ldots, p_{\mathsf{enc}}(\lambda)\}$. When $\mathcal{Z}_{\mathcal{S}}$ issues the j_1-th encryption query $(\mathsf{sid}, \textsc{Enc}, m, \tau_{\mathsf{dec}})$ to an honest party simulated by $\mathcal{B}$, $\mathcal{B}$ proceeds as follows: If $\tau_{\mathsf{dec}} > \mathsf{Cl}$ ($\mathcal{B}$ simulates $\mathcal{G}_{\mathsf{clock}}$), then he sends $(\textsc{Challenge}, \tau_{\mathsf{dec}} - \mathsf{Cl})$ to Ch. When $\mathcal{B}$ receives $(\textsc{Challenge}, \tau_{\mathsf{dec}} - \mathsf{Cl}, c_1)$ from Ch, $\mathcal{B}$ picks c_2, c_3 exactly as $\mathcal{F}_{\mathsf{TLE}}$ and returns $(\mathsf{sid}, \textsc{Enc}, m, \tau, c \leftarrow (c_1, c_2, c_3))$ to $\mathcal{Z}_{\mathcal{S}}$. Then, $\mathcal{B}$ picks $j_2 \xleftarrow{\$} \{1, \ldots, p_{\mathcal{H}}(\lambda)\}$. When $\mathcal{Z}_{\mathcal{S}}$ issues the j_2-th random oracle

query $(\mathsf{sid}, \text{QUERY}, x)$ to a corrupted party, $\mathcal{B}$ sends x to Ch as the answer to the challenge. It can be seen that the probability x to be the answer of the challenge is at least $1/(p_{\mathsf{enc}}(\lambda)p_{\mathcal{H}}(\lambda)) \cdot \tilde{\alpha}(\lambda)$. Note that, although that the ciphertexts of the honest parties simulated by $\mathcal{B}$ are created based on the $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ simulated by $\mathcal{B}$ as well in contrast with the challenged one that is created from Ch trough $\mathcal{O}_{\mathsf{eval}}$ the distribution are exactly the same and the probability for collision on inputs in negligible. □

On the importance of instantiating $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ with $\mathcal{F}^*_{\mathsf{RO}}$: In our proof, we instantiate the functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ with $\mathcal{F}^*_{\mathsf{RO}}$, so that $\mathcal{Z}$ cannot bypass the interaction with the functionality wrapper and thus breaches the security argument of our proof. Let us suppose for instance that $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ was not instantiated with $\mathcal{F}^*_{\mathsf{RO}}$, instead it was instantiated by any other functionality parameterized by a constant distribution $\mathbf{D}_x$. In that case, $\mathcal{Z}$ could simply sample values from that distribution locally, solve the puzzle, and encrypt/decrypt any messages in a single round. Specifically, in Rivest et al. (1996), the procedure for solving a time-lock puzzle consists in repeatedly squaring a base a specific polynomial number of times. However, this computation is deterministic. So any PPT Turing machine, including $\mathcal{Z}$, can produce identical results if they engaged in the same computation without necessarily interacting with the functionality wrapper at all, breaching the security argument of our proof.

A promising way to tackle such deterministic $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ could be to allow the encryption/decryption algorithms to interact with the oracle through the functionality wrapper, verifying that the provided solution for the puzzle was constructed through the evaluation oracle. Of course, this would require more modelling assumptions such as the definition of the encryption/decryption algorithms as ITMs so that they could interact with the oracle. On the other hand, if we instantiate $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ with $\mathcal{F}^*_{\mathsf{RO}}$ then the modelling is more natural. We address the limitations of Rivest et al. (1996) by defining a new construction, namely *Astolabous*, defined in the Section 4.3. It is worth mentioning that both random oracles in the protocol Π_{TLE} are local and thus programmable.

4.3 Astrolabous: a UC-secure TLE construction

We present and prove that our relative TLE construction is a secure time-lock encryption scheme according to Definition 4.2.1. Our scheme combines the construction of Mahmoody et al. (2011) and Rivest et al. (1996).

First, we present our TLE construction, namely *Astrolabous*, and the proof of security, i.e. Astrolabous satisfies Definition 4.2.1. Finally, for the sake of completeness, we present the equivocable Astrolabous algorithm, which is the algorithm that is used in the hybrid protocol in Figure 4.4.

We did not adopt any of the TLE constructions provided in Rivest et al. (1996) and Mahmoody et al. (2011) because they can not provide us with the

necessary security properties we are seeking in our theoretical framework so that we can UC realise $\mathcal{F}_{\mathsf{TLE}}$, as we explain in detail in the next paragraph.

Neccesity for defining Astrolabous: The construction in Rivest et al. (1996) is very simple and easily implementable, which is not the case in our theoretical framework (e.g. UC framework). The security of the construction is based on the repeated squaring problem, which states that: "Given a composite number n and an element $b \in \mathbb{Z}_n$ it is hard to compute b^{2^τ} with less than τ repeated squaring". To define this construction in UC, we have to introduce this new hardness assumption and we have to correlate it with the pair of encryption/decryption algorithms. Specifically, we would have to define an oracle, like the $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, that is responsible for that computation. The algorithms must communicate with the oracle to ensure that a provided witness is created only from queries through the oracle rather than local computations, where we can not restrict the access via a functionality wrapper and thus can not capture the whole concept of TLE in UC framework. If we want to formulate the communication of the encryption/decryption algorithms with the functionality oracle, we have to define them as ITMs rather than just plain algorithms. This approach is rather new to UC and out of the scope of this work. Instead, we searched solutions where the functionality oracle is the random oracle, such as in Mahmoody et al. (2011). With that approach, the algorithm need not communicate with the oracle because the computations to solve the time-lock puzzle are not deterministic (e.g. like in Rivest et al. (1996)), in fact, they are probabilistic. So, even if the adversary knows the distribution where the oracle responses to the queries, he can not predict the actual outcome. As a result, when the adversary tries to decrypt the message that is created based on the random oracle functionality, it is impossible to do so without interacting with the oracle first. However, we can not adapt directly the construction from Mahmoody et al. (2011) because the adversary can learn parts of the plaintext before the desired decryption time, leading to a weak encryption scheme concerning cryptographic standards Kościelny et al. (2013). That is why we use the construction from Mahmoody et al. (2011) to encrypt not the actual message but the key that is used to encrypt our message with some symmetric encryption scheme, like AES, in the same spirit as Rivest et al. (1996). Although that this construction without the extension presented previously is enough if we want to stress the security of Astrolabous against a standalone definition, like the one in Subsection 4.3.3, for a UC realization is not enough as we have already discussed.

Description of the *Astrolabous* scheme Initially, we provide the necessary glossary in Table 4.2. We name our construction Astrolabous from the ancient Greek clock device Astrolabe, which was used by the astronomers of that era to perform different types of calculations including the measurement of the altitude

above the horizon of a celestial body, identification of stars and the determination of the local time.

We refer to the encryption/decryption algorithms of the Astrolabous scheme in Subsection 4.3.1 as $\mathsf{AST.enc}, \mathsf{AST.dec}$ where AST is the abbreviation of Astrolabous. In Subsection 4.3.2, we refer to the equivocable encryption/decryption algorithms as $\mathsf{EAST.enc}, \mathsf{EAST.dec}$ where the letter E indicates the extended algorithms of the Astrolabous scheme.

Table 4.2: The glossary of Astrolabous scheme.

Notation	Description
$\mathsf{E} = (\mathsf{enc}, \mathsf{dec})$	a symmetric key encryption scheme
$\mathcal{H}, \mathcal{G}$	two hash functions (modelled as random oracles)
$b \xleftarrow{\$} \mathsf{D}$	b is sampled uniformly at random from D
$\mathsf{X.enc}, \mathsf{X.dec}$	encryption and decryption algorithm respectively of scheme X
$\oplus$	the XOR bit operation, e.g. $0 \oplus 1 = 1, 1 \oplus 1 = 0$
$x\|\|y$	the concatenation of two bit strings x and y

4.3.1 The $(\mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}, \mathsf{AST.dec}_{\mathsf{E},\mathcal{H}})$ scheme

$\mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}(m, \tau_{\mathsf{dec}})$**:** The algorithm accepts as input the message m and the time-lock's puzzle difficulty τ_{dec} [1] and does:

- Picks $k_{\mathsf{E}} \xleftarrow{\$} \mathsf{K_E}$, where $\mathsf{K_E}$ is the key space of the symmetric encryption scheme E and the size of the key is equal to the domain of the hash function $\mathcal{H}$ equal to $p_1(\lambda)$. Then compute $c_{m,k_{\mathsf{E}}} \leftarrow \mathsf{enc}(m, k_{\mathsf{E}})$.
- It picks $r_0||r_1||\ldots||r_{q\tau_{\mathsf{dec}}-1} \xleftarrow{\$} \{0,1\}^{\mathsf{p}_2(\lambda)}$ and computes $c_{k_{\mathsf{E}},\tau_{\mathsf{dec}}} \leftarrow (r_0, r_1 \oplus \mathcal{H}(r_0), r_2 \oplus \mathcal{H}(r_1), \ldots, k_{\mathsf{E}} \oplus \mathcal{H}(r_{q\tau_{\mathsf{dec}}-1})$[2].
- It outputs $c = (\tau_{\mathsf{dec}}, c_{m,k_{\mathsf{E}}}, c_{k_{\mathsf{E}},\tau_{\mathsf{dec}}})$ as the ciphertext.

$\mathsf{AST.dec}_{\mathsf{E},\mathcal{H}}(c, w_{\tau_{\mathsf{dec}}})$**:** The algorithm accepts as input the ciphertext c of the form $(\tau_{\mathsf{dec}}, c_{m,k_{\mathsf{E}}}, c_{k_{\mathsf{E}},\tau_{\mathsf{dec}}})$ and the witness $w_{\tau_{\mathsf{dec}}} = (r_0, \mathcal{H}(r_0), \mathcal{H}(r_1), \ldots, \mathcal{H}(r_{q\tau_{\mathsf{dec}}-1}), c)$ that can be computed by issuing $q\tau_{\mathsf{dec}}$ random oracle queries. Specifically, to solve the puzzle the first oracle query is r_0 and the response $\mathcal{H}(r_0)$. Then, the decryptor computes the value r_1 from $c_{k_{\mathsf{E}}}$ by using the *XOR* operation such as $r_1 \leftarrow c_{k_{\mathsf{E}},\tau_{\mathsf{dec}}}[1] \oplus \mathcal{H}(r_0)$. Similarly, it computes the pair of values $(r_2, \mathcal{H}(r_2)), \ldots, (r_{q\tau_{\mathsf{dec}}-1}, \mathcal{H}(r_{q\tau_{\mathsf{dec}}-1}))$. Then it does:

[1] Note that this time difficulty is relative, that means that it specifies the duration for solving the puzzle rather than the specific date at which the puzzle should be solved.

[2] To do this efficiently all the hash queries can be performed simultaneously as k_{E} and $r_0||r_1||\ldots||r_{q\tau_{\mathsf{dec}}-1}$ are known. In the UC setting, the party sends $(\mathsf{sid}, \textsc{Evaluate}, \tau_{\mathsf{dec}})$ to $\mathcal{W}_q$ and receives back $(\mathsf{sid}, \textsc{Evaluate}, \tau_{\mathsf{dec}}, \{(r_j, y_j)\}_{j=0}^{q\tau_{\mathsf{dec}}-1})$.

- It computes $k_{\mathsf{E}} = \mathcal{H}(r_{q\tau_{\mathsf{dec}}-1}) \oplus c_{k_{\mathsf{E}},\tau_{\mathsf{dec}}}[q\tau_{\mathsf{dec}}]$, where $c_{k_{\mathsf{E}},\tau_{\mathsf{dec}}}[j]$ indicates the jth element in vector $c_{k_{\mathsf{E}},\tau_{\mathsf{dec}}}$.
- It computes and outputs $m \leftarrow \mathsf{dec}(c_{m,k_{\mathsf{E}}}, k_{\mathsf{E}})$.

In Table 4.3, we summarize the oracle, algorithms, functions and relation that define a TLE scheme as in Definition 4.2.1. We instantiate these to specify our TLE construction.

Table 4.3: Oracle, algorithms, functions and relation that define a TLE construction.

TLE items	Description
$\mathcal{O}_{\mathsf{eval}}$	the oracle to which the parties issue queries for solving/creating time-lock puzzles
$(e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}})$	the pair of encryption/decryption algorithms with respect to the oracle $\mathcal{O}_{\mathsf{eval}}$
$\mathsf{f}_{\mathsf{state}}$	the state function that prepares the next oracle query to $\mathcal{O}_{\mathsf{eval}}$
$\mathsf{f}_{\mathsf{puzzle}}$	the puzzle function that extracts the time-lock puzzle from a given ciphertext
$\mathsf{wit\_con}$	the witness construction function that returns the solution of the puzzle or the witness if that is possible
$\mathsf{R}_{\mathcal{O}_{\mathsf{eval}}}$	the relation that specifies when a witness w is a solution to a puzzle c with difficulty τ

We instantiate the items from Table 4.3 based on our construction as shown below.

1. The oracle $\mathcal{O}_{\mathsf{eval}}$ is the random oracle RO.
2. The encryption and decryption algorithms $(e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}})$ are described as $\mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}, \mathsf{AST.dec}_{\mathsf{E},\mathcal{H}}$. Our algorithm is relative, meaning that we define the difficulty of the time-lock puzzle rather than the specific time that the message will eventually open. For our algorithms to be compatible with the UC setting, for a given time τ_{dec} we must define the difficulty of the puzzle. In that case, given the current time is Cl, the puzzle complexity is $\tau_{\mathsf{dec}} - \mathsf{Cl}$. The time τ_{dec} gives us the essence of absolute time that a ciphertext should be opened. On the other hand, both constructions in Mahmoody et al. (2011); Rivest et al. (1996) function in relative time. To compute relative time, both values Cl and τ_{dec} are provided to $e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}$.
3. The state function $\mathsf{f}_{\mathsf{state}}$ for a ciphertext $c = (\tau_{\mathsf{dec}}, c_{m,k_{\mathsf{E}}}, c_{k_{\mathsf{E}},\tau_{\mathsf{dec}}})$ as described previously, is defined as:

$$\mathsf{f}_{\mathsf{state}}(c, 0, \mathsf{Null}) = c_{k_{\mathsf{E}},\tau_{\mathsf{dec}}}[0] \tag{4.2}$$

and $\forall j \in \{1, \ldots, q(\tau_{\mathsf{dec}} - \mathsf{Cl}) - 1\}$ it holds that:

$$\mathsf{f}_{\mathsf{state}}(c, j, y = \mathcal{H}(r_{j-1})) = y \oplus c_{k_{\mathsf{E}}, \tau_{\mathsf{dec}}}[j] \tag{4.3}$$

4. The puzzle function $\mathsf{f}_{\mathsf{puzzle}}$ for a ciphertext $c = (\tau_{\mathsf{dec}}, c_{m,k_{\mathsf{E}}}, c_{k_{\mathsf{E}},\tau_{\mathsf{dec}}})$ is defined as:

$$\mathsf{f}_{\mathsf{puzzle}}(c) = c_{k_{\mathsf{E}},\tau_{\mathsf{dec}}} \tag{4.4}$$

5. The witness construction function $\mathsf{wit\_con}$ accepts the input described in Figure 4.4 and outputs the witness described in the same figure.

6. A pair $(w_{\tau_{\mathsf{dec}}} = (r_0, \mathcal{H}(r_0), \mathcal{H}(r_1), \ldots, \mathcal{H}(r_{q(\tau_{\mathsf{dec}}-\mathsf{Cl})-1})), (\mathsf{f}_{\mathsf{puzzle}}(c), \tau))$ is in $\mathsf{R}_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}$, where $w_{\tau_{\mathsf{dec}}}$ and c have the same form as in the description of $\mathsf{AST.dec}_{\mathsf{E},\mathcal{H}}$, if $|w_{\tau_{\mathsf{dec}}}| = |\mathsf{f}_{\mathsf{puzzle}}(c)|$ and $w[j] = c_{k_{\mathsf{E}},\tau_{\mathsf{dec}}}[j] \oplus \mathcal{H}(w[j-1])$ for all $j \in [0, q(\tau_{\mathsf{dec}} - \mathsf{Cl}) - 2]$, where $w[-1] = 1$.

The following theorem states that our TLE construction satisfies Definition 4.2.1.

Theorem 2. *Let* $\mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}, \mathsf{AST.dec}_{\mathsf{E},\mathcal{H}}$ *be the pair of* encryption/decryption *algorithms just described. If the underlying symmetric encryption scheme* E *satisfies* $\mathsf{IND-CPA}$ *security and* correctness*, then the pair* $(\mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}, \mathsf{AST.dec}_{\mathsf{E},\mathcal{H}})$ *is a secure TLE scheme according to Definition 4.2.1 in the random oracle model.*

Proof. In order to prove that the pair $\mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}, \mathsf{AST.dec}_{\mathsf{E},\mathcal{H}}$ satisfies Definition 4.2.1 we need to prove that it satisfies both Correctness and qSecurity.

Proving Correctness: We know that the decryption algorithm of the symmetric scheme E returns the correct plaintext with probability 1 Daemen and Rijmen (2002). Specifically it holds $\forall m \in \mathsf{M}$:

$$\Pr[k_{\mathsf{E}} \xleftarrow{\$} \mathsf{K}_{\mathsf{E}}; m' \leftarrow \mathsf{dec}(\mathsf{enc}(m, k_{\mathsf{E}}), k_{\mathsf{E}}) : m = m'] = 1$$

where K_{E} and M is the key space and message space of the E respectively.

Let $\mathsf{R}_{\mathcal{H}}$ be the relation as defined in Subsection 4.3.1 with $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ instantiated by the random oracle, abbreviating here as $\mathcal{H}$, that correlates the time τ_{dec} and the puzzle c with the correct witness for decryption $w_{\tau_{\mathsf{dec}}}$. Because the correct decryption of $\mathsf{AST.dec}_{\mathsf{E},\mathcal{H}}$ is solely based on the correct decryption of the underlying symmetric scheme E, $\forall m \in \mathsf{M}$ and $\tau_{\mathsf{dec}} \leftarrow \mathbb{N}$ it holds that:

$$\Pr\left[\begin{array}{l} m' \leftarrow \mathsf{AST.dec}_{\mathsf{E},\mathcal{H}}(\mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}(m, \tau_{\mathsf{dec}}), w_{\tau_{\mathsf{dec}}}) \\ \mathsf{R}_{\mathcal{H}}(w_{\tau_{\mathsf{dec}}}, \mathsf{f}_{\mathsf{puzzle}}((\mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}(m, \tau_{\mathsf{dec}})), \tau_{\mathsf{dec}})) \end{array} : m' = m\right] = 1$$

Proving qSecurity: We argue about qSecurity by defining a new experiment, similar to the one in Figure 4.5, where the decryption key used in the symmetric encryption scheme E does not appear at all but still the distribution of messages the adversary sees in both experiments are statistically close based on the security parameter λ. Thus, there is no way for the adversary to learn the real key with less queries than the maximum allowed number and as long as E is secure, the adversary can retrieve the plaintext only with negligible probability.

First, let us define the event that the adversary $\mathcal{B}$ wins in the experiment $\mathbf{EXP}_{\mathsf{TLE}}$ as $\mathsf{Win}_{\mathbf{EXP}_{\mathsf{TLE}}}$ and the event to make less oracle queries than the expected ones for the challenged ciphertext (e.g. $\tau > \mathsf{Cl}_{\mathsf{exp}}$, see Figure 4.5) as Bad. Note that it holds that $\mathsf{Win}_{\mathbf{EXP}_{\mathsf{TLE}}} \subseteq \mathsf{Bad}$ because the necessary requirements for the adversary to win the $\mathbf{EXP}_{\mathsf{TLE}}$ is by making less oracle queries than the expected ones for the challenged ciphertext. Thus, it holds that

$$\Pr[\mathsf{Win}_{\mathbf{EXP}_{\mathsf{TLE}}}] = \Pr[\mathsf{Win}_{\mathbf{EXP}_{\mathsf{TLE}}} \wedge \mathsf{Bad}] \tag{4.5}$$

Thus, we need to show that $\Pr[\mathsf{Win}_{\mathbf{EXP}_{\mathsf{TLE}}} \wedge \mathsf{Bad}]$ is negligible with respect to λ. Let us define the experiment $\mathbf{EXP}^{*}_{\mathsf{TLE}}$ which is the same as $\mathbf{EXP}_{\mathsf{TLE}}$ except that the challenged ciphertext does not contain the key that is used to encrypt the message with the symmetric encryption scheme. Specifically, the last part of the time-lock puzzle in the challenged ciphertext in $\mathbf{EXP}_{\mathsf{TLE}}$ is $k_{\mathsf{E}} \oplus \mathcal{H}(r_{q\tau_{\mathsf{dec}}-1})$, whereas in $\mathbf{EXP}^{*}_{\mathsf{TLE}}$ it is $\mathcal{H}(r_{q\tau_{\mathsf{dec}}-1})$ instead. Observe that the distribution of messages that $\mathcal{B}$ receives in the two experiments are exactly the same, in the case the adversary did less oracle queries for the challenged ciphertext (the event Bad), because we are in the random oracle model. So we have:

$$\Pr[\mathsf{Win}_{\mathbf{EXP}_{\mathsf{TLE}}} \wedge \mathsf{Bad}] = \Pr[\mathsf{Win}_{\mathbf{EXP}^{*}_{\mathsf{TLE}}} \wedge \mathsf{Bad}] \tag{4.6}$$

In the case event Bad does not happen, $\mathcal{B}$ can retrieve the key of the challenged ciphertext from the puzzle. As a result, the distributions of messages in the two experiments are no longer the same because the key that the challenged ciphertext was created with and the key that $\mathcal{B}$ retrieved from the puzzle in $\mathbf{EXP}^{*}_{\mathsf{TLE}}$ do not match.

We argue that the event $\mathsf{Win}_{\mathbf{EXP}^{*}_{\mathsf{TLE}}} \wedge \mathsf{Bad}$ happens with negligible probability. Let us assume that:

$$\Pr[\mathsf{Win}_{\mathbf{EXP}^{*}_{\mathsf{TLE}}} \wedge \mathsf{Bad}] > \alpha(\lambda) \tag{4.7}$$

where α is a non-negligible function. We construct an adversary $\mathcal{B}_{\mathsf{IND-CPA}}$ that uses the adversary $\mathcal{B}$ to win in the $\mathsf{IND-CPA}$ game of the symmetric scheme E with non-negligible probability. Specifically, $\mathcal{B}_{\mathsf{IND-CPA}}$ works as follows:

He initializes the algorithms $e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}}$, responds to $\mathcal{B}$ and keeps the same counters/database as if it was Ch and $\mathcal{O}_{\mathsf{eval}}$ in the experiment $\mathbf{EXP}^{*}_{\mathsf{TLE}}$ except when he receives the challenged query from $\mathcal{B}$ for a labelling τ. When the latter happens, $\mathcal{B}_{\mathsf{IND-CPA}}$ chooses two random messages $m_0, m_1 \xleftarrow{\$} \mathbf{M}_{\lambda}$ and sends them to the challenger of the $\mathsf{IND-CPA}$ game. Upon receiving the ciphertext

c back from the challenger, $\mathcal{B}_{\mathsf{IND-CPA}}$ picks $(r_0||r_1||\dots||r_{q\tau-1}) \xleftarrow{\$} \{0,1\}^{\mathsf{p}_2(\lambda)}$(see description: 4.3.1) and computes $c_\tau \leftarrow (r_0, r_1 \oplus \mathcal{H}(r_0), r_2 \oplus \mathcal{H}(r_1), \dots, \mathcal{H}(r_{q\tau-1}))$ where the random oracle calls $\mathcal{H}(\cdot)$ are simulated by $\mathcal{B}_{\mathsf{IND-CPA}}$. Then, he returns (τ, c, c_τ) to $\mathcal{B}$ as if it was Ch. Observe that, $\mathcal{B}_{\mathsf{IND-CPA}}$ does not know the key that it is used for the production of the ciphertext c and thus the probability to create a time puzzle c_τ where the actual key appears in the last XOR operation would be negligible. For that reason it was necessary to define the intermediate experiment $\mathbf{EXP}^*_{\mathsf{TLE}}$.

At some point, $\mathcal{B}_{\mathsf{IND-CPA}}$ receives the answer for the challenged ciphertext, namely $\tilde{m}$, from $\mathcal{B}$. If $\tilde{m} = m_0 \vee \tilde{m} = m_1$, $\mathcal{B}_{\mathsf{IND-CPA}}$ returns $\tilde{m}$ to the challenger of $\mathsf{IND-CPA}$ as the answer to the challenged ciphertext, else it returns m_b where $b \xleftarrow{\$} \{0,1\}$.

Let us define the event $\mathcal{B}_{\mathsf{IND-CPA}}$ to win the experiment $\mathsf{IND-CPA}$ as $\mathsf{Win}_{\mathsf{IND-CPA}}$. Observe that, if $\mathcal{B}$ correctly finds the message in experiment $\mathbf{EXP}^*_{\mathsf{TLE}}$ and the event Bad holds then $\mathcal{B}_{\mathsf{IND-CPA}}$ wins as well in the experiment $\mathsf{IND-CPA}$. Specifically:

$$\Pr[\mathsf{Win}_{\mathsf{IND-CPA}}] = \Pr[\mathsf{Win}_{\mathsf{IND-CPA}}|\mathsf{ABad}]\Pr[\mathsf{ABad}] + \Pr[\mathsf{Win}_{\mathsf{IND-CPA}}|\overline{\mathsf{ABad}}]\Pr[\overline{\mathsf{ABad}}] \tag{4.8}$$

where ABad is the abbreviation for the event $\mathsf{Win}_{\mathbf{EXP}^*_{\mathsf{TLE}}} \wedge \mathsf{Bad}$.

By the description of the adversary $\mathcal{B}_{\mathsf{IND-CPA}}$, we have that $\Pr[\mathsf{Win}_{\mathsf{IND-CPA}}|\mathsf{ABad}] = 1$ and $\Pr[\mathsf{Win}_{\mathsf{IND-CPA}}|\overline{\mathsf{ABad}}] \geq 1/2$. Therefore, by Equation (4.8), it holds that:

$$\Pr[\mathsf{Win}_{\mathsf{IND-CPA}}] \geq 1/2 + 1/2\Pr[\mathsf{ABad}] \tag{4.9}$$

By Equations (4.7),(4.9) it holds that:

$$\Pr[\mathsf{Win}_{\mathsf{IND-CPA}}] > 1/2 + \alpha(\lambda)/2 \tag{4.10}$$

which is a contradiction. As a result it holds that:

$$\Pr[\mathsf{Win}_{\mathbf{EXP}^*_{\mathsf{TLE}}} \wedge \mathsf{Bad}] = \mathsf{negl}(\lambda) \tag{4.11}$$

Finally, by Equations (4.5),(4.6),(4.11) we have:

$$\Pr[\mathsf{Win}_{\mathbf{EXP}_{\mathsf{TLE}}}] = \mathsf{negl}(\lambda) \tag{4.12}$$

which completes the proof. □

4.3.2 Equivocable $(\mathsf{EAST.enc}_{\mathsf{E},\mathcal{H},\mathcal{G}}, \mathsf{EAST.dec}_{\mathsf{E},\mathcal{H},\mathcal{G}})$ scheme

For our purposes, it is not enough to adopt directly a TLE construction and make security claims in the UC framework because we cannot equivocate, which is essential. For that reason, we have shown in our hybrid protocol in Figure 4.4 how to extend any TLE construction in order for our security claims to be compatible with the UC framework. We provide the description below. $\mathsf{EAST.enc}_{\mathsf{E},\mathcal{H},\mathcal{G}}$**:** The algorithm accepts as input the message m and the time-lock puzzle difficulty τ_{dec} and does the following:

- It picks $r_1 \xleftarrow{\$} \{0,1\}^{\mathsf{p}_3(\lambda)}$ and computes $c_1 \leftarrow \mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}(r_1, \tau_{\mathsf{dec}})$.

- It computes $c_2 \leftarrow \mathcal{G}(r_1) \oplus m$ and $c_3 \leftarrow \mathcal{G}(r_1||m)$.

- It outputs $c = (c_1, c_2, c_3)$.

$\mathsf{EAST.dec}_{\mathsf{E},\mathcal{H},\mathcal{G}}(c, w_{\tau_{\mathsf{dec}}})$**:** The algorithm accepts as input the ciphertext c and the witness $w_{\tau_{\mathsf{dec}}}$:

- It computes $r_1 \leftarrow \mathsf{AST.dec}_{\mathsf{E},\mathcal{H}}(c_1, w_{\tau_{\mathsf{dec}}})$ and $m \leftarrow \mathcal{G}(r_1) \oplus c_2$.

- If $c_3 \neq \mathcal{G}(r_1||m)$ it outputs $\perp$, else it outputs m.

4.3.3 IND-CPA-TLE security

Game-based definitions are often natural and easy to use. Unfortunately, the experiment $\mathbf{EXP}_{\mathsf{TLE}}$ presented in Figure: 4.5 is not enough to argue about the security of a TLE scheme on its own, and is only useful in the context of the Theorem: 1. The reason is that $\mathbf{EXP}_{\mathsf{TLE}}$ argues about only the onewayness of a TLE scheme, leaving aside any semantic security. On the other hand, it is enough for the proof of Theorem: 1 as we use an extension of the TLE scheme in the random oracle model and not the scheme as it is.

Below, we present the analogous experiment of the IND-CPA security notion in the time-lock setting. In a nutshell, this experiment is the same as the one in Figure: 4.5 except that the adversary in the CHALLENGE command specifies two messages (m_0, m_1) as in the classical IND-CPA game. Again, in order to win the game, the adversary $\mathcal{B}$ must guess correctly which of the two messages is encrypted by the challenger Ch without engaging with the oracle more than the desired amount of times. In case he wins, that would mean that he managed to "break" the TLE scheme in the sense that he decrypted the message before its decryption time.

The experiment $\mathbf{EXP}_{\mathsf{IND-CPA-TLE}}(\mathcal{B}, \mathcal{O}_{\mathsf{eval}}, e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}}, \mathsf{f}_{\mathsf{state}}, \mathsf{f}_{\mathsf{puzzle}}, q)$

Initialization Phase.

- Ch is initialized with $e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}}$ and sends them to $\mathcal{B}$. In addition, it creates a local time counter $\mathsf{Cl}_{\mathsf{exp}}$.

Learning Phase.

- When $\mathcal{B}$ issues the query $(\textsc{Evaluate}, x)$ to $\mathcal{O}_{\mathsf{eval}}$ through the Ch, he gets back $(\textsc{Evaluate}, x, y)$.
- $\mathcal{B}$ can request the encryption of a message $m \in \mathbf{M}_\lambda$ with time label τ_{dec} by sending $(\textsc{Enc}, m, \tau_{\mathsf{dec}})$ to Ch.
- When Ch receives a $(\textsc{Enc}, m, \tau_{\mathsf{dec}})$ request from $\mathcal{B}$, it runs the algorithm $e_{\mathcal{O}_{\mathsf{eval}}}(m, \tau_{\mathsf{dec}}) \rightarrow c$ and returns c to $\mathcal{B}$.
- Ch increases $\mathsf{Cl}_{\mathsf{exp}}$ by 1 every time $\mathcal{B}$ queries $\mathcal{O}_{\mathsf{eval}}$ q times.
- $\mathcal{B}$ can request the decryption of a ciphertext c by sending $(\textsc{Dec}, c, w)$ to Ch. Then, Ch just runs the algorithm $d_{\mathcal{O}_{\mathsf{eval}}}(c, w) \rightarrow y \in \{m, \perp\}$ and returns to $\mathcal{B}$ $(\textsc{Dec}, c, w, y)$.

Challenge Phase.

- $\mathcal{B}$ can request for a single time a challenge from Ch by sending $(\textsc{Challenge}, (m_0, m_1), \tau)$. Then, Ch picks a value $b \xleftarrow{\$} \{0,1\}$ and sends $(\textsc{Challenge}, \tau, c \leftarrow e_{\mathcal{O}_{\mathsf{eval}}}(m_b, \tau - \mathsf{Cl}_{\mathsf{exp}}))$ to $\mathcal{B}$. Then, $\mathcal{B}$ is free to repeat the *Learning Phase*.
- $\mathcal{B}$ sends as the answer of the challenge the message $(\textsc{Challenge}, \tau, c, m_{b^*})$ to Ch.
- If $(m_{b^*} = m_b) \wedge (\tau > \mathsf{Cl}_{\mathsf{exp}})$ (i.e. $\mathcal{B}$ manages to decrypt c_r before the decryption time comes) then $\mathbf{EXP}_{\mathsf{TLE}}$ outputs 1. Else, $\mathbf{EXP}_{\mathsf{TLE}}$ outputs 0.

Figure 4.6: Experiment $\mathbf{EXP}_{\mathsf{IND-CPA-TLE}}$ for a number of queries q, function $\mathsf{f}_{\mathsf{state}}$, message domain $\mathbf{M}_\lambda$, algorithms $e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}}$ in the presence of an adversary $\mathcal{B}$, oracle $\mathcal{O}_{\mathsf{eval}}$ and a challenger Ch all parameterized by 1^λ.

Definition 4.3.1. A pair of TLE algorithms $(e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}})$ as described in Definition 4.2.1 is *IND-CPA-TLE*, if for every PPT adversary $\mathcal{B}$ the probability to win the experiment described in Figure 4.5 is $1/2 + negl(\lambda)$.

Mahmoody *et al.*'s construction is not IND-CPA-TLE: Recall the construction in Mahmoody et al. (2011) for encrypting a message m or a secret in general. It can be easily seen that it does not satisfy Definition 4.3.1, as the secret is spread across the puzzle, and thus part of it is leaked as the puzzle is solved. Specifically, the encryption $e_{\mathsf{MM0.1}}(m, \tau)$ for a message m and difficulty τ works as follows:

1. It uses an encoding function F_{e} to divide m into $\tau q + 1$ bit-blocks, $\mathsf{F}_{\mathsf{e}}(m, \tau, q) \rightarrow (m_0, \ldots, m_{\tau q})$.
2. Then it computes $c = (m_0, m_1 \oplus \mathcal{H}(m_0), \ldots, m_{\tau q} \oplus \mathcal{H}(m_{\tau q - 1})$ as the ciphertext of the plaintext m.

The decryption algorithm $d_{\mathsf{MM0.1}}(c, (m_0, \mathcal{H}(m_0), \ldots, \mathcal{H}(m_{\tau q-1}))$ for a ciphertext c and witness $(m_0, \mathcal{H}(m_0), \ldots, \mathcal{H}(m_{\tau q-1})$ which acts as the secret key, works as follows:

1. It computes the $\tau q + 1$ blocks of message m as $m_j = c_j \oplus \mathcal{H}_{j-1}$.
2. It computes the message m with the decoding function $\mathsf{F_d}((m_0, \ldots, m_{\tau q})) \rightarrow m$

It is worth mentioning that this algorithm as presented in Mahmoody et al. (2011) , it was not intended to be used as an encryption algorithm rather than a puzzle creation one. Observe that the message is spread all over the puzzle. As a result, the adversary $\mathcal{B}$ can easily win the IND-CPA-TLE game with probability 1. Specifically, he chooses the messages m_0 and m_1 such that the leading bit is different. Next he starts to solve the puzzle. As the message is revealed in a progressive way, when he finds either the bit 0 or 1 first he will know with probability 1 which of the two messages is without depleting all the available oracle queries and thus wins the game.

Next, we show both Astrolabous and an enhanced version of the construction in Mahmoody et al. (2011), we call it MMV 2.0 from the first letter of each author, are IND-CPA-TLE.

MMV 2.0**:** As we explained above, the construction in Mahmoody et al. (2011) does not satisfy IND-CPA-TLE security because it spreads the message all over the puzzle. A natural question is if it satisfies our game based definition when the message is not spread across all over the puzzle, but instead, it is XORed in the last hash evaluation. Specifically, $e_{\mathsf{MM0.1}}(m, \tau) \rightarrow (r_0, r_1 \oplus \mathcal{H}(r_0), \ldots, m \oplus \mathcal{H}(r_{\tau q-1}))$, where $r = r_0 || \ldots || r_{\tau q-1}$ is a random string. In that case, as we see next, the MMV 2.0 satisfies IND-CPA-TLE.

Theorem 3. *The construction MMV* 2.0 *as described above is IND-CPA-TLE secure.*

Proof. The reasoning of the proof is very similar with the one in Theorem 2. Specifically, we define a second experiment where the last *XOR* instead of containing the message it is just a random hash evaluation. Again, with exactly the same reasoning we argue that:

$$\Pr[\mathsf{Win}_{\mathbf{EXP}_{\mathsf{IND-CPA-TLE}}} \wedge \mathsf{Bad}] = \Pr[\mathsf{Win}_{\mathbf{EXP}^*_{\mathsf{IND-CPA-TLE}}} \wedge \mathsf{Bad}] \tag{4.13}$$

In $\mathbf{EXP}^*_{\mathsf{IND-CPA-TLE}}$ the challenged message it does not appear at all (in contrast with Astrolabous where it appears in the symmetric encryption scheme), so the probability to win there is $1/2$ exactly. This completes the proof. □

Next we show that Astrolabous is also IND-CPA-TLE secure. The reasoning again is exactly the same as the one in theorem 2 except that the IND-CPA

adversary sends the messages m_0, m_1 received from the IND-CPA-TLE adversary to the challenger instead of choosing his own. The rest are exactly the same and thus we omit the proof.

Theorem 4. *Astrolabous is IND-CPA-TLE secure.*

Even if both Astrolabous and MMV 2.0 are IND-CPA-TLE secure, Astrolabous has a potential advantage in terms of efficiency. Namely, Astrolabous hides the key of the symmetric cryptosystem that it uses into the puzzle, instead of the message itself as in MMV 2.0. As a result, many messages can be encrypted under the same key and be opened at the same time solving just one puzzle. In contrast, with MMV 2.0, for every message, a new puzzle must be generated, making the encryption more time-consuming. For example, for a puzzle with difficulty that should last 24-hours, an 8-core CPU can generate it in 3 hours (24/8). The total time for encrypting two messages with MMV 2.0 with the above difficulty is 3 hours for the first message and 2.625 hours (24-1.5/8) for the second, in total 5.625 hours. With Astrolabous one puzzle can be used for both messages, making the total encryption time just 3 hours. The gap becomes even bigger if we consider several encryptions instead of just two. In both examples with did not consider the time to perform AES, as in practice is very efficient.

Asymmetry of puzzle generation and puzzle solving time with Astrolabous: A natural question is if the puzzle generation time is significantly smaller than the time that is required for solving the puzzle. The answer is positive. Specifically, there are hash functions that are not meant to have an efficient evaluation, such as *Argon2* Biryukov et al. (2016). Equipped with such function we can create puzzles that are small (in terms of space) and fast, but at the same time difficult enough. For example, Argon2 can be parameterized in such a way that a single hash evaluation can take roughly 60 seconds Toponce (2016), meaning that an 8-core processor can generate a puzzle that meant to be solved in 4 hours (equably 14.400 seconds or $14.400/60 = 240$ hash evaluations) in just 30 minutes (puzzle generation is parallelizable so an 8-core processor can do 8 hash evaluation simultaneously which each one of them takes 60 seconds. So 240 hash evaluations can be done in 30 minutes.). As the number of CPU cores increases the puzzle generation can become even smaller but at the same time, the time for solving the puzzle remains unchanged (no parallelization for puzzle solving).

Chapter 5

E-cclesia: a self-tallying classical e-voting protocol

In general, an e-voting protocol consists of a set of parties, namely an election authority responsible for generating the protocol parameters, a set of voters generating and cast their ballot and a set of talliers responsible for producing the election result. The key properties that an e-voting protocol should satisfy as a minimum standard are: i) `Verifiability`: there is an auditing mechanism that verifies if the election has been tempered; ii) `Privacy`: the link between the votes and the voters remains unknown even after the end of the election procedure; iii) `Fairness`: partial results should not be leaked before all the voters have cast their ballot. One drawback of many e-voting constructions is the existence of talliers, which are responsible for the task of tallying. Ideally, we want to keep the trust and the number of dedicated participating parties as low as possible to withstand stronger threat models that capture the adversarial influence Canetti (2001b) over a protocol. To this end, special instances of e-voting protocols with not talliers involved proposed. Such protocols are called self-tallying protocols.

As explained in Chapter 3, the most prominent self-tallying proposals suffer from fairness issues. Specifically, the adversary can obtain some partial result before casting his ballot thus allowing him to change his vote based on that leakage. To tackle this issue, we need a cryptographic primitive that provides us semantic security of the encrypted ballot for the whole duration of the casting phase. In addition, we need to be sure that the ballots will eventually open at the tallying phase. These concerns can be addressed in a decentralized setting by a special type of encryption which is called time-lock encryption, as explained in the previous Chapter.

Our goal in this Chapter is to further investigate the self-tallying paradigm, in light of the recent developments in distributed systems that we have been witnessing with the development of Blockchain technologies. Indeed, Nakamoto's Bitcoin protocol Nakamoto (2008) for a decentralized banking system has paved a new way for a decentralized future. With this as our credo too, we develop E-CCLESIA, a new family of self-tallying election schemes that satisfy the stan-

dard `eligibility`, `fairness`, `vote-privacy`, and `one-voter-one-vote` requirements for electronic voting. We formally prove that E-CCLESIA satisfies these properties in the Universal Composability paradigm proposed by Canetti Canetti (2001b), which is a state-of-the-art framework for specifying and analyzing cryptographic protocols, especially when these are run under concurrent sessions. E-CCLESIA further satisfies *individual*, *universal verifiability* and *eligibility verifiability* as a direct consequence of its decentralized nature, its reliance on a broadcast channel, and *ballot authentication*, albeit our functionality does not capture these properties. Finally, we provide the first provably secure concrete instance of E-CCLESIA, which we call E-CCLESIA 1.0 . In particular, inspired by Zerocoin Miers et al. (2013), we rely on RSA dynamic accumulators for unlinking cast votes from the voters who cast them. For `fairness`, we rely on *Astrolabous*, the *time-lock encryption* (TLE) scheme that we developed based on ideas from Rivest et al. (1996) and Mahmoody et al. (2011) and presented in Subsection 4.3.

We formalize security of self-tallying election (STE) schemes and provide an abstract description of the E-CCLESIA family along the lines of modular UC design described in Subsection 2.4.2. Subsequently, we provide a specific instantiation, namely E-CCLESIA 1.0 , that UC-realises our $\mathcal{F}_{\mathsf{STE}}$ functionality by providing sub-protocols that in turn UC-realise the smaller module functionalities. Our approach to formally describing the E-CCLESIA family is to first capture `eligibility`, `fairness`, and `vote-privacy` via two main ideal modules, namely $\mathcal{F}_{\mathsf{elig}}$ and $\mathcal{F}_{\mathsf{vm}}$ as they are presented in Section 5.2.

In Figure 5.1 we provide the roadmap of our UC treatment of STE schemes, enhancing the readability of the Chapter before entering the actual technical development of our protocol. It can be inferred from the Figure that the E-CCLESIA 1.0 protocol UC realizes $\mathcal{F}_{\mathsf{STE}}$ in the $(\mathcal{F}_{\mathsf{SoK}}, \mathcal{F}_{\mathsf{NIC}}, \mathcal{F}_{\mathsf{an.BC}}, \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{Crs}}^{Gen}, \mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}), \{\mathcal{F}_{\mathsf{cert}}^{P}\})$-hybrid model. For the functionalities $(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}, \mathcal{W}_q, \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{Crs}}^{Gen})$ there is not a UC treatment in the literature as they capture modelling assumptions (e.g. the functionality wrapper captures the limited computational resourcesa party has at its disposal in each round). However, for the functionalities $(\mathcal{F}_{\mathsf{SoK}}, \mathcal{F}_{\mathsf{NIC}}, \{\mathcal{F}_{\mathsf{cert}}^{P}\})$ there are UC treatments where available in the literature Camenisch et al. (2016); Chase and Lysyanskaya (2006); Canetti (2001b) but not in the standard model Naccache (2011). Finally, for the functionality $\mathcal{F}_{\mathsf{an.BC}}$, currently, there is no UC realization in the literature. Nevertheless, there are many protocols for anonymous broadcast that have been deployed and used in applications Dingledine et al. (2004); Garay et al. (2014); Kobusiska et al. (01 Jun. 2016).

A brief description of E-cclesia 1.0 : In E-CCLESIA 1.0 the participating parties are the setup authority SA and the voters in the set $\mathbf{V} = \{V_1, \ldots, V_n\}$. Each voter V_j has hard-coded algorithms that are required in different phases of E-CCLESIA. Specifically, the required algorithms are `GenCred`, the credential generation algorithm, `AuthBallot`, the ballot authentication algorithm, `UpState`, the function that updates the current state (accumulated value) of voter's

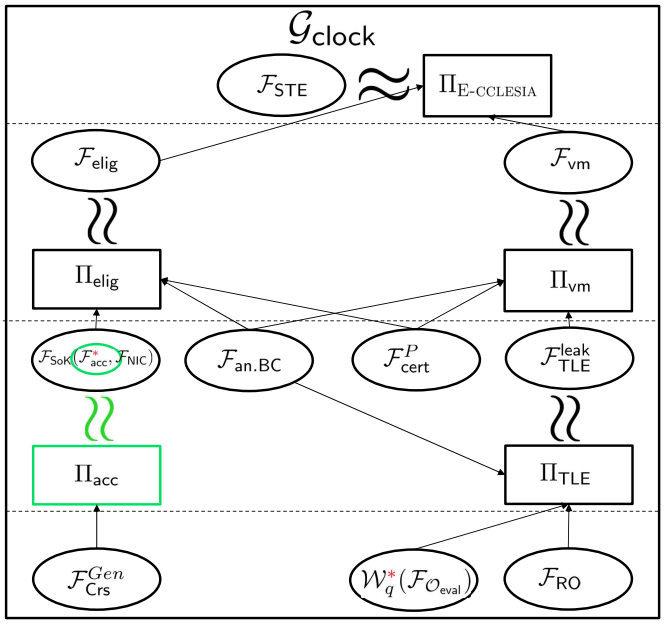

Figure 5.1: The UC roadmap of our UC treatment of our STE schemes. With "$\approx$" we indicate the UC realization of an ideal functionality from a protocol. With "$\leftarrow$" we indicate the inclusion of the ideal functionalities (start of the arrow) in the hybrid protocol (end of the arrow).

credential and the initial state St_{gen}, the `GenBallot`, the generation ballot algorithm, and the `OpenBallot`, the ballot open algorithm.

The E-CCLESIA 1.0 protocol consists of four disjoint consecutive phases, namely *Setup*, *Credential generation*, *Cast* and *Tally* phase.

In the *Setup* phase, the SA sets the duration of the election and the list of eligible voters. SA then sends this information to the voters via a broadcast channel. In the *Credential generation* phase, the voters generate the public and a private part of their credential with algorithm `GenCred`. Specifically, the private part is just a random number and the public part is the commitment over that number. Then, voters broadcast the public part of the credential to each other. After all, voters have broadcast their public credential, each voter can "compress" the list of credentials into a single value $\mathsf{St}_{\mathsf{fin}}$ by using the function `UpState` which is the update algorithm of a *dynamic accumulator* scheme. In the *Cast* phase, each voter generates their ballot by using the algorithm `GenBallot` which is a *time-lock encryption* scheme and authenticates it by using the algorithm `AuthBallot` which is a *signature of knowledge* over the statement that "I know the committed value of a commitment that is part of the accumulated value $\mathsf{St}_{\mathsf{fin}}$". Observe that only eligible voters can generate such signatures. Then, the voter broadcasts the ballot along with the signature to the other voters via an anonymous broadcast channel. Finally, in the *Tally* phase, the voters drop the non-valid received ballots (e.g. with a non-valid signature). This check is done by using the function `AuthBallot`. They then decrypt the ballots with the `OpenBallot` which is the time-lock decryption algorithm of our scheme and produce the election result. Such decryption is possible since the time-lock encryption scheme enables each voter to decrypt the message when the tally phase comes.

In order to avoid repetition and excessive formalization, this Chapter is organized as follows. In Section 5.1, we present the general *ideal STE functionality* $\mathcal{F}_{\mathsf{STE}}$ that captures `eligibility`, `fairness`, `vote-privacy`, `one-voter-one-vote`, properties. To provide intuition, we describe $\mathcal{F}_{\mathsf{STE}}$ in a complete yet not strictly formal manner. The functionalities $\mathcal{F}_{\mathsf{vm}}$ and $\mathcal{F}_{\mathsf{elig}}$ are defined formally in Subsections 5.2.2 and 5.2.1, respectively, with precise references to $\mathcal{F}_{\mathsf{STE}}$'s steps that are essentially handled by $\mathcal{F}_{\mathsf{vm}}$ and $\mathcal{F}_{\mathsf{elig}}$. Finally, in Subsection 5.3.1, we present the E-CCLESIA family as a protocol $\Pi_{\text{E-CCLESIA}}^{\mathcal{F}_{\mathsf{elig}},\mathcal{F}_{\mathsf{vm}},\mathcal{G}_{\mathsf{clock}}}$ that UC-realizes $\mathcal{F}_{\mathsf{STE}}$ in the $\{\mathcal{F}_{\mathsf{elig}}, \mathcal{F}_{\mathsf{vm}}, \mathcal{G}_{\mathsf{clock}}\}$-hybrid model in a straightforward manner. We note that in Sections 5.4 and 5.5, we provide two instantiations of $\mathcal{F}_{\mathsf{elig}}$ and $\mathcal{F}_{\mathsf{vm}}$, respectively, that specify the first version of E-CCLESIA, namely E-CCLESIA 1.0, as a provably secure STE scheme.

5.1 The STE functionality $\mathcal{F}_{\mathsf{STE}}^{\mathsf{delay}}$

In this Section, we describe the functionality $\mathcal{F}_{\mathsf{STE}}$ which captures our security requirements for STE elections (`eligibility`, `fairness`, `vote-privacy`,

one-voter-one-vote). The functionality $\mathcal{F}_{\mathsf{STE}}$ interacts with the setup authority SA, the voters in the set $\mathbf{V} = \{V_1, \ldots, V_n\}$ and the simulator $\mathcal{S}$. The functionality operates as described in the next Subsection and formally presented in Figure 5.2.

In addition, it forwards all (sid, ADVANCE_CLOCK) and (sid, READ_CLOCK) commands to $\mathcal{G}_{\mathsf{clock}}$ (see Figure 2.1) on behalf of the (dummy) voters, while it also reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ whenever necessary. In the spirit of Canetti (2001b), we allow $\mathcal{S}$ to provide $\mathcal{F}_{\mathsf{STE}}$ with the election algorithms, noting that this is *only for consistency* of the output format that $\mathcal{F}_{\mathsf{STE}}$ sends to the voters. As it will be clear in the description, the main security properties of eligibility, fairness, voter privacy and one-voter-one vote are preserved by $\mathcal{F}_{\mathsf{STE}}$, independently of the security of the algorithms provided by $\mathcal{S}$.

The functionality is parameterized by the set of voters $\mathbf{V}$, the function that defines the concrete time-points of the election `define_time`, a delay variable that shows the ballot's generation time and the predicate that shows the current phase of the election `Status`. The function `define_time` accepts as input the times $t_{\mathsf{cast}}, t_{\mathsf{open}}$ that define the duration of the election procedure, and outputs a vector $\boldsymbol{t}$. The vector $\boldsymbol{t}$ includes various time-points that are important for the distinction of the phases of the elections, where the latter is achieved with the function `Status`. For example, in protocol E-CCLESIA the vector $\boldsymbol{t} = (t_{\mathsf{cast}}, t_{\mathsf{open}})$ wherein other protocols more time points needs to be introduced. The reason of why not just two distinct times (in our case $t_{\mathsf{cast}}, t_{\mathsf{open}}$) were not enough to capture a broader spectrum of $\mathcal{F}_{\mathsf{TLE}}$ realizations is that we use time-lock encryption (see Chapter 4) in order to guarantee that the `fairness` condition is satisfied. Specifically, there are TLE constructions, such as in Liu et al. (2018), where the adversary might decrypt the ballots before the tally phase, learn the election outcome and change its vote (violation of `fairness`) and thus making the realization of $\mathcal{F}_{\mathsf{TLE}}$ impossible.

Fortunately, even these time-lock encryption schemes are useful in voting if we introduce a *time-wait* period, that's it, a period where no one can cast a ballot and ballots can not be opened either. In that period, the adversary can open the ballot but it can not change its vote because the casting period is over. This is why we defined the function `define_time`, `Status`. As we see next, this function and predicate respectively are instantiated with the leakage function in order for $\mathcal{F}_{\mathsf{TLE}}$ to be able to realise $\mathcal{F}_{\mathsf{vm}}$ in Section 5.5. Finally, in reality, the ballot generation time also takes some time. There are settings which this time is zero or very small (delay $= 0$), or it might take a unit of time (delay $= 1$). We capture these settings with the delay variable.

Setup: The functionality initializes as empty the lists of: eligible voters' credentials L_{elig}, generated ballots L_{gball}, cast ballots L_{cast}, algorithms L_{vm}, and ballots included in the tally set L_{tally}. It also initializes state St_{fin} as 0 and stores the set $\mathbf{V}_{\mathsf{corr}} \subseteq \mathbf{V}$ of corrupted voters provided by $\mathcal{S}$. Then, it operates as follows:

Upon receiving $(\mathsf{sid}, \text{ELECTION\_INFO}, \mathbf{V}_{\mathsf{elig}}, t_{\mathsf{cast}}, t_{\mathsf{open}})$ from SA, where i) $\mathbf{V}_{\mathsf{elig}}$

is the list of eligible voters and ii) $t_{\mathsf{cast}} < t_{\mathsf{open}}$ are moments that determine the beginning of the **Cast** and **Tally** phases (see below), it acts as follows:

Upon receiving $(\mathsf{sid}, \textsc{Eligible})$ from SA, it informs $\mathcal{S}$, which replies with the *eligibility algorithms* and an *initial credential state*. Then it provides $\langle V_i \rangle_{i\in[n]}$ and $\mathcal{S}$ with the *registration parameters* $\mathsf{reg.par}$.

Credential generation: This phase is active if $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cred}) = \top$. The predicate `Status` shows us in which phase of the protocol we are based on the current time Cl. It takes as input the current time Cl, the time vector $\boldsymbol{t}$ and the phase we want to check if it is active at the moment (all the phases are *Credential*, *Cast* and *Tally* and can be checked with the acronym $\mathsf{Cred}, \mathsf{Cast}$ and Tally respectively.). If this phase is active, `Status` outputs the special symbol $\top$, else it outputs $\bot$. The predicate is instantiated according to the protocol we realize. For example, this predicate might perform the check $t_{\mathsf{cast}} < \mathsf{Cl} < t_{\mathsf{open}}$, concluding if we are in the Cast phase or not. Moreover all phases are disjointed, e.g if $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cast}) = \top$ then $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cred}) = \bot$.

When $\mathcal{F}_{\mathsf{STE}}$ receives the $\textsc{Gen\_Cred}$ command message from $V \in \mathbf{V}_{\mathsf{elig}} \setminus \mathbf{V}_{\mathsf{corr}}$ for the first time and after permission from $\mathcal{S}$ via public delayed output, it runs $(\mathsf{cr}, \mathsf{rc}) \leftarrow \mathtt{GenCred}(1^\lambda, \mathsf{reg.par})$ and adds $(V, \mathsf{cr}, \mathsf{rc}, 1)$ to L_{elig}. Here, cr, rc play the role of the *private and public part* of the credential, respectively. If there are already tuples $(\cdot, \mathsf{cr}, \cdot, \cdot)$ or $(\cdot, \cdot, \mathsf{rc}, \cdot)$ in L_{elig} or $(\mathsf{cr}, \mathsf{rc}) = \bot$, it sends $(\mathsf{sid}, \textsc{Gen\_Cred}, \bot)$ to V and halts. Else, it adds $(V, \mathsf{cr}, \mathsf{rc}, 1)$ to L_{elig}. If $\mathcal{F}_{\mathsf{STE}}$ receives some credential pair $(\mathsf{sid}, \textsc{Gen\_Cred}, \mathsf{cr}, \mathsf{rc}, V)$ from $\mathcal{S}$ on behalf of a corrupted yet eligible voter V for the first time, if there are no tuples $(\cdot, \mathsf{cr}, \cdot, 1)$ or $(\cdot, \cdot, \mathsf{rc}, 1)$ in L_{elig}, then $\mathcal{F}_{\mathsf{STE}}$ adds $(V, \mathsf{cr}, \mathsf{rc}, 0)$ to L_{elig}. In any case, if a new record $(V, \mathsf{cr}, \mathsf{rc}, \cdot)$ is included in L_{elig} then it sends $(\mathsf{sid}, \textsc{Gen\_Cred}, V, \mathsf{rc})$ to $\langle V_j \rangle_{j\in[n]}$ and $\mathcal{S}$ after permission of $\mathcal{S}$ via public delayed output.

Cast: This phase is active if $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cast}) = \top$ and manages ballot generation, authentication and broadcasting. For ballot authentication, $\mathcal{F}_{\mathsf{STE}}$ once computes the *final credential state* St_{fin} by running `UpState` on input St_{gen} and the set of public credentials rc included in L_{elig}.

Upon receiving a message $\big(\mathsf{sid}, \textsc{Cast}, o\big)$ from an honest and eligible voter V for the first time such that $(V, \mathsf{cr}, \mathsf{rc}, 1) \in L_{\mathsf{elig}}$, it handles the request to $\mathcal{S}$ without revealing the message. This step captures the semantic security of the encryption algorithm. If the delay of generating the ciphertext is zero, it executes step ***Cast*** (as described below), else informs $\mathcal{Z}$ that the ballot is preparing to be cast by sending the message $(\mathsf{sid}, \textsc{Casting})$. When the voter advances the clock, the simulator can send the ballot of the preparing message. Observe that the simulator might not give the ballot when receives the token from $\mathcal{F}_{\mathsf{STE}}$ for the first time and might decide to give it when receives a clock advancement command from $\mathcal{G}_{\mathsf{clock}}$. For that reason, a tag is necessary to be given to $\mathcal{S}$ at first place, when $\mathcal{S}$ receives the token from $\mathcal{F}_{\mathsf{vm}}$. As a result, $\mathcal{S}$ can specify for which message

generated a ballot by presenting the tag. Upon receiving $(\mathsf{sid}, \textsc{Cast}?)$ from an honest and eligible voter, the functionality checks if the ballot has been received by $\mathcal{S}$ and that the delay has been passed (e.g that the ballot has been created and can be cast). Then it checks if it is already cast. If yes, it returns CAST. Else it executes step ***Cast***: (i) It generates the *authentication receipt* σ for the ballot v. (ii) It asks the permission of $\mathcal{S}$ via public delayed output. Finally, $\mathcal{F}_{\mathsf{STE}}$ broadcasts the ballot "anonymously", e.g. without revealing V's identity.

In addition, $\mathcal{F}_{\mathsf{STE}}$ allows $\mathcal{S}$ to cast ballots arbitrarily on behalf of a corrupted voter V: when it receives $(\mathsf{sid}, \textsc{Cast}, o, v, \sigma, V)$ from $\mathcal{S}$, if there is a tuple $(V, \mathsf{cr}, \mathsf{rc}, 0)$ in L_{elig}, it adds $(V, o, v, \mathsf{cr}, \sigma, 0)$ to L_{cast}, else it adds $(V, o, v, \perp, \sigma, 0)$ to L_{cast}. In any case, it sends $(\mathsf{sid}, \textsc{Cast}, v, \sigma)$ to $\langle V_j \rangle_{j\in[n]}$ and $\mathcal{S}$.

Tally: This phase is active if $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Tally}) = \top$ and manages the opening of all the valid ballots that should be included in the tally (thus, capturing `fairness`). When $\mathcal{F}_{\mathsf{STE}}$ receives the command message TALLY from a voter or the adversary, it performs the security check-steps (i),(ii) for all the accepted ballots using the verification algorithm provided by the adversary in previous steps. Step (i) checks if `eligibility` has been breached. Step (ii) checks if forgery of some honest ballot has occurred. Note that this check implies that the ballot generation algorithm should satisfy `unforgeability` Rabin (1978); Canetti (2003). As a result, at the end of these steps, the list of recorded ballots contains successfully verified ballots cast only by the eligible voters. Note that multiple valid ballots coming from corrupted eligible voters may still exist after the end of steps (i),(ii), something which is handled at the next step, which is important to ensure the one-voter-one-vote property. Thus, after the end of steps (i),(ii) the L_{tally} contains valid tuples in one-to-one correspondence with the eligible voters that participated in the election. $\mathcal{F}_{\mathsf{STE}}$ allows $\mathcal{S}$ to open the ballot of any corrupted voter to an alternative opening. Then, it checks the correctness of the honest voters' ballot generation, which implies the correctness of the ballot opening algorithm. Finally, it outputs the election result in the form of a multi-set of eligible voters' tallied options. The command message LEAKAGE models the fact that in the "wait" time period, the period where the protocol is between the **Cast** and **Tally** phase, the adversary might be able to open the ballots depending on the cryptographic primitives that it is used in a real-world protocol. On the other hand, this information at that point is not very useful (the adversary cannot break fairness) because the adversary cannot change its ballot, as the **Cast** phase is over. In protocols where no such waiting periods exist the condition $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cred}) = \mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cast}) = \mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Open}) = \perp$ is never satisfied.

The self-tallying election functionality $\mathcal{F}_{\mathsf{STE}}^{\mathsf{delay}}(\mathbf{V}, \mathtt{define\_time}, \mathtt{Status})$.

The functionality initializes as empty the lists of: eligible voters' credentials L_{elig}, generated ballots L_{gball}, cast ballots L_{cast}, algorithms L_{vm}, state St_{fin} as 0, and ballots included in the tally set L_{tally}.

■ Upon receiving $(\mathsf{sid}, \textsc{Corrupt}, \mathbf{V}_{\mathsf{corr}})$ from $\mathcal{S}$, if $\mathbf{V}_{\mathsf{corr}} \subseteq \mathbf{V}$, it fixes $\mathbf{V}_{\mathsf{corr}}$ as the set of corrupted voters.

■ Upon receiving $(\mathsf{sid}, \textsc{Election\_Info}, \mathbf{V}_{\mathsf{elig}}, t_{\mathsf{cast}}, t_{\mathsf{open}})$ from SA, it computes $\boldsymbol{t} \leftarrow \mathtt{define\_time}(t_{\mathsf{cast}}, t_{\mathsf{open}})$ and if $\boldsymbol{t} \neq \perp$, it sends $(\mathsf{sid}, \textsc{Setup\_Ok}, \mathbf{V}_{\mathsf{elig}}, t_{\mathsf{cast}}, t_{\mathsf{open}}, \boldsymbol{t})$ to SA after the permission of $\mathcal{S}$ via a public delayed output.

■ Upon receiving $(\mathsf{sid}, \textsc{Eligible})$ from SA, it forwards the message to $\mathcal{S}$. Upon receiving $(\mathsf{sid}, \textsc{Eligible}, \mathtt{GenCred}, \mathtt{AuthBallot}, \mathtt{VrfyBallot}, \mathtt{UpState}, St_{\mathsf{gen}})$, it sets $\mathsf{reg.par} := (\{\mathbf{V}_{\mathsf{elig}}, t_{\mathsf{cast}}, t_{\mathsf{open}}, \boldsymbol{t}\}, St_{\mathsf{gen}})$. Then it sends $(\mathsf{sid}, \textsc{Elig\_Par}, \mathsf{reg.par})$ to $\langle V_j \rangle_{j\in[n]}$ and $\mathcal{S}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Gen\_Cred})$ from $V \in \mathbf{V}_{\mathsf{elig}} \setminus \mathbf{V}_{\mathsf{corr}}$ for the first time and after permission from $\mathcal{S}$ via public delayed output, it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. If $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cred}) = \top$, it runs $(\mathsf{cr}, \mathsf{rc}) \leftarrow \mathtt{GenCred}(1^\lambda, \mathsf{reg.par})$ and adds $(V, \mathsf{cr}, \mathsf{rc}, 1) \rightarrow L_{\mathsf{elig}}$. If there are already tuples $(\cdot, \mathsf{cr}, \cdot, \cdot)$ or $(\cdot, \cdot, \mathsf{rc}, \cdot)$ in L_{elig} or $(\mathsf{cr}, \mathsf{rc}) = \perp$, it sends $(\mathsf{sid}, \textsc{Gen\_Cred}, \perp)$ to V. Else, it adds $(V, \mathsf{cr}, \mathsf{rc}, 1)$ to L_{elig}.

■ Upon receiving $(\mathsf{sid}, \textsc{Gen\_Cred}, \mathsf{cr}, \mathsf{rc}, V)$ from $\mathcal{S}$ for $V \in \mathbf{V}_{\mathsf{corr}} \cap \mathbf{V}_{\mathsf{elig}}$ for the first time, it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. If $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cred}) = \top$, it checks if there are no tuples $(\cdot, \mathsf{cr}, \cdot, 1)$ or $(\cdot, \cdot, \mathsf{rc}, 1)$ in L_{elig}. In these cases, it adds $(V, \mathsf{cr}, \mathsf{rc}, 0)$ to L_{elig} and sends $(\mathsf{sid}, \textsc{Gen\_Cred}, V, \mathsf{rc})$ to $\langle V_j \rangle_{j\in[n]}$ and $\mathcal{S}$ after permission of $\mathcal{S}$ via public delayed output.

■ Upon receiving $(\mathsf{sid}, \textsc{Cast}, o)$ from $V \in \mathbf{V}_{\mathsf{elig}} \setminus \mathbf{V}_{\mathsf{corr}}$ for the first time such that $(V, \mathsf{cr}, \mathsf{rc}, 1) \in L_{\mathsf{elig}}$, it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. If $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cast}) = \top$, it (a) picks $\mathsf{tag} \xleftarrow{\$} \mathsf{TAG}$ and it adds the tuple $\big(V, \mathsf{Null}, o, \mathsf{tag}, \mathsf{Cl}, 1\big)$ to L_{gball}, (b) sends $(\mathsf{sid}, \textsc{Gen\_Ballot}, \mathsf{tag}, \mathsf{Cl}, 0^{|o|})$ to $\mathcal{S}$. Upon receives the token back from from $\mathcal{S}$, if $\mathsf{delay} = 0$ it executes step *Cast* as described bellow, else it returns $(\mathsf{sid}, \textsc{Casting})$ to V. In any other case, it returns $(\mathsf{sid}, \textsc{Cast}, o, \perp)$ to V.

■ Upon receiving $(\mathsf{sid}, \textsc{Update}, \{(v_j, \mathsf{tag}_j)\}_{j=1}^{p(\lambda)})$ from $\mathcal{S}$, for all $v_j \neq \mathsf{Null}$ it updates each tuple $(V, \mathsf{Null}, o_j, \tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, 1)$ to $(V, v_j, o_j, \tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, 1)$

■ Upon receiving $(\mathsf{sid}, \textsc{Cast?})$ from $V \in \mathbf{V}_{\mathsf{elig}} \setminus \mathbf{V}_{\mathsf{corr}}$ such that $(V, \mathsf{cr}, \mathsf{rc}, 1) \in L_{\mathsf{elig}}$, it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. If $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cast}) = \top$, it executes the following steps:

1. It searches for a unique tuple $(V, v, o, \mathsf{tag}, \mathsf{Cl}', 1) \in L_{\mathsf{gball}}$ such that $\mathsf{Cl} - \mathsf{Cl}' \geq \mathsf{delay}$. If does not find one, it returns $(\mathsf{sid}, \textsc{No\_Cast})$ to V.
2. If $(V, o, v, \mathsf{cr}, \sigma, 1) \in L_{\mathsf{cast}}$, it returns $(\mathsf{sid}, \textsc{Cast})$ to V.
3. ***Cast:*** Else, It generates $\sigma \leftarrow \mathtt{AuthBallot}(v, \mathsf{cr}, \mathsf{rc}, St_{\mathsf{fin}})$ for ballot v. If $\mathtt{VrfyBallot}(v, \sigma, St_{\mathsf{fin}}, \mathsf{reg.par}) = 0$, it sends $(\mathsf{sid}, \textsc{Auth\_Ballot}, \perp)$ to V. It sends $(\mathsf{sid}, \textsc{Cast}, v, \sigma)$ to $\mathcal{S}$. Upon receiving $(\mathsf{sid}, \textsc{Cast\_Allowed})$ from $\mathcal{S}$, it adds $(V, o, v, \mathsf{cr}, \sigma, 1)$ to L_{cast} as the authenticated ballot tuple for V. It sends $(\mathsf{sid}, \textsc{Cast}, v, \sigma)$ to $\langle V_j \rangle_{j\in[n]}$ and $\mathcal{S}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Cast}, o, v, \sigma, V)$ from $\mathcal{S}$, it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. If $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cast}) = \top$ it does:

1. If there is a tuple $(V, \mathsf{cr}, \mathsf{rc}, 0) \in L_{\mathsf{elig}}$, it adds $(V, o, v, \mathsf{cr}, \sigma, 0)$ to L_{cast}.
2. Else it adds $(V, o, v, \perp, \sigma, 0)$ to L_{cast}.
3. It sends $(\mathsf{sid}, \textsc{Cast}, v, \sigma)$ to $\langle V_j \rangle_{j \in [n]}$ and $\mathcal{S}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Tally})$ from a voter V or $\mathcal{S}$, $\mathcal{F}_{\mathsf{STE}}$, if $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Tally}) = \top$ it executes the following steps:

1. For all $(V, o, v, \mathsf{cr}, \sigma, b) \in L_{\mathsf{cast}}$, it runs $x \leftarrow \mathtt{VrfyBallot}(v, \sigma, St_{\mathsf{fin}})$. If $x = 1$, then it does:
   (i). If there is no tuple $(V, \mathsf{cr}, \mathsf{rc}, \cdot)$ in L_{elig}, it returns $\big(\mathsf{sid}, \textsc{Tally}, \perp\big)$ to V or $\mathcal{S}$.
   (ii). If there is a tuple $(V, \mathsf{cr}, \mathsf{rc}, 1)$ in L_{elig} and there is a tuple $(\cdot, o', v', \mathsf{cr}', \sigma', 1)$ in L_{cast} such that $(\mathsf{cr}' = \mathsf{cr}) \wedge (v' \neq v)$, it returns $(\mathsf{sid}, \textsc{Tally}, \perp)$ to V or $\mathcal{S}$.
   (iii). Otherwise, it adds (V, o, v, b) to L_{tally}.
2. For every tuple $(V, \mathsf{cr}, \mathsf{rc}, \cdot)$ in L_{elig} such that there are multiple tuples $(V, o^1, v^1, \cdot), \ldots, (V, o^n, v^n, \cdot)$ in L_{tally}, it removes all multiple tuples from L_{tally} except the first one it recorded.
3. For every tuple $(V, o, v, \cdot)$ in L_{tally} it does:
   (a) If the tuple is of the form $(V, o, v, 0)$ it forwards the message $(\mathsf{sid}, \textsc{Tally}, (V, o, v, 0))$ to $\mathcal{S}$. Upon receiving $(\mathsf{sid}, \textsc{Tally}, (V, o, v, 0), \tilde{o})$, it updates the tuple $(V, o, v, 0)$ to $(V, \tilde{o}, v, 0)$.
   (b) If there are two tuples $(V, o, v, 1)$, $(V', o', v', 1)$ in L_{tally} such that $(v' = v) \wedge (o \neq o')$, it returns $(\mathsf{sid}, \textsc{Tally}, \perp)$ to V. Else it returns $\big(\mathsf{sid}, \textsc{Tally}, \{(o, v) | (V, o, v, \cdot) \in L_{\mathsf{tally}}\}\big)$ to V or $\mathcal{S}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Leakage})$ from $\mathcal{S}$, it reads the time Cl from $\mathcal{G}_{\mathsf{Clock}}$. If $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cred}) = \mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cast}) = \mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Open}) = \perp$, it returns to $\mathcal{S}$ all the tuples $(v, o, 1)$ such that $(V, v, o, 1) \in L_{\mathsf{gball}} \wedge (V, 0, v, \mathsf{cr}, \sigma, 1) \in L_{\mathsf{cast}}$.

Figure 5.2: The self-tallying election functionality $\mathcal{F}_{\mathsf{STE}}^{\mathsf{delay}}$ interacting with voters $\mathbf{V}$, SA, and the simulator $\mathcal{S}$.

5.2 Decomposing $\mathcal{F}_{\mathsf{STE}}^{\mathsf{delay}}$ into $\mathcal{F}_{\mathsf{elig}}$ and $\mathcal{F}_{\mathsf{vm}}^{\mathsf{delay}}$

In this section we decompose $\mathcal{F}_{\mathsf{STE}}$ into two modules, namely $\mathcal{F}_{\mathsf{elig}}$ and $\mathcal{F}_{\mathsf{vm}}$. Our modular approach makes the development of the E-CCLESIA protocol presented

in Section 5.3 easily accessible for future developments as the substitution of one cryptographic primitive related to one of the two functionalities modules does not lead to reproving the whole security of our protocol. Instead, only the functionality module that is related to that cryptographic primitive needs a new proof of security.

Morally we decompose $\mathcal{F}_{\mathsf{STE}}$ as follows: i) a *vote management functionality* $\mathcal{F}_{\mathsf{vm}}$ that handles ballot encryption, anonymous broadcast and opening. We stress that ballot encryption and opening run by $\mathcal{F}_{\mathsf{vm}}$ ensure fairness; ii) an *eligibility functionality* $\mathcal{F}_{\mathsf{elig}}$ that is responsible for generating anonymous credentials and authenticating the ballots of the eligible voters. The functionality can also link two ballots originating from the same (corrupted) voter. The credential generation and ballot authentication run by $\mathcal{F}_{\mathsf{elig}}$ guarantee eligibility. Note that both functionalities combined safeguard voter privacy by implicitly featuring anonymous ballot authentication and casting. In addition, one-voter-one-vote is achieved through $\mathcal{F}_{\mathsf{elig}}$ for discarding multiple ballots that are linked to the same (corrupted) voter. Given the aforementioned approach, the major technical challenge in designing E-CCLESIA instantiations is to devise real-world protocols that UC-realize $\mathcal{F}_{\mathsf{vm}}$ and $\mathcal{F}_{\mathsf{elig}}$. Upon completion of this task, the step towards full STE security is merely a careful composition in terms of the interface by specifying how the STE entities interact with $\mathcal{F}_{\mathsf{vm}}$ and $\mathcal{F}_{\mathsf{elig}}$ and check the information they obtain from their engagement in the overall execution so that the four security properties are preserved.

In the following Subsections we describe the functionalities $\mathcal{F}_{\mathsf{elig}}$ and $\mathcal{F}_{\mathsf{vm}}$.

5.2.1 Eligibility functionality $\mathcal{F}_{\mathsf{elig}}$

The eligibility functionality $\mathcal{F}_{\mathsf{elig}}$ takes over the following parts of the $\mathcal{F}_{\mathsf{STE}}$ execution: i) in **Setup** the eligibility algorithms and the initial credential state; ii) the entire (private and public) **Credential generation**; iii) in **Cast** the creation of ballot authentication receipts, (iv) in **Tally** the ballot verification step, so that unforgeability is preserved (see Step 1. in $\mathcal{F}_{\mathsf{STE}}$'s description for $(\mathsf{sid}, \textsc{Tally})$ messages), and v) the linkability of two ballots to the same voter, so that one-voter-one-vote is preserved. The functionality is presented in Figure 5.3.

$\mathcal{F}_{\mathsf{elig}}$ ensures that each voter receives at most one credential and that the signatures that are issued from $\mathcal{F}_{\mathsf{elig}}$ on behalf of a voter always pass the verification procedure.

The functionality $\mathcal{F}_{\mathsf{elig}}$ is instantiated with the functions `define_time`, `Status` similar to $\mathcal{F}_{\mathsf{STE}}$.

In the beginning, $\mathcal{F}_{\mathsf{elig}}$ initializes as empty the list of eligible voters L_{elig}, that stores the voters' identities and the public and private part of their credential, the list of authenticated ballots L_{auth} that stores the voters' identities with the ballot and their private part of their credential given that the voter is honest and

eligible. Moreover, it initializes the *state* value St_{fin} as 0, that is a state that can describe all the private parts of the credential that $\mathcal{S}$ allowed to be broadcast.

At the beginning, $\mathcal{S}$ provides the corruption vector to $\mathcal{F}_{\mathsf{elig}}$. Next, $\mathcal{F}_{\mathsf{elig}}$ receives the eligibility list $\mathsf{V}_{\mathsf{elig}}$ accompanied with the times $t_{\mathsf{cast}}, t_{\mathsf{open}}$ that define the duration of the election procedure. Then, $\mathcal{F}_{\mathsf{elig}}$ requests from $\mathcal{S}$ the credential generation algorithm, `GenCred`, the authentication ballot algorithm, `AuthBallot`, the verification ballot algorithm, `VrfyBallot`, the state update algorithm, `UpState`, and the initial state, St_{gen}. Then, it computes the vector $\boldsymbol{t}$ with the function `define_time`, sets as the registration parameters $\mathsf{reg.par} := (\mathbf{V}_{\mathsf{elig}}, t_{\mathsf{cast}}, t_{\mathsf{open}}, \boldsymbol{t}, St_{\mathsf{gen}})$ and sends them to all parties. When receiving a credential generation request from an honest and eligible voter it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. If the status of the protocol is in *credential generation* phase given by function `Status`), then it checks if that voter made such request for the first time. If yes, then $\mathcal{F}_{\mathsf{elig}}$ generates both the public and the private part of the credential for that voter after permission of $\mathcal{S}$ via public delayed output. Then, $\mathcal{F}_{\mathsf{elig}}$ sends the public part of the credential to the other parties again after the permission of $\mathcal{S}$. In applications where such distinction of the credential, e.g. public and private part, does not exist the missing part is the Null value. In case a corrupted voter requests a credential generation then $\mathcal{F}_{\mathsf{elig}}$ forwards the message to $\mathcal{S}$. Then, after receiving back the credential values from $\mathcal{S}$ on behalf of that voter it records them with a distinct flag 0 and sends again the public part of the credential to the other voters only if it was the first time that corrupted voter made such request. Note that the credential returned by $\mathcal{S}$ does not comply necessarily with the algorithms that $\mathcal{S}$ provided $\mathcal{F}_{\mathsf{elig}}$ with at the beginning of the protocol; capturing the fact that $\mathcal{S}$ can deviate from the description of the credential generation algorithm. Finally, when an honest and eligible voter requests to authenticate a ballot, $\mathcal{F}_{\mathsf{elig}}$ checks if the status of the protocol is *cast* by reading the current time Cl and using the function `Status`. If yes, then it computes the final state St_{fin} with the function `UpState` and all the public parts of the credentials. This value in a sense represents all the public parts of the credential. In protocols where such a feature does not exist then this state is equal to the Null value. Then, $\mathcal{F}_{\mathsf{elig}}$ checks if there is a signature for that ballot. If not, it signs the ballot with the algorithm `AuthBallot` and checks if the signature is valid by running the algorithm `VrfyBallot`. If this is the case then it adds the voter's identity with the ballot, the private part of the voter's credential and the signature to the list of the authenticated ballots L_{auth}. This captures the fact that the algorithms provided by $\mathcal{S}$ are consistent, meaning that signatures generated with algorithm `AuthBallot` must pass the verification test with the algorithm `VrfyBallot`, else $\mathcal{F}_{\mathsf{elig}}$ returns $\perp$ to that voter.

The eligibility functionality $\mathcal{F}_{\mathsf{elig}}(\mathbf{V}, \mathtt{define\_time}, \mathtt{Status})$.

The functionality initializes the lists of eligible voters $L_{\mathsf{elig}} \leftarrow \emptyset$, of authenticated ballots of eligible voters $L_{\mathsf{auth}} \leftarrow \emptyset$, the value $St_{\mathsf{fin}} = 0$, and its status to 'init'.

■ Upon receiving $(\mathsf{sid}, \textsc{Corrupt}, \mathbf{V}_{\mathsf{corr}})$ from $\mathcal{S}$, if $\mathbf{V}_{\mathsf{corr}} \subseteq \mathbf{V}$ and status=init, it fixes $\mathbf{V}_{\mathsf{corr}}$ as the set of corrupted voters.

■ Upon receiving $(\mathsf{sid}, \textsc{Eligible}, \mathbf{V}_{\mathsf{elig}}, t_{\mathsf{cast}}, t_{\mathsf{open}})$ from SA, if $\mathbf{V}_{\mathsf{elig}} \subseteq \mathbf{V}$ and status=init, it sends $(\mathsf{sid}, \textsc{Setup\_Elig})$ to $\mathcal{S}$. Upon receiving $(\mathsf{sid}, \textsc{Setup\_Elig}, \mathtt{GenCred}, \mathtt{AuthBallot}, \mathtt{VrfyBallot}, \mathtt{UpState}, St_{\mathsf{gen}})$ from $\mathcal{S}$, if status=init, then:

1. It computes $\boldsymbol{t} \leftarrow \mathtt{define\_time}(t_{\mathsf{cast}}, t_{\mathsf{open}})$. If $\boldsymbol{t} \neq \perp$, it sets $\mathsf{reg.par} := (\mathbf{V}_{\mathsf{elig}}, t_{\mathsf{cast}}, t_{\mathsf{open}}, \boldsymbol{t}, St_{\mathsf{gen}})$ as registration parameters.
2. It sets its status to 'credential' and sends $(\mathsf{sid}, \textsc{Elig\_Par}, \mathsf{reg.par})$ to $\langle V_j \rangle_{j \in [n]}$ and $\mathcal{S}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Gen\_Cred})$ from $V \in \mathbf{V}_{\mathsf{elig}} \setminus \mathbf{V}_{\mathsf{corr}}$, if status=credential, then it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. If $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cast}) = \top$, it sets its status to 'cast'. Otherwise, if $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cred}) = \top$, it executes the following steps:

1. If there is no tuple $(V, \mathsf{cr}', \mathsf{rc}', 1)$ in L_{elig}, it runs $(\mathsf{cr}, \mathsf{rc}) \leftarrow \mathtt{GenCred}(1^{\lambda}, \mathsf{reg.par})$. If there are tuples $(\cdot, \mathsf{cr}, \cdot, \cdot)$ or $(\cdot, \cdot, \mathsf{rc}, \cdot)$ in L_{elig} or $(\mathsf{cr}, \mathsf{rc}) = \perp$, it sends $(\mathsf{sid}, \textsc{Gen\_Cred}, \perp)$ to V and halts. Else, it adds $(V, \mathsf{cr}, \mathsf{rc}, 1)$ to L_{elig} after permission of $\mathcal{S}$ via public delayed output.
2. It sends $(\mathsf{sid}, \textsc{Gen\_Cred}, V, \mathsf{rc})$ to $\langle V_j \rangle_{j \in [n]}$ and $\mathcal{S}$ after permission of $\mathcal{S}$ via public delayed output.

■ Upon receiving $(\mathsf{sid}, \textsc{Gen\_Cred})$ from $V \in \mathbf{V}_{\mathsf{corr}}$, it forwards the message $(\mathsf{sid}, \textsc{Gen\_Cred}, V)$ to $\mathcal{S}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Gen\_Cred}, V, \mathsf{cr}, \mathsf{rc})$ from $\mathcal{S}$, if $V \in \mathbf{V}_{\mathsf{corr}}$, then:

1. If $\mathbf{V}_{\mathsf{elig}}$ and there are no tuples $(V, \mathsf{cr}', \mathsf{rc}', 0)$, $(\cdot, \mathsf{cr}, \cdot, 1)$ or $(\cdot, \cdot, \mathsf{rc}, 1)$ in L_{elig}, then it adds $(V, \mathsf{cr}, \mathsf{rc}, 0)$ to L_{elig}.
2. It sends $(\mathsf{sid}, \textsc{Gen\_Cred}, V, \mathsf{rc})$ to $\langle V_j \rangle_{j \in [n]}$ and $\mathcal{S}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Auth\_Ballot}, v)$ from $V \in \mathbf{V}_{\mathsf{elig}} \setminus \mathbf{V}_{\mathsf{corr}}$, if status=cast, then it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. If $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Open}) = \top$, it sets its status to 'open'. Otherwise, if $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cast}) = \top$, it executes the following steps:

1. If $St_{\mathsf{fin}} = 0$, then it runs $St_{\mathsf{fin}} \leftarrow \mathtt{UpState}(St_{\mathsf{gen}}, \{\mathsf{rc} | (\cdot, \cdot, \mathsf{rc}, \cdot) \in L_{\mathsf{elig}}\})$.
2. If there is a tuple $(V, \mathsf{cr}, \mathsf{rc}, 1) \in L_{\mathsf{elig}}$ but no $(V, v', \mathsf{cr}, \sigma', 1) \in L_{\mathsf{auth}}$, then it runs $\sigma \leftarrow \mathtt{AuthBallot}(v, \mathsf{cr}, \mathsf{rc}, St_{\mathsf{fin}}, \mathsf{reg.par})$. If $\mathtt{VrfyBallot}(v, \sigma, St_{\mathsf{fin}}, \mathsf{reg.par}) = 0$, it sends $(\mathsf{sid}, \textsc{Auth\_Ballot}, \perp)$ to V and halts. Else, it (a) adds $(V, v, \mathsf{cr}, \sigma, 1)$ to L_{auth}, and (b) returns $(\mathsf{sid}, \textsc{Auth\_Ballot}, v, \sigma)$ to V.

■ Upon receiving $(\mathsf{sid}, \textsc{Auth\_Ballot})$ from $V \in \mathbf{V}_{\mathsf{corr}}$, it forwards the message $(\mathsf{sid}, \textsc{Auth\_Ballot}, V)$ to $\mathcal{S}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Auth\_Ballot}, V, v, \sigma)$ from $\mathcal{S}$, if there is a tuple $(V, \mathsf{cr}, \mathsf{rc}, 0) \in L_{\mathsf{elig}}$, then it adds $(V, v, \mathsf{cr}, \sigma, 0)$ to L_{auth}. It returns $(\mathsf{sid}, \textsc{Auth\_Ballot}, V, v, \sigma)$ to V.

■ Upon receiving $(\mathsf{sid}, \textsc{Ver\_Ballot}, v, \sigma)$ from $V \in \mathbf{V}$:

1. It computes $x \leftarrow \mathtt{VrfyBallot}(v, \sigma, St_{\mathsf{fin}}, \mathsf{reg.par})$.
2. If $x = 1$ and there is no cr such that there are tuples $(\cdot, \mathsf{cr}, \cdot, \cdot) \in L_{\mathsf{elig}}$ and $(\cdot, v, \mathsf{cr}, \sigma, \cdot) \in L_{\mathsf{auth}}$, it sends $(\mathsf{sid}, \textsc{Ver\_Ballot}, v, \sigma, \perp)$ to V and halts.
3. If $x = 1$ and there are tuples $(\cdot, v, \mathsf{cr}, \sigma, \cdot)$, $(\cdot, v', \mathsf{cr}', \sigma', 1) \in L_{\mathsf{auth}}$ such that $\mathsf{cr} = \mathsf{cr}'$ and $v \neq v'$, it sends $(\mathsf{sid}, \textsc{Ver\_Ballot}, v, \sigma, \perp)$ to V and halts.
4. Else, it sends $(\mathsf{sid}, \textsc{Ver\_Ballot}, v, \sigma, x)$ to V.

■ Upon receiving $\big(\mathsf{sid}, \textsc{Link\_Ballots}, (v, \sigma), (v', \sigma')\big)$ from $V \in \mathbf{V}$, if there are tuples $(\cdot, v, \mathsf{cr}, \sigma, \cdot)$, $(\cdot, v', \mathsf{cr}', \sigma', \cdot) \in L_{\mathsf{auth}}$ such that $\mathsf{cr} = \mathsf{cr}'$, then it sets $x = 1$, else $x = 0$. Then, it sends $\big(\mathsf{sid}, \textsc{Link\_Ballots}, (v, \sigma), (v', \sigma'), x\big)$ to V.

Figure 5.3: The eligibility functionality $\mathcal{F}_{\mathsf{elig}}(\mathbf{V})$ interacting with voters $\mathbf{V}$, SA, and the simulator $\mathcal{S}$.

5.2.2 Vote management funtionality $\mathcal{F}_{\mathsf{vm}}^{\mathsf{delay}}$

The vote management functionality $\mathcal{F}_{\mathsf{vm}}$ takes over the following parts of the $\mathcal{F}_{\mathsf{STE}}$ execution: i) in **Setup** the configuration of the vote management algorithms; ii) in **Cast** the secure ballot generation and casting and iv) in **Tally** the ballot opening step, so that fairness and ballot correctness is preserved (see Step 3. in $\mathcal{F}_{\mathsf{STE}}$'s description for $(\mathsf{sid}, \textsc{Tally})$ messages). The functionality is presented in Figure 5.4.

This functionality captures the privacy of each voter (identities and votes are not leaked), the ballot casting (we allow an adversary to drop ballots), and the fairness of the outcome (votes are not revealed unless the status is 'open'). The functionality is parameterized by the variable delay which shows how much time a ballot needs to be generated, similar as it appears in $\mathcal{F}_{\mathsf{TLE}}$. In the beginning, the functionality initializes as empty the lists L_{gball}, the list of generated ballots from honest parties through $\mathcal{F}_{\mathsf{vm}}$, L_{cast}, the list of the cast ballots from honest parties, L_{vm}, the list of ballot generation and open algorithms received by $\mathcal{S}$. The simulator provides the corruption vector and the setup authority provides the eligibility list. Moreover, $\mathcal{F}_{\mathsf{vm}}$ requests from $\mathcal{S}$ to generate ballots for honest voters only one time. Similarly to $\mathcal{F}_{\mathsf{STE}}$, the simulator returns the ballot either when the honest voter sends a ballot generation command to $\mathcal{F}_{\mathsf{vm}}$ or at some point after that voter advances the clock. Again, the ballot becomes available to

the voter only after delay number of rounds has been passed by the time of the creation of the ballot. Similar to $\mathcal{F}_{\mathsf{STE}}$, the simulator along with the request for ballot generation, receives an identification tag.

Next, $\mathcal{F}_{\mathsf{vm}}$ anonymously broadcasts a ballot with a signature of honest voters to other parties after asking the simulator if it allows such an action. Similarly, generation ballot requests from corrupted voters are passed to $\mathcal{S}$, and $\mathcal{F}_{\mathsf{vm}}$ returns to that voter whatever received from $\mathcal{S}$. Again, the provided ballot from $\mathcal{S}$ does not have to be produced by using the algorithm provided by $\mathcal{S}$ in the previous steps, capturing the fact that the adversary might use arbitrary code to generate a ballot. Next, when receiving a cast request on behalf of an uncorrupted voter, it checks if the status is not *open* by reading the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and using the function `Status`. Then, if this is the first time that this voter made a cast request, $\mathcal{F}_{\mathsf{vm}}$ broadcasts the received values to all parties; upon permission of $\mathcal{S}$. Here we capture the fact that $\mathcal{S}$ controls the network and can decide to either block or allow a message. It is worth noting that the identity of the voter is kept hidden from $\mathcal{S}$, meaning that we assume a level of anonymity. Finally, when $\mathcal{F}_{\mathsf{vm}}$ receives Open from voters and the status is 'open', it opens the ballot or returns $\perp$ in the case that the provided algorithm by the adversary does not satisfy correction, e.g. the same ballot for two different votes.

The vote management functionality $\mathcal{F}_{\mathsf{vm}}^{\mathsf{delay}}(\mathbf{V}, \mathtt{define\_time}, \mathtt{Status})$.

The functionality initializes the lists of generated ballots L_{gball}, cast ballots L_{cast} as empty. Then it sets its status to 'exec'.

■ Upon receiving $(\mathsf{sid}, \textsc{Corrupt}, \mathbf{V}_{\mathsf{corr}})$ from $\mathcal{S}$, if $\mathbf{V}_{\mathsf{corr}} \subseteq \mathbf{V}$ and status=exec, it fixes $\mathbf{V}_{\mathsf{corr}}$ as the set of corrupted voters.

■ Upon receiving $(\mathsf{sid}, \textsc{Setup\_Info}, \mathbf{V}_{\mathsf{elig}}, t_{\mathsf{cast}}, t_{\mathsf{open}})$ from SA it computes $\boldsymbol{t} \leftarrow \mathtt{define\_time}(t_{\mathsf{cast}}, t_{\mathsf{open}})$. If status=exec and $\boldsymbol{t} \neq \perp$, it sets $\mathsf{vote.par} := (\mathbf{V}_{\mathsf{elig}}, t_{\mathsf{cast}}, t_{\mathsf{open}}, \boldsymbol{t})$ as voting parameters and sends $(\mathsf{sid}, \textsc{Setup\_OK}, \mathsf{vote.par})$ to SA, after permission from $\mathcal{S}$ via public delayed output.

■ Upon receiving $(\mathsf{sid}, \textsc{Gen\_Ballot}, o)$ from $V \notin \mathbf{V}_{\mathsf{corr}}$, it reads the time Cl and does:

1. If there is no $\big(V, v', o', \mathsf{tag}', \mathsf{Cl}', 1\big) \notin L_{\mathsf{gball}}$, it (a) picks $\mathsf{tag} \xleftarrow{\$} \mathsf{TAG}$ and it inserts the tuple $\big(V, \mathsf{Null}, o, \mathsf{tag}, \mathsf{Cl}, 1\big) \rightarrow L_{\mathsf{gball}}$, (b) sends $(\mathsf{sid}, \textsc{Gen\_Ballot}, \mathsf{tag}, \mathsf{Cl}, 0^{|o|})$ to $\mathcal{S}$. Upon receives the token back from $\mathcal{S}$ it returns $(\mathsf{sid}, \textsc{Generating})$ to V.

2. Else it returns $(\mathsf{sid}, \textsc{Gen\_Ballot}, o, \perp)$ to V.

■ Upon receiving $(\mathsf{sid}, \textsc{Gen\_Ballot})$ from $V \in \mathbf{V}_{\mathsf{corr}}$, it sends the message $(\mathsf{sid}, \textsc{Gen\_Ballot}, V)$ to $\mathcal{S}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Gen\_Ballot}, o, v, V)$ from $\mathcal{S}$, it sends $(\mathsf{sid}, \textsc{Gen\_Ballot}, o, v)$ to V.

■ Upon receiving $(\mathsf{sid}, \textsc{Update}, \{(v_j, \mathsf{tag}_j)\}_{j=1}^{p(\lambda)})$ from $\mathcal{S}$, for all $v_j \neq \mathsf{Null}$ it updates each tuple $(V, \mathsf{Null}, o_j, \tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, 1)$ to $(V, v_j, o_j, \tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, 1)$

■ Upon receiving $(\mathsf{sid}, \textsc{Retrieve})$ from $V \notin \mathbf{V}_{\mathsf{corr}}$ it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and it searches for a tuple $\big(V, v, o, \mathsf{tag}, \mathsf{Cl}', 1\big) \in L_{\mathsf{gball}}$ with $v \neq \mathsf{Null}$ and $\mathsf{Cl} - \mathsf{Cl}' \geq \mathsf{delay}$. It returns $(\mathsf{sid}, \textsc{Retrieve}, (o, v))$ to V.

■ Upon receiving $\big(\mathsf{sid}, \textsc{Cast}, v, \sigma\big)$ from V, if $V \notin \mathbf{V}_{\mathsf{corr}}$ and status=exec, it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. If $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Open}) = \top$, it sets the status to 'open', otherwise:

1. If $(V, v, o', \mathsf{tag}, \mathsf{Cl}', 1) \notin L_{\mathsf{gball}}$ or $\mathsf{Cl} - \mathsf{Cl}' < \mathsf{delay}$, it returns $(\mathsf{sid}, \textsc{Cast}, v, \sigma, \perp)$ to V.

2. If there is no $(V, v', \sigma', 1) \notin L_{\mathsf{cast}}$, it sends $(\mathsf{sid}, \textsc{Allow\_Cast}, v, \sigma)$ to $\mathcal{S}$. Upon receiving $(\mathsf{sid}, \textsc{Cast\_Allowed})$ from $\mathcal{S}$, it adds $(V, v, \sigma, 1)$ to L_{cast} and sends $(\mathsf{sid}, \textsc{Cast}, v, \sigma)$ to $\langle V_j \rangle_{j \in [n]}$ and $\mathcal{S}$.

3. If there is a tuple $(V, v', \sigma', 1)$ in L_{cast}, it returns $(\mathsf{sid}, \textsc{Cast}, v, \sigma, \perp)$ to V.

■ Upon receiving $\big(\mathsf{sid}, \textsc{Cast}\big)$ from $V \in \mathbf{V}_{\mathsf{corr}}$, it sends $\big(\mathsf{sid}, \textsc{Cast}, V\big)$ to $\mathcal{S}$.

■ Upon receiving $\big(\mathsf{sid}, \textsc{Cast}, v, \sigma, V\big)$ from $\mathcal{S}$, if $V \in \mathbf{V}_{\mathsf{corr}}$ and status=exec, it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. If $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Open}) = \top$ it sets status to 'open', otherwise it adds $(V, v, \sigma, 0)$ to L_{cast} and sends $(\mathsf{sid}, \textsc{Cast}, v, \sigma)$ to $\langle V_j \rangle_{j \in [n]}$ and $\mathcal{S}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Open}, v)$ from any party $P \in \mathbf{V} \cup \{\mathcal{S}\}$, it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. If $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Open}) = \top$, it sets the status to 'open' and executes the following steps:

1. If there is a tuple $(V, v, \sigma, \cdot) \in L_{\mathsf{cast}}$, and a *unique* $(V, v, o, 1) \in L_{\mathsf{gball}}$, it sends $(\mathsf{sid}, \textsc{Open}, v, o)$ to P.
2. Else, if there is a tuple $(V, v, \sigma, \cdot) \in L_{\mathsf{cast}}$ and at least two tuples $(V, v, o, 1), (V', v', o', 1) \in L_{\mathsf{gball}}$ such that $(v = v') \wedge (o \neq o')$, it sends $(\mathsf{sid}, \textsc{Open}, v, \perp)$ to P.
3. Else, if there is a tuple $(V, v, \sigma, \cdot) \in L_{\mathsf{cast}}$ but there is no tuple $(V, v, o, 1) \in L_{\mathsf{gball}}$, it sends $(\mathsf{sid}, \textsc{Open}, v)$ to $\mathcal{S}$. Then it sends the reply it gets from $\mathcal{S}$ to P.

■ Upon receiving $(\mathsf{sid}, \textsc{Leakage})$ from $\mathcal{S}$, it reads the time Cl from $\mathcal{G}_{\mathsf{Clock}}$. If $\mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cred}) = \mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cast}) = \mathtt{Status}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Open}) = \perp$ then it returns to $\mathcal{S}$ all the triples $(v, o, 1)$ such that $(V, v, o, 1) \in L_{\mathsf{gball}} \wedge (V, v, \sigma, 1) \in L_{\mathsf{cast}}$.

Figure 5.4: The vote management functionality $\mathcal{F}_{\mathsf{vm}}^{\mathsf{delay}}(\mathbf{V})$ interacting with voters $\mathbf{V}$, SA and the simulator $\mathcal{S}$.

5.3 The E-cclesia family: relalization of $\mathcal{F}_{\mathsf{STE}}^{\mathsf{delay}}$ in the $(\mathcal{F}_{\mathsf{elig}}, \mathcal{F}_{\mathsf{vm}}^{\mathsf{delay}}, \mathcal{G}_{\mathsf{clock}})$-hybrid model

5.3.1 Description of the E-cclesia family

We provide a description of the E-CCLESIA family $\Pi_{\text{E-CCLESIA}}^{\mathcal{F}_{\mathsf{elig}}, \mathcal{F}_{\mathsf{vm}}, \mathcal{G}_{\mathsf{clock}}}$ of STE schemes as a hybrid protocol that makes use of the main modules $\mathcal{F}_{\mathsf{elig}}, \mathcal{F}_{\mathsf{vm}}$. Any pair of real-world protocols $\Pi_{\mathsf{elig}}, \Pi_{\mathsf{vm}}$ that UC-realize $\mathcal{F}_{\mathsf{elig}}, \mathcal{F}_{\mathsf{vm}}$ respectively, specifies a member of the family. The description of $\Pi_{\text{E-CCLESIA}}^{\mathcal{F}_{\mathsf{elig}}, \mathcal{F}_{\mathsf{vm}}, \mathcal{G}_{\mathsf{clock}}}$ follows the phases and command interface of $\mathcal{F}_{\mathsf{STE}}$ in Section 5.1 as described below:

The protocol $\Pi_{\text{E-CCLESIA}}(\mathcal{F}_{\mathsf{elig}}, \mathcal{F}_{\mathsf{vm}}^{\mathsf{delay}}, \mathcal{G}_{\mathsf{clock}})$.

Setup.

■ Upon receiving $(\mathsf{sid}, \text{ELECTION\_INFO}, \mathbf{V}_{\mathsf{elig}}, t_{\mathsf{cast}}, t_{\mathsf{open}})$ from $\mathcal{Z}$, if $\mathbf{V}_{\mathsf{elig}} \subseteq \mathbf{V}$ and $t_{\mathsf{cast}} < t_{\mathsf{open}}$, SA sends $(\mathsf{sid}, \text{SETUP\_INFO}, \mathbf{V}_{\mathsf{elig}}, t_{\mathsf{cast}}, t_{\mathsf{open}})$ to $\mathcal{F}_{\mathsf{vm}}$.

■ Upon receiving $(\mathsf{sid}, \text{ELIGIBLE})$ from the environment $\mathcal{Z}$, if SA has received $(\mathbf{V}_{\mathsf{elig}}, t_{\mathsf{cast}}, t_{\mathsf{open}})$, it sends $(\mathsf{sid}, \text{ELIGIBLE}, \mathbf{V}_{\mathsf{elig}}, t_{\mathsf{cast}}, t_{\mathsf{open}})$ to $\mathcal{F}_{\mathsf{elig}}$ which sends $\mathsf{reg.par} := (\mathbf{V}_{\mathsf{elig}}, t_{\mathsf{cast}}, t_{\mathsf{open}}, \boldsymbol{t}, St_{\mathsf{gen}})$ to $\langle V_i \rangle_{i\in[n]}$. Upon receiving $\mathsf{reg.par}$ from $\mathcal{F}_{\mathsf{elig}}$, each voter $V \in \mathbf{V}$ stores it as the registration parameters $\mathsf{reg.par}$.

Credential generation. This phase is completely managed by $\mathcal{F}_{\mathsf{elig}}$.

■ Upon receiving $(\mathsf{sid}, \text{GEN\_CRED})$ from $\mathcal{Z}$, V sends $(\mathsf{sid}, \text{GEN\_CRED})$ to $\mathcal{F}_{\mathsf{elig}}$, which in turn sends $(\mathsf{sid}, \text{GEN\_CRED}, V, \mathsf{rc})$ to $\langle V_j \rangle_{j\in[n]}$ (or sends $(\mathsf{sid}, \text{GEN\_CRED}, \perp)$ to V and halts).

Cast. Here, $\mathcal{F}_{\mathsf{vm}}$ and $\mathcal{F}_{\mathsf{elig}}$ combined carry out the ballot generation, authentication and broadcasting tasks.

■ Upon receiving $\big(\mathsf{sid}, \text{CAST}, o\big)$ from $\mathcal{Z}$, V executes the following steps:

1. She sends $(\mathsf{sid}, \text{GEN\_BALLOT}, o)$ to $\mathcal{F}_{\mathsf{vm}}$ which replies either with $(\mathsf{sid}, \text{GENERATING})$ or $(\mathsf{sid}, \text{GEN\_BALLOT}, o, \perp)$, which in the second case she forwards the message to $\mathcal{Z}$.
2. If $\mathsf{delay} = 0$ she sends $(\mathsf{sid}, \text{RETRIEVE})$ to $\mathcal{F}_{\mathsf{vm}}$. Upon receiving $(\mathsf{sid}, \text{RETRIEVE}, (o, v))$ from $\mathcal{F}_{\mathsf{vm}}$ she does the *Cast* step as described below.
3. In any other case, she returns $(\mathsf{sid}, \text{CASTING})$ to $\mathcal{Z}$.

■ Upon receiving $(\mathsf{sid}, \text{CAST?})$ from $\mathcal{Z}$, V sends $(\mathsf{sid}, \text{RETRIEVE})$ from $\mathcal{F}_{\mathsf{vm}}$. Upon receiving $(\mathsf{sid}, \text{RETRIEVE}, (o, v))$ from $\mathcal{F}_{\mathsf{vm}}$ she does:

- ***Cast:*** She sends $(\mathsf{sid}, \text{AUTH\_BALLOT}, v)$ to $\mathcal{F}_{\mathsf{elig}}$ which replies with the authentication receipt for v as $(\mathsf{sid}, \text{AUTH\_BALLOT}, v, \sigma)$ (or sends $(\mathsf{sid}, \text{AUTH\_BALLOT}, \perp)$ to V and halts). Finally, she sends $\big(\mathsf{sid}, \text{CAST}, v, \sigma\big)$ to $\mathcal{F}_{\mathsf{vm}}$ which broadcasts the message to $\langle V_j \rangle_{j\in[n]}$. In turn, the voters store the received pair (v, σ).

Tally. In order for the voter to perform self-tallying, she accesses $\mathcal{F}_{\mathsf{elig}}$ for ballot verification and linkability and $\mathcal{F}_{\mathsf{vm}}$ for ballot opening.

- Upon receiving a message $(\mathsf{sid}, \text{TALLY})$ from $\mathcal{Z}$, V executes the following steps:

1. **For** every tuple $(\mathsf{sid}, \text{CAST}, v, \sigma)$ she has obtained from $\mathcal{F}_{\mathsf{vm}}$, V sends $(\mathsf{sid}, \text{VER\_BALLOT}, v, \sigma)$ to $\mathcal{F}_{\mathsf{elig}}$ which replies with $(\mathsf{sid}, \text{VER\_BALLOT}, v, \sigma, x)$, where $x \in \{0, 1, \perp\}$.

   If there is any ballot verification request such that $\mathcal{F}_{\mathsf{elig}}$ replied with $x = \perp$, then V sets tally to $\perp$. Otherwise, she includes in her tally set all pairs (v, σ) such that $\mathcal{F}_{\mathsf{elig}}$ replied with $x = 1$.

2. V discards multiple ballots as follows: for every pair (v, σ), (v', σ') in her tally set, she sends $(\mathsf{sid}, \text{LINK\_BALLOTS}, (v, \sigma), (v', \sigma'))$ to $\mathcal{F}_{\mathsf{elig}}$. If she gets $(\mathsf{sid}, \text{LINK\_BALLOTS}, (v, \sigma), (v', \sigma'), 1)$ as a response, then she discards the ballot she received the last out of those two. Clearly, after this pairwise check is completed, all except one of ballots that are linked will be removed from the tally set, so that one-voter-one vote is guaranteed.

3. **For** every pair (v, σ) in the tally set, V sends $(\mathsf{sid}, \text{OPEN}, v)$ to $\mathcal{F}_{\mathsf{vm}}$, which replies with the opening $(\mathsf{sid}, \text{OPEN}, v, o)$. Then, V adds o to the multi-set of all opened options (initialized as empty). If at any time $\mathcal{F}_{\mathsf{vm}}$ replies with $(\mathsf{sid}, \text{OPEN}, v, \perp)$, then V sets tally to $\perp$.

4. Finally, she sets the tally result as the multi-set of all opened options.

Figure 5.5: Description of the protocol $\Pi_{\text{E-CCLESIA}}$ in $(\mathcal{F}_{\mathsf{elig}}, \mathcal{F}_{\mathsf{vm}}^{\mathsf{delay}}, \mathcal{G}_{\mathsf{clock}})$-hybrid model.

Observe that the voters discard the same ballots after the pairwise check-in Tally phase. Specifically, they receive the ballots in the same order because of the way $\mathcal{F}_{\mathsf{vm}}$ works (either the adversary allows a ballot to reach everyone or blocks it). Thus, the tally for all voters is identical.

5.3.2 Realization of $\mathcal{F}_{\mathsf{STE}}^{\mathsf{delay}}$ via $\mathcal{F}_{\mathsf{elig}}$ and $\mathcal{F}_{\mathsf{vm}}^{\mathsf{delay}}$

Security: As already mentioned in the introduction of Section 5.2, proving that $\Pi_{\text{E-CCLESIA}}^{\mathcal{F}_{\mathsf{elig}}, \mathcal{F}_{\mathsf{vm}}, \mathcal{G}_{\mathsf{clock}}}$ UC-realizes $\mathcal{F}_{\mathsf{STE}}$ in the $\{\mathcal{F}_{\mathsf{elig}}, \mathcal{F}_{\mathsf{vm}}, \mathcal{G}_{\mathsf{clock}}\}$-hybrid model is straightforward given the description of $\mathcal{F}_{\mathsf{STE}}$, $\mathcal{F}_{\mathsf{elig}}$ and $\mathcal{F}_{\mathsf{vm}}$. Below, we provide the theorem statement and the proof. The proof idea is that for any real-world adversary $\mathcal{A}$ against $\Pi_{\text{E-CCLESIA}}^{\mathcal{F}_{\mathsf{elig}}, \mathcal{F}_{\mathsf{vm}}, \mathcal{G}_{\mathsf{clock}}}$ and environment $\mathcal{Z}$, we can construct a simulator $\mathcal{S}$ that interacts with $\mathcal{F}_{\mathsf{STE}}$ and emulates an execution of $\Pi_{\text{E-CCLESIA}}^{\mathcal{F}_{\mathsf{elig}}, \mathcal{F}_{\mathsf{vm}}, \mathcal{G}_{\mathsf{clock}}}$ in the presence of $\mathcal{A}$ by playing the role of the honest parties, $\mathcal{F}_{\mathsf{elig}}$ and $\mathcal{F}_{\mathsf{vm}}$. In addition, $\mathcal{S}$ acts as a proxy between the interaction of $\mathcal{A}$ and $\mathcal{Z}$.

Theorem 5. *The protocol* $\Pi_{\text{E-CCLESIA}}^{\mathcal{F}_{\mathsf{elig}},\mathcal{F}_{\mathsf{vm}}^{\mathsf{delay}},\mathcal{G}_{\mathsf{clock}}}$ *described in Subsection 5.3.1 UC-realizes* $\mathcal{F}_{\mathsf{STE}}^{\mathsf{delay}}$ *in the* $\{\mathcal{F}_{\mathsf{elig}}, \mathcal{F}_{\mathsf{vm}}^{\mathsf{delay}}, \mathcal{G}_{\mathsf{clock}}\}$*-hybrid model.*

Proof. We define a simulator $\mathcal{S}$ as follows: In the cases of public delayed outputs, we assume that $\mathcal{S}$ forwards the message to the adversary $\mathcal{A}$ as if it was from either $\mathcal{F}_{\mathsf{elig}}$ or $\mathcal{F}_{\mathsf{vm}}$ and responds to $\mathcal{F}_{\mathsf{STE}}$ in the same way as $\mathcal{A}$. Moreover, whatever command $\mathcal{F}_{\mathsf{STE}}$ receives on behalf of a corrupted voter, we assume that $\mathcal{F}_{\mathsf{STE}}$ forwards that message to $\mathcal{S}$. Then $\mathcal{S}$ forwards that message to $\mathcal{A}$ as if it was from either $\mathcal{F}_{\mathsf{elig}}$ or $\mathcal{F}_{\mathsf{vm}}$ and he returns to $\mathcal{F}_{\mathsf{STE}}$ whatever he receives from $\mathcal{A}$. We describe the details below.

During the **Setup** phase, when $\mathcal{S}$ receives the corruption set $\mathbf{V}_{\mathsf{corr}}$ from $\mathcal{Z}$, he forwards it to $\mathcal{A}$ as if it was from $\mathcal{Z}$. Then $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{elig}}$ and $\mathcal{F}_{\mathsf{vm}}$ and receives back the corruption set from $\mathcal{A}$. Next, $\mathcal{S}$ forwards the corruption set to $\mathcal{F}_{\mathsf{STE}}$. Upon receiving $(\mathsf{sid}, \textsc{Election\_Info}, \mathbf{V}_{\mathsf{elig}}, t_{\mathsf{cast}}, t_{\mathsf{open}})$ from $\mathcal{F}_{\mathsf{STE}}$, $\mathcal{S}$ stores the parameters and he sends the message $(\mathsf{sid}, \textsc{Setup\_Info}, \mathbf{V}_{\mathsf{elig}}, t_{\mathsf{cast}}, t_{\mathsf{open}})$ to $\mathcal{A}$ as if it was from $\mathcal{F}_{\mathsf{vm}}$. Then, $\mathcal{S}$ returns to $\mathcal{F}_{\mathsf{STE}}$ whatever he receives from $\mathcal{A}$. After $\mathcal{S}$ receives $(\mathsf{sid}, \textsc{Eligible})$ from $\mathcal{F}_{\mathsf{STE}}$, he plays the role of $\mathcal{F}_{\mathsf{elig}}$ and he sends $(\mathsf{sid}, \textsc{Setup\_Elig})$ to $\mathcal{A}$. Then, upon receiving the eligibility algorithms $(\mathsf{sid}, \textsc{Setup\_Elig}, \texttt{GenCred}, \texttt{AuthBallot}, \texttt{VrfyBallot}, \texttt{UpState}, St_{\mathsf{gen}})$ from $\mathcal{A}$, $\mathcal{S}$ forwards the message to $\mathcal{F}_{\mathsf{STE}}$.

During the **Credential generation** phase, when $\mathcal{S}$ receives a credential generation request from $\mathcal{F}_{\mathsf{STE}}$ on behalf of a corrupted voter V, $\mathcal{S}$ forwards the request to $\mathcal{A}$ as if it was from $\mathcal{F}_{\mathsf{elig}}$. Then, upon receiving $(\mathsf{sid}, \textsc{Gen\_Cred}, V, \mathsf{cr}, \mathsf{rc})$ from $\mathcal{A}$, $\mathcal{S}$ forwards the message to $\mathcal{F}_{\mathsf{STE}}$.

During the **Cast** phase, when $\mathcal{S}$ receives $(\mathsf{sid}, \textsc{Gen\_Ballot}, \mathsf{tag}, \mathsf{CI}, 0^{|o|})$ from $\mathcal{F}_{\mathsf{STE}}$, he forwards the message to $\mathcal{A}$ as if it was $\mathcal{F}_{\mathsf{vm}}$ and returns to $\mathcal{F}_{\mathsf{STE}}$ whatever receives from $\mathcal{A}$. Upon receiving a $\textsc{Clock\_Advance}$ command from $\mathcal{G}_{\mathsf{clock}}$ on behalf of a voter V, he forwards the message to $\mathcal{A}$ as if he was $\mathcal{G}_{\mathsf{clock}}$. Upon receiving an $\textsc{Update}$ command from $\mathcal{A}$ as if he was $\mathcal{F}_{\mathsf{vm}}$, $\mathcal{S}$ forwards the message to $\mathcal{F}_{\mathsf{STE}}$. When $\mathcal{S}$ receives $(\mathsf{sid}, \textsc{Cast}, v, \sigma)$ from $\mathcal{F}_{\mathsf{STE}}$, he forwards the message to $\mathcal{A}$ as if it was from $\mathcal{F}_{\mathsf{vm}}$. Then $\mathcal{S}$ returns whatever he receives from $\mathcal{A}$. Upon receiving a cast ballot request from $\mathcal{F}_{\mathsf{STE}}$ on behalf of a corrupted voter V, $\mathcal{S}$ forwards the message to $\mathcal{A}$ as if it was from $\mathcal{F}_{\mathsf{vm}}$. After $\mathcal{S}$ receives $\big(\mathsf{sid}, \textsc{Cast}, \tilde{v}, \tilde{\sigma}, V\big)$ from $\mathcal{A}$, he returns the message $\big(\mathsf{sid}, \textsc{Cast}, o, \tilde{v}, \tilde{\sigma}, V\big)$ to $\mathcal{F}_{\mathsf{STE}}$ for some o. The choice of o is irrelevant because during the **Tally** phase $\mathcal{S}$ will be asked by $\mathcal{F}_{\mathsf{STE}}$ for a new opening of $\tilde{v}$. This happens because every time $\mathcal{F}_{\mathsf{STE}}$ ($\mathcal{F}_{\mathsf{vm}}$ as well) receives a request for opening a malicious ballot, it gives the token to the simulator and it answers according to what $\mathcal{S}$ provides.

During the **Tally** phase, when $\mathcal{S}$ is asked by $\mathcal{F}_{\mathsf{STE}}$ for an alternative ballot opening of the tuple $(V, o, v, \cdot) \in L_{\mathsf{tally}}$ where $V \in \mathbf{V}_{\mathsf{corr}}$, $\mathcal{S}$ sends the message $(\mathsf{sid}, \textsc{Open}, v)$ to $\mathcal{A}$ as if it was from $\mathcal{F}_{\mathsf{vm}}$ and he returns to $\mathcal{F}_{\mathsf{STE}}$ the alternative opening received from $\mathcal{A}$. Upon receiving $(\mathsf{sid}, \textsc{Open}, v)$ from $\mathcal{Z}$ for the first time, $\mathcal{S}$ sends $(\mathsf{sid}, \textsc{Tally})$ to $\mathcal{F}_{\mathsf{STE}}$. Upon receiving $\big(\mathsf{sid}, \textsc{Tally}, \{(o, v) | (V, o, v, \cdot) \in L_{\mathsf{tally}}\}\big)$ from $\mathcal{F}_{\mathsf{STE}}$, $\mathcal{S}$ records the tally. Next, $\mathcal{S}$ forwards the message $(\mathsf{sid}, \textsc{Open}, v)$

to $\mathcal{A}$ as if it was from $\mathcal{Z}$. Upon receiving $(\mathsf{sid}, \textsc{Open}, v)$, $\mathcal{S}$ playing the role of $\mathcal{F}_{\mathsf{vm}}$ returns the plaintext for that ballot to $\mathcal{A}$. If the message does not exist in the list previously provided by $\mathcal{F}_{\mathsf{STE}}$, then $\mathcal{S}$ asks $\mathcal{A}$ for the opening of that message as before.

The distribution of messages is the same in both settings since the algorithms that are used are the same. As a result, the simulation is perfect. □

5.4 Realizing $\mathcal{F}_{\mathsf{elig}}$ via accumulators

This section describes the primitives needed to build a real protocol that realizes the eligibility functionality. Since cryptographic accumulators do not have a suitable UC treatment in the literature that would fit our purpose, Subsections 5.4.1 and 5.4.2 provide an ideal accumulator functionality and link it to a standard game-based definition. [1]

Note that recently, Baldimtsi et al. (2018) provided a UC functionality for accumulators, however, it requires the accumulation operation to be managed centrally by some authority. Subsection 5.4.3 presents the Π_{elig} protocol, which is not suitable for our purpose as we require the accumulation operation to be managed by every party and not only the accumulation manager. The authors in Baldimtsi et al. (2018) remarked in a dedicated Paragraph in Subsection 3.1 how they can capture such cases by allowing every party to invoke the accumulation operation in their functionality without filling up the details. Moreover, their functionality captures a broader spectrum of accumulators (e.g. accumulators that support proof of non-membership Li et al. (2007)) but at the expense of being complicated. On the other hand, our functionality fills up the details of the decentralized accumulation operations. Moreover, our functionality is simpler than the one of Baldimtsi et al. (2018), but it captures a more limited spectrum of protocols; which is sufficient for our purpose.

5.4.1 Definition of $\mathcal{F}_{\mathsf{acc}}$

The purpose of secure accumulators is to provide an object representing a set and create witnesses for specific items being in the set. Our starting point for this functionality is the property-based definition by Camenisch and Lysyanskaya (2002) provided below. In this model, it is assumed that a trusted accumulator manager runs `Gen` to generate both the public parameters and the secret trapdoor.

Definition 5.4.1 (Accumulator)**.** $\mathsf{AC} = (\mathcal{X}_\lambda, \mathtt{Gen})$ is an accumulator scheme for a family of inputs $\{\mathcal{X}_\lambda\}$ if it has the following properties:

[1] The definition of the ideal functionality $\mathcal{F}_{\mathsf{acc}}$, the protocol Π_{acc} and the proof of UC-realization is also presented in Lenka Marekova's honours thesis that I co-supervised Marekova (2018).

1. **Efficient generation**

   $\texttt{Gen}$ is an efficient probabilistic algorithm such that $\texttt{Gen}(1^\lambda) \to (f, aux_f)$ for random $f \in F_\lambda$, where F_λ is a family of functions and aux_f some auxiliary information with respect to f (e.g a secret key sk_f that is required for the generation of the accumulated value after the deletion of an element from the accumulation set.)

2. **Efficient evaluation**

   $f \in F_\lambda$ is a polynomial-size circuit such that $f(w, x) \to a$ for $(w, x) \in \mathcal{U}_f \times \mathcal{X}_\lambda$ and $a \in \mathcal{U}_f$ where $\mathcal{U}_f$ is an efficiently samplable domain for f.

3. **Correctness**

   $w \in \mathcal{U}_f$ is a witness for $x \in \mathcal{X}_\lambda$ in the accumulator $a \in \mathcal{U}_f$ under f if $\texttt{Verify}_f(w, x, a) \to \top$ where $\texttt{Verify}_f(w, x, a) := f(w, x) = a$.

4. **Quasi-commutativity**

   For all $f \in F_\lambda$, $u \in \mathcal{U}_f$, $x_1, x_2 \in \mathcal{X}_\lambda$: $f(f(u, x_1), x_2) = f(f(u, x_2), x_1)$. So for $X = \{x_1, \ldots, x_m\} \subset \mathcal{X}_\lambda$, $f(u, X) := f(f(\ldots(u, x_1), \ldots), x_m)$.

5. **Witness unforgeability**

   Let $\mathcal{U}'_f \times \mathcal{X}'_\lambda$ denote the domains for which the computational procedure for function $f \in F_\lambda$ is defined (so $\mathcal{U}_f \subseteq \mathcal{U}'_f, \mathcal{X}_\lambda \subseteq \mathcal{X}'_\lambda$).

   AC is secure if for all PPT adversaries $\mathcal{A}_\lambda$:

   $$Pr[f \leftarrow \texttt{Gen}(1^\lambda); u \leftarrow \mathcal{U}_f; (x, w, X) \leftarrow \mathcal{A}_\lambda(f, \mathcal{U}_f, u) : X \subset \mathcal{X}_\lambda; w, a \in \mathcal{U}'_f; x \in \mathcal{X}'_\lambda; x \notin X; \texttt{Verify}_f(u, X, a) = \texttt{Verify}_f(w, x, a) = \top] = \mathsf{negl}(\lambda)\,.$$

6. **Efficient deletion**

   AC is dynamic if there exist efficient algorithms $\texttt{Delete}$, $\texttt{Update}$ such that if $x, x' \in X$ and $\texttt{Verify}_f(u, X, a) = \texttt{Verify}_f(w, x, a) = \top$, then:
   $\texttt{Delete}(aux_f, a, x') \to a'$ such that $\texttt{Verify}_f(u, X \backslash \{x'\}, a') = \top$,
   and $\texttt{Update}(f, a, a', x, x') \to w'$ such that $\texttt{Verify}_f(w', x, a') = \top$.

Remark 1. *Definition 5.4.1 is well known, but it is not the only definition of accumulators in the standard model Derler et al. (2015a). The reason for matching our functionality to this particular definition is that it suits the public setting that we wanted to model the best. As opposed to the more common definitions which involve a central authority responding to requests for accumulation, here the accumulator is a public function which any party can use.*

This is not without caveats, as we still rely on trusted setup, and efficient deletions require the manager to use his secret trapdoor. Coming up with a better definition for our use-case (and constructions that satisfy it) is an interesting problem, but out of the scope of this thesis. We also note that quasi-commutativity

is often handled as an optional property, but it is implicitly embedded in the way Camenisch and Lysyanskaya (2002) defines witnesses. Extending to non-commutative constructions would hence also require a new definition.

The definition of the ideal functionality $\mathcal{F}_{\mathsf{acc}}$ is shown in Figure 5.6. $\mathcal{F}_{\mathsf{acc}}$, parameterized by the input set $\mathcal{X}_\lambda$, stores a list L_{acc} of (a, X, w) entries where a is the accumulator value, X is the set of accumulated items and w is either another accumulator or the basis u, a fixed value representing the empty set. Since $\mathcal{F}_{\mathsf{acc}}$ is not producing the witnesses itself, to ensure correctness it needs to keep some auxiliary information. L_{sets} is a list of $(a, [S_1, S_2, \dots])$ entries for each a in L_{acc}, where $[S_1, S_2, \dots]$ is a list of all known representations of the contents of the set accumulated in a. L_{ref} is a list of (a, A_r) entries which represents references to accumulators that have not been expanded yet. We call an accumulator *expanded* if it has an entry in L_{sets}, i.e. there exists some representation of its contents. If it is not expanded, there is no such entry, but A_r may appear (syntactically) as part of some list S_i for another entry. We denote all such unexpanded references in capitals and with the $\_r$ suffix. We call a set *fully-expanded* if it does not contain references to any unexpanded accumulators.

To illustrate the use of these lists, we give an example execution for the accumulation of items x_1, x_2, x_3:

1. *Accumulate* $\{x_2, x_3\}$ *with witness* a*:* $\mathcal{F}_{\mathsf{acc}}$ computes $f(a, \{x_2, x_3\}) = b$, adds $(b, \{x_2, x_3\}, a)$ to L_{acc}, $(b, [A_r \cup \{x_2, x_3\}])$ to L_{sets} and (a, A_r) to L_{ref}.

2. *Accumulate* x_1 *with witness* u*:* $\mathcal{F}_{\mathsf{acc}}$ computes $f(u, x_1) = a$ and adds $(a, \{x_1\}, u)$ to L_{acc} and $(a, [\{x_1\}])$ to L_{sets}. Since (a, A_r) is in L_{ref}, the first entry in L_{sets} expands to $(b, [\{x_1, x_2, x_3\}])$ and (a, A_r) is removed from L_{ref}.

3. *Accumulate* $\{x_1, x_2, x_3\}$ *with witness* u*:* $\mathcal{F}_{\mathsf{acc}}$ computes $f(u, \{x_1, x_2, x_3\}) = c$. If $c = b$, it adds $(c, \{x_1, x_2, x_3\}, u)$ to L_{acc}, otherwise it aborts.

We can now describe the individual command interface of $\mathcal{F}_{\mathsf{acc}}$. When $\mathcal{F}_{\mathsf{acc}}$ receives $(\mathsf{sid}, \textsc{Setup})$ from the accumulation manager M for the first time it requests from $\mathcal{S}$ the accumulation value space $\mathcal{U}_\mathsf{f}$ the initial seed accumulation value u, the accumulation function f , the auxiliary information aux_f,the verification algorithm `Verify` and the deletion algorithm `Delete`, `Update` and returns them to M.

Upon receiving $(\mathsf{sid}, \textsc{Accum}, w, X)$ from any party $P \in \mathbf{P}$, $\mathcal{F}_{\mathsf{acc}}$ checks if the pair (w, X) are from the correct sampling space and if the value X has been accumulated before on w. If the latter is the case; it returns the recorded value, else $\mathcal{F}_{\mathsf{acc}}$ computes with the function f the accumulated value a. In case that a does not belong to the correct sampling space or the verification check that w is a valid witness that the set X is accumulated in value a is false, $\mathcal{F}_{\mathsf{acc}}$ returns the special abort symbol $\perp$ to party P. If the value w is the initial seed u of the accumulator then $\mathcal{F}_{\mathsf{acc}}$ creates the list L_a and inserts the set X to keep track of

which set the value a represents. Similarly, if there exists already a list L_w for the value w, $\mathcal{F}_{\mathsf{acc}}$ updates the list L_a with the set X and the contents of the list L_w. Note that there might be many representations for one accumulated value, that is why a list instead of a set is the correct mathematical object to keep track of which sets this value represents. The case that neither the value w is equal to the initial seed u nor it exists a recorded set that w is the accumulated value means that w is not a legitimate accumulated value (e.g an accumulation value which is not issued through $\mathcal{F}_{\mathsf{acc}}$ and specifically via the function f). In that case, $\mathcal{F}_{\mathsf{acc}}$ creates the set W_r, which is associated with the value w, for reference reasons (e.g contains only the value "unknown"), and updates the list L_a as the union of W_r and X.

Next, $\mathcal{F}_{acc}$ checks if the value a has already representation sets and if that is the case it *updates* the existing list with these sets. If $\mathcal{F}_{\mathsf{acc}}$ detects that there exists a representative set with two different accumulated values (in this case the value a and a value a', such that $a \neq a'$), it returns $\perp$ to P. If the value a was previously appeared in a pair (a, A_r) in the list L_{ref}, $\mathcal{F}_{\mathsf{acc}}$ updates all entries in L_{sets}. If some of these updated entries have different accumulated values, it returns $\perp$. Else, it removes the entry (a, A_r) from the list L_{ref} and adds the entry (a, L_a) to the list L_{sets}. In case that the value a did not appear in the list L_{ref}, $\mathcal{F}_{\mathsf{acc}}$ simply adds the pair (a, L_a) to L_{sets}. Finally, $\mathcal{F}_{\mathsf{acc}}$ adds the triple (a, X, w) to L_{acc} and returns a to P. This step ensures the consistency of output with the previous values, and that incomplete set references updated correctly.

Upon receiving $(\mathsf{sid}, \textsc{Delete}, a, x, A)$ from M, $\mathcal{F}_{\mathsf{acc}}$ checks if the set A and the accumulated value a are from the correct sampling space and if x belongs to the set A and returns $\perp$ if the check fails. Next, it checks if indeed x belongs to a set X such that the representative value is a and if the value a represents the set A and returns $\perp$ if the check fails. After that, it computes the new accumulated value a' by using the `Delete` function and verifies if a' is the accumulation value of set $A \backslash \{x\}$. Next, it checks if the value a' appearing in the list L_{ref} and does the necessary steps as mentioned previously in the *update* operation. Finally, it returns the value a' to M.

Upon receiving $(\mathsf{sid}, \textsc{Update}, a, a', x, x', A)$ from any party $P \in \mathbf{P}$, $\mathcal{F}_{\mathsf{acc}}$ checks again if all values are from the correct sampling space, then it follows a similar procedure as when it receives the DELETE command. Next, it computes the new witness w' by using the function `Update` and verifies with the algorithm `Verify` if w' is a witness that the value x has been accumulated in a'. Finally, $\mathcal{F}_{\mathsf{acc}}$ updates all the existing lists based on w' and a' for consistency, similar to steps we described when $\mathcal{F}_{\mathsf{acc}}$ receives the ACCUM command and returns the witness w' to P. In a nutshell, the command Update subtracts from the accumulator the value x' and inserts the value x instead.

Upon receiving $(\mathsf{sid}, \textsc{Verify}, w, X, a)$ from $P \in \mathbf{P}$, $\mathcal{F}_{\mathsf{acc}}$ checks if the w is a witness that the set X has been accumulated in a. If there is a recorded triple-$(a, X, w) \in L_{\mathsf{acc}}$, $\mathcal{F}_{\mathsf{acc}}$ returns $\top$ to P. Else, the functionality checks if forgery has

occurred. Specifically, if the result of the verification is $\top$ but: i) the accumulated ordered set X is not a subset of a fully expanded set A for value a, then adversary managed to forge at least one element into the value a; ii) in case that the value a had not a fully expanded set, which means that $\mathcal{F}_{\mathsf{acc}}$ is not fully aware which elements it accumulates, $\mathcal{F}_{\mathsf{acc}}$ checks if there exists a fully expanded set (e.g. a complete track history of the accumulated elements) but for the set $V = A - W$ where $W \in L_w$. If there exists, $\mathcal{F}_{\mathsf{acc}}$ checks if the ordered sets V and X represent the same elements. If not, then a forgery has occurred in the sense that the adversary provided a "fake" witness.

In both cases security has been breached (violation of `correctness`) and $\mathcal{F}_{\mathsf{acc}}$ returns $\perp$.

In order to capture various settings, where command lines could be omitted, we indicate with red the command lines that can be totally omitted from $\mathcal{F}_{\mathsf{acc}}$. We can re-define our $\mathcal{F}_{\mathsf{acc}}$ depending if we choose to omit from its description the red command lines or not (e.g $\mathcal{F}^{*}_{\mathsf{acc}}$ is the same as $\mathcal{F}_{\mathsf{acc}}$ but the command lines in red are omitted from the description of $\mathcal{F}_{\mathsf{acc}}$) depending on the actual protocol we prefer to realize. We stress that in E-CCLESIA deletion commands never occur, so the manager is not required after the **Setup** phase.

The accumulator functionality $\mathcal{F}_{\mathsf{acc}}(M, \mathbf{P}, \mathcal{X}_\lambda))$

The functionality initializes the lists of accumulated values L_{acc}, of relation between accumulation values and ordered sets L_{sets} and the list of accumulators that are not expanded L_{ref} as empty.

■ Upon receiving (sid, SETUP) from the manager M for the first time, it sends (sid, SETUP, $\mathcal{X}_\lambda$) to $\mathcal{S}$. Upon receiving (sid, ALGS, ($\mathcal{U}_f$, u), f, aux_f, `Verify`, `Delete`, `Update`) from $\mathcal{S}$, it sets params = ($\mathcal{X}_\lambda$, $\mathcal{U}_f$, u) and sends (sid, ALGS, params, f, aux_f, `Verify`, `Delete`, `Update`) to M.

■ Upon receiving (sid, PARAMS) from any party P, it returns (sid, PARAMS, params, f, `Verify`, `Update`) to P.

■ Upon receiving (sid, ACCUM, w, X) from party $P \in \mathbf{P}$:

1. If $X \not\subseteq \mathcal{X}_\lambda$ or $w \notin \mathcal{U}_f$, it returns (sid, ACCUM, w, $X, \perp$) to P.
2. If there is (a, X, w) $\in L_{\mathsf{acc}}$ for some a, it returns (sid, ACCUM, a) to P.
3. It computes $a \leftarrow f(w, X)$. If $a \notin \mathcal{U}_f$ or `Verify`(w, X, a) $\neq \top$, it returns (sid, ACCUM, w, X,$\perp$) to P.
4. If $w = u$, it creates $L_a \leftarrow [X]$. Else if there is $(w, L_w) \in L_{\mathsf{sets}}$ for some L_w, it creates $L_a \leftarrow [W \cup X : W \in L_w]$.
   Else it adds (w, W_r) to L_{ref}, and creates $L_a \leftarrow [W_r \cup X]$.
5. If there is $(a, L'_a) \in L_{\mathsf{sets}}$ for some L'_a, it replaces the pair with $(a, L'_a \,||\, L_a)$ (i.e. it appends L_a to L'_a) unless $L'_a = L_a$. Else if for any $A \in L_a$, there is (a', L'_a) $\in L_{\mathsf{sets}}$ for some $a' \neq a$ and $A \in L'_a$, it returns (sid, ACCUM, w, X,$\perp$) to P. Else if $(a, A_r) \in L_{\mathsf{ref}}$ for some A_r:
   (a) For every $(a', L'_a) \in L_{\mathsf{sets}}$, it replaces each occurrence of A_r in L'_a with items from L_a (so each $A' = A_r \cup \ldots$ can expand into multiple $E = A \cup \ldots$). For any E, if there is (a'', L''_a) $\in L_{\mathsf{sets}}$ for some $a'' \neq a'$ such that $E \in L''_a$, it returns (sid, ACCUM, w, $X, \perp$) to P.
   (b) It removes (a, A_r) from L_{ref} and it adds (a, L_a) to L_{sets}.

   Else, it adds (a, L_a) to L_{sets}.
6. It adds (a, X, w) to L_{acc} and returns (sid, ACCUM, a) to P.

- Upon receiving (sid, DELETE, a, x, A) from manager M:
  1. If $A \not\subseteq \mathcal{X}_\lambda$, $a \notin \mathcal{U}_f$ or $x \notin A$, it returns (sid, DELETE, a, x, $A, \perp$) to M.
  2. If there is no $(a, X, w) \in L_{\mathsf{acc}}$ for some w such that $x \in X$, and no $(a, L_a) \in L_{\mathsf{sets}}$ such that $A \in L_a$, it returns (sid, DELETE, a, x, $A, \perp$) to M.
  3. It computes $a' \leftarrow \mathtt{Delete}(a, x, aux_f)$.
  4. If $a' = u$ and $A = \{x\}$, it skips to Step 7.
  5. If $\mathtt{Verify}(u, A\backslash\{x\}, a') \neq \top$, it returns (sid, DELETE, a, x, $A, \perp$) to M.
  6. It runs Step 6 of ACCUM for a' and $L_a = [A\backslash\{x\}]$.
  7. It returns (sid, DELETED, a') to M.
- Upon receiving (sid, UPDATE, a, a', x, x', A) from $P \in \mathbf{P}$:
  1. If $A \not\subseteq \mathcal{X}'_\lambda$, a, $a' \notin \mathcal{U}_f$ or x, $x' \notin A$, it returns (sid, UPDATE, a, a', x, x', $A, \perp$) to P.
  2. It runs Step 2 of DELETE.
  3. It computes $w' \leftarrow \mathtt{Update}(a, a', x, x')$. If $\mathtt{Verify}(w', \{x\}, a') \neq \top$, it returns (sid, UPDATE, a, a', x, x', $A, \perp$) to P.
  4. It runs Steps 5 to 7 of ACCUM for w' and a'.
  5. It returns (sid, UPDATED, w') to P.
- Upon receiving (sid, VERIFY, w, X, a) from party $P \in \mathbf{P}$:
  1. If $X \not\subseteq \mathcal{X}'_\lambda$, $a \notin \mathcal{U}_f$ or $w \notin \mathcal{U}'_f$, it returns (sid, VERIFY, w, X, $a, \perp$) to P.
  2. If $(a, X, w) \in L_{\mathsf{acc}}$, it returns (sid, VERIFIED, $\top$) to P.
  3. If $\mathtt{Verify}(w, X, a) = \top$ and (a) or (b) holds, it returns (sid, VERIFY, w, X, $a, \perp$) to P.:
     (a) There is $(a, L_a) \in L_{\mathsf{sets}}$ such that for some fully-expanded $A \in L_a$ we have $X \not\subseteq A$.
     (b) There are (w, L_w), $(a, L_a) \in L_{\mathsf{sets}}$ such that for some $W \in L_w$ and $A \in L_a$ we have $A = W \cup V$ where V is a fully-expanded set with $V \not\subseteq X$.
  4. It returns (sid, VERIFIED, $\mathtt{Verify}(w, X, a)$) to P.

Figure 5.6: The accumulator functionality $\mathcal{F}_{\mathsf{acc}}(M, \mathbf{P})$ interacting with an accumulation manager M and a set of parties $\mathbf{P}$.

Next, we describe the protocol Π_{AC}. When M receives from $\mathcal{Z}$ (sid, SETUP)

for the first time it sends $(\mathsf{sid}, \textsc{Crs})$ to $\mathcal{F}_{\mathsf{Crs}}^{Gen}$ so that the initial parameters for the accumulator can be generated (e.g accumulator's seed, trapdoor information etc.). Then, M returns the parameters along with the hard-coded algorithms $f, \texttt{Verify}, \texttt{Delete}, \texttt{Update}$ to $\mathcal{Z}$. Similarly, when a party P receives the command message PARAMS from $\mathcal{Z}$ it gets the accumulator parameters that has been either already generated from a previous invocation by other party or are generated by this invocation, in any case the parameters are consistent between the parties. Then, P returns the parameters along with the hard-coded algorithms $f, \texttt{Verify}, \texttt{Update}$ to $\mathcal{Z}$. When P receives the command $(\mathsf{sid}, \textsc{Accum}, w, X)$ from $\mathcal{Z}$ it returns the accumulated value $f(w, X)$ after the necessary checks(e.g checks if w, X are sampled from the correct domain). In a similar manner M acts when it receives a delete command message from $\mathcal{Z}$. Upon receiving $(\mathsf{sid}, \textsc{Update}, a, a', x, x', A)$ from $\mathcal{Z}$, P checks again if the values are sampled from the correct domain and if a is the accumulated value of the ordered set A. If the checks fails, P returns $\perp$ to $\mathcal{Z}$ else it returns the updated witness to $\mathcal{Z}$. Finally, when P receives a VERIFY command message from $\mathcal{Z}$ it does the necessary checks and returns the result of the verification function $\texttt{Verify}$ to $\mathcal{Z}$.

The accumulation protocol Π_{AC} *for* $\mathsf{AC} = (\mathcal{X}_\lambda, \texttt{Gen})$.

- Upon receiving (sid, SETUP) for the first time from $\mathcal{Z}$, M sends (sid, CRS) to $\mathcal{F}_{\mathsf{CRS}}^{\mathtt{Gen}}$. Upon receiving (sid, CRS, $(f, \mathcal{U}_f, u, aux_f)$) from $\mathcal{F}_{\mathsf{CRS}}^{\mathtt{Gen}}$, M sets $\mathsf{params} = (\mathcal{X}_\lambda, \mathcal{U}_f, u)$ and returns (sid, ALGS, params, f, $\texttt{Verify}$, $\texttt{Delete}(aux_f, \cdot, \cdot)$, $\texttt{Update}(f, \cdot, \cdot, \cdot, \cdot)$) to $\mathcal{Z}$.
- Upon receiving (sid, PARAMS) for the first time from $\mathcal{Z}$, P sends (sid, CRS) to $\mathcal{F}_{\mathsf{CRS}}^{\mathtt{Gen}}$. Upon receiving (CRS, sid, $(f, \mathcal{U}_f, u)$) from $\mathcal{F}_{\mathsf{CRS}}^{\mathtt{Gen}}$, P sets $\mathsf{params} = (\mathcal{X}_\lambda, \mathcal{U}_f, u)$ and returns (sid, PARAMS, params, f, $\texttt{Verify}$, $\texttt{Update}$) to $\mathcal{Z}$.
- Upon receiving (sid, ACCUM, w, X) for the first time from $\mathcal{Z}$ if $X \not\subseteq \mathcal{X}'_\lambda$ or $w \notin \mathcal{U}'_f$, P returns (sid, ACCUM, w, $X, \perp$) to $\mathcal{Z}$. Otherwise it returns (sid, ACCUM, $f(w, X)$) to $\mathcal{Z}$.
- Upon receiving (sid, DELETE, a, x', A) for the first time from $\mathcal{Z}$, if $A \not\subseteq \mathcal{X}'_\lambda$, $a \notin \mathcal{U}'_f$, $x' \notin A$ or if $\texttt{Verify}(u, A, a) \neq \top$, M returns (sid, DELETE, a, x', $A, \perp$) to $\mathcal{Z}$. Otherwise it returns (sid, DELETED, $\texttt{Delete}(aux_f, a, x')$) to $\mathcal{Z}$.
- Upon receiving (sid, UPDATE, a, a', x, x', A) from $\mathcal{Z}$, if $A \not\subseteq \mathcal{X}'_\lambda$, a, $a' \notin \mathcal{U}'_f$, x, $x' \notin A$ or if $\texttt{Verify}(u, A, a) \neq \top$, P returns (sid, UPDATE, a, a', x, x', $A), \perp$ to $\mathcal{Z}$. Otherwise it returns (sid, UPDATED, $\texttt{Update}(a, a', x, x')$).
- Upon receiving (sid, VERIFY, w, X, a) from $\mathcal{Z}$, if $X \not\subseteq \mathcal{X}'_\lambda$, $w \notin \mathcal{U}'_f$, P returns (sid, VERIFY, w, X, $a, \perp$) to $\mathcal{Z}$. Else it returns (sid, VERIFIED, $\texttt{Verify}(w, X, a)$) to $\mathcal{Z}$.

Figure 5.7: Definition of Π_{AC} in the $\mathcal{F}_{\mathsf{CRS}}^{\mathtt{Gen}}$-hybrid model.

5.4.2 A protocol that realizes $\mathcal{F}_{\mathsf{acc}}$

In Figure 5.7, we define a real protocol Π_{AC} that uses an accumulator scheme in the CRS model. Theorem 6 captures the equivalence between our functionality and secure accumulator schemes, and in particular, implies that $\mathcal{F}_{\mathsf{acc}}$ can be realized by a strong RSA accumulator construction Camenisch and Lysyanskaya (2002). The proof can be found in Marekova (2018).

Theorem 6. Π_{AC} *UC-realizes* $\mathcal{F}_{\mathsf{acc}}$ *in the* $\mathcal{F}_{\mathsf{CRS}}^{\mathsf{Gen}}$*-hybrid model if and only if* $\mathsf{AC} = (\mathcal{X}_\lambda, \mathsf{Gen})$ *is a secure dynamic accumulator scheme.*

Commitment functionality $\mathcal{F}_{\mathsf{NIC}}$.

(sid, SETUP) → (sid, PARAMS, params, `Verify`) delayed output.
(sid, COMM, m) → (sid, COMM, c, o).
(sid, VERIFY, c, m, o) → (sid, VERIFIED, b).

Signature of knowledge functionality $\mathcal{F}_{\mathsf{SoK}}(\mathcal{F}_{\mathsf{acc}}, \mathcal{F}_{\mathsf{NIC}})$.

(sid, SETUP) → (sid, ALGS, `Sign`, `Verify`).
(sid, SIGN, m, (a, S), (c, w, r)) → (sid, SIGN, m, (a, S), σ).
(sid, VERIFY, m, (a, S), σ) → (sid, VERIFIED, b).

Figure 5.8: Interfaces provided by $\mathcal{F}_{\mathsf{NIC}}$ and $\mathcal{F}_{\mathsf{SoK}}(\mathcal{F}_{\mathsf{acc}}, \mathcal{F}_{\mathsf{NIC}})$.

5.4.3 A protocol that realizes $\mathcal{F}_{\mathsf{elig}}$

To realize $\mathcal{F}_{\mathsf{elig}}$, we need two other primitives besides accumulators, both of which have UC formulations in the literature: *non-interactive commitments* Camenisch et al. (2016) and (non-interactive zero-knowledge) *signatures of knowledge* Chase and Lysyanskaya (2006).

We note the command interfaces that they provide in Figure 5.8. Note that Figure 5.8 does not fully describe $\mathcal{F}_{\mathsf{NIC}}$ and $\mathcal{F}_{\mathsf{SoK}}$ instead, it presents only the command lines that are necessary for our purpose. $\mathcal{F}_{\mathsf{NIC}}$ allows a party to commit to a message m via a commitment c that can be verified using the opening o, and $\mathcal{F}_{\mathsf{SoK}}$ allows a party to produce a signature σ on message m if they know the opening r to some commitment c (which commits to some value S) which has been accumulated in a with witness w. $\mathcal{F}_{\mathsf{SoK}}$ can internally use any functionality that represents a language of statements on which signatures of knowledge can be made. In our case, it uses both $\mathcal{F}_{\mathsf{acc}}^{*}$ and $\mathcal{F}_{\mathsf{NIC}}$. The model of Chase and Lysyanskaya (2006) is easily extended to the case of two "internal" functionalities, since we can define the language that our $\mathcal{F}_{\mathsf{SoK}}$ accepts as $\mathcal{L} = \{(a, S, c, w, r) : (w, \{c\}, a) \in \mathcal{L}_{\mathsf{acc}} \wedge (c, S, r) \in \mathcal{L}_{\mathsf{NIC}}\}$, where $\mathcal{L}_{\mathsf{acc}}$, $\mathcal{L}_{\mathsf{NIC}}$ are the languages accepted by $\mathcal{F}_{\mathsf{acc}}^{*}$ and $\mathcal{F}_{\mathsf{NIC}}$, respectively. Figure 5.9 defines an

eligibility protocol that realizes $\mathcal{F}_{\mathsf{elig}}$ according to Theorem 7 where the proof can be found in Supplementary Material A.1. For ease of notation, when we say P calls $\mathcal{F}^{*}_{\mathsf{acc}}$ or $\mathcal{F}_{\mathsf{NIC}}$, we mean that the communication goes through $\mathcal{F}_{\mathsf{SoK}}$ since the functionalities are embedded within. Chase and Lysyanskaya (2006) describes formally how this is achieved. The full description of $\mathcal{F}_{\mathsf{SOK}}$ and $\mathcal{F}_{\mathsf{NIC}}$ can be found in Appendix A.

The eligibility protocol $\Pi^{\mathcal{F}_{\mathsf{SoK}}(\mathcal{F}^{*}_{\mathsf{acc}},\mathcal{F}_{\mathsf{NIC}}),\mathcal{F}_{\mathsf{an.BC}},\{\mathcal{F}^{P}_{\mathsf{cert}}\},\mathcal{G}_{\mathsf{clock}}}_{\mathsf{elig}}$.

All parties initialize the set $C \leftarrow \emptyset$. Moreover, all parties have hard coded the pair $(\texttt{define\_time}, \texttt{Status})$ If at any point a hybrid functionality aborts, the party returns $\perp$ to $\mathcal{Z}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Eligible}, \mathbf{V}_{\mathsf{elig}}, t_{\mathsf{cast}}, t_{\mathsf{open}})$ from $\mathcal{Z}$, if $\mathbf{V}_{\mathsf{elig}} \subseteq \mathbf{V}$ and SA's status=init, SA runs:

1. It sets its status to 'credential'.
2. It computes $\boldsymbol{t} \leftarrow \texttt{define\_time}(t_{\mathsf{cast}}, t_{\mathsf{open}})$.
3. It sends $(\mathsf{sid}, \textsc{Setup})$ to $\mathcal{F}^{*}_{\mathsf{acc}}$ to get $(\mathsf{sid}, \textsc{Algs}, \mathsf{params}_{\mathsf{acc}}, f, \texttt{Verify}_{\mathsf{acc}}, \texttt{Delete}, \texttt{Update})$. It parses $\mathsf{params}_{\mathsf{acc}}$ as $(\mathcal{X}_\lambda, \mathcal{U}_f, u)$. It sends $(\mathsf{sid}, \textsc{Setup})$ to $\mathcal{F}_{\mathsf{NIC}}$ to get $(\mathsf{sid}, \textsc{Params}, \mathsf{params}_{\mathsf{NIC}}, \texttt{Verify}_{\mathsf{NIC}})$. It parses $\mathsf{params}_{\mathsf{NIC}}$ as $(\mathcal{M}, \mathcal{R})$. It sends $(\mathsf{sid}, \textsc{Setup})$ to $\mathcal{F}_{\mathsf{SoK}}$ to get $(\mathsf{sid}, \textsc{Algs}, \texttt{Sign}, \texttt{Verify}_{\mathsf{SoK}})$. It saves all algorithms and parameters except $\texttt{Delete}$ in St_{gen}.
4. It sets $\mathsf{reg.par} := (\mathbf{V}_{\mathsf{elig}}, t_{\mathsf{cast}}, t_{\mathsf{open}}, \boldsymbol{t}, St_{\mathsf{gen}})$.
5. It sends $((\mathsf{SA}, \mathsf{sid}), \textsc{Setup})$ to $\mathcal{F}^{\mathsf{SA}}_{\mathsf{cert}}$, waits for confirmation and then sends $((\mathsf{SA}, \mathsf{sid}), \textsc{Sign}, (\mathsf{SA}, \mathsf{reg.par}))$ back to receive $((\mathsf{SA}, \mathsf{sid}), \textsc{Signature}, (\mathsf{SA}, \mathsf{reg.par}), s)$. Then it sends $(\mathsf{sid}, \textsc{Broadcast}, (\mathsf{SA}, \mathsf{reg.par}, s))$ to $\mathcal{F}_{\mathsf{an.BC}}$ [a].
6. It returns $(\mathsf{sid}, \textsc{Elig\_Par}, \mathsf{reg.par})$ to $\mathcal{Z}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Broadcast}, (\mathsf{SA}, \mathsf{reg.par}, s))$ from $\mathcal{F}_{\mathsf{an.BC}}$, V sends $((\mathsf{SA}, \mathsf{sid}), \textsc{Verify}, (\mathsf{SA}, \mathsf{reg.par}), s)$ to $\mathcal{F}^{\mathsf{SA}}_{\mathsf{cert}}$. If it returns 1, V stores $\mathsf{reg.par}$ and sets status to 'credential'.

[a] A simple anonymous channel is enough as at that point the voter needs to state its identity. In order to keep the format simple, we use only the anonymous broadcast functionality in our protocol (instead of both simple and anonymous broadcast) but we leak the identity of the voter at that point.

■ Upon receiving $(\mathsf{sid}, \textsc{Gen\_Cred})$ from $\mathcal{Z}$, if V's status=credential, V reads CI from $\mathcal{G}_{\mathsf{clock}}$. If $\mathtt{Status}(\mathsf{CI}, \boldsymbol{t}, \mathsf{Cast}) = \top$, V sets status to 'cast'. Otherwise, if $\mathtt{Status}(\mathsf{CI}, \boldsymbol{t}, \mathsf{Cred}) = \top$, then V does:

1. She computes $S \xleftarrow{\$} \mathcal{M}$ and sends $(\mathsf{sid}, \textsc{Comm}, S)$ to $\mathcal{F}_{\mathsf{NIC}}$ to get $(\mathsf{sid}, \textsc{Comm}, c, r)$. If $c \notin \mathcal{X}_\lambda$, she repeats this step until it does.
2. She sends $((V, \mathsf{sid}), \textsc{Setup})$ to $\mathcal{F}_{\mathsf{cert}}^{V}$, waits for confirmation and then sends $((V, \mathsf{sid}), \textsc{Sign}, (V, c))$ back to receive $((V, \mathsf{sid}), \textsc{Signature}, (V, c), s)$. Then she sends $(\mathsf{sid}, \textsc{Broadcast}, (V, (c, s)))$ to $\mathcal{F}_{\mathsf{an.BC}}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Broadcast}, (V', (c', s')))$ from $\mathcal{F}_{\mathsf{an.BC}}$, V sends $((V', \mathsf{sid}), \textsc{Verify}, (V', c'), s')$ to $\mathcal{F}_{\mathsf{cert}}^{V'}$. If it returns 1, she adds c' to C and returns $(\mathsf{sid}, \textsc{Gen\_Cred}, V', c')$ to $\mathcal{Z}$. ■ Upon receiving $(\mathsf{sid}, \textsc{Auth\_Ballot}, v)$ from $\mathcal{Z}$, if V's status=cast, she reads CI from $\mathcal{G}_{\mathsf{clock}}$. If $\mathtt{Status}(\mathsf{CI}, \boldsymbol{t}, \mathsf{Open}) = \top$, she sets its status to 'open'. Otherwise, if $\mathtt{Status}(\mathsf{CI}, \boldsymbol{t}, \mathsf{Cast}) = \top$, then V does:

1. She sends $(\mathsf{sid}, \textsc{Accum}, u, C\backslash\{c\})$ to $\mathcal{F}_{\mathsf{acc}}^{*}$ to get $(\mathsf{sid}, \textsc{Accum}, w)$, and $(\mathsf{sid}, \textsc{Accum}, w, \{c\})$ to get $(\mathsf{sid}, \textsc{Accum}, a)$.
2. She sends $(\mathsf{sid}, \textsc{Sign}, v, (a, S), (c, w, r))$ to $\mathcal{F}_{\mathsf{SoK}}$ to get $(\mathsf{sid}, \textsc{Sign}, v, (a, S), \phi)$. She sets $\sigma := (\phi, S)$.
3. She returns $(\mathsf{sid}, \textsc{Auth\_Ballot}, v, \sigma)$ to $\mathcal{Z}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Ver\_Ballot}, v, \sigma)$ from $\mathcal{Z}$, V:

1. She parses σ as (ϕ, S) and sends $(\mathsf{sid}, \textsc{Accum}, u, C)$ to $\mathcal{F}_{\mathsf{acc}}^{*}$ to get $(\mathsf{sid}, \textsc{Accum}, a)$ and $(\mathsf{sid}, \textsc{Verify}, v, (a, S), \phi)$ to $\mathcal{F}_{\mathsf{SoK}}$ to get $(\mathsf{sid}, \textsc{Verified}, x)$.
2. She returns $(\mathsf{sid}, \textsc{Ver\_Ballot}, v, \sigma, x)$ to $\mathcal{Z}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Link\_Ballots}, (v, \sigma), (v', \sigma'))$ from $\mathcal{Z}$, V:

1. She parses σ as (ϕ, S) and σ' as (ϕ', S'). If $S = S'$, She sets $x = 1$, otherwise $x = 0$.
2. She returns $(\mathsf{sid}, \textsc{Link\_Ballots}, (v, \sigma), (v', \sigma'), x)$ to $\mathcal{Z}$.

Figure 5.9: Definition of Π_{elig} in the $\{\mathcal{F}_{\mathsf{SoK}}(\mathcal{F}_{\mathsf{acc}}^{*}, \mathcal{F}_{\mathsf{NIC}}), \mathcal{F}_{\mathsf{an.BC}}, \{\mathcal{F}_{\mathsf{cert}}^{P}\}, \mathcal{G}_{\mathsf{clock}}\}$-hybrid model in the presence of $\mathbf{V}$ and SA.

Π_{elig} also uses the functionalities for anonymous broadcast, the global clock and the certification functionality instantiated for SA and all V which we denote by $\{\mathcal{F}_{\mathsf{cert}}^{P}\}$, where $P \in \{V, \mathsf{SA}\}$. Note that when we write $\mathcal{F}_{\mathsf{cert}}^{P}$ we mean $\mathcal{F}_{\mathsf{cert}}(P, \mathbf{V} \cup \mathsf{SA})$.

Remark 2. *Note that in the definitions of $\mathcal{F}_{\mathsf{elig}}$ and Π_{elig}, we implicitly assume that the commands with which the environment can activate the corrupted parties*

are the same as for the honest parties. However, for corrupted parties, $\mathcal{F}_{\mathsf{elig}}$ will first reach out to the simulator to request their internal state, which the simulator will provide (this part is captured explicitly), and then $\mathcal{F}_{\mathsf{elig}}$ will output as usual based on what $\mathcal{S}$ provides.

Theorem 7. Π_{elig} *UC-realizes* $\mathcal{F}_{\mathsf{elig}}$ *in the* $\{\mathcal{F}_{\mathsf{SoK}}(\mathcal{F}^{*}_{\mathsf{acc}}, \mathcal{F}_{\mathsf{NIC}}), \mathcal{F}_{\mathsf{an.BC}}, \{\mathcal{F}^{P}_{\mathsf{cert}}\}, \mathcal{G}_{\mathsf{clock}}\}$*-hybrid model.*

5.5 Realizing $\mathcal{F}^{\mathsf{delay}}_{\mathsf{vm}}$ via time-lock puzzles

In this section we construct a real-world protocol that UC-realizes the vote management functionality $\mathcal{F}_{\mathsf{vm}}$ (cf. Subsection 5.2.2) via $\mathcal{F}_{\mathsf{TLE}}$.

We present the description of the protocol $\Pi^{\mathcal{F}_{\mathsf{TLE}},\{\mathcal{F}^{P}_{\mathsf{cert}}\},\mathcal{F}_{\mathsf{an.BC}},\mathcal{G}_{\mathsf{clock}}}_{\mathsf{vm}}$ and we provide a proof that it UC-realizes $\mathcal{F}_{\mathsf{vm}}$ in the $\{\mathcal{F}_{\mathsf{TLE}}, \{\mathcal{F}^{P}_{\mathsf{cert}}\}, \mathcal{F}_{\mathsf{an.BC}}, \mathcal{G}_{\mathsf{clock}}\}$-hybrid model. The description of $\Pi^{\mathcal{F}_{\mathsf{TLE}},\{\mathcal{F}^{P}_{\mathsf{cert}}\},\mathcal{F}_{\mathsf{an.BC}},\mathcal{G}_{\mathsf{clock}}}_{\mathsf{vm}}$ shown in Figure 5.10 follows the phases and the command interface of $\mathcal{F}_{\mathsf{vm}}$ in Subsection 5.2.2.

It is worth mentioning that we use the same $\mathcal{F}_{\mathsf{TLE}}$ introduced in the previous chapter, except that the simulator does not learn the identity of the party that encrypts a message. This small tweak is necessary to guaranty the unlinkability between the ballot and the voter, and thus realizing $\mathcal{F}_{\mathsf{vm}}$. On the other hand, this was not necessary to capture the general concept of TLE, that is why we did not include it as a leakage to the simulator. Despite that, the proof of realization of this "new" $\mathcal{F}_{\mathsf{TLE}}$ is the same as in the previous chapter except that, in the hybrid protocol, instead of using a plain broadcast channel we need to use an anonymous one. The proof is obvious and thus omitted. For the rest of this section, we refer to $\mathcal{F}_{\mathsf{TLE}}$ as the one described before.

Definition 5.5.1. Let leak be a leakage function and $t_{\mathsf{cast}}, t_{\mathsf{open}}$ be time points. We say that the pair of functions $\mathtt{Status}_{\mathsf{leak}}, \mathtt{define\_time}_{\mathsf{leak}}$ with respect to a leakage function leak is `phase preserving` if there is no time s.t. the **Cast** and **Tally** phases are simultaneously active. Formally, $\forall \mathsf{CI}$ s.t $\mathtt{Status}_{\mathsf{leak}}(\mathsf{CI}, \boldsymbol{t}, \mathsf{Cast}) = \top \Rightarrow \mathtt{Status}_{\mathsf{leak}}(\mathsf{leak}(\mathsf{CI}), \boldsymbol{t}, \mathsf{Open}) = \bot$, where $\boldsymbol{t} \leftarrow \mathtt{define\_time}_{\mathsf{leak}}(t_{\mathsf{cast}}, t_{\mathsf{open}})$.

5.5.1 A protocol $\Pi_{\mathsf{vm}}^{\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}}$ that realizes $\mathcal{F}_{\mathsf{vm}}^{\mathsf{delay}}$

The vote management protocol $\Pi_{\mathsf{vm}}^{\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}},\mathcal{F}_{\mathsf{an.BC}},\{\mathcal{F}_{\mathsf{cert}}^{i}\},\mathcal{G}_{\mathsf{clock}}}$.

Each voter V maintains a list of the cast ballots L_{cast}^{V} initially as empty. Each voter initializes its status to 'exec'.

■ Upon receiving $(\mathsf{sid}, \textsc{Setup\_Info}, t_{\mathsf{cast}}, t_{\mathsf{open}}, \mathbf{V}_{\mathsf{elig}})$ for the first time from $\mathcal{Z}$, SA sends $((\mathsf{SA}, \mathsf{sid}), \textsc{Setup})$ to $\mathcal{F}_{\mathsf{cert}}^{\mathsf{SA}}$, and waits to receive $((\mathsf{SA}, \mathsf{sid}), \textsc{Setup})$ from $\mathcal{F}_{\mathsf{cert}}^{\mathsf{SA}}$. Then SA computes $\boldsymbol{t} \leftarrow \mathtt{define\_time}_{\mathsf{leak}}(t_{\mathsf{cast}}, t_{\mathsf{open}})$ and if $\boldsymbol{t} \neq \perp$ it sends $((\mathsf{SA}, \mathsf{sid}), \textsc{Sign}, (\mathsf{SA}, t_{\mathsf{cast}}, t_{\mathsf{open}}, \boldsymbol{t}, \mathbf{V}_{\mathsf{elig}}))$ to $\mathcal{F}_{\mathsf{cert}}^{\mathsf{SA}}$. Upon receiving $((\mathsf{SA}, \mathsf{sid}), \textsc{Signature}, (\mathsf{SA}, t_{\mathsf{cast}}, t_{\mathsf{open}}, \boldsymbol{t}, \mathbf{V}_{\mathsf{elig}}), s)$ from $\mathcal{F}_{\mathsf{cert}}^{\mathsf{SA}}$, it sends $(\mathsf{sid}, \textsc{Broadcast}, (\mathsf{SA}, t_{\mathsf{cast}}, t_{\mathsf{open}}, \boldsymbol{t}, \mathbf{V}_{\mathsf{elig}}, s))$ to $\mathcal{F}_{\mathsf{an.BC}}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Broadcast}, (\mathsf{SA}, t_{\mathsf{cast}}, t_{\mathsf{open}}, \boldsymbol{t}, \mathbf{V}_{\mathsf{elig}}, s))$ from $\mathcal{F}_{\mathsf{an.BC}}$, V sends $((\mathsf{SA}, \mathsf{sid}), \textsc{Verify}, (\mathsf{SA}, t_{\mathsf{cast}}, t_{\mathsf{open}}, \boldsymbol{t}, \mathbf{V}_{\mathsf{elig}}), s)$ to $\mathcal{F}_{\mathsf{cert}}^{\mathsf{SA}}$. If it returns 1, she stores $(t_{\mathsf{cast}}, t_{\mathsf{open}}, \boldsymbol{t}, \mathbf{V}_{\mathsf{elig}})$.

■ Upon receiving $(\mathsf{sid}, \textsc{Gen\_Ballot}, o)$ from $\mathcal{Z}$, V does:

1. If this is the first time receiving this command-message and $V \in \mathbf{V}_{\mathsf{elig}}$, V sends to $\mathcal{F}_{\mathsf{TLE}}$ $(\mathsf{sid}, \textsc{Enc}, o, t_{\mathsf{open}})$. Upon receiving $(\mathsf{sid}, \textsc{Encrypting})$ from $\mathcal{F}_{\mathsf{TLE}}$, V returns $(\mathsf{sid}, \textsc{Generating})$ to $\mathcal{Z}$.
2. Else, V returns to $\mathcal{Z}$ $(\mathsf{sid}, \textsc{Gen\_Ballot}, o, \perp)$.

■ Upon receiving $(\mathsf{sid}, \textsc{Retrieve})$ from $\mathcal{Z}$, V sends $(\mathsf{sid}, \textsc{Retrieve})$ to $\mathcal{F}_{\mathsf{TLE}}$. Upon receiving $(\mathsf{sid}, \textsc{Retrieve}, (o, v, t_{\mathsf{open}}))$ from $\mathcal{F}_{\mathsf{TLE}}$, she records the tuple $(V, v, o, 1)$ and sends $(\mathsf{sid}, \textsc{Retrieve}, (o, v))$ to $\mathcal{Z}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Cast}, v, \sigma)$ from $\mathcal{Z}$, if her status is 'exec', V reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. If $\mathtt{Status}_{\mathsf{leak}}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Open}) = \top$, V sets the status to 'open'. Otherwise V does:

1. She sends $(\mathsf{sid}, \textsc{Retrieve})$ to $\mathcal{F}_{\mathsf{TLE}}$. Upon receiving $(\mathsf{sid}, \textsc{Retrieve}, (o', v', t_{\mathsf{open}}))$ she records the tuple $(V, v', o', 1)$.
2. If there is a tuple of the form $(V, v, \cdot, 1)$ stored and it is the first time receiving this command-message, V sends $(\mathsf{sid}, \textsc{Cast}, v, \sigma)$ to $\mathcal{F}_{\mathsf{an.BC}}$. Upon receiving $(\mathsf{sid}, \textsc{Cast}, v, \sigma)$ from $\mathcal{F}_{\mathsf{an.BC}}$, V^* stores the tuple (v, σ) to $L^{V^*}_{\mathsf{cast}}$.
3. Else, V returns $(\mathsf{sid}, \textsc{Cast}, v, \sigma, \perp)$ to $\mathcal{Z}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Open}, v^*)$ from $\mathcal{Z}$, if there is a tuple $(v^*, \sigma^*) \in L^{V}_{\mathsf{cast}}$, V sends $(\mathsf{sid}, \textsc{Dec}, v^*, t_{\mathsf{open}})$ to $\mathcal{F}_{\mathsf{TLE}}$.

1. Upon receiving $(\mathsf{sid}, \textsc{Dec}, v^*, t_{\mathsf{open}}, o^*)$ from $\mathcal{F}_{\mathsf{TLE}}$, V returns the message $(\mathsf{sid}, \textsc{Open}, v^*, o^*)$ to $\mathcal{Z}$.
2. Upon receiving $(\mathsf{sid}, \textsc{Dec}, v^*, t_{\mathsf{open}}, \perp)$ from $\mathcal{F}_{\mathsf{TLE}}$, V returns the message $(\mathsf{sid}, \textsc{Open}, v^*, \perp)$ to $\mathcal{Z}$.

Figure 5.10: Description of Π_{vm} in the $\{\mathcal{F}^{\mathsf{leak}}_{\mathsf{TLE}}, \mathcal{F}_{\mathsf{an.BC}}, \{\mathcal{F}^{P}_{\mathsf{cert}}\}, \mathcal{G}_{\mathsf{clock}}\}$-hybrid model in the presence of $\mathbf{V}$ and SA.

Theorem 8. Π_{vm} *UC-realizes* $\mathcal{F}^{\mathsf{delay}}_{\mathsf{vm}}$ *in the* $\{\mathcal{F}^{\mathsf{leak,delay}}_{\mathsf{TLE}}, \mathcal{F}_{\mathsf{an.BC}}, \{\mathcal{F}^{P}_{\mathsf{cert}}\}, \mathcal{G}_{\mathsf{clock}}\}$*-hybrid model given that the pair of functions* $\mathtt{Status}_{\mathsf{leak}}, \mathtt{define\_time}_{\mathsf{leak}}$ *is* ***phase preserving****.*

Proof. In cases where a corrupted party receives input and we do not describe her behaviour, we assume that the message is sent to $\mathcal{S}$ from $\mathcal{F}_{\mathsf{vm}}$ and $\mathcal{S}$ forwards that message to $\mathcal{A}$ as if he was from that party. Then $\mathcal{S}$ returns to $\mathcal{F}_{\mathsf{vm}}$ whatever he receives from $\mathcal{A}$.

We describe the ideal adversary $\mathcal{S}$. When $\mathcal{S}$ receives the corruption vector from $\mathcal{Z}$, $\mathcal{S}$ forwards it to $\mathcal{A}$ as if he was from $\mathcal{Z}$. When $\mathcal{S}$ receives back the corruption vector from $\mathcal{A}$ playing the role of both of $\mathcal{F}_{\mathsf{TLE}}, \mathcal{F}^{\mathsf{SA}}_{\mathsf{cert}}$, $\mathcal{S}$ forwards it to $\mathcal{F}_{\mathsf{vm}}$. When $\mathcal{S}$ receives the setup information $(\mathsf{sid}, \textsc{Setup\_Info}, t_{\mathsf{cast}}, t_{\mathsf{open}}, \mathbf{V}_{\mathsf{elig}})$

from $\mathcal{F}_{\mathsf{vm}}$, $\mathcal{S}$ sends $((\mathsf{SA}, \mathsf{sid}), \textsc{Setup})$ to $\mathcal{A}$ as if he was $\mathcal{F}_{\mathsf{cert}}^{\mathsf{SA}}$. Upon receiving $((\mathsf{SA}, \mathsf{sid}), \textsc{Algorithms}, \mathtt{Verify}, \mathtt{Sign})$ from $\mathcal{A}$ playing the role of $\mathcal{F}_{\mathsf{cert}}^{\mathsf{SA}}$, $\mathcal{S}$ stores the algorithms $(\mathtt{Verify}, \mathtt{Sign})$ and produces the signature $\sigma \leftarrow \mathtt{Sign}(t_{\mathsf{cast}}, t_{\mathsf{open}}, \mathbf{V}_{\mathsf{elig}})$. Then $\mathcal{S}$ asks $\mathcal{A}$ if he allows the cast of the message $(t_{\mathsf{cast}}, t_{\mathsf{open}}, \mathbf{V}_{\mathsf{elig}}, \sigma)$ as if he was $\mathcal{F}_{\mathsf{an.BC}}$.

Then, $\mathcal{S}$ responds to $\mathcal{F}_{\mathsf{vm}}$ according to the answer of $\mathcal{A}$.

Upon receiving a $\textsc{Gen\_Ballot}$ request from $\mathcal{F}_{\mathsf{vm}}$ on behalf of a voter V, $\mathcal{S}$ forwards the message to $\mathcal{A}$ as if it was from $\mathcal{F}_{\mathsf{TLE}}$ and he returns the response of $\mathcal{A}$ to $\mathcal{F}_{\mathsf{vm}}$. Upon receiving an $\textsc{Advance\_Clock}$ command from $\mathcal{G}_{\mathsf{clock}}$ on behalf of a voter V, $\mathcal{S}$ forwards the message to $\mathcal{A}$ as if he was $\mathcal{G}_{\mathsf{clock}}$. Upon receiving an $\textsc{Update}$ command as if he was $\mathcal{F}_{\mathsf{TLE}}$ from $\mathcal{A}$, he forwards the message to $\mathcal{F}_{\mathsf{vm}}$. Upon receiving $(\mathsf{sid}, \textsc{Allow\_Cast}, v, \sigma)$ from $\mathcal{F}_{\mathsf{vm}}$, $\mathcal{S}$ asks $\mathcal{A}$ if he allows the broadcast of (v, σ) as if it was from $\mathcal{F}_{\mathsf{an.BC}}$. If the broadcast is allowed, $\mathcal{S}$ sends $(\mathsf{sid}, \textsc{Cast\_Allowed})$ to $\mathcal{F}_{\mathsf{vm}}$. When $\mathcal{S}$ receives a $\textsc{Cast}$ request from $\mathcal{F}_{\mathsf{vm}}$ on behalf of a corrupted voter V, he forwards the message to $\mathcal{A}$ as if it was from $\mathcal{F}_{\mathsf{an.BC}}$ and he returns the message he received from $\mathcal{A}$ back to $\mathcal{F}_{\mathsf{vm}}$.

Upon receiving $(\mathsf{sid}, \textsc{Open}, v)$ from $\mathcal{F}_{\mathsf{vm}}$ (where v is a ballot not generated by $\mathcal{F}_{\mathsf{vm}}$), $\mathcal{S}$ sends $(\mathsf{sid}, \textsc{Dec}, v, t_{\mathsf{open}})$ to $\mathcal{A}$ as if it was from $\mathcal{F}_{\mathsf{TLE}}$. When $\mathcal{S}$ receives $(\mathsf{sid}, \textsc{Dec}, v, t_{\mathsf{open}}, o)$ from $\mathcal{A}$, he returns the message $(\mathsf{sid}, \textsc{Open}, v, o)$ to $\mathcal{F}_{\mathsf{TLE}}$.

Upon receiving $(\mathsf{sid}, \textsc{Leakage})$ from $\mathcal{Z}$, $\mathcal{S}$ forwards the message to $\mathcal{A}$ as if it was from $\mathcal{Z}$. Upon receiving $(\mathsf{sid}, \textsc{Leakage})$ from $\mathcal{A}$, $\mathcal{S}$ reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. Then $\mathcal{S}$ playing the role of $\mathcal{F}_{\mathsf{TLE}}$ returns to $\mathcal{A}$ all the maliciously generated ciphertexts with time labelling until time $\mathsf{leak}(\mathsf{Cl})$. If $\mathtt{Status}_{\mathsf{leak}}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Open}) = \mathtt{Status}_{\mathsf{leak}}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cast}) = \mathtt{Status}_{\mathsf{leak}}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cred}) = \bot$, then $\mathcal{S}$ can request and give also the honest generated ciphertexts from $\mathcal{F}_{\mathsf{vm}}$ and returns them to $\mathcal{A}$ as if there were from from $\mathcal{F}_{\mathsf{TLE}}$. Note that all honest parties will use $\mathcal{F}_{\mathsf{TLE}}$ with the same time labelling for encryption requests and at most once. The only way simulation fails is when the $\mathtt{Status}_{\mathsf{leak}}(\mathsf{Cl}, \boldsymbol{t}, \mathsf{Cast}) = \top$ and $\mathtt{Status}_{\mathsf{leak}}(\mathsf{leak}(\mathsf{Cl}), \boldsymbol{t}, \mathsf{Open}) = \top$ because in that case $\mathcal{S}$ can not retrieve the honestly generated plaintexts from $\mathcal{F}_{\mathsf{vm}}$ so he cannot give a response on behalf of $\mathcal{F}_{\mathsf{TLE}}$ to $\mathcal{A}$ on a $(\mathsf{sid}, \textsc{Leakage})$ command. By the definition of $\mathtt{Status}_{\mathsf{leak}}$ and $\mathtt{define\_time}_{\mathsf{leak}}$ this is impossible to happen.

The distribution of messages is the same in both the ideal and the hybrid setting, as the algorithms $\mathtt{GenBallot} \equiv e_{\mathcal{A}}$ and $\mathtt{OpenBallot} \equiv d_{\mathcal{A}}$ that are used are the same. As a result, the simulation is perfect. □

Based on all previous Sections in this Chapter, we can state the following corollary as can be inferred from Figure 5.1.

Corollary 8.1. *The protocol* E-CCLESIA 1.0 *, which is defined as the union of the protocols* $\Pi_{\mathsf{elig}}(\mathcal{F}_{\mathsf{Crs}}^{\mathsf{Gen}}, \mathcal{F}_{\mathsf{an.BC}}, \mathcal{F}_{\mathsf{cert}}^{\mathsf{P}}, \mathcal{F}_{\mathsf{SOK}}, \mathcal{F}_{\mathsf{NIC}})$ *and* $\Pi_{\mathsf{vm}}(\mathcal{F}_{\mathsf{an.BC}}, \mathcal{F}_{\mathsf{cert}}^{\mathsf{P}}, \mathcal{W}_q^*(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}), \mathcal{F}_{\mathsf{RO}})$,

UC realizes $\mathcal{F}_{\mathsf{STE}}$ *in the* $(\mathcal{F}_{\mathsf{Crs}}^{\mathsf{Gen}}, \mathcal{F}_{\mathsf{an.BC}}, \mathcal{F}_{\mathsf{cert}}^{\mathsf{P}}, \mathcal{F}_{\mathsf{SOK}}, \mathcal{F}_{\mathsf{NIC}}, \mathcal{W}_q^*(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}), \mathcal{F}_{\mathsf{RO}})$ *hybrid model* [2].

[2]Note that there exist a UC realization of the functionalities $(\mathcal{F}_{\mathsf{SOK}}, \mathcal{F}_{\mathsf{NIC}}, \mathcal{F}_{\mathsf{cert}}^{\mathsf{P}})$ in the literature as we mentioned above.

Chapter 6

Quantum e-voting & limitations

Voting is fundamental in democratic societies. With the technological advances of the computer era, voting could benefit to become more secure and efficient and as a result more democratic. For this reason, over the last two decades, several cryptographic protocols for electronic voting were proposed and implemented, see e.g. Adida (2008); Chaum et al. (2009); Juels et al. (2005); Kiayias et al. (2015); Ryan and Schneider (2006); Cortier et al. (2019). The security of all these systems relies on computational assumptions such as the hardness of integer factorization and the discrete logarithm problem. But, these are easy to solve with quantum computers using Shor's algorithm Shor (1994). Although not yet available, recent technological advances indicate that quantum computers will soon be threatening existing cryptographic protocols. In this context, researchers have proposed to use quantum communication to implement primitives like key distribution, bit commitment and oblivious transfer. Unfortunately, perfect security without assumptions has proven to be challenging in the quantum setting Lo and Chau (1998); Mayers (1997), and the need to study different corruption models has emerged. This includes limiting the number of dishonest parties and introducing different non-colluding authorities.

More than a decade of studies on quantum electronic voting has resulted in several protocols that use the properties of quantum mechanical systems. However, all these new protocols are studied against different and not well-articulated corruption models and claim security using ad-hoc proofs that are not formalized and backed only against limited classes of quantum attacks. In particular, none of the proposed schemes provides rigorous definitions of `privacy` and `verifiability`, nor formal security proofs against specific, well defined (quantum) attacker models. When it comes to electronic voting schemes, it is particularly hard to ensure that all the, somehow conflicting, properties hold Chevallier-Mames et al. (2010); it is therefore important that these new quantum protocols be rigorously and mathematically studied and the necessary assumptions and limitations formally established.

This is precisely what we set to address in this work. We systematize and assess the security of existing e-voting protocols based on quantum technology.

	Privacy	Correctness	Verifiability	Corruption
Dual basis (Section 6.1)	✗	?	?	ϵ fraction of voters
Travelling ballot (Section 6.2)	✗	✗	✗	two voters
Distributed ballot (Section 6.3)	?	✗*	✗	ϵ fraction of voters
Conjugate coding (Section 6.4)	✗	?	?	election authority

Table 6.1: Properties of the different categories of quantum e-voting protocols. ✗: Insecure, **?**: Unexplored Area, *:Protocol runs less than $exp(\Omega(N))$ rounds.

Unfortunately, our analyses uncover vulnerabilities in all the proposed schemes. While some of them suffer from trivial attacks due to inconsistencies in the security definitions, the main contribution of the paper is to argue that sophisticated attacks can exist even in protocols that "seem secure" if the security is proven ad hoc, and not in a formal framework. We argue that the cause of these failures is the absence of an appropriate security framework.

Therefore, this work follows previous works Barnum et al. (2002); Portmann (2017); Unruh (2010); Moran and Naor (2006); Gallegos-Garcia et al. (2016) in their effort to highlight the importance of formally defining and proving security in the relatively new field of quantum cryptography. This also includes studying classical protocols that are secure against unbounded attackers Broadbent and Tapp (2007), as well as ones based on problems believed to be hard even for quantum computers e.g. lattice-based Chillotti et al. (2016). However, it is out of the scope of this study to review such classical protocols, as we are focusing on the possible contribution of quantum computers to the security of e-voting.

We systematize the proposed quantum e-voting approaches according to key technical features. To our knowledge, our study covers all relevant research in the field, identifying four main families. Table 6.1 summarises our results.

- *Two measurement bases protocols* - These protocols rely on two measurement bases to verify the correct distribution of an entangled state. We specifically prove that the probability that a number of corrupted states are not tested and used later in the protocol, is non-negligible, which leads to a violation of voters' privacy. Furthermore, even if the states are shared by a trusted authority, we show that `privacy` can still be violated in case of abort.

- *Traveling ballot protocols* - In these protocols the "ballot box" circulates among all voters who add their vote by applying a unitary to it. We show how colluding voters can break honest voters' vote privacy just by measuring the ballot box before and after the victim has cast their ballot. These protocols further suffer as we will see from double voting attacks, whereby a dishonest voter can simply apply multiple times the voting operator.

- *Distributed ballot protocols* - These schemes exploit properties of entangled states that allow voters to cast their votes by applying operations on parts

of them. We present an attack that allows the adversary to double-vote and therefore changes the outcome of the voting process with a probability of at least 0.25 if the protocol runs fewer than exponentially many rounds in the number of voters. The intuition behind this attack is that an adversary does not need to find exactly how the ballots have been created to influence the outcome of the election; it suffices to find a specific relation between them from left-over voting ballots provided by the corrupted voters.

- *Conjugate coding protocols* - These protocols exploit BB84 states adding some verification mechanism. The main issue with these schemes, as we show, is that ballots are malleable, allowing an attacker to modify the part of the ballot which encodes the candidate choice to their advantage.

In this work, we will be dealing with protocols involving one election authority EA, a set of voters $\mathcal{V} = \{V_k\}_{k=1}^{N}$ and a set of talliers T (which may overlap with $\mathcal{V}$). EA sets the parameters of the protocol, each voter $V_k \in \mathcal{V}$ casts vote v_k and T gathers the ballots, computes the election outcome and announces it. Informally, a voting protocol Π has three distinct phases (*setup*, *casting*, and *tally*) and running time proportional to a security parameter δ_0.

6.1 Dual basis measurement based protocols

In this section, we discuss protocols that use the dual basis measurement technique Huang et al. (2014); Wang et al. (2016), and use as a blank ballot an entangled state with an interesting property: when measured in the computational basis, the sum of the outcomes is equal to zero, while when measured in the Fourier basis, all the outcomes are equal. Both of these protocols use cut-and-choose techniques to verify that the state was distributed correctly. This means that a large number of states are checked for correctness and a remaining few are kept at the end unmeasured, to proceed with the rest of the protocol. Although a cut-and-choose technique with just one verifying party is secure if the states that are sampled are exponentially many and the remaining ones are constant, it is not clear how this generalizes to multiple verifying parties. Specifically, we show that if the corrupted parties sample their states last, then the probability with which the corrupted states are not checked and remain after all the honest parties sample is at least a constant to the security parameter of the protocol.

6.1.1 Protocol specification

We will now present the self-tallying protocol of Wang et al. (2016), which is based on the classical protocol of Kiayias and Yung (2002). The voters $\{V_k\}_{k=1}^{N}$, without the presence of any trusted authority or tallier, need to verify that they share specific quantum states. At the end of the verification process, the voters

share a classical matrix; every cast vote is equal to the sum of the elements of a row in the matrix.

`Setup` phase:

1. One of the voters, not necessarily trusted, prepares $N + N2^{\delta_0}$ states:

$$|D_1\rangle = \frac{1}{\sqrt{c^{N-1}}} \sum_{\sum_{k=1}^{N} i_k = 0 \mod c} |i_1\rangle |i_2\rangle \dots |i_N\rangle$$

   where m is the dimension of the qudits' Hilbert space, c is the number of the possible candidates such that $m \geq c$ and δ_0 the security parameter. The voter also shares $1 + N2^{\delta_0}$ states of the form:

$$|D_2\rangle = \frac{1}{\sqrt{N!}} \sum_{(i_1, i_2, \dots, i_N) \in \mathcal{P}_N} |i_1\rangle |i_2\rangle \dots |i_N\rangle$$

   where $\mathcal{P}_N$ is the set of all possible permutations with N elements. Each V_k receives the k^{th} particle from each of the states.

2. The voters agree that the states they receive are truly $|D_1\rangle, |D_2\rangle$ by using a cut-and choose technique. Specifically, voter V_k chooses at random 2^{δ_0} of the $|D_1\rangle$ states and asks the other voters to measure half of their particles in the computational and half in the Fourier basis. Whenever the chosen basis is computational, the measurement results need to add up to 0, while when the basis is the Fourier, then the measurement results are all the same. All voters simultaneously broadcast their results and if one of them notices a discrepancy, the protocol aborts. The states $|D_2\rangle$ are similarly checked. Specifically, if measured either in the computational or the Fourier base all the values should be different and in fact, constitute a random permutation in $\mathcal{P}_N$.

3. The voters are left to share N copies of $|D_1\rangle$ states and one $|D_2\rangle$ state. Each voter holds one qudit for each state. They now all measure their qudits in the computational basis. As a result, each V_k holds a "blank ballot" of dimension N with the measurement outcomes corresponding to parts of $|D_1\rangle$ states:

$$B_k = [\xi_k^1 \cdots \xi_k^{sk_k} \cdots \xi_k^N]^\intercal$$

   and a unique index, $sk_k \in \{1, \dots, N\}$, from the measurement outcome of the qudit that belongs to $|D_2\rangle$. The set of all the blank ballots has the property $\sum_{k=1}^{N} \xi_k^j = 0 \mod c$ for all $j = 1, \dots, N$.

`Casting` phase:

4. Based on sk_k, all voters add their vote, $v_k \in \mathbb{Z}_c$, to the corresponding row of their "secret" column. Specifically, V_k applies $\xi_k^{sk_k} \to \xi_k^{sk_k} + v_k$.

5. All voters simultaneously broadcast their columns, resulting in a public $N \times N$ table, whose k-th column encodes V_k's candidate choice.

$$B = \begin{bmatrix} & & \xi_k^1 & & \\ & & \vdots & & \\ B_1^{v_1} & \cdots & \xi_k^{sk_k} + v_k & \cdots & B_N^{v_N} \\ & & \vdots & & \\ & & \xi_k^N & & \end{bmatrix}$$

`Tally` phase:

6. Each V_k verifies that their vote is counted by checking that the corresponding row of the matrix adds up to their vote. If this fails, the protocol aborts.
7. Each voter can tally the outcome of the election by computing the sum of the elements of each row of the public $N \times N$ table. The resulting N elements are the result of the election.

6.1.2 Vulnerabilities of dual basis measurement Protocols

In this section, we present an attack on the cut-and-choose technique of the protocol in the `setup` phase, that can be used to violate `privacy`. We consider a static adversary that corrupts t voters, including the one that distributes the states. Suppose that the adversary corrupts N out of $N + N2^{\delta_0}$ states $|D_1\rangle$. We denote with *Bad*, the event that all the corrupted voters choose last which states they want to test, and with *Win*, the event that the N corrupted states are not checked. We want to compute the probability that event *Win* happens, given event *Bad*, *i.e.* the probability none of the N corrupted states is checked by the honest voters, and therefore remain intact until the corrupted voters' turn. The corrupted voters will of course not sample any of the corrupted states and therefore the corrupted states will be accepted as valid.

The number of corrupted states that an honest voter will check, follows a mixture distribution with each mixture component being one of the hypergeometric distributions $\{\mathsf{HG}(L_{i_k}, b_{i_k}, 2^{\delta_0}) : 0 \leq b_{i_k} \leq N\}$, where L_{i_k} is the number of states left to sample from the previous voter and b_{i_k} the number of the remaining corrupted states. We can therefore define the random variable X_{i_k} that follows the above mixture distribution, where $i_1, \ldots, i_{N-t}$ is a permutation of the honest voters' indices (by slightly abusing notation, we consider the first $N - t$ voters to be honest). The following lemma is proven by induction:

Lemma 9. *Let X_{i_k} be a random variable that follows the previous mixture distribution. Then,*

$$\Pr[\sum_{k=1}^{N-t} X_{i_k} = 0] = \prod_{k=1}^{N-t} \Pr[X_{i_k}^* = 0] \text{ where } X_{i_k}^* \sim \mathsf{HG}(L_{i_k}, N, 2^{\delta_0}).$$

We are now ready to prove that with at least a constant probability, the corrupted states will remain intact until the end of the verification process.

Proposition 9.1. *For $0 < \varepsilon < 1$, let $t = \varepsilon N$ be the fraction of voters controlled by the adversary. It holds that :*

$$\Pr[Win \mid Bad] > \left(\frac{\varepsilon}{2}\right)^N$$

Proof.

$$\begin{aligned}
&\Pr[Win \mid Bad] = \Pr[\sum_{k=1}^{N-t} X_{i_k} = 0] = \prod_{k=1}^{N-t} P[X^*_{i_k} = 0] \\
&= \prod_{k=0}^{N-t-1} \binom{N + N2^{\delta_0} - N - k2^{\delta_0}}{2^{\delta_0}} \Big/ \binom{N + N2^{\delta_0} - k2^{\delta_0}}{2^{\delta_0}} \\
&= \frac{(N + t2^{\delta_0} - N + 1) \cdot \ldots \cdot (N + t2^{\delta_0})}{(N + N2^{\delta_0} - N + 1) \cdot \ldots \cdot (N + N2^{\delta_0})} \\
&> \left(\frac{t2^{\delta_0} + 1}{N + N2^{\delta_0}}\right)^N = \left(\frac{t2^{\delta_0}}{N + N2^{\delta_0}} + \frac{1}{N + N2^{\delta_0}}\right)^N \\
&> \left(\frac{t2^{\delta_0}}{N + N2^{\delta_0}}\right)^N = \left(\frac{\varepsilon}{2^{-\delta_0} + 1}\right)^N > \left(\frac{\varepsilon}{2}\right)^N
\end{aligned}$$

□

The question now is with what probability event *Bad* occurs, *i.e* how likely is the fact that voters controlled by the adversary are asked to sample last? The answer is irrelevant, because this probability depends on N and t, and are both independent of δ_0. As a result,

$$\Pr[Win] > \Pr[Win \mid Bad] \Pr[Bad] = (\varepsilon/2)^N f(N, t)$$

where $f(N, t)$ is a constant function with respect to the security parameter δ_0, making $\Pr[Win]$ non-negligible in δ_0. As a matter of fact, a static adversary will corrupt the voters that maximize $\Pr[Bad]$. Therefore, we can assume that the honest voters sample the states at random, in order to not favor sets of corrupted voters. Now let us examine how this affects the privacy of the scheme.

Theorem 10. *Let $\Pi(N, t, \delta_0)$ be an execution of the self-tallying protocol with N voters, t of them corrupted, and δ_0 the security parameter. We can construct an adversary $\mathcal{A}$, which with non-negligible probability in δ_0 violates* ***privacy****.*

Proof. Let $\mathcal{C}_{\mathcal{A}}$ be the set of indices of the corrupted voters with $|\mathcal{C}_{\mathcal{A}}| = t$. Suppose the voter distributing the states is also corrupted, and prepares $1 + N2^{\delta_0}$ states of the form of $|D_2\rangle$, $N2^{\delta_0}$ states of the form $|D_1\rangle$ and N states of the form:

$$|D_{\text{Corrupt}}\rangle = |\xi_1\rangle \otimes \ldots \otimes |\xi_N\rangle$$

where $\xi_k \in_R \{0, \ldots, c-1\}$ for all $k \in \{2, \ldots, N\}$, $\xi_1 \in \{0, \ldots, c-1\}$ such that[1]:

$$\xi_1 + \ldots + \xi_N = 0 \mod c$$

From Proposition 9.1 and the previous observations we know that the probability that states $|D_{\text{Corrupt}}\rangle$ remain intact after the verification procedure in step 2 (*i.e.* event *Win*), happens with non-negligible probability in the security parameter δ_0. Therefore, with non-negligible probability, the remaining states in step 3 are: one of the form $|D_2\rangle$ and N of the form $|D_{\text{Corrupt}}\rangle$. All honest voters V_k measure their qudits in the computational basis and end up with a secret number sk_k (from measuring the corresponding part of $|D_2\rangle$) and a column

$$B_k = [\xi_k^1 \ \ldots \ \xi_k^{sk_k} \ \ldots \ \xi_k^N]^\intercal$$

(from measuring states $|D_{\text{Corrupt}}\rangle$), that is known to the adversary. Now all voters apply their vote v_k to the B_k according to sk_k. As a result:

$$B_k^{v_k} = [\xi_k^1 \ \ldots \ \xi_k^{sk_k} + v_k \ \ldots \ \xi_k^N]^\intercal$$

At this point all voters simultaneously broadcast their $B_k^{v_k}$, as the protocol specifies, and end up with the matrix $B = (B_1^{v_1} \ldots B_N^{v_N})$. Each $V_k, k \notin \mathcal{C}_{\mathcal{A}}$ checks that

$$\sum_{j=1}^{N} B[sk_k, j] = v_k \mod c$$

which happens with probability 1 from the description of the attack in the previous steps. As a result, each voter accepts the election result. The adversary knowing both the pre-vote matrix and the post-vote matrix can therefore extract the vote of all honest voters. □

A similar attack can be mounted if the adversary instead of corrupting N out of $N + N2^{\delta_0}$ $|D_1\rangle$ states, corrupts just 1 of the $|D_2\rangle$ states. The attack is similar to the one mentioned above but in this case, the adversary knows the row in which each voter voted instead of the pre-vote matrix. Moreover, the probability of theorem 9.1 is improved from $(\varepsilon/2)^N$ to $\varepsilon/2$ (the proof works similarly).

So far we have seen how voters' privacy can be violated if an adversary distributes the quantum states in the protocol. However, even if the sharing of the states is done honestly by a trusted authority, still an adversary $\mathcal{A}$ can violate the privacy of a voter. This is done by replacing one element in a column of one of the players controlled by $\mathcal{A}$ with a random number. As a result, in step, 6, the honest voter whose row doesn't pass the test, will abort the protocol by broadcasting it. $\mathcal{A}$ will therefore know the identity of the voter aborting and their corresponding vote since it knows the matrix before the modification of the column element.

[1] $\in_R$ denotes that the element is chosen uniformly at random from a specific domain.

A possible solution might be to use a classical anonymous broadcast channel, so that the voters can anonymously broadcast abort if they detect any misbehaviour at step 6. However, this might open a path to other types of attacks, like denial-of-service, and requires further study to be a viable solution.

6.2 Travelling ballot based protocols

In this section, we discuss the travelling ballot family of protocols for referendum type elections. Here, T also plays the role of EA, as it sets up the parameters of the protocol in addition to producing the election result. Specifically, it prepares two entangled qudits and sends one of them (the *ballot qudit*) to travel from voter to voter. When the voters receive the ballot qudit, they apply some unitary operation according to their vote and forward the qudit to the next voter. When all voters have voted, the ballot qudit is sent back to T who measures the whole state to compute the result of the referendum. The first quantum scheme in this category was introduced by Vaccaro *et al.* Vaccaro et al. (2007) and later improved Bonanome et al. (2011); Hillery et al. (2006); Li and Zeng (2008).

6.2.1 Protocol specification

Here we present the travelling ballot protocol of Hillery et al. (2006); an alternative form Vaccaro et al. (2007) encodes the vote in a phase factor rather than in the qudit itself.

`Setup` phase:

1. T prepares the state $|\Omega_0\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} |j\rangle_V |j\rangle_T$, keeps the second qudit and passes the first (the ballot qudit) to voter V_1.

`Casting` phase:

2. For $k = 1, \ldots, N$, V_k receives the ballot qudit and applies the unitary $U^{v_k} = \sum_{j=0}^{N-1} |j+1\rangle \langle j|$, where $v_k = 1$ signifies a "yes vote and $v_k = 0$ a "no" vote (i.e. applying the identity operator). Then, V_k forwards the ballot qudit to the next voter V_{k+1} and V_N to T.

`Tally` phase:

3. The global state held by T after all voters have voted, is:

$$|\Omega_N\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} |j+m\rangle_V |j\rangle_T$$

where m is the number of "yes" votes. T measures the two qudits in the computational basis, subtracts the two results and obtains the outcome m.

6.2.2 Vulnerabilities of travelling ballot based protocols

The first obvious weakness of this type of protocol is that they are subject to double voting. A corrupted voter can apply the "yes" unitary operation many times without being detected (this issue is addressed in the next session, where we study the distributed ballot voting schemes). As a result, we can easily construct an adversary $\mathcal{A}$ that wins in the correctness experiment described in the Appendix (Figure 6.6) with probability 1. Furthermore, these protocols are subject to privacy attacks, when several voters are colluding. In what follows, we describe such an attack on `privacy`, in the case of two colluding voters. Figure 6.1 depicts this attack.

Let us assume that the adversary corrupts voters V_{k-1} and V_{k+1} for any k. Upon receipt of the ballot qudit, instead of applying the appropriate unitary, V_{k-1} performs a measurement on the travelling ballot on a computational basis. As a result the global state becomes $|\Omega_{k-1}\rangle = |h+m\rangle_V \otimes |h\rangle_T$, where $|h+m\rangle_V$ is one of the possible eigenstates of the observable $O = \sum_{j=0}^{N-1} |j\rangle\langle j|$, and m is the number of "yes" votes cast by the voters $V_1, \ldots, V_{k-2}$ (note that V_{k-1} does not get any other information about the votes of the previous voters, except number $h+m$). Then V_{k-1} passes the ballot qudit $|h+m\rangle_V$ to V_k, who applies the respective unitary for voting "yes" or "no". As a result the ballot qudit is in the state $|h+m+v_k\rangle_V$. Next, the ballot qudit is forwarded to the corrupted voter V_{k+1}, who measures it again on the computational basis and gets the result $h+m+v_k$. $\mathcal{A}$ can now infer vote v_k from the two measurement results and figure out how V_{k+1} voted. The same attack can also be applied in the case where there are many voters between the two corrupted parties. In this case, the adversary can't learn the individual votes but only the total votes. One suggestion presented in Vaccaro et al. (2007) is to allow T to perform extra measurements to detect a malicious action during the protocol's execution. However, this only identifies an attack and does not prevent the adversary from learning some of the votes, as described above. Furthermore, the probability of detecting a deviation from the protocol is constant and as such does not depend on the security parameter and does not lead to a substantial improvement of security. It should also be noted that verifiability of the election result is not addressed in any of these works, since T is assumed to generate the initial state honestly. In the case where T is corrupted,`privacy` is trivially violated.

All travelling ballot protocols proposed (Vaccaro et al. (2007); Bonanome et al. (2011); Hillery et al. (2006); Li and Zeng (2008)) suffer from the above privacy

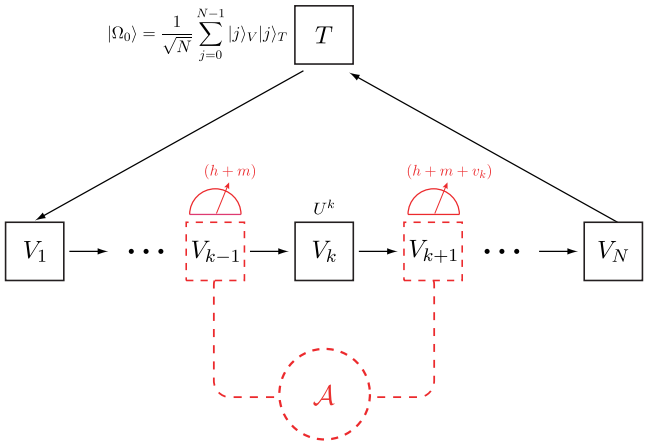

Figure 6.1: $\mathcal{A}$ corrupts voters V_{k-1},V_{k+1} and learns how voter V_k voted with probability 1.

attack. Next, we discuss how this issue has been addressed by revisiting the structure of the protocols. Unfortunately, as we will see, new issues arise.

6.3 Distributed ballot based protocols

Here we describe the family of quantum distributed ballot protocols Bonanome et al. (2011); Hillery et al. (2006); Vaccaro et al. (2007). In these schemes, T prepares and distributes to each voter a blank ballot, and gathers it back after all voters have cast their vote to compute the outcome. This type of protocol give strong guarantees for `privacy` against other voters but not against a malicious T which is trusted to prepare correctly specific states. So it is not hard to see that if the states are not the correct ones, then the privacy of a voter can be violated.

A first attempt presented in Vaccaro et al. (2007) suffers from double voting similarly to the discussion in the previous section. The same problem also appears in Dolev et al. (2006). Later works Bonanome et al. (2011); Hillery et al. (2006) address this issue with a very elaborate countermeasure. The intuition behind the proposed technique is that T chooses a secret number δ according to which it prepares two different quantum states: the "yes" and the "no" states. This δ value is hard to predict due to the non-orthogonality of the shared states and the no-cloning theorem. The authors suggest that many rounds of the protocol be executed. As a result, any attempt of the adversary to learn δ gives rise to a different result in each round. However, the number of required rounds, as well as rigorous proof are not presented in the study.

More importantly, a careful analysis reveals that the proposed solution is still vulnerable to double voting. As we will see, an adversary can mount what we call a d-transfer attack, and transfer d votes for one option of the referendum election to the other. To achieve this attack, the adversary does not need to find

the exact value of δ (as the authors believed), but knowing the difference of the angles used to create the "yes" and "no" states suffices. We construct a quantum polynomial-time adversary that performs the d-transfer attack with a probability of at least 0.25 if the number of rounds is smaller than exponential in the number of voters. As a result, this makes the protocol practically unrealistic for large scale elections.

6.3.1 Protocol specification

We first present the protocol from Bonanome et al. (2011); Hillery et al. (2006):

`Setup` phase:

1. T prepares an N-qudit ballot state: $|\Phi\rangle = \frac{1}{\sqrt{D}} \sum_{j=0}^{D-1} |j\rangle^{\otimes N}$, where the states $|j\rangle, j = 0, ..., D-1$, form an orthonormal basis for the D-dimensional Hilbert space, and $D > N$. The k-th qudit of $|\Phi\rangle$ corresponds to V_k's blank ballot.
2. T sends to V_k the corresponding blank ballot together with two option qudits, one for the "yes" and one for the "no" option:

   $$\text{yes:} \quad |\psi(\theta_y)\rangle = \frac{1}{\sqrt{D}} \sum_{j=0}^{D-1} e^{ij\theta_y} |j\rangle, \quad \text{no:} \quad |\psi(\theta_n)\rangle = \frac{1}{\sqrt{D}} \sum_{j=0}^{D-1} e^{ij\theta_n} |j\rangle$$

   For $v \in \{y, n\}$ we have $\theta_v = (2\pi l_v/D) + \delta$, where $l_v \in \{0, \ldots, D-1\}$ and $\delta \in [0, 2\pi/D)$. Values l_y and δ are chosen uniformly at random from their domain and l_n is chosen such that $N(l_y - l_n \mod D) < D$. These values are known only to T.

`Casting` phase:

3. Each V_k decides on "yes" or "no" by appending the corresponding option qudit to the blank ballot and performing a 2-qudit measurement $R = \sum_{r=0}^{D-1} rP_r$, where:

   $$P_r = \sum_{j=0}^{D-1} |j+r\rangle \langle j+r| \otimes |j\rangle \langle j|$$

   According to the result r_k, V_k performs a unitary correction $U_{r_k} = \mathbb{I} \otimes \sum_{j=0}^{D-1} |j + r_k\rangle \langle j|$ and sends the 2-qudits ballot along with r_k back to T.

`Tally` phase:

4. The global state of the system (up to normalization) is:

$$\frac{1}{\sqrt{D}} \sum_{j=0}^{D-1} \prod_{k=1}^{N} \alpha_{j,r_k} |j\rangle^{\otimes 2N}$$

where

$$\alpha_{j,r_k} = \begin{cases} e^{i(D+j-r_k)\theta_v^k}, & 0 \leq j \leq r_k - 1 \\ e^{i(j-r_k)\theta_v^k} & r_k \leq j \leq D-1 \end{cases}$$

5. For every k, using the announced results r_k, T applies the unitary operator:

$$W_k = \sum_{j=0}^{r_k-1} e^{-iD\delta} |j\rangle \langle j| + \sum_{j=r_k}^{D-1} |j\rangle \langle j|$$

on one of the qudits in the global state (it is not important on which one, since changes to the phase factor of a qudit that is part of a bigger entangled state take effect globally). Now T has the state:

$$|\Omega_m\rangle = \frac{1}{\sqrt{D}} \sum_{j=0}^{D-1} e^{ij(m\theta_y + (N-m)\theta_n)} |j\rangle^{\otimes 2N}$$

where m is the number of "yes" votes.

6. By applying the unitary operator $\sum_{j=0}^{D-1} e^{-ijN\theta_n} |j\rangle \langle j|$ on one of the qudits and setting $q = m(l_y - l_n)$, we have:

$$|\Omega_q\rangle = \frac{1}{\sqrt{D}} \sum_{j=0}^{D-1} e^{2\pi ijq/D} |j\rangle^{\otimes 2N}$$

We note here that q must be between 0 and $D-1$ so that the different outcomes be distinguishable. Now with the corresponding measurement, T can retrieve q. Since T knows values l_y and l_n, it can derive the number m of "yes" votes. Note that if a voter does not send back a valid ballot, the protocol execution aborts.

6.3.2 Vulnerabilities of distributed ballot based protocols

In this section, we show how the adversary can perform the d-transfer attack in favour of the "yes" outcome. We proceed as follows. We first show that this is possible if the adversary knows the difference $l_y - l_n$. We then show how the adversary can find out this value, and conclude the section with the probabilistic analysis of our attack which establishes that it can be performed with overwhelming probability in the number of voters.

The d-transfer attack: Given the difference $l_y - l_n$, a dishonest voter can violate the no-double-voting. From the definition of l_y and l_n it holds that:

$$2\pi(l_y - l_n)/D = \theta_y - \theta_n \tag{6.1}$$

If a corrupted voter (e.g. V_1) knows $l_y - l_n$, then they proceed as follows (w.l.o.g. we assume that they want to increase the number of "yes" votes by d):

1. V_1 applies the unitary operator: $C_d = \sum_{j=0}^{D-1} e^{ijd(\theta_y-\theta_n)} |j\rangle \langle j|$ to the received option qudit $|\psi(\theta_y)\rangle$. As a result, the state becomes:

$$C_d |\psi(\theta_y)\rangle = \frac{1}{\sqrt{D}} \sum_{j=0}^{D-1} e^{ijd(\theta_y-\theta_n)} e^{ij\theta_y} |j\rangle$$

items V_1 now performs the 2-qudit measurement specified in the Casting phase of the protocol and obtains the outcome r_1.

2. V_1 performs the unitary correction U_{r_1}. For $\tilde{\theta} = d(\theta_y - \theta_n) + \theta_y$, the global state now is:

$$U_{r_1} P_{r_1}\big(|\Phi\rangle \otimes C_d |\psi(\theta_y)\rangle \big) = \frac{1}{\sqrt{D}}\Big[\sum_{j=0}^{r_1-1} e^{i(D+j-r_1)\tilde{\theta}} |j\rangle^{\otimes N+1} + \sum_{j=r_1}^{D-1} e^{i(j-r_1)\tilde{\theta}} |j\rangle^{\otimes N+1} \Big]$$

3. Before sending the two qudit ballot and the value r_1 to T, V_1 performs the following operation to the option qudit:

$$\mathbf{Correct}_{r_1} = \begin{cases} e^{-iDd(\theta_y-\theta_n)} |j\rangle \langle j|, & 0 \le j \le r_1 - 1 \\ |j\rangle \langle j| & r_1 \le j \le D-1 \end{cases}$$

4. After all voters have cast their ballots to T, the global state of the system (up to normalization) is:

$$\frac{1}{\sqrt{D}}\Big(\sum_{j=0}^{r_1-1} e^{i(j-r_1)d(\theta_y-\theta_n)} e^{i(D+j-r_1)\theta_y} \prod_{k=2}^{N} \alpha_{j,r_k} |j\rangle^{\otimes 2N} + \sum_{j=r_1}^{D-1} e^{i(j-r_1)d(\theta_y-\theta_n)} e^{i(j-r_1)\theta_y} \prod_{k=2}^{N} \alpha_{j,r_k} |j\rangle^{\otimes 2N}\Big)$$

where,

$$\alpha_{j,r_k} = \begin{cases} e^{i(D+j-r_k)\theta_v^k}, & 0 \le j \le r_k - 1 \\ e^{i(j-r_k)\theta_v^k} & r_k \le j \le D-1 \end{cases}$$

and θ_v^k describes the vote of voter V_k, where $v \in \{y, n\}$. T just follows the protocol specification. It applies some corrections on the state given the announced results r_k and finally the state becomes:

$$\frac{1}{\sqrt{D}} \sum_{j=0}^{D-1} e^{i(j-r_1)d(\theta_y-\theta_n)} e^{i(j-r_1)\theta_y} \cdot \ldots \cdot e^{i(j-r_n)\theta_v^n} |j\rangle^{\otimes 2N}$$

which under a global phase factor is equivalent to:

$$\frac{1}{\sqrt{D}}\sum_{j=0}^{D-1} e^{ijd(\theta_y-\theta_n)}e^{ij(m\theta_y+(N-m)\theta_n)}\left|j\right\rangle^{\otimes 2N}$$

5. T removes the unwanted factor $e^{ijN\theta_n}$ as prescribed by the protocol, and the final state is:

$$\begin{aligned}\left|\Omega_{m+d}\right\rangle &= \frac{1}{\sqrt{D}}\sum_{j=0}^{D-1} e^{ijd(\theta_y-\theta_n)}e^{ijm(\theta_y-\theta_n)}\left|j\right\rangle^{\otimes 2N}\\ &= \frac{1}{\sqrt{D}}\sum_{j=0}^{D-1} e^{2\pi ij(m+d)(l_y-l_n)/D}\left|j\right\rangle^{\otimes 2N}\end{aligned}$$

6. After measuring the state, the result is $m+d$ instead of m.

Finding the difference between l_y and l_n: What remains in order to complete our attack is to find the difference $l_y - l_n$. We now show how an adversary can learn this difference with overwhelming probability in N. We assume that the adversary controls a fraction ε of the voters ($0 < \varepsilon < 1$), who are (all but one) instructed to vote half the times "yes" and the other half "no". Instead of destroying the remaining option qudits (exactly $\varepsilon N/2$ "yes" and $\varepsilon N/2$ "no" votes), the adversary keeps them to run Algorithm 1. In essence, the algorithm is

Algorithm 1 Adversary's algorithm

Input: $D, \left|\psi(\theta_v)\right\rangle_1, \cdots, \left|\psi(\theta_v)\right\rangle_{\varepsilon N/2}$
Output: $\tilde{l} \in \{0,\ldots,D-1\}$
1: $\texttt{Record} = [0,\ldots,0] \in \mathbb{N}^{1\times D}$; ▷ This vector shows us how many values are observed in each interval
2: $\texttt{Solution} = [\text{"}Null\text{"},\text{"}Null\text{"}] \in \mathbb{N}^{1\times 2}$;
3: $i, l, m = 0$;
4: **while** $i \leq \varepsilon N/2$ **do**
5: Measure $\left|\psi(\theta_v)\right\rangle_i$ by using POVM operator $E(\theta)$ from Eq.(6.2), the result is y_i;
6: Find the interval for which $\frac{2\pi j}{D} \leq y_i \leq \frac{2\pi(j+1)}{D}$;
7: $\texttt{Record}[j] =++$;
8: $i++$;
9: **end while**
10: **while** $l < D$ **do**
11: **if** $\texttt{Record}[l] \geq 40\%(\varepsilon N/2)$ **then**
12: $\texttt{Solution}[m] = l$;
13: $m++$;
14: **end if**
15: $l++$;
16: **end while**
17: **if** $\texttt{Solution} == [0, D-1]$ **then**
18: $\texttt{Solution} = [\texttt{Solution}[1], \texttt{Solution}[0]]$;
19: **end if**
20: **return** $\tilde{l} = \texttt{Solution}[0]$;

executed twice - once for each set of option qudits $\{\left|\psi(\theta_v)\right\rangle\}_{\varepsilon N/2}$, where $v \in \{y,n\}$. It measures the states in each set and attributes to each one an integer. After all,

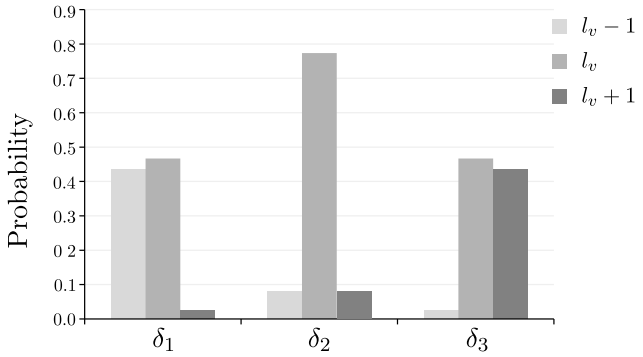

Figure 6.2: The probabilities with which Algorithm 1 records a value in $\{l_v - 1, l_v, l_v + 1\}$ after measuring state $|\psi(\theta_v)\rangle$ for $\delta_1 = \frac{\pi}{2^{35}}$, $\delta_2 = \frac{\pi}{2^{30}}$, and $\delta_3 = \frac{\pi(2^6-1)}{2^{35}}$.

states have been measured, the algorithm creates a vector `Record`, which contains the number of times each integer appeared during the measurements. Finally, Algorithm 1 creates the vector `Solution` in which it registers the values that appeared at least 40% times during the measurements, equivalently the values for which the `Record` vector assigned a number greater or equal than 40% of times. The algorithm outputs the first value in the `Solution` vector. As we see in Figure 6.2, with high probability the value that algorithm outputs is either l_v or $l_v - 1$, for both values of v. Hence, we can find the difference $l_y - l_n$. After having acquired knowledge of $l_y - l_n$, the adversary can instruct the last corrupted voter to change the outcome of the voting process as previously described.

Probabilistic analysis We prove here that the adversary's algorithm succeeds with overwhelming probability in N, where N is the number of voters. Therefore, as we later prove in Theorem 18, the election protocol needs to run at least exponentially many times to N to guarantee that the success probability of the adversary is at most 0.25. We present here the necessary lemmas and give the full proofs in the Quantum Supplementary Material.

In order to compute the success probability of the attack, we first need to compute the probability of measuring a value in the interval (x_l, x_{l+w}), where $x_l = \dfrac{2\pi l}{D}, \quad l \in \{0, 1, \ldots, D-1\}$[2].

Lemma 11. *Let $\Theta^v_{D,\delta} \in [0, 2\pi]$ be the continuous random variable that describes the outcome of the measurement of an option qudit $|\psi(\theta_v)\rangle, v \in \{y, n\}$ using operators:*

$$E(\theta) = \frac{D}{2\pi} |\Phi(\theta)\rangle \langle\Phi(\theta)| \tag{6.2}$$

[2]It is convenient to think of l as the D^{th} roots of unity.

where $|\Phi(\theta)\rangle = \frac{1}{\sqrt{D}} \sum_{j=0}^{D-1} e^{ij\theta} |j\rangle$. *It holds that:*

$$\Pr[x_l < \Theta^v_{D,\delta} < x_{l+w}] = \frac{1}{2\pi D} \int_{x_l}^{x_{l+w}} \frac{\sin^2[D(\theta - \theta_v)/2]}{\sin^2[(\theta - \theta_v)/2]} d\theta$$

According to Algorithm 1, an option qudit is attributed with the correct value l_v when the result of the measurement is in the interval $[x_{l_v}, x_{l_v+1}]$. Using Lemma 11, we can prove the following:

Lemma 12. *Let* $|\psi(\theta_v)\rangle$ *be an option qudit of the protocol. Then it holds:*

$$\Pr[x_{l_v} < \Theta^v_{D,\delta} < x_{l_v+1}] > 0.405$$

Lemma 12 shows that with probability at least 0.405, the result of the measurement is in the interval (x_{l_v}, x_{l_v+1}). Since Algorithm 1 inserts an integer to the `Solution` vector if it corresponds to at least 40% of the total measured values, l_v will most likely be included in the vector (we formally prove it later). Furthermore, we prove now that with high probability, there will be no other values to be inserted in `Solution`, except the neighbours of the value l_v (namely $l_v \pm 1$).

Lemma 13. *Let* $|\psi(\theta_v)\rangle$ *be an option qudit of the protocol. Then it holds:*

$$Pr[x_{l_v-1} < \Theta^v_{D,\delta} < x_{l_v+2}] > 0.9$$

Here we need to note that we are aware of the cases $l_v \in \{0, D-1\}$ where the members x_{l_v-1} and x_{l_v+2} are not defined. It turns out not to be a problem and the same thing can be proven for these values (see Quantum Supplementary Material).

We have shown that the probability the measurement outcome lies in the interval (x_{l_v-1}, x_{l_v+2}), and therefore gets attributed with a value of $l_v - 1$, l_v or $l_v + 1$, is larger than 0.9. If we treat each measurement performed by Algorithm 1 on each option qudit $|\psi(\theta_v)\rangle$, as an independent Bernoulli trial with success probability $p_l = Pr[x_l < \Theta^v_{D,\delta} < x_{l+1}]$, we can prove the following theorem:

Theorem 14. *With overwhelming probability in the number of voters* N*, Algorithm 1 includes* l_v *in the* `Solution` *vector*

$$\Pr[\texttt{Solution}[0] = l_v \vee \texttt{Solution}[1] = l_v] > 1 - 1/exp(\Omega(N))$$

We have proven that with overwhelming probability in N, integer l_v occupies one of the two positions of vector `Solution`, but what about the other value? In the next theorem, we show that with overwhelming probability in N, the other value is one of the neighbours of l_v, namely $l_v + 1$ or $l_v - 1$.

Theorem 15. *With negligible probability in the number of voters* N*, Algorithm 1 includes a value other than* $(l_v - 1, l_v, l_v + 1)$ *in the* `Solution` *vector,* i.e. $\forall w \in \{0, \ldots, l_v - 2, l_v + 2, \ldots, D - 1\}$ *:*

$$\Pr[\texttt{Solution}[0] = w \vee \texttt{Solution}[1] = w] < 1/exp(\Omega(N))$$

Lemma 16. *With overwhelming probability in N, the* `Solution` *vector in Algorithm 1, is equal to* $[l_v - 1, l_v], [l_v, "Null"]$ *or* $[l_v, l_v + 1]$. *Specifically,*

$$\Pr[\texttt{Solution} \in \{[l_v - 1, l_v], [l_v, "Null"], [l_v, l_v + 1]\}] > 1 - 1/exp(\Omega(N))$$

Now consider we have two executions of the Algorithm 1, one for the "yes" and one for the "no" option qudits. It turns out that the values in the positions $l_y - 1$ and $l_n - 1$ of the vector `Record`, follow the same Binomial distribution (it is easy to see that $p_{l_y-1} = p_{l_n-1}$). Also, each of them can be seen as a function of δ which is a monotonic decreasing function that takes a maximum value for $\delta = 0$ (the proof technique is similar to Lemma 12). At this point the probability is equal to p_{l_v}, which is at least 0.405 as we have proven in Lemma 12[3]. Armed with this observation we can prove the next theorem.

Theorem 17. *If we define the event "Cheat" as:*

$$Cheat = \left[Algo(y) - Algo(n) = l_y - l_n\right]$$

where $Algo(v)$ is the execution of Algorithm 1 with $v \in \{y, n\}$, then it holds that:

$$Pr["Cheat"] > 1 - 1/exp(\Omega(N))$$

Proof. (sketch) We have seen that there exists a δ_0 such that the probability p_{l_v-1} is equal to 0.4 for both values of v. It holds that:

$$\begin{aligned} Pr["Cheat"] &= Pr["Cheat"|\delta \in [0, \delta_0)] \cdot Pr[\delta \in [0, \delta_0)] \\ &+ Pr["Cheat"|\delta = \delta_0] \cdot Pr[\delta = \delta_0] \\ &+ Pr["Cheat"|\delta \in (\delta_0, 2\pi/D)] \cdot Pr[\delta \in (\delta_0, 2\pi/D)] \end{aligned}$$

For the first interval, for both values of v, Algorithm 1 registers `Solution` $= [l_v - 1, l_v]$ with overwhelming probability in N. This holds because of Theorem 16 and the previous observation. Therefore, for both values of v the algorithm outputs the values $l_v - 1$. As a result, $l_y - 1 - (l_n - 1) = l_y - l_n$.

For the second term, $Pr[\delta = \delta_0] = 0$, because δ is a continuous random variable. Finally, in the last term, the probability that the algorithm registers `Solution` $= [l_v - 1, l_v]$ is negligible in N, and by Theorem 16, `Solution` has the form $[l_v]$ or $[l_v, l_v + 1]$. So for both values of v, the printed values are l_y and l_n. □

At this point, we have proven that the adversary succeeds with overwhelming probability in N to perform the d-transfer attack in one round. But how many rounds should the protocol run to prevent this attack?

In the next theorem we prove that if the number of rounds is at most $exp(\Omega(N))$, the adversary succeeds with a probability of at least 0.25. Although

[3]The same holds for the p_{l_y+1}, p_{l_n+1} except that probability is a monotonic increasing function with maximum value at point $\delta = 2\pi/D$ and value equal to p_{l_v}.

in a small election these numbers might not be big, in a large scale election it is infeasible to run the protocol as many times, making it either inefficient or insecure. We also note that the probabilistic analysis for one round is independent of the value D, so cannot be used to improve the security of the protocol.

Theorem 18. *Let* $(|\Phi\rangle, |\psi(\theta_y)\rangle, |\psi(\theta_n)\rangle, \delta, D, N)$ *define one round of the protocol. If the protocol runs* ρ *rounds, where* $2 \leq \rho \leq exp(\Omega(N))$ *, the d-transfer attack succeeds with probability at least* 0.25.

Proof. According to Theorem 17 the probability that an adversary successfully performs the d-transfer attack is:

$$Pr[\text{"}Cheat\text{"}] > 1 - 1/exp(\Omega(N))$$

Now, for ρ protocol runs, where $2 \leq \rho \leq exp(\Omega(N))$, this probability becomes:

$$(Pr[\text{"}Cheat\text{"}])^{\rho} > (1 - 1/exp(\Omega(N)))^{\rho} \geq (1 - 1/\rho)^{\rho} > 0.25$$

□

6.4 Quantum voting based on conjugate coding

This section looks at protocols based on conjugate coding Okamoto et al. (2008); Zhou and Yang (2013). The participants in this family of protocols are one or more election authorities, the tallier and the voters. The election authorities are only trusted for eligibility; `privacy` should be guaranteed by the protocol against both malicious EA and T. Unlike the previous protocols, here the voters do not share any entangled states with neither EA nor T to cast their ballots. One of the main differences between the two protocols is that Okamoto et al. (2008) does not provide any verification of the election outcome, while Zhou and Yang (2013) does, but at the expense of receipt freeness, which Okamoto et al. (2008) satisfies. Specifically, in Zhou and Yang (2013) each V_k establishes two keys with T in an anonymous way by using part of protocol Okamoto et al. (2008) as a subroutine. It's worth mentioning that for these keys to be established, further interaction between the voters and EA is required and EA is assumed trusted for that task. At the end of execution, V_k encrypts the ballot with one of the keys and sends it to T over a quantum anonymous channel. T announces the result of each ballot accompanied with the second key so that the voters can verify that their ballot has been counted. This makes it also possible for a coercer to verify how a voter voted, by showing them the second key used as a receipt. It is worth mentioning that protocol Okamoto et al. (2008) could easily be made to satisfy the same notion of `verifiability`.

6.4.1 Protocol specification

`Setup` phase:

1. EA picks a vector $\bar{b} = (b_1, \ldots, b_{n+1}) \in_R \{0,1\}^{n+1}$, where n is the security parameter of the protocol. This vector will be used by EA for the encoding of the ballots and it will be kept secret from T until the end of the ballot casting phase.
2. For each V_k, EA prepares $w = poly(n)$ blank ballot fragments each of the form $|\phi_{\bar{a}_j,\bar{b}}\rangle = |\psi_{a_j^1,b_1}\rangle \otimes \ldots \otimes |\psi_{a_j^{n+1},b_{n+1}}\rangle, j \in \{1,\ldots,w\}$, where $\bar{a}_j = (a_j^1, \ldots, a_j^{n+1})$ such that:

$$(a_j^1, \ldots, a_j^n) \in_R \{0,1\}^n, a_j^{n+1} = a_j^1 \oplus \ldots \oplus a_j^n$$

and: $|\psi_{0,0}\rangle = |0\rangle, |\psi_{1,0}\rangle = |1\rangle, |\psi_{0,1}\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), |\psi_{1,1}\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.
These w fragments will constitute a blank ballot (e.g the first row of Figure 6.3 is a blank ballot fragment).
3. EA sends one blank ballot to each V_k over an authenticated channel.

`Casting` phase:

4. After reception of the blank ballot, each V_k re-randomizes it by picking for each fragment a vector $\bar{d}_j = (d_j^1, \ldots, d_j^{n+1})$ such that:

$$(d_j^1, \ldots, d_j^n) \in_R \{0,1\}^n, d_j^{n+1} = d_j^1 \oplus \ldots \oplus d_j^n.$$

$\forall j \in \{1,\ldots,w\}$, V_k applies unitary $U_j^{\bar{d}_j} = Y^{d_j^1} \otimes \ldots \otimes Y^{d_j^{n+1}}$ to the blank ballot fragment $|\phi_{\bar{a}_j,\bar{b}}\rangle$, where:

$$Y^1 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, Y^0 = \mathbb{I}$$

5. V_k encodes the candidate of choice in the $(n+1)^{th}$-qubit of the last blank ballot fragments[4]. For example, if we assume a referendum type election, V_k votes for $c \in \{0,1\}$ by applying to the blank ballot fragment $|\phi_{\bar{a}_w,\bar{b}}\rangle$ the unitary operations $U_w^{\bar{c}}$ respectively, where: $\bar{c} = (0, \ldots, 0, c)$ (see Figure 6.3).
6. V_k sends the ballot to T over an anonymous channel.

`Tally` phase:

7. Once the ballot casting phase ends, EA announces $\bar{b}$ to T.

[4]Candidate choices are encoded in binary format (e.g 3 → 011

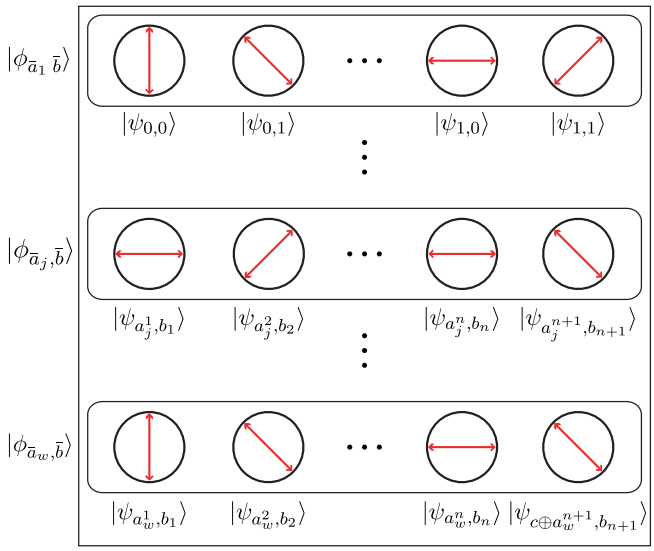

Figure 6.3: The ballot consisting of w ballot fragments, which encode the binary choice "0...01" in a referendum type election example.

8. With this knowledge, T can decode each cast ballot on the correct basis. Specifically, T decodes each ballot fragment by measuring it in the basis described by vector $\bar{b}$ and XORs the resulting bits. After doing this to each ballot fragment, T ends up with a string, which is the actual vote cast.
9. T announces the election result.

6.4.2 Vulnerabilities of conjugate coding protocols

The technique underlying this protocol is closely related to the one used in the first quantum key distribution protocols Bennett and Brassard (1984); Shor and Preskill (2000). However, it has some limitations in the context of these voting schemes.

Malleable blank ballots: An adversary can change the vote of an eligible voter when the corresponding ballot is cast over the anonymous channel. Assume V_k has applied the appropriate unitary on the blank ballot to vote for the candidate of their choice. And let us consider that the last m ballot fragments encode the candidate. When the adversary sees the cast ballot over the quantum anonymous channel, they apply the unitary $U_{w-(m-1)}^{\bar{c}_1}, \ldots, U_w^{\bar{c}_m}$, where c_r is either 0 or 1, depending on their choice to flip the candidate bit or not. As a result, the adversary modifies the ballot of V_k such that it decodes to a different candidate than the intended one. This is possible because the adversary is aware of the ballot fragments used to encode the candidate choice. Furthermore, if the adversary has

side-channel information about the likely winning candidate (from pre-election polls for instance), they will be able to change the vote encoded in the ballot into one of their desire. This is possible because the adversary is aware of which bits are encoded in the ballot more frequently and knows exactly which unitary operator to apply to decode to a specific candidate.

Violation of privacy: It is already acknowledged by the authors of Okamoto et al. (2008) that the EA can introduce a "serial number" in a blank ballot to identify a voter, e.g. some of the blank ballot fragments in the head of the ballot decode to "1" instead of "0". This allows the EA to decode any ballot cast over the quantum anonymous channel, linking the identity of the voters with their choice.

One-more-unforgeability: The security of the protocol relies on a quantum problem introduced in Okamoto et al. (2008), named *one-more-unforgeability* and the assumption that it is computationally hard for a quantum adversary. The game that captures this assumption goes as follows: a challenger encodes w blank ballot fragments in a basis $\bar{b}$ and gives them to the adversary. The adversary wins the game if they produce $w + 1$ valid blank ballot fragments in the basis $\bar{b}$. The authors claim the probability of the adversary winning this game is at most $1/2 + 1/2(negl(n))$.

On the security parameter: Because of the ballots' malleability, an adversary could substitute the parts of the corrupted voters' blank ballot fragments that encode a candidate, with blank ballot fragments in a random base. Of course, these ballots would open into random candidates in a specific domain but would still be valid, since the leading zeros would not be affected by this change. This is because blank ballots contain no entanglement. Now the adversary can keep these valid spare blank ballot fragments to create new valid blank ballots. To address this problem, the size of blank ballots needs to be substantially big compared to the number of voters and the size of the candidate space ($Nm << w$).

6.5 Other protocols

Other protocols have also been proposed, with the main characters that the EA controls when ballots get counted. This can be achieved with either the use of shared entangled states between EA, T and V_k Thapliyal et al. (2017); Xue and Zhang (2017) or Bell pairs Thapliyal et al. (2017) between T and V_k with EA knowing the identity of the holder of each pair particle. However, we do not fully analyse these protocols in this review, as they have many and serious flaws making even the `correctness` arguable. The protocol of Xue and Zhang (2017) claims to provide verifiability of the election outcome, but without explaining how this can be achieved. From our understanding of the protocol, this seems unlikely to

be the case. From the description of the protocol, each voter can change their mind and announce a different vote from the original cast one. This is possible because the function every voter uses to encode their vote is not committed in any way. Two protocols introduced in Thapliyal et al. (2017) have similar limitations. For instance, there is no mechanism for verifiability of the election outcome. In addition, `privacy` against T is not satisfied in contrast to protocols we saw in section 6.3. This is because each voter's vote is handled individually and not in a homomorphic manner. All of these could be achieved just by a classical secure channel. Last, the protocol appearing in Horoshko and Kilin (2011) shares many of the limitations of the former protocols as well as some further ones. The method introduced for detecting eavesdropping in the election process is insecure, as trust is put into another voter to detect any deviation from the protocol specification. Moreover, the way each voter casts their vote is not well defined in the protocol, which makes `privacy` and `correctness` trivially violated.

Finally, we note that there exist protocols that consider elections with quantum input, see e.g. Bao and Yunger Halpern (2017). This type of protocol is more relevant to quantum game theory and less to election schemes with classical input/voting choice, and we have not considered them in this study.

As we see, the absence of formal treatment in quantum settings leads to security flaws, especially for the concept of verifiability, privacy, integrity. Just to initiate such discussion, we present in the next section the first definitions of the mentioned properties. Although these security definitions capture security guarantees for protocols that use a specific tally function, this restriction is a natural assumption based on the state-of-the-art e-voting protocols Arapinis et al. (2016). Regarding the soundness of the security definitions, they need to be compared extensively with the security definitions in the classical literature and evaluated by arguing about their security in comparison with an ideal functionality similar to Theorem 1 in this thesis. For that reason, we only present them to initiate a discussion around the formal treatment of quantum e-voting rather than suggest them as a solution for evaluating the security of future implementations.

6.6 Discussion: definitions for secure quantum e-voting

In this section, we present formal definitions for vote privacy and universal verifiability in the quantum setting considering adaptive corruption. Even if we inspired by other works in classical settings Cortier et al. (2016); Bernhard et al. (2015) there are many challenges to define a security definition in the quantum setting that is related to the sensitive nature of quantum information, for example after measuring a quantum state we can not revert it to its original state before the measurement occurs, leading us to many limitations. Contrary, these

limitations are not appearing in the classical setting, making security modelling in the quantum setting inherently more challenging to be defined.

Next we present three security definitions, the first is for *Universal verifiability* Cortier et al. (2016) in the quantum setting and it is captured via the experiment $\mathbf{EXP}^{\Pi}_{\mathsf{Qver}}$ described in Figure 6.4. Our definition is inspired by its classical counterpart in Cortier et al. (2016) where the authors reviewing extensively the state-of-the-art of the notions of verifiability in the classical setting. Specifically, our predicate $\mathsf{P}^{\Pi}_{\mathsf{VCounted}}$ captures the security properties of Definition 2, p.9 in Cortier et al. (2016). Our second security definition is for *Privacy* and is captured via the experiment $\mathbf{EXP}^{\Pi}_{\mathsf{Qpriv}}$ described in Figure 6.5. It is built on classical Definition 1 in Bernhard et al. (2015). Specifically, in our experiment we allow the adversary to define the votes of the honest parties and by the end of the experiment, he should guess if the votes have been permuted or not, similar to definition 1 in Bernhard et al. (2015) in the two-party case. However, our security definition is more general as it captures a wider spectrum of both classical and quantum e-voting protocol, such as protocols that rely such part of their security in *anonymous channels* Dingledine et al. (2004). Finally, we define *Correctness* which is captured in Figure 6.6. In a nutshell, the experiment is the same as $\mathbf{EXP}^{\Pi}_{\mathsf{Qver}}$ except that there is not any verifiable mechanism of the tallying, which is provided by the protocol Π, and we do not allow the adversary to corrupt the tallier, e.g. in settings where a tallier exists. The motivation defining this experiment was the impact of the adversarial influence in various treat models where the adversary can not corrupt the tallier, e.g. the ability of a corrupted voter to double vote or alternate other voters vote.

we intend to present these definitions, not as a security tool for arguing about the security of a quantum e-voting protocol, rather as a starting point for further research.

For formalising security we adopt the standard game-based security framework. The security of a protocol is captured by a game between a challenger $\mathcal{C}$ that models the honest parties, and a quantum polynomial-time adversary $\mathcal{A}$ that captures the corrupted parties. $\mathcal{A}$ can adaptively corrupt a fraction ϵ of the voters. We assume that the eligibility list is provided with a priori in a trusted manner and that $\mathcal{A}$ chooses the votes of all voters, to provide stronger definitions Juels et al. (2005). We denote with $\mathcal{X}_a$ the local register of each party a. Communication between parties is done using *communication oracles* Dupuis et al. (2012); a call to the oracle $\mathcal{O}_{\Pi,\mathcal{A}}(\mathcal{X}_S, \mathcal{X}_R)$ takes contents of the sender's register $\mathcal{X}_S$ and copies them to the receiver's register $\mathcal{X}_R$, according to Π. In between, it allows the communication to be processed by $\mathcal{A}$, and it also erases the quantum information from the respective 'sent' registers, to respect the no-cloning theorem (classical information can still be transmitted without being erased from the sender). Note that the way $\mathcal{A}$ can treat the transmitted information is specified by the protocol Π and quantum mechanics, e.g. in case of a quantum authen-

tication channel between parties, $\mathcal{A}$ is only allowed to erase the content of the quantum register but nothing more.

Two limitations of our definitions are: (i) they capture only e-voting protocols which the tally function defines explicitly the voting set (e.g no weigh voting); (ii) the adversary can corrupt adaptively voters only during the casting phase, and not initially.

***Setup* phase:** $\mathcal{A}$ defines the voting choices of all voters. $\mathcal{C}$ and $\mathcal{A}$ generate the protocol parameters and store them in the global register $\mathcal{X}$ (comprising of all local registers $\mathcal{X}_a$). This is done according to Π, therefore if the latter specifies that the parameters are generated by a trusted third party, then this interaction is void. Instead, in protocols like the one in section 6.1 where the parameters are generated interactively among the parties, this interaction illustrates the fact that some of these parties might be corrupt. To also capture protocols that use anonymous channels, we ask $\mathcal{C}$ to randomly choose a permutation ρ from the set $\mathcal{F}$ of all permutations of N elements (the uniformly at random choice is denoted with $\rho \in_R \mathcal{F}$). Permutation ρ specifies the casting order of the voters and is initially unknown to $\mathcal{A}$; information about ρ could however be leaked during the next phase.

***Casting* phase:** The protocol Π specifies the algorithm CastBallot for generating the ballots. $\mathcal{C}$ generates ballots according to the CastBallot algorithm on behalf of honest voters and $\mathcal{A}$ on behalf of the corrupted ones.

***Tally* phase:** The protocol Π specifies the tallying algorithm Tally and the verification algorithm Verify. The Verify algorithm is the protocol-specific public test parties can run to verify the election. Note that if such an algorithm is not explicitly provided in the protocol's description, then it can be modelled by the True predicate (the algorithm will always return 1). If this test is successful (return 1), the tally is then computed. If all talliers are honest, $\mathcal{C}$ computes the election result on behalf of them by running the Tally algorithm. If some of the talliers are corrupt, $\mathcal{A}$ and $\mathcal{C}$ interactively produce the tally. If none of the talliers is honest, $\mathcal{A}$ computes the tally instead. We capture all these cases with function Tally$'$, where honest talliers controlled by $\mathcal{C}$ act according to Π and dishonest talliers act as $\mathcal{A}$ dictates.

Ideally, an e-voting protocol will satisfy at least the following properties Bernhard et al. (2015); Cortier et al. (2016); Delaune et al. (2006); Gallegos-Garcia et al. (2016). `Correctness`: corrupted voters cannot modify the honest votes, all voters vote at most once and the protocol does not abort[5]. `Privacy`: keep the vote of

[5]All of the quantum protocols we analyse in this work are subject to abort by a single party except the one presented in Okamoto et al. (2008); Zhou and Yang (2013). Despite that, we present and focus on attacks that violate properties such as `privacy` and `correctness` by attacking the one-voter-one-vote policy.

a voter private. `Verifiability`: allow for verification of the results by voters and external auditors. We focus on privacy and verifiability type properties. We note that our tally function is defined similarly to Bernhard et al. (2015). Our definitions, therefore, capture only tally functions where the election result corresponds to a unique set of votes (e.g. it doesn't capture privacy for 'weighted voting'); however, this restriction is a natural assumption based on the state-of-the-art on e-voting based protocols Arapinis et al. (2016).

Universal verifiability: Our definition of `universal verifiability` is similar to Cortier et al. (2016) and is captured by the experiment $\mathbf{EXP}^{\Pi}_{\mathsf{Qver}}$.

The experiment $\mathbf{EXP}^{\Pi}_{\mathsf{Qver}}(\mathcal{A}, \epsilon, \delta_0)$

- `Setup` phase: $\mathcal{C}$ and $\mathcal{A}$ generate the parameters in register $\mathcal{X}$ as specified by Π and the adversarial model. Furthermore, $\mathcal{A}$ chooses the votes for all voters $\{v_k\}_{V_k \in \mathcal{V}}$ and $\mathcal{C}$ the casting order $\rho \in_R \mathcal{F}$.
- `Casting` phase: For $k = 1, \ldots, N$
  - If $\mathcal{A}$ chooses to corrupt $V_{\rho(k)}$, they are added to $\mathcal{V}_{\mathcal{A}}$, where $|\mathcal{V}_{\mathcal{A}}| \leq \epsilon|\mathcal{V}|$.
  - If $V_{\rho(k)} \notin \mathcal{V}_{\mathcal{A}}$, then $\mathcal{C}$ runs $(\mathcal{X}_{\rho(k)}, \perp) \leftarrow \mathsf{CastBallot}(v_{\rho(k)}, \mathcal{X}_{\rho(k)}, \delta_0)$. If not $\perp$, $\mathcal{C}$ calls $\mathcal{O}_{\Pi,\mathcal{A}}(\mathcal{X}_{\rho(k)}, \mathcal{X}_R)$, where R is the receiver designated by Π.
  - If $V_{\rho(k)} \in \mathcal{V}_{\mathcal{A}}$, $\mathcal{A}$ performs some operation on register $\mathcal{X}_{\mathcal{A}}$ and calls $\mathcal{O}_{\Pi,\mathcal{A}}(\mathcal{X}_{\mathcal{A}}, \mathcal{X}_R)$, where R is any receiver designated by $\mathcal{A}$.
- `Tally` phase: $\mathcal{A}$ and $\mathcal{C}$ call the Tally' function, which depends on Π and the adversarial model, to compute the election outcome $X \leftarrow \mathsf{Tally}'(\mathcal{X}_{\mathcal{C}}, \mathcal{X}_{\mathcal{A}}, \delta_0)$.
- If $(\mathsf{Verify}^{\Pi}(X, \mathcal{X}_{\mathcal{C}}, \delta_0) = 1)$ and $(\mathsf{P}^{\Pi}_{\mathsf{VCounted}}(\{v_k\}_{V_k \notin \mathcal{V}_{\mathcal{A}}}, X) = 0 \vee \mathsf{Nballots}^{\Pi}(X) > N)$ then `output 1`, else `output 0`.

Figure 6.4: The experiment $\mathbf{EXP}^{\Pi}_{\mathsf{Qver}}$

First, $\mathcal{A}$ defines how honest voters vote. Then, $\mathcal{C}$ and $\mathcal{A}$ generate the protocol parameters $\mathcal{X}$ according to Π and the corruption model of $\mathcal{A}$. Moreover, $\mathcal{C}$ chooses at random a permutation ρ that specifies the casting order of all voters. In the `casting` phase, $\mathcal{A}$ can choose to corrupt voters adaptively. Specifically, $\mathcal{A}$ decides to either corrupt or not the casting slot without knowing which party it is in the case that the order is unknown to him (e.g anonymous channels). For honest voters, $\mathcal{C}$ follows the $\mathsf{CastBallot}$ algorithm as specified by Π to generate the ballot and sends it to $\mathcal{A}$. Depending on the protocol specification, $\mathcal{A}$ might then perform some quantum operation on the received ballot and further forward it to the designated receiver. The operation should always be consistent with the protocol, e.g. if Π uses quantum authenticated channels, $\mathcal{A}$ will not be able to

modify the ballot. This process is captured by calling the communication oracle $\mathcal{O}$. For corrupted voters, $\mathcal{A}$ casts the ballot on their behalf. After all, votes have been cast, the election outcome is computed by Tally' defined above, which depends on the protocol Π and the adversarial model. $\mathbf{EXP}^{\Pi}_{\mathsf{Qver}}$ outputs 1 if the election outcome is accepted by $\mathcal{C}$, while either an honest vote has not been counted in the outcome, or the number of cast votes exceeds the number of voters; Otherwise, the experiment outputs 0. To account for these events, we define three predicates; Verify which is the protocol-specific public test parties can run to verify the election, the predicate $\mathsf{P}^{\Pi}_{\mathsf{VCounted}}$ reveals if honest votes are discarded from or altered in the final outcome[6] and $\mathsf{Nballots}^{\Pi}$ reveals the number of votes accounted for the election result X. Specifically, $\mathsf{P}^{\Pi}_{\mathsf{VCounted}}$ captures exactly the two security properties of definition 2, p.9 in Cortier et al. (2016). If $\mathcal{A}$ has tampered with the election outcome, the predicate Verify should return `false`.

Definition 6.6.1. A quantum e-voting protocol Π satisfies ϵ-**quantum verifiability** if for every quantum polynomial-time $\mathcal{A}$ the probability of winning experiment $\mathbf{EXP}^{\Pi}_{\mathsf{Qver}}(\mathcal{A}, \epsilon, \delta_0)$ is negligible with respect to δ_0:

$$\Pr[1 \leftarrow \mathbf{EXP}^{\Pi}_{\mathsf{Qver}}(\mathcal{A}, \epsilon, \delta_0)] = \mathsf{negl}(\delta_0)$$

6.6.1 Game based definition for quantum privacy

Vote-privacy: The experiment $\mathbf{EXP}^{\Pi}_{\mathsf{Qpriv}}$ captures vote privacy which ensures that the adversary $\mathcal{A}$ cannot link honest voters to their votes. We build upon definition 1 in Bernhard et al. (2015); however, a problem with this definition is that it requires the honest voters controlled by the challenger to send their ballot in two separate ballot boxes. Such a process is not possible with quantum information due to the no-cloning theorem; we solve this issue with our definition. Moreover, we also capture self-tallying protocols, where Bernhard et al. (2015) states explicitly that a secret key is required for the production of the tally. Finally, our definition also treats protocols that use anonymous channels to provide privacy, while Bernhard et al. (2015) leaves out such protocols since it states that the adversary can call the oracle $\mathcal{O}_{\mathsf{cast}}$ with the ballot and the voter's ID. These fundamental differences lead to a completely new experiment. The goal of $\mathbf{EXP}^{\Pi}_{\mathsf{Qpriv}}$ is to capture that $\mathcal{A}$ cannot distinguish between two worlds, one where the voters vote as $\mathcal{A}$ tells them, and another where their votes have been permuted.

[6] We require in particular that $\mathsf{P}^{\Pi}_{\mathsf{VCounted}}(\{v_k\}_{V_k \notin \mathcal{V}_{\mathcal{A}}}, \perp) = 0$

The experiment $\mathbf{EXP}^{\Pi}_{\mathsf{Qpriv}}(\mathcal{A}, \epsilon, \delta_0)$

■ **Setup phase:** $\mathcal{C}$ and $\mathcal{A}$ generate the parameters in register $\mathcal{X}$ as specified by Π and the adversarial model. $\mathcal{C}$ chooses the casting order $\rho \in_R \mathcal{F}$ and a bit $\beta \in_R \{0,1\}$. Furthermore, $\mathcal{A}$ chooses the votes for all voters $\{v_k\}_{V_k \in \mathcal{V}}$ and a permutation $\mathsf{F} \in \mathcal{F}$.

■ **Casting phase:** For $k = 1, \ldots, N$

- If $\mathcal{A}$ chooses to corrupt $V_{\rho(k)}$, $V_{\rho(k)}$ is added to $\mathcal{V}_{\mathcal{A}}$, where $|\mathcal{V}_{\mathcal{A}}| \leq \epsilon|\mathcal{V}|$.
- If $V_{\rho(k)} \notin \mathcal{V}_{\mathcal{A}}$, then $\mathcal{C}$ runs $\{\mathcal{X}_{\rho(k)}, \perp\} \leftarrow \mathsf{CastBallot}(v^{\beta}_{\mathsf{F}(\rho(k))} \cdot v^{1-\beta}_{\rho(k)}, \mathcal{X}_{\rho(k)}, \delta_0)$. If not $\perp$, $\mathcal{C}$ calls $\mathcal{O}_{\Pi,\mathcal{A}}(\mathcal{X}_{\rho(k)}, \mathcal{X}_R)$, where R is the receiver designated by Π.
- If $V_{\rho(k)} \in \mathcal{V}_{\mathcal{A}}$, $\mathcal{A}$ performs some operations on register $\mathcal{X}_{\mathcal{A}}$ and calls $\mathcal{O}_{\Pi,\mathcal{A}}(\mathcal{X}_{\mathcal{A}}, \mathcal{X}_R)$, where R is any receiver designated by $\mathcal{A}$.

■ **Tally phase:** If $\{v_k : V_k \notin \mathcal{V}_{\mathcal{A}}\} = \{v_{\mathsf{F}(k)} : V_k \notin \mathcal{V}_{\mathcal{A}}\}$, $\mathcal{C}$ announces the election outcome $X \leftarrow \mathsf{Tally}(\mathcal{X}_{\mathcal{C}}, \delta_0)$ to $\mathcal{A}$. Else `output 0`. $\mathcal{A}$ guesses bit β^*. If $\beta^* = \beta$ then `output 1`, else `output 0`.

Figure 6.5: Experiment $\mathbf{EXP}^{\Pi}_{\mathsf{Qpriv}}$

$\mathcal{A}$ defines how honest voters vote and chooses a permutation $\mathsf{F} \in \mathcal{F}$ over the voting choices of all voters in $\mathcal{V}$. After the parameters of the protocol $\mathcal{X}$ are generated, $\mathcal{C}$ chooses a random bit β which defines two worlds; when $\beta = 0$, the honest voters vote as specified by $\mathcal{A}$, while when $\beta = 1$, the honest voters swap their votes according to permutation F again specified by $\mathcal{A}$. Again, $\mathcal{C}$ chooses at random a permutation ρ which defines the casting order of all voters. If the choices of the honest voters during the `casting phase` are still a permutation of their initial choices the experiment proceeds to the next phase, else it outputs 0. In the tally phase, $\mathcal{C}$ computes the election outcome. Finally, $\mathcal{A}$ tries to guess if the honest voters controlled by $\mathcal{C}$ have permuted their votes ($\beta = 1$) or not ($\beta = 0$), by outputting the guess bit β^*. If $\mathcal{A}$ guessed correctly $\mathbf{EXP}^{\Pi}_{\mathsf{Qpriv}}$ outputs 1; otherwise $\mathbf{EXP}^{\Pi}_{\mathsf{Qpriv}}$ outputs 0.

Definition 6.6.2. A quantum e-voting protocol Π satisfies ϵ-**quantum privacy** if for every quantum polynomial-time $\mathcal{A}$ the probability of winning the experiment $\mathbf{EXP}^{\Pi}_{\mathsf{Qpriv}}(\mathcal{A}, \epsilon, \delta_0)$ is negligibly close to $1/2$ with respect to δ_0 :

$$\Pr[1 \leftarrow \mathbf{EXP}^{\Pi}_{\mathsf{Qpriv}}(\mathcal{A}, \epsilon, \delta_0)] \leq 1/2 + \mathsf{negl}(\delta_0)$$

The most established game-based definitions for privacy in the classical setting Bernhard et al. (2015) assume two ballot boxes, where one holds the real tally and the other holds either the real or the fake tally. In the quantum case, the adaptation is not straightforward mainly because of the no-cloning theorem.

The existence of two such boxes assumes that information is copyable, which is not the case with quantum information. Similarly, we cannot assume that the experiment runs two times because $\mathcal{A}$ could correlate the two executions by entangling their parameters, something that a classical adversary cannot do. We address this difficulty by introducing quantum communication oracles to capture the network activity and model the special handling of quantum information (e.g. entangled states). Moreover, the election result is produced on the actual ballots rather than the intended ones. With this, we capture a broader spectrum of attacks (e.g Helios replay attack), at the same time introduce trivial distinctions corresponding to false attacks. We tackle this by allowing the experiment to output `-1` in such undesired cases which are mainly artefacts of the model. So an advantage of our privacy definition is that it allows the analysis of self-tallying type protocols in contrast with previous definitions of privacy Bernhard et al. (2015). In self-tallying elections, the adversary can derive the election outcome on their own without the need for secret information.

Note: Our definitions of `verifiability` and `privacy` capture both classical and quantum protocols. For the classical case, the quantum registers will be used for storing and communicating purely classical information. Devising our definitions for the quantum setting was not a trivial task as many aspects are hard to define, like bulletin boards, and others that need to be introduced, like quantum registers, potentially containing entangled quantum states. Moreover, our experiments capture protocols that use anonymous channels, by assuming that the casting order is unknown to $\mathcal{A}$, as well as self-tallying protocols. We leave as future work the investigation of the soundness of our definitions. To this end, one would define an ideal functionality capturing privacy and verifiability, and prove the soundness relation between our game-based definitions and this ideal functionality similarly to Bernhard et al. (2015).

6.6.2 Game based definition for quantum integrity

The correctness experiment $\mathbf{EXP}^{\Pi}_{\mathsf{Qcorr}}$ is the same as $\mathbf{EXP}^{\Pi}_{\mathsf{Qver}}$ with the only exceptions that there is not a predicate $\mathsf{P}^{\Pi}_{\mathsf{Verify}}$ and we do not allow $\mathcal{A}$ to corrupt the tallier (if there exist in the protocol Π). As a result, we do not capture universal verifiability in $\mathbf{EXP}^{\Pi}_{\mathsf{Qcorr}}$, but only double voting and vote deletion/alteration of honest ballots. Moreover, in the case that the election result X is equal "$\perp$" the adversary wins the experiment.

The experiment $\mathbf{EXP}^{\Pi}_{\mathsf{Qcorr}}(\mathcal{A}, \epsilon, \delta_0)$

- **Set up phase:** $\mathcal{C}$ and $\mathcal{A}$ generate the parameters in register $\mathcal{X}$ as specified by Π and the adversarial model. Furthermore, $\mathcal{A}$ chooses the votes for all voters $\{v_k\}_{V_k \in \mathcal{V}}$ and $\mathcal{C}$ the casting order $\rho \in_R \mathcal{F}$.
- **Casting phase:** For each $k = 1, \ldots, N$
  - If $\mathcal{A}$ chooses to corrupt $V_{\rho(k)}$, they are added to $\mathcal{V}_{\mathcal{A}}$, where $|\mathcal{V}_{\mathcal{A}}| \leq \epsilon|\mathcal{V}|$.
  - If $V_{\rho(k)} \notin \mathcal{V}_{\mathcal{A}}$, then $\mathcal{C}$ runs $(\mathcal{X}_{\rho(k)}, \perp) \leftarrow \mathsf{CastBallot}(v_{\rho(k)}, \mathcal{X}_{\rho(k)}, \delta_0)$. If not $\perp$, $\mathcal{C}$ calls $\mathcal{O}_{\Pi,\mathcal{A}}(\mathcal{X}_{\rho(k)}, \mathcal{X}_R)$, , where R is the receiver designated by Π.
  - If $V_{\rho(k)} \in \mathcal{V}_{\mathcal{A}}$, $\mathcal{A}$ performs some operation on register $\mathcal{X}_{\mathcal{A}}$ and calls $\mathcal{O}_{\Pi,\mathcal{A}}(\mathcal{X}_{\mathcal{A}}, \mathcal{X}_R)$, where R is any receiver designated by $\mathcal{A}$.
- **Tally phase:** $\mathcal{C}$ computes $X \leftarrow \mathsf{Tally}(\mathcal{X}_{\mathcal{C}}, \delta_0)$:
  - If $(X = \perp)$ or $(\mathsf{P}^{\Pi}_{\mathsf{VCounted}}(\{v_k\}_{V_k \notin \mathcal{V}_{\mathcal{A}}}, X) = 0 \vee \mathsf{Nballots}^{\Pi}(X) > N)$ then `output 1`, else `output 0`.

Figure 6.6: The experiment $\mathbf{EXP}^{\Pi}_{\mathsf{Qcorr}}$

Definition 6.6.3. We say that a quantum e-voting protocol Π satisfies ϵ-**quantum correctness** if for every quantum polynomial-time $\mathcal{A}$, the probability to win the experiment $\mathbf{EXP}^{\Pi}_{\mathsf{Qcorr}}(\mathcal{A}, \epsilon, \delta_0)$ is negligible with respect to δ_0:

$$\Pr[1 \leftarrow \mathbf{EXP}^{\Pi}_{\mathsf{Qcorr}}(\mathcal{A}, \epsilon, \delta_0)] = \mathsf{negl}(\delta_0).$$

Chapter 7

Discussion

In this chapter, we summarize our findings across all the chapters of this thesis. Moreover, we point out the limitation of our proposals related to the security assumptions and the modelling choices that we made. Finally, we discuss the possibilities of further improvements and likely future research directions.

7.1 Summary

We have examined the current state-of-the-art for both classic self-tallying and quantum e-voting protocols. We defined new security definitions, modelled and developed new cryptographic primitives and protocols, presented the most prominent proposals and analysed their security.

Classical cryptography: In Chapter 4 we defined the ideal functionality $\mathcal{F}_{\mathsf{TLE}}$ that captures all the security properties that a TLE scheme should ideally satisfy. Moreover, we have introduced security definitions in a game-based style that, again, both in the concept of UC and the stand-alone setting. Usually, game-based definitions are more convenient and flexible when proving security, in contrast with their counterpart, the UC framework. Although the UC framework is rigorous and captures with precision the desired security properties, usually it lacks simplicity. Next, we defined a protocol that UC realises the functionality $\mathcal{F}_{\mathsf{TLE}}$ given that the underlying encryption scheme satisfies one of our security definitions. We presented a concrete TLE construction, namely Astrolabous, that satisfies our security definition, making the whole construction complete. Finally, we presented a new game based definition in the stand-alone setting and showed that both Astrolabous and a variant of the construction in Mahmoody et al. (2011) satisfies it.

In Chapter 5 we captured the notion of self-tallying elections into the ideal functionality $\mathcal{F}_{\mathsf{STE}}$ and we presented our protocol E-CCLESIA, that UC realises $\mathcal{F}_{\mathsf{STE}}$. We followed a modular approach by "breaking down" $\mathcal{F}_{\mathsf{STE}}$ into two sub-functionalities, namely $\mathcal{F}_{\mathsf{vm}}$ and $\mathcal{F}_{\mathsf{elig}}$. The functionality $\mathcal{F}_{\mathsf{vm}}$ captures the

semantic security of the encrypted votes and the network activity during the election whereas the functionality $\mathcal{F}_{\mathsf{elig}}$ captures the security property of eligibility (e.g. only votes originating from eligible voters will be counted). Because of our modular approach, the substitution of one cryptographic primitive does not lead to the need to reprove the security of the whole protocol, making E-CCLESIA easily accessible for future development. For example, one can change the Astrolabous TLE scheme with another construction, not necessarily a TLE-based one. In that case, we can avoid proving the whole security of the new E-CCLESIA from scratch. Instead, we need to prove that our new protocol UC realises $\mathcal{F}_{\mathsf{vm}}$.

Quantum setting: In Chapter 6 we have systematized and analyzed the most prominent quantum e-voting protocols in the literature based on their special features. What we have found is that all the proposed protocols fail to satisfy the cryptographic security standards so that they can be implemented in the future. We have presented concrete novel attacks in each of them. Notwithstanding these limitations, we believe that our analysis opens new research directions for the study of the quantum cut-and-choose technique, which plays a fundamental role in the secure distribution of quantum information.

7.2 Limitations

In this section, we present the limitation of the work presented in the thesis Chapter by Chapter.

Time-lock encryption, Chapter 4: The way the $\mathcal{F}_{\mathsf{TLE}}$ functionality handles the decryption queries is limited as it does not specify which party is allowed to decrypt the message. Instead, all parties can eventually decrypt and retrieve the original message. This approach is similar to the concept of "time-lock commitments" rather than of time-lock encryption. Nevertheless, in the literature, the proposed constructions are termed time-lock encryption and we keep that terminology.

We have defined a relativistic TLE construction, Astrolabous, that UC realises $\mathcal{F}_{\mathsf{TLE}}$. As an intermediate step, we had to prove that Astrolabous satisfies our game-based definition. On the other hand, we do not know if our game-based definition is general enough to capture absolute time TLE constructions such as the one in Liu et al. (2018). Also, in Theorem 1 we proved that if a construction satisfies our security definition, then we have a UC realization of a protocol that uses these TLE algorithms, without proving the other direction. That means that the security definition provides more security guarantees than $\mathcal{F}_{\mathsf{TLE}}$ itself, thus it is "stronger". Nevertheless, the security definition seems reasonable enough to claim that it is not "too" strong.

Our modelling does not support constructions where the functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ uses a constant distribution $\mathbf{D}_x$ to sample values upon request. For example,

in Rivest et al. (1996), solving a time-lock puzzle consists of repeatedly squaring a base. This computation is deterministic so any PPT Turing machine can produce identical results if they engaged in the same computation. Even if we abstract this computation into the functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ and we restrict the access to it by using the functionality wrapper $\mathcal{W}$, there is no way for the decryption algorithm to know if the provided decryption key (witness in our case) is created through queries to $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ or is locally computed by the adversary. As a result, either the algorithm returns the actual message, without necessarily the provided decryption key be created through the oracle, meaning that the model does not capture the concept of TLE, or it returns the special symbol $\perp$, meaning that even in optimal scenarios where the $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ is engaged for the creation of the decryption key, the algorithm aborts.

E-cclesia: A self-tallying classical e-voting protocol, Chapter 5: Our ideal functionality $\mathcal{F}_{\mathsf{vm}}$ captures correctness, privacy and fairness. Another very important security property a protocol should satisfy is called *coercion resistant*. Informally, the property considers adversaries called the coercers that they try to influence a voter's choice. The voter has to obey the coercer's will because of the relationship they have (e.g. the relationship between employer and employee). The coercer can instruct a voter on how to vote before the start of the election. After the end of the election, the coercer checks the election outcome and tries to conclude if the coercion was successful (*passive coercion*). However, he can be all along with the voter during the whole election process (*active* coercion). Unfortunately, in $\mathcal{F}_{\mathsf{STE}}$ we do not capture if a party becomes subject to coercion. Coercion, as a security property, is crucially important, especially in the context of on-line based elections. One of the main advantages of on-line based elections is the accessibility properties it provides (e.g. convenient voting process through a personal computer). On the other hand, the absence of a voting booth leaves the voters vulnerable to potential coercers, opening the door for rigged election results. Moreover, even if we believe that our protocol E-CCLESIA 1.0 satisfies some notion of verifiability due to its decentralized nature we do not provide any formal proof.

In the current version of the E-CCLESIA protocol, E-CCLESIA 1.0 , a single authority executes the setup phase. Specifically, one of the roles of this authority is to generate and distribute the RSA numbers which are mandatory for the parameterization of the dynamic accumulator. On the other hand, the authority knows the trapdoor information for these numbers (e.g. the factorization of n), allowing it to create eligible votes originated from voters that do not belong to the eligibility list. As a result, the eligibility property can be violated. Similarly, the authority can break the hiding property of the commitment scheme as it holds trapdoor information.

Moreover, in E-CCLESIA 1.0 , the computational burden for solving Astro-

labou's time puzzle is handled by the voters, making their participation in the whole duration of the protocol mandatory.

Quantum e-voting & limitations, Chapter 6: In quantum e-voting, we saw that, unless combined with some new technique, the travelling ballot protocols do not seem to provide a viable solution, as double-voting is always possible, and there is no straightforward way to guarantee privacy.

On the other hand, the distributed ballot protocols give us very strong privacy guarantees because of the entanglement between the ballot states, but verifiability against malicious talliers might be hard to achieve. In fact, one of the most intriguing questions in quantum e-voting is whether we can achieve all desired properties simultaneously. For instance, every classical definition of verifiability Cortier et al. (2016) assumes a trusted bulletin board that the participants can read, write on, and finally verify the outcome of the election. However, implementing a quantum bulletin board to achieve the same properties is not straightforward, since reading a quantum state can 'disturb' it in an irreversible way.

Regarding the protocols in Section 6.1, the cut-and-choose technique used is both inefficient and insecure. An inherent problem with this type of technique with one prover and one verifier is that the number of sampled states grows exponentially to the protocol parameter to achieve a satisfying level of security. With the protocols in Section 6.1, even if the sampled states are as many as in the single prover/single verifier case, the protocol still is not secure. Finally, most of the protocols we have seen are subject to denial-of-service attacks.

7.3 Future directions

In this section, we discuss possible solutions to the mentioned limitations. Moreover, we give ideas for further development based on the literature.

Time-lock encryption: Regarding the functionality $\mathcal{F}_{\mathsf{TLE}}$, we can enrich its glossary by allowing the encryptor to specify the recipient party of the generated ciphertext. Next, we can define a hybrid protocol to our "old" $\mathcal{F}_{\mathsf{TLE}}$ and prove that it UC realises the "new" one, making $\mathcal{F}_{\mathsf{TLE}}$ a special instance of the new functionality.

To realise constructions such as the one in Liu et al. (2018), we need to capture the concept of the *computational reference clock* (CRC) via an ideal functionality. Informally, the computational reference clock, as presented in Liu et al. (2018), is a way to define the essence of time in the execution of a protocol, besides the $\mathcal{G}_{\mathsf{clock}}$, in terms of the computational effort that is needed for solving a puzzle. For example, in Bitcoin's case, after x blocks have been announced we know that y time has elapsed in the real world. Of course, the ideal functionality for CRC, namely $\mathcal{F}_{\mathsf{CRC}}$, should "live" below the presence of the $\mathcal{G}_{\mathsf{clock}}$ and not above it, as

CRC produces puzzles when a specific amount of time has been passed and not the opposite. That means that the $\mathcal{F}_{\mathsf{CRC}}$ should be parameterized by $\mathcal{G}_{\mathsf{clock}}$. We leave the definition of $\mathcal{F}_{\mathsf{CRC}}$, the realization of the construction in Liu et al. (2018) and the proof that our game-based definition and $\mathcal{F}_{\mathsf{TLE}}$ are equivalent as future work.

Finally, in TLE constructions where the way of solving the time-puzzle is deterministic, for example, the construction in Rivest et al. (1996), might be able to be proven secure in the *Generic Group Model* (GGM), similar to Baum et al. (2021). We leave it as future work.

E-cclesia: A self tallying classical e-voting protocol: To enhance E-CCLESIA with the coercion resistant property we can borrow ideas from Juels et al. (2005). Specifically, the authors provide a mechanism, namely PETS (see Subsection 3.1.1), where the tallying authorities can discard the tallied ballots that are created with "fake" credentials without the identity of the ballot issuer be revealed. The innovative part about this technique is the fact that the coercer does not know if the voter provided him with the correct credential or the fake one, making the coerced vote not countable in the outcome of the election. The underline assumption is that the voter needs "one moment of privacy" to cast her real ballot. We believe that we can take advantage of that technique or similar end further develop E-CCLESIA 1.0 . However, this technique was meant to be used in a centralized setting, thus cannot be adapted directly to E-CCLESIA. It would be interesting to explore the trade-off between centralization and coercion resistance as a security property. We leave it as future work.

In addition, in (Canetti, 2000, Version:11/02/2020), Canetti provides modelling of coercion resistance in the UC framework. Following these steps, we can further develop our $\mathcal{F}_{\mathsf{STE}}$ functionality to capture that property. We leave both of these as future work.

We can make the generation phase of E-CCLESIA 1.0 decentralized by exploring the idea of *RSA-UFO* from Sander (1999) as mentioned in a dedicated Paragraph on page 37. Of course, we have to prove the security of such construction in a well-articulated framework, such as the UC. Alternatively, we can explore other constructions, such as the one in Groth and Kohlweiss (2015) for realizing $\mathcal{F}_{\mathsf{elig}}$. Fortunately, we can do this kind of modification quite easily thanks to our modular approach, without reproving the security of the whole E-CCLESIA protocol.

We can use other TLE constructions to realize our $\mathcal{F}_{\mathsf{vm}}$ to transfer the computational burden of solving a time-puzzle either to external entities to the protocol Liu et al. (2018) or we can take a totally different approach, outside the concept of time-puzzles, and realize $\mathcal{F}_{\mathsf{TLE}}$ via distributed key generation protocols Abe and Fehr (2004).

In addition, we can explore the possibility of defining verifiable auctions Nojoumian and Stinson (2014) via E-CCLESIA. As a matter of fact, auctions are not

opposed to coercion thus no additional enhancement of E-CCLESIA is needed for that task. Still, E-CCLESIA should be adapted to the concept of auctions.

Quantum e-voting & limitations: In Hillery et al. (2006), the authors present an interesting solution, which however might still allow an adversary to vote multiple times, by taking advantage of unused ballots. To prevent this, the protocol would need to run exponentially many times to the number of voters, which would in practice be inefficient. It might be possible to overcome this specific issue, but to prevent other forms of attacks, formal treatment is essential in well-studied models Unruh (2010); Hallgren et al. (2011); Canetti (2001a). One plausible idea of how to improve the protocol to be resistant against the attack we presented in Section 6.3 is to choose different values of δ for each ballot. If this was possible, then the attack we described would not be applicable as it relies on the fact that all ballot states share the same δ. On the other hand, this is not an easy task because the δ's are different for each ballot, it is not obvious how the tallier could compute the correct result since the protocol's correctness is based on a homomorphic property that allows getting the correct result out of the combination of all ballots. It is a very intriguing open question of whether we can achieve these two properties at the same time. In Section 6.1 the verification process of the distributed states is done in a very specific way, and it would be interesting to study if there are other options to explore to improve it. A possible solution could be to provide some type of randomness to the voters (in the form of a common random string for example), which will define if a state should be verified or used for the voting phase (a similar testing process has been studied in the past for other types of entangled states, both in theory Pappa et al. (2012) and experimentally McCutcheon et al. (2016)). However, even if the problem with the cut-and-choose technique is addressed in future works, privacy can still be violated as we have seen, and possible corrections might require the use of more advanced techniques. Moreover, we could avoid assumptions such as a simultaneously classical broadcast channel by using other quantum cryptographic primitives such as *relativistic quantum bit commitments* Lunghi et al..

Our analysis is based on e-voting protocols implemented on quantum computers without concerning classical protocols that use primitives which are believed to be quantum-resistant. So an interesting question that needs to be answered is what is the level of security these protocols del Pino et al. (2017) provide against our security definitions.

Appendix A

Supplementary material for Section 5.4

Theorem A.1. Π_{elig} *UC-realizes* $\mathcal{F}_{\mathsf{elig}}$ *in the* $\{\mathcal{F}_{\mathsf{SoK}}(\mathcal{F}^{*}_{\mathsf{acc}}, \mathcal{F}_{\mathsf{NIC}}), \mathcal{F}_{\mathsf{an.BC}}, \{\mathcal{F}^{P}_{\mathsf{cert}}\}, \mathcal{G}_{\mathsf{clock}}\}$*-hybrid model.*

Proof. In a direct proof of UC-realizability, given a real adversary $\mathcal{A}$ we have to construct a simulator $\mathcal{S}$ that can make the ideal execution indistinguishable from the real ones from the point of view of the environment $\mathcal{Z}$. Since we are in a hybrid model, $\mathcal{S}$ will internally simulate the real execution for $\mathcal{A}$ by playing the part of voters and authorities as well as the part of the hybrid functionalities. In the case of corrupted voters, it will pass any requests from $\mathcal{A}$ onto $\mathcal{F}_{\mathsf{elig}}$.

Since both $\mathcal{F}_{\mathsf{elig}}$ and all the functionalities we are working with expect the adversary to provide algorithms and parameters during setup, $\mathcal{S}$ will have to ask $\mathcal{A}$ for the algorithms for $\mathcal{F}_{\mathsf{acc}}$, $\mathcal{F}_{\mathsf{NIC}}$ and $\mathcal{F}_{\mathsf{SoK}}$ and use them to define the algorithms that $\mathcal{F}_{\mathsf{elig}}$ expects, as shown in Figure A.1.

We do not describe the reasoning behind the hybrid functionalities' algorithms here, as we are referring to the definitions established in Camenisch et al. (2016) and Chase and Lysyanskaya (2006) respectively for $\mathcal{F}_{\mathsf{NIC}}$ and $\mathcal{F}_{\mathsf{SoK}}$. `TrapCom`, `TrapOpen` and $\mathtt{Verify}_{\mathsf{NIC}}$ belong to $\mathcal{F}_{\mathsf{NIC}}$ and `SimSign`, $\mathtt{Verify}_{\mathsf{SoK}}$ and `Extract` are from $\mathcal{F}_{\mathsf{SoK}}$, and $\mathtt{Verify}_{\mathsf{cert}}$ and $\mathtt{Sign}_{\mathsf{cert}}$ are given to all $\mathcal{F}^{P}_{\mathsf{cert}}$ (which maybe different for each party). The rest have been explained in the context of $\mathcal{F}_{\mathsf{acc}}$ in Subsection 5.4.1.

Then $\mathcal{S}$ continues observing the ideal execution as allowed by $\mathcal{F}_{\mathsf{elig}}$ and its task is to use this to play the part of real voters in the simulated execution with $\mathcal{A}$. Using the commitment algorithms obtained earlier by $\mathcal{S}$, $\mathcal{F}_{\mathsf{elig}}$ can generate valid credentials for all eligible voters who have been activated in the ideal execution.

$\texttt{GenCred}(1^\lambda, \mathsf{reg.par})$:

1. $(\mathsf{comm}, \mathsf{info}) \leftarrow \texttt{TrapCom}(\mathsf{sid}, \mathsf{params}_{\mathsf{NIC}}, \mathsf{trapdoor})$.
2. Let $(\mathcal{M}, \mathcal{R}) \leftarrow \mathsf{params}_{\mathsf{NIC}}$ and compute $S \xleftarrow{\$} \mathcal{M}$.
3. $\mathsf{open} \leftarrow \texttt{TrapOpen}(\mathsf{sid}, S, \mathsf{info})$.
4. If $\texttt{Verify}_{\mathsf{NIC}}(\mathsf{params}_{\mathsf{NIC}}, \mathsf{comm}, S, \mathsf{open}) \neq \top$, return $\perp$.
5. Return $(\mathsf{cr}, \mathsf{rc}) := ((S, \mathsf{open}), (\mathsf{comm}))$.

$\texttt{UpState}(St_{\mathsf{gen}}, C)$:

1. Return $St_{\mathsf{fin}} := C$.

$\texttt{AuthBallot}(v, \mathsf{cr}, \mathsf{rc}, St_{\mathsf{fin}}, \mathsf{reg.par})$:

1. Let $(S, r) := \mathsf{cr}$, $(c, s) := \mathsf{rc}$ and $C := St_{\mathsf{fin}}$.
2. Compute $w \leftarrow f(u, C \backslash \{c\})$ and $a \leftarrow f(w, \{c\})$.
3. Compute $a' \leftarrow f(u, C)$. If $a' \neq a$, return $\perp$.
4. If $\texttt{Verify}_{\mathsf{acc}}(u, C \backslash \{c\}, w) \neq \top$, $\texttt{Verify}_{\mathsf{acc}}(w, \{c\}, a) \neq \top$ or $\texttt{Verify}_{\mathsf{acc}}(u, C, a) \neq \top$, return $\perp$.
5. Else compute $\phi \leftarrow \texttt{SimSign}(\mathcal{L}, v, (a, S))$.
6. If $\texttt{Verify}_{\mathsf{SoK}}(\mathcal{L}, v, (a, S), \phi) \neq \top$, return $\perp$.
7. Else return $\sigma := (\phi, S)$.

$\texttt{VrfyBallot}(v, \sigma, St_{\mathsf{fin}}, \mathsf{reg.par})$:

1. Let $C := St_{\mathsf{fin}}$ and $(\phi, S) := \sigma$. Compute $a \leftarrow f(u, C)$.
2. If $\texttt{Verify}_{\mathsf{acc}}(u, C, a) \neq \top$, return $\perp$.
3. Else compute $(c, w, r) \leftarrow \texttt{Extract}(\mathcal{L}, v, (a, S), \phi)$.
4. If $\texttt{Verify}_{\mathsf{acc}}(w, \{c\}, a) \neq \top$ or $\texttt{Verify}_{\mathsf{NIC}}(\mathsf{params}_{\mathsf{NIC}}, c, S, r) \neq \top$, return $\perp$.
5. Else return $x \leftarrow \texttt{Verify}_{\mathsf{SoK}}(\mathcal{L}, v, (a, S), \phi)$.

Figure A.1: Algorithms supplied by $\mathcal{S}$ to $\mathcal{F}_{\mathsf{elig}}$ during Setup, using algorithms supplied by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{acc}}$, $\mathcal{F}_{\mathsf{NIC}}$ and $\mathcal{F}_{\mathsf{SoK}}$.

On the other hand, in the ideal execution when $\mathcal{S}$ is asked by $\mathcal{F}_{\mathsf{elig}}$ if it allows the credential to be generated via public delayed output, $\mathcal{S}$ signs the credential and checks the signature with the algorithms provided by $\mathcal{A}$. Note that in the

ideal execution there are not any signatures involved with each voter's credential because we assume that $\mathcal{F}_{\mathsf{elig}}$ broadcasts and authenticates the public part of the credential at the same time. If the verification fails, $\mathcal{S}$ does not allow the generation of the credential. Similarly, when $\mathcal{S}$ is asked by $\mathcal{F}_{\mathsf{elig}}$ if it allows sending the credential via public delay output to the other parties, $\mathcal{S}$ asks $\mathcal{A}$ if it allows the broadcast as if it was from $\mathcal{F}_{\mathsf{an.BC}}$ and responds appropriately to $\mathcal{F}_{\mathsf{elig}}$. Note that $\mathcal{S}$ has no visibility into the interaction between voters and $\mathcal{F}_{\mathsf{elig}}$ for AUTH_BALLOT and VER_BALLOT commands, just like $\mathcal{A}$ would not in the real protocol. That is not the case for $\mathcal{Z}$, as it can activate parties at will and supplies their inputs, so indistinguishability of the outputs of those commands is ensured by the matching of the algorithms described earlier. In this way, properties guaranteed by $\mathcal{F}_{\mathsf{elig}}$ directly translate to (one or several) properties provided by the hybrid functionalities, e.g. eligibility relies on binding commitments and ballot unforgeability depends on the unforgeability of signatures of knowledge.

$\mathcal{S}$ proceeds as follows:

1. Send $(\mathsf{sid}, \text{CORRUPT}, \mathbf{V}_{\mathsf{corr}})$ to $\mathcal{F}_{\mathsf{elig}}$ where $\mathbf{V}_{\mathsf{corr}}$ is the set of voters that $\mathcal{A}$ has decided to corrupt at the beginning of the real execution.

2. Wait to receive $(\mathsf{sid}, \text{SETUP\_ELIG})$ from $\mathcal{F}_{\mathsf{elig}}$.

3. Simulate the 'init' stage of Π_{elig}:

   - Simulate setup of the hybrid functionalities $\mathcal{F}_{\mathsf{acc}}$, $\mathcal{F}_{\mathsf{NIC}}$, $\{\mathcal{F}_{\mathsf{cert}}^{\mathsf{SA}}\}$ and $\mathcal{F}_{\mathsf{SoK}}$ by requesting algorithms and parameters from $\mathcal{A}$ for each of them. Store all except the accumulator `Delete` algorithm in St_{gen}.
   - Use the received algorithms to define the new algorithms as in Figure A.1 and send $(\mathsf{sid}, \text{SETUP\_ELIG}, \texttt{GenCred}, \texttt{AuthBallot}, \texttt{VrfyBallot}, \texttt{UpState}, St_{\mathsf{gen}})$ to $\mathcal{F}_{\mathsf{elig}}$.
   - Wait to receive $(\mathsf{sid}, \text{ELIG\_PAR}, \mathsf{reg.par})$ from $\mathcal{F}_{\mathsf{elig}}$.

4. Begin receiving $(\mathsf{sid}, \text{GEN\_CRED}, V_i, \mathsf{rc}_i)$ from $\mathcal{F}_{\mathsf{elig}}$ for honest voters V_i. Sign and verify the credential with the algorithms provided upon request from $\mathcal{A}$ by playing the role of $\mathcal{F}_{\mathsf{cert}}^{V_i}$. If verification succeeds, then collate their public credentials in the set C.

5. Send $(\mathsf{sid}, \text{GEN\_CRED}, V_i', \mathsf{cr}_i', \mathsf{rc}_i')$ to $\mathcal{F}_{\mathsf{elig}}$ for each corrupted voter V_i', where rc_i' is the public credential generated by $\mathcal{A}$ and cr_i' is the private credential generated by $\mathcal{A}$ or a random unique value, if $\mathcal{A}$ has generated the public part in another way.

6. $\mathcal{S}$ has no visibility into AUTH_BALLOT queries between $\mathcal{F}_{\mathsf{elig}}$ and the dummy honest voters.

7. Simulate the 'cast' stage of Π_{elig}:

- Respond to $\mathcal{A}$'s requests to hybrid functionalities involved in authentication.

- Verification and linking of ballots is performed by the voters themselves, so $\mathcal{S}$ can again only respond to $\mathcal{A}$'s requests to hybrid functionalities. The answers should be consistent with those of $\mathcal{F}_{\mathsf{elig}}$, since they are computed using the same algorithms.

8. $\mathcal{S}$ can provide the authentication receipts on behalf of corrupted voters by sending (sid, AUTH_BALLOT, V_i', v_i', σ_i') to $\mathcal{F}_{\mathsf{elig}}$, using the outputs given by $\mathcal{A}$ in the simulation.

Hence the ideal and the real distribution will be the same. □

In the next figures we give the full description of the functionality for non-interactive commitments $\mathcal{F}_{\mathsf{NIC}}$ Camenisch et al. (2016) and signature of knowledge $\mathcal{F}_{\mathsf{SOK}}$ Chase and Lysyanskaya (2006).

The non-interactive commitment functionality $\mathcal{F}_{\mathsf{NIC}}(\mathbf{P})$.

■ Upon receiving $(\mathsf{sid}, \textsc{Setup})$ from $P \in \mathbf{P}$ it does:

1. If $(sid, params, trapdoor, algs)$ already stored, it inserts P in the set $\mathbb{P}$ (initially empty) and sends $(\mathsf{sid}, \textsc{Parameters}, params)$ to P as delayed output.
2. Else, it generates a random $ssid$ and stores $(ssid, P)$. Then, it sends $(\mathsf{sid}, \textsc{Request\_Algorithms}, ssid)$ to $\mathcal{S}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Request\_Algorithms}, ssid, params, trapdoor, \texttt{TrapCom}, \texttt{TrapOpen}, \texttt{Verify})$ from $\mathcal{S}$ it does:

1. If no $(ssid, P)$ is stored for some P, it sends $(\mathsf{sid}, \textsc{Setup}, \perp)$ to P.
2. It Deletes $(ssid, P)$.
3. If $(\mathsf{sid}, params, trapdoor, algs)$ not already stored, it stores the tuple.
4. It includes P in the set $\mathbb{P}$.
5. It sends $(\mathsf{sid}, \textsc{Parameters}, params)$ to P as delayed output.

■ Upon receiving $(\mathsf{sid}, \textsc{Verification\_Algorithm})$ from party $P \in \mathbf{P}$, it sends $(\mathsf{sid}, \textsc{Verification\_Algorithm}, \texttt{Verify}(\mathsf{sid}, params, \cdot))$ to P. ■ Upon receiving $(\mathsf{sid}, \textsc{Commit}, msg)$ from party $P \in \mathbf{P}$, it does:

1. If $P \notin \mathbb{P}$ or $msg \notin M$ (for M defined in $params$), it sends $(\mathsf{sid}, \textsc{Commit}, msg, \perp)$ to P.
2. It computes $(comm, info) \leftarrow \texttt{TrapCom}(\mathsf{sid}, params, trapdoor)$.
3. If there is an entry $[comm, msg', open', 1]$ with $msg \neq msg'$, it sends $(\mathsf{sid}, \textsc{Commit}, msg, \perp)$ to P .
4. It computes $open \leftarrow \texttt{TrapOpen}(\mathsf{sid}, msg, info)$.
5. If $\texttt{Verify}(\mathsf{sid}, params, comm, msg, open) \neq 1$, it sends $(\mathsf{sid}, \textsc{Commit}, msg, \perp)$ to P.
6. It records the entry $[comm, msg, open, 1]$.
7. It sends $(\mathsf{sid}, \textsc{Commitment}, comm, open)$ to P.

■ Upon receiving $(\mathsf{sid}, \textsc{Verify}, comm, msg, open)$ from $P \in \mathbf{P}$, it does:

1. If $P \notin \mathbb{P}$, $msg \notin M$ or $open \notin R$ (for M, R defined in *params*), it sends $(\mathsf{sid}, \texttt{Verify}, comm, msg, open, \perp)$ to P.
2. If there is an entry $[comm, msg, open, u]$, set $v \leftarrow u$. Else it does:

   If there is $[comm, msg', open', 1]$ with $msg \neq msg'$, it sets $v \leftarrow 0$.

   Else it sets $v \leftarrow \texttt{Verify}(sid, params, comm, msg, open)$.
3. It sends $(\mathsf{sid}, \texttt{Verified}, v)$ to P.

Figure A.2: The non-interactive commitment functionality $\mathcal{F}_{\mathsf{NIC}}(\mathbf{P})$ interacting with parties in $\mathbf{P}$ and the simulator $\mathcal{S}$.

The signature of knowledge functionality $\mathcal{F}_{\mathsf{SOK}}(\mathcal{F}_{\mathsf{NIC}}, \mathcal{F}_{\mathsf{acc}})$.

■ Upon receiving $(\mathsf{sid}, \textsc{Setup})$ from party $P \in \mathbf{P}$, it does:

If $\mathsf{sid} = (\mathcal{F}_{\mathsf{NIC}}, \mathcal{F}_{\mathsf{acc}}, \mathsf{sid}_{\mathsf{NIC}}, \mathsf{sid}_{\mathsf{acc}}, \mathsf{sid}')$ for some $\mathsf{sid}_{\mathsf{NIC}}, \mathsf{sid}_{\mathsf{acc}}, \mathsf{sid}'$:

(a) It sets $\mathcal{M}_{\mathsf{NIC}} \leftarrow \texttt{GetLanguage}(\mathcal{F}_{\mathsf{NIC}}, \mathsf{sid}_{\mathsf{NIC}})$.

(b) It sets $\mathcal{M}_{\mathsf{acc}} \leftarrow \texttt{GetLanguage}(\mathcal{F}_{\mathsf{acc}}, sid_{\text{ACC}})$.

(c) It defines $\mathcal{M}_L((a, S), (c, w, r)) = \mathcal{M}_{\mathsf{acc}}((a, \{c\}), w) \wedge \mathcal{M}_{\mathsf{NIC}}((c, S), r)$.

(d) It sends $(\mathsf{sid}, \textsc{Setup})$ to $\mathcal{S}$.

> $\texttt{GetLanguage}(\mathcal{F}_0, \mathsf{sid}_0)$:
>
> - It creates an instance of $\mathcal{F}_0^w$ with session id sid_0.
> - It sends $(\mathsf{sid}_0, \textsc{Setup})$ and $(\mathsf{sid}_0, \textsc{VerificationAlgorithm})$ to $\mathcal{F}_0$ on behalf of P.
> - Upon receiving $(\mathsf{sid}_0, \texttt{VerificationAlgorithm}, \mathcal{M})$ from $\mathcal{F}_0$, it outputs $\mathcal{M}$.

■ Upon receiving input $(\mathsf{sid}, \textsc{Algorithms}, \texttt{Verify}, \texttt{Sign}, \texttt{SimSign}, \texttt{Extract})$ from $\mathcal{S}$ for deterministic polynomial time (DPT) ITM `Verify` and the rest PPT ITMs:

It stores the algorithms and sends $(\mathsf{sid}, \textsc{Algorithms}, \texttt{Sign}(\mathcal{M}_L, \cdot, \cdot, \cdot), \texttt{Verify}(\mathcal{M}_L, \cdot, \cdot, \cdot))$ to P.

■ Upon receiving $(\mathsf{sid}, \textsc{Sign}, m, (a, S), (c, w, r))$ from party $P \in \mathbf{P}$, it does:

1. If $\mathcal{F}_{\mathsf{acc}}$ accepts $(\mathsf{sid}_{\mathsf{acc}}, \mathtt{Verify}, a, \{c\}, w)$ and $\mathcal{F}_{\mathsf{NIC}}$ accepts $(\mathsf{sid}_{\mathsf{NIC}}, \mathtt{Verify}, c, S, r)$ when queried by P, it does:
   (a) If $\mathcal{M}_L((a, S), (c, w, r)) \neq 1$, it sends $(\mathsf{sid}, \textsc{Sign}, m, (a, S), (c, w, r), \perp)$ to P.
   (b) It computes $\sigma \leftarrow \mathtt{SimSign}(\mathcal{M}_L, m, (a, S))$.
   (c) If $\mathtt{Verify}(\mathcal{M}_L, m, (a, S), \sigma) \neq 1$, it sends $(\mathsf{sid}, \textsc{Sign}, m, (a, S), (c, w, r))$ to P.
   (d) It records $[m, (a, S), \sigma]$ and outputs $(\mathsf{sid}, \textsc{Signature}, m, (a, S), \sigma)$ to P.

   $\mathcal{F}_{\mathsf{SOK}}$ forwards all queries between $\mathcal{F}_{\mathsf{acc}}, \mathcal{F}_{\mathsf{NIC}}$ and P or $\mathcal{S}$ (the formal details of this can be found in Chase and Lysyanskaya (2006).

■ Upon receiving $(\mathsf{sid}, \textsc{Verify}, m, (a, S), \sigma)$ from party P, it does:

1. If $[m, (a, S), \sigma']$ is recorded for some σ':

   It sends $(\mathsf{sid}, \textsc{Verified}, \mathtt{Verify}(\mathcal{M}_L, m, (a, S), \sigma))$ to P.

   Else:

   (a) It sets $(c, w, r) \leftarrow \mathtt{Extract}(\mathcal{M}_L, m, (a, S), \sigma)$.
   (b) If $\mathcal{M}_L((a, S), (c, w, r)) = 1$, it does:
      i. If $\mathcal{F}_{\mathsf{acc}}$ does not accept $(\mathsf{sid}_{\mathsf{acc}}, \textsc{Verify}, a, \{c\}, w)$ or $\mathcal{F}_{\mathsf{NIC}}$ does not accept $(\mathsf{sid}_{\mathsf{NIC}}, \textsc{Verify}, c, S, r)$, it sends $(\mathsf{sid}, \textsc{Verify}, m, (a, S), \sigma, \perp)$ to P.
      ii. It outputs $(\mathsf{sid}, \textsc{Verified}, \mathtt{Verify}(\mathcal{M}_L, m, (a, S), \sigma))$ to V.

   Else if $\mathtt{Verify}(\mathcal{M}_L, m, (a, S), \sigma) \neq 1$:

   It outputs $(\mathsf{sid}, \textsc{Verified}, 0)$ to P.

   Else it sends $(\mathsf{sid}, \textsc{Verify}, m, (a, S), \sigma, \perp)$ to P.

Figure A.3: The signature of knowledge functionality $\mathcal{F}_{\mathsf{SOK}}$ parameterized by the functionalities $\mathcal{F}_{\mathsf{NIC}}$,$\mathcal{F}_{\mathsf{acc}}$, interacting with parties in $\mathbf{P}$ and the simulator $\mathcal{S}$.

Appendix B

Supplementary material for Section 6.3

B.1 Proof of attack on distributed ballot protocols

Now we give detailed proofs of the theorems and lemmas of Section 6.3.

Lemma B.1. *11 Let* $\Theta^{v}_{D,\delta} \in [0, 2\pi]$ *be the continuous random variable that describes the outcome of the measurement of a vote state* $|\psi(\theta_v)\rangle, v \in \{y, n\}$ *using operators*

$$E(\theta) = \frac{D}{2\pi} |\Phi(\theta)\rangle \langle\Phi(\theta)| \tag{B.1}$$

where $|\Phi(\theta)\rangle = \frac{1}{\sqrt{D}} \sum_{j=0}^{D-1} e^{ij\theta} |j\rangle$. *It holds that:*

$$Pr[x_l < \Theta^{v}_{D,\delta} < x_{l+w}] = \frac{1}{2\pi D} \int_{x_l}^{x_{l+w}} \frac{\sin^2[D(\theta - \theta_v)/2]}{\sin^2[(\theta - \theta_v)/2]} d\theta \tag{B.2}$$

Proof.

$$Pr[x_l < \Theta^v_{D,\delta} < x_{l+w}] = \langle\phi(\theta_v)| \int_{x_l}^{x_{l+w}} E(\theta)d\theta \, |\phi(\theta_v)\rangle$$
$$= \int_{x_l}^{x_{l+w}} \langle\phi(\theta_v)| \, E(\theta) \, |\phi(\theta_v)\rangle \, d\theta$$
$$= \frac{D}{2\pi D^2} \int_{x_l}^{x_{l+w}} |\sum_{j=0}^{D-1} e^{(\theta-\theta_v)ij}|^2 d\theta$$
$$= \frac{1}{2\pi D} \int_{x_l}^{x_{l+w}} ([\sum_{j=0}^{D-1} \cos[(\theta-\theta_v)j]]^2$$
$$+ [\sum_{j=0}^{D-1} \sin[(\theta-\theta_v)j]]^2) d\theta$$

For any $x \in \mathbb{R}$, the following two equations hold:

$$\sum_{j=0}^{D-1} \cos[jx] = \frac{\sin[Dx/2]}{\sin[x/2]} \cos[(D-1)x/2]$$

$$\sum_{j=0}^{D-1} \sin[jx] = \frac{\sin[Dx/2]}{\sin[x/2]} \sin[(D-1)x/2]$$

So finally we have:

$$Pr[x_l < \Theta^v_{D,\delta} < x_{l+w}] = \frac{1}{2\pi D} \int_{x_l}^{x_{l+w}} \frac{\sin^2[D(\theta-\theta_v)/2]}{\sin^2[(\theta-\theta_v)/2]} d\theta$$

□

Lemma B.2. *12 Let $|\psi(\theta_v)\rangle$ be a voting state of the protocol. Then it holds:*

$$Pr[x_{l_v} < \Theta^v_{D,\delta} < x_{l_v+1}] > 0.405$$

Proof. A simple change of variables in Eq.(B.2) gives us:

$$Pr[x_{l_v} < \Theta^v_{D,\delta} < x_{l_v+1}] = \frac{1}{2\pi D} \int_0^{2\pi/D} \frac{\sin^2[D(\theta-\delta)/2]}{\sin^2[(\theta-\delta)/2]} d\theta$$

By setting $(\theta-\delta)/2 = y$, we get:

$$Pr[x_{l_v} < \Theta^v_{D,\delta} < x_{l_v+1}] = \frac{1}{\pi D} \int_{-\delta/2}^{(2\pi/D-\delta)/2} \frac{\sin^2[Dy]}{\sin^2[y]} dy$$

The above is just a function of δ, which we denote as $F(\delta)$. In order to lower-bound $F(\delta)$ we need to find its derivative:

$$\frac{dF(\delta)}{d\delta} = \frac{1}{2\pi D}\left(\frac{\sin^2[D\delta/2]}{\sin^2[\delta/2]} - \frac{\sin^2[D\delta/2]}{\sin^2[(2\pi/D-\delta)/2]}\right)$$

It is easy to check that:

$$\begin{aligned}
&\frac{dF(\delta)}{d\delta} = 0, \quad \text{when} \quad \delta = 0 \quad \text{or} \quad \delta = \pi/D \\
&\frac{dF(\delta)}{d\delta} > 0, \quad \text{when} \quad 0 < \delta < \pi/D \\
&\frac{dF(\delta)}{d\delta} < 0, \quad \text{when} \quad \pi/D < \delta < 2\pi/D
\end{aligned}$$

It also holds that $F(0) = F(2\pi/D)$, so the minimum extreme points of our function are equal. As a result we have:

$$F(\delta) \geq \lim_{\delta \to 0^-} F(\delta) = F(0) \tag{B.3}$$

From the fact that:

$$\begin{aligned}
&|\sin[x]| \leq |x|, \forall x \in \mathbb{R} \\
&|\sin[x]| \geq |(2/\pi)x|, \forall x \in [0, \pi/2] \\
&|\sin[x]| \geq |-(2/\pi)x + 2|, \forall x \in [\pi/2, \pi]
\end{aligned}$$

It follows:

$$\begin{aligned}
F(0) &\geq \frac{1}{\pi D}\int_0^{\frac{\pi}{2D}} \left(\frac{2}{\pi D y}\right)^2 / y^2 dy + \int_{\frac{\pi}{2D}}^{\frac{\pi}{D}} \left(\frac{2}{\pi D y} + 2\right)^2 / y^2 dy \\
&\geq \frac{4}{\pi^2} \\
&> 0.405
\end{aligned}$$

□

Now in order to prove lemma 13, we need the following proposition:

Proposition B.2.1. *$\forall x \in [-2\pi, 2\pi]$ it holds that:*

$$\sin^2[x] > \sum_{n=1}^{20} (-1)^{n+1} \frac{2^{2n-1} x^{2n}}{(2n)!} \tag{B.4}$$

Proof. From the Taylor series expansion at point 0 of $\cos[x]$, we know that:

$$\cos[x] = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}, \quad \forall x \in \mathbb{R}$$

Then:

$$\sin^2[x] = \frac{1}{2} - \frac{\cos[2x]}{2} = \frac{1}{2} - \frac{1}{2}\sum_{n=0}^{\infty}(-1)^n \frac{2^{2n}x^{2n}}{(2n)!}$$
$$= \sum_{n=1}^{\infty}(-1)^{n+1}\frac{2^{2n-1}x^{2n}}{(2n)!}$$

Given the above equation, in order to prove Eq.(B.4), we simply need to show:

$$\sum_{n=21}^{\infty}(-1)^{n+1}\frac{2^{2n-1}x^{2n}}{(2n)!} > 0$$

If we think of the above as a sum of terms a_n $(n = 21, \ldots, \infty)$, for integer $j \geq 10$, it holds that:

$$a_n > 0, \text{ when } n = 2j + 1,$$
$$a_n < 0, \text{ when } n = 2j.$$

We therefore need to prove that $\sum_{n=21}^{\infty} a_n > 0$, which in turn is equivalent to proving that:

$$|a_n| > |a_{n+1}| \iff 2^{2n-1}x^{2n}/(2n)! > 2^{2n+1}x^{2n+2}/(2n+2)!$$
$$\iff 1 > 4x^2/((2n+1)(2n+2))$$
$$\iff (2n+1)(2n+2)/4 > x^2$$

In this case, the above holds, because the minimum value of n is 21 and the maximum value of x^2 is $4\pi^2$. □

Lemma B.3. *13 Let $|\psi(\theta_v)\rangle$ be a voting state of the protocol. Then it holds:*

$$Pr[x_{l_v-1} < \Theta^v_{D,\delta} < x_{l_v+2}] > 0.9$$

Proof. We follow exactly the same procedure as lemma 12 and get:

$$Pr[x_{l_v-1} < \Theta^v_{D,\delta} < x_{l_v+2}] \tag{B.5}$$
$$= \frac{1}{2\pi D}\int_{x_{l_v-1}}^{x_{l_v+2}} \frac{\sin^2[D(\theta-\theta_v)/2]}{\sin^2[(\theta-\theta_v)/2]}d\theta$$
$$= \frac{1}{2\pi D}\int_{-2\pi/D}^{4\pi/D} \frac{\sin^2[D(\theta-\delta)/2]}{\sin^2[(\theta-\delta)/2]}d\theta$$
$$= \frac{1}{\pi D}\int_{-\pi/D-\delta/2}^{2\pi/D-\delta/2} \frac{\sin^2[Dy]}{\sin^2[y]}dy \tag{B.6}$$

where $(\theta - \delta)/2 = y$. Again the above probability depends only on δ and can therefore be denoted with $F(\delta)$. In a similar way as before, we can prove that the minimum of this function is at $\delta = 0$ and compute $F(0)$.

$$\begin{aligned}
F(0) &= \frac{1}{\pi D}\int_{-\pi/D}^{2\pi/D}\frac{\sin^2[Dy]}{\sin^2[y]}dy \\
&\geq \frac{1}{\pi D}\int_{-\pi/D}^{2\pi/D}\frac{\sum_{n=1}^{20}\frac{(-1)^{n+1}2^{2n-1}(Dy)^{2n}}{(2n)!}}{y^2}dy \\
&= \frac{1}{\pi D}\sum_{n=1}^{20}\int_{-\pi/D}^{2\pi/D}\frac{(-1)^{n+1}2^{2n-1}D^{2n}y^{2n}}{y^2(2n)!}dy \\
&= \frac{1}{\pi D}\sum_{n=1}^{20}\frac{(-1)^{n+1}2^{2n-1}D^{2n}}{(2n)!}\int_{-\pi/D}^{2\pi/D}y^{2(n-1)}dy \\
&= \frac{1}{\pi D}\sum_{n=1}^{20}\frac{(-1)^{n+1}2^{2n-1}D^{2n}}{(2n)!}[y^{2n-1}/(2n-1)]_{-\pi/D}^{2\pi/D} \\
&= \sum_{n=1}^{20}\frac{(-1)^{n+1}2^{2n-1}}{(2n)!}\frac{\pi^{2n-2}(2^{2n-2}+1)}{2n-1} \\
&\approx 0.9263
\end{aligned} \tag{B.7}$$

□

Theorem B.4. *14 With overwhelming probability in the number of voters N, algorithm 1 includes l_v in the* `Solution` *vector (i.e. it measures a value in the interval $[x_{l_v}, x_{l_v+1}]$ more than 40% of the time).*

$$\Pr[\texttt{Solution}[0] = l_v \vee \texttt{Solution}[1] = l_v] > 1 - 1/exp(\Omega(N))$$

Proof. We can see each measurement that algorithm 1 performs at each vote state $|\psi(\theta_v)\rangle$, as an independent Bernoulli trial X_l with probability of success $p_l = \Pr[x_l < \Theta^v_{D,\delta} < x_{l+1}]$. Then the value of `Record`$[l]$ follows the binomial distribution:

$$X_{\texttt{Record}[l]} \sim B(\frac{\varepsilon N}{2}, p_l)$$

We can therefore compute:

$$\begin{aligned}
&\Pr\left[\texttt{Solution}[0] = l_v \vee \texttt{Solution}[1] = l_v\right] \\
&= \Pr\left[\texttt{Record}[l_v] \geq 0.4\varepsilon N/2\right] \\
&\geq 1 - \Pr\left[\texttt{Record}[l_v] \leq 0.4\varepsilon N/2\right] \\
&\overset{1}{=} 1 - \Pr\left[\texttt{Record}[l_v] \leq (1-\gamma)p_{l_v}\varepsilon N/2\right] \\
&\overset{2}{\geq} 1 - exp(-\gamma^2 p_{l_v}\varepsilon N/6) \\
&= 1 - (exp(-\gamma^2 p_{l_v}\varepsilon/6))^N \\
&= 1 - 1/exp(\Omega(N))
\end{aligned}$$

□

Theorem B.5. *15 With negligible probability in the number of voters N, algorithm 1 includes a value other than $(l_v - 1, l_v, l_v + 1)$ in the* `Solution` *vector, i.e. $\forall w \in \{0, \ldots, l_v - 2, l_v + 2, \ldots, D - 1\}$:*

$$Pr[\texttt{Solution}[0] = w \vee \texttt{Solution}[1] = w] < 1/exp(\Omega(N))$$

Proof. Let $w \in \{0, \ldots, D-1\} \setminus \{l_v - 1, l_v, l_v + 1\}$, then it holds:

$$\begin{aligned}
&Pr[\texttt{Solution}[0] = w \vee \texttt{Solution}[1] = w] \\
&= Pr[X_{\texttt{Record}[w]} \geq 0.4\varepsilon N/2]
\end{aligned}$$

We know from lemma 13 that $p_w < 0.1$, so $\exists \gamma > 0$ such that:[1]

$$\begin{aligned}
&Pr[X_{\texttt{Record}[w]} \geq 0.4\varepsilon N/2] \\
&= Pr[X_{\texttt{Record}[w]} \geq (1+\gamma)p_w\varepsilon N/2] \\
&< exp(-\gamma p_w \varepsilon N/6) \\
&= (exp(-\gamma p_w \varepsilon/6))^N \\
&= 1/exp(\Omega(N))
\end{aligned}$$

□

Lemma B.6. *16 With overwhelming probability in N, the* `Solution` *vector in algorithm 1, is equal to $[l_v - 1, l_v]$, $[l_v, \text{“}Null\text{”}]$ or $[l_v, l_v + 1]$. Specifically,*

$$\begin{aligned}
&\Pr[\texttt{Solution} \in \{[l_v - 1, l_v], [l_v, \text{“}Null\text{”}], [l_v, l_v + 1]\}] \\
&> 1 - 1/exp(\Omega(N))
\end{aligned}$$

[1] $p_{l_v} > 0.405 \implies \exists \gamma > 0$ s.t $0.4 = (1-\gamma)p_{l_v}$

[2] The Chernoff bound for a random variable $X \sim B(N, p)$ and expected value $E[X] = \mu$ is: $Pr[X \leq (1-\gamma)\mu] \leq exp(-\gamma^2\mu/3)$

[1] The Chernoff bound for a random variable $X \sim B(N, p)$ and expected value $E[X] = \mu$ is: $Pr[X \leq (1+\gamma)\mu] \leq exp(-\gamma\mu/3), \gamma > 1$

Proof. Let as define the following events:

$$A = \big[\texttt{Solution}[0] = w \vee \texttt{Solution}[1] = w, \\ w \in \{0, \ldots, l_v - 2, l_v + 2, \ldots, D-1\}\big]$$

$$B = \big[\texttt{Solution}[0] = l_v \vee \texttt{Solution}[1] = l_v\big]$$

Since the cases $\texttt{Solution} = [l_v, l_v - 1]$ and $\texttt{Solution} = [l_v + 1, l_v]$ are impossible from the construction of the algorithm, from theorems 14 and 15 it holds:

$$\begin{aligned}
&\Pr[\texttt{Solution} \in \{[l_v - 1, l_v], [l_v, \text{“}Null\text{”}], [l_v, l_v + 1]\}] \\
&= \Pr[B \wedge \neg A] \\
&= \Pr[B] - \Pr[B \wedge A] \\
&> 1 - 1/exp(\Omega(N))
\end{aligned}$$

□

Lemma B.7. *Let $|\psi(\theta_v)\rangle$ be a voting state with $\delta \in [0, 2\pi/D)$ and $l_v = D - 1$,where δ is a continuous random variable .Then it holds:*

$$\Pr[x_{D-2} < \Theta^v_{D,\delta} < x_D] + Pr[x_0 < \Theta^v_{D,\delta} < x_1] > 0.9$$

Proof.

$$\Pr[x_0 < \Theta^v_{D,\delta} < x_1] \tag{B.8}$$

$$= 1/(2\pi D) \int_{x_0}^{x_1} (\frac{Sin[D/2(\theta - \theta_v)]}{Sin[1/2(\theta - \theta_v)]})^2 d\theta \tag{B.9}$$

Now we set $\theta = \theta - x_D$ to B.9 and we have:

$$\Pr[x_0 < \Theta^v_{D,\delta} < x_1] \tag{B.10}$$

$$= 1/(2\pi D) \int_{x_D}^{x_D + x_1} (\frac{Sin[-D\pi + D/2(\theta - \theta_v)]}{Sin[-\pi + 1/2(\theta - \theta_v)]})^2 d\theta \tag{B.11}$$

$$= 1/(2\pi D) \int_{x_D}^{x_D + x_1} (\frac{Sin[D/2(\theta - \theta_v)]}{Sin[1/2(\theta - \theta_v)]})^2 d\theta \tag{B.12}$$

Finally we have:

$$\Pr[x_{D-2} < \Theta^v_{D,\delta} < x_D] + \Pr[x_0 < \Theta^v_{D,\delta} < x_1] \tag{B.13}$$

$$= 1/(2\pi D) \int_{x_{D-2}}^{x_D + x_1} (\frac{Sin[D/2(\theta - \theta_v)]}{Sin[1/2(\theta - \theta_v)]})^2 d\theta \tag{B.14}$$

$$= 1/(2\pi D) \int_{-2\pi/D}^{4\pi/D} \frac{Sin^2[D(\theta - \delta)/2]}{Sin^2[(\theta - \delta)/2]} d\theta \tag{B.15}$$

From lemma 13 this integral is at least 0.9. The proof is similar for $l_v = 0$. □

Lemma B.8. *Let* `Solution` *be the matrix of algorithm 1, then it holds:*

$$\begin{aligned}&\Pr[\texttt{Solution} \in \{\{l_v - 1, l_v\}, \{l_v\}, \{l_v, l_v + 1\}\}]\\ &= \Pr[\texttt{Solution} \in \{[l_v - 1, l_v], [l_v], [l_v, l_v + 1]\}]\end{aligned}$$

Proof. (sketch)It holds that:

$$\Pr[\texttt{Solution} \in \{\{l_v - 1, l_v\}, \{l_v\}, \{l_v, l_v + 1\}\}] \tag{B.16}$$
$$= \Pr[\texttt{Solution} \in \{l_v - 1, l_v\}] \tag{B.17}$$
$$+ \Pr[\texttt{Solution} \in \{l_v\}] \tag{B.18}$$
$$+ \Pr[\texttt{Solution} \in \{l_v, l_v + 1\}] \tag{B.19}$$

We need to prove that:

$$\Pr[\texttt{Solution} \in \{l_v - 1, l_v\}] = \Pr[\texttt{Solution} = [l_v - 1, l_v]] \tag{B.20}$$

From the construction of the algorithm 1 we know that:

$$\Pr[\texttt{Solution} = [l_v, l_v - 1] | \texttt{Solution} \in \{l_v - 1, l_v\}] = 0 \tag{B.21}$$

This is true because the values of the `Solution` are from the matrix `Record` in a progressive manner. So under the assumption that both $l_v, l_v - 1$ had appeared at least 40% times, they inserted in a progressive order. The only time they will not is the case in which $l_v = 0$ and $l_v - 1 = D - 1$. At first, the order is $[0, D - 1]$, but because of the special condition we had in our algorithm the order switches to $[D - 1, 0]$.

It holds that:

$$\Pr[\texttt{Solution} = [l_v, l_v - 1] | \texttt{Solution} \in \{l_v - 1, l_v\}] \tag{B.22}$$
$$= \Pr[\texttt{Solution} = [l_v, l_v - 1]] + \Pr[\emptyset] \tag{B.23}$$
$$= \Pr[\texttt{Solution} = [l_v, l_v - 1]] \tag{B.24}$$
$$= 0 \tag{B.25}$$

Similar are the other cases. □

References

Masayuki Abe and Serge Fehr. Adaptively secure feldman vss and applications to universally-composable threshold cryptography. In Matt Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, pages 317–334, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

Ben Adida. Helios: Web-based open-audit voting. In *USENIX security symposium*, volume 17, pages 335–348, 2008.

Joël Alwen, Rafail Ostrovsky, Hong-Sheng Zhou, and Vassilis Zikas. Incoercible multi-party computation and universally composable receipt-free voting. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, pages 763–780, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. ISBN 978-3-662-48000-7.

Myrto Arapinis, Véronique Cortier, and Steve Kremer. When are three voters enough for privacy properties? 21st European Symposium on Research in Computer Security, Heraklion, Crete, Greece, 2016. Springer.

Myrto Arapinis, Elham Kashefi, Nikolaos Lamprou, and Anna Pappa. Definitions and analysis of quantum e-voting protocols. *arXiv*, pages arXiv–1810, 2018.

Charles Arthur. Estonian e-voting shouldn't be used in european elections, say security experts, 2014. URL https://www.theguardian.com/technology/2014/may/12/estonian-e-voting-security-warning-european-elections-research.

Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In *CRYPTO*, pages 324–356, 2017.

Foteini Baldimtsi, Ran Canetti, and Sophia Yakoubov. Universally composable accumulators. *IACR Cryptology ePrint Archive*, 2018:1241, 2018.

Jonathan Bannet, David W. Price, Algis Rudys, Justin Singer, and Dan S. Wallach. Hack-a-vote: Security issues with electronic voting systems. *IEEE Security & Privacy*, 2(1):32–37, 2004.

Ning Bao and Nicole Yunger Halpern. Quantum voting and violation of arrow's impossibility theorem. *Phys. Rev. A*, 95:062306, Jun 2017.

Howard Barnum, Claude Crépeau, Daniel Gottesman, Adam Smith, and Alain Tapp. Authentication of quantum messages. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pages 449–458. IEEE, 2002.

Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. Craft: Composable randomness and almost fairness from time. Cryptology ePrint Archive, Report 2020/784, 2020. https://eprint.iacr.org/2020/784.

Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. Tardis: A foundation of time-lock puzzles in uc. Advances in Cryptology - EUROCRYPT, 2021. https://eprint.iacr.org/2020/537.

Charles H. Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. In *Proceedings of IEEE International Conference on Computers, Systems and Signal Processing*, volume 175, page 8. New York, 1984.

David Bernhard, Veronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. Sok: A comprehensive analysis of game-based ballot privacy definitions. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 499–516. IEEE, 2015.

Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: New generation of memory-hard functions for password hashing and other applications. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 292–302, 2016.

Marianna Bonanome, Vladimír Bužek, Mark Hillery, and Mário Ziman. Toward protocols for quantum-ensured privacy and secure voting. volume 84, page 022331. APS, 2011.

Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156189, October 1988. ISSN 0022-0000.

Anne Broadbent and Alain Tapp. Information-theoretic security without an honest majority. In Kaoru Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2007*, pages 410–426. Springer Berlin Heidelberg, 2007.

Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, pages 61–76, 2002.

Jan Camenisch, Maria Dubovitskaya, and Alfredo Rial. UC commitments for modular protocol design and applications to revocation and attribute tokens. In *CRYPTO*, 2016.

Jan Camenisch, Anja Lehmann, Gregory Neven, and Kai Samelin. Uc-secure non-interactive public-key encryption. In *CSF 2017*, pages 217–233, 2017.

Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. https://eprint.iacr.org/2000/067.

Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001a.

Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001b.

Ran Canetti. Universally composable signatures, certification and authentication. *IACR Cryptology ePrint Archive*, 2003:239, 2003.

Ran Canetti and Rosario Gennaro. Incoercible multiparty computation. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 504–513. IEEE, 1996.

Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 337–351. Springer, 2002.

Ran Canetti, Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 98–116, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *TCC*, pages 61–85, 2007a.

Ran Canetti, Rafael Pass, and Abhi Shelat. Cryptography from sunspots: How to use an imperfect reference string. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 249–259, 2007b.

Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In *CRYPTO*, pages 78–96, 2006.

David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology*, pages 199–203, Boston, MA, 1983. Springer US. ISBN 978-1-4757-0602-4.

David Chaum. Surevote: technical overview. In *Proceedings of the workshop on trustworthy elections (WOTE01)*, 2001.

David Chaum. Secret-ballot receipts: True voter-verifiable elections. *Security & Privacy, IEEE*, 2:38 – 47, 02 2004.

David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II: end-to-end verifiability by voters of optical scan elections through confirmation codes. *IEEE Trans. Information Forensics and Security*, 4(4):611–627, 2009.

Jung Hee Cheon, Nicholas Hopper, Yongdae Kim, and Ivan Osipkov. Timed-release and key-insulated public key encryption. In Giovanni Di Crescenzo and Avi Rubin, editors, *Financial Cryptography and Data Security*, pages 191–205, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-46256-9.

Benoît Chevallier-Mames, Pierre-Alain Fouque, David Pointcheval, Julien Stern, and Jacques Traoré. On some incompatible properties of voting schemes. In David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter Y. A. Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida, editors, *Towards Trustworthy Elections: New Directions in Electronic Voting*, pages 191–199, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. A homomorphic lwe based e-voting scheme. In *International Workshop on Post-Quantum Cryptography*, pages 245–265. Springer, 2016.

Michael Clarkson, Stephen Chong, and Andrew Myers. Civitas: A secure remote voting system. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2008.

V. Cortier, D. Galindo, R. Ksters, J. Mller, and T. Truderung. Sok: Verifiability notions for e-voting protocols. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 779–798, 2016.

Véronique Cortier and Ben Smyth. Attacking and fixing helios: An analysis of ballot secrecy. *J. Comput. Secur.*, 21(1):89148, January 2013. ISSN 0926-227X.

Véronique Cortier, P. Gaudry, and S. Glondu. Belenios: A simple private and verifiable electronic voting system. In Guttman J.and Landwehr C.and Meseguer J.and Pavlovic D., editor, *Foundations of Security, Protocols, and Equational Reasoning*, volume 11565 of *Lecture Notes in Computer Science*. Springer, Cham, 2019.

Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag, Berlin, Heidelberg, 2002. ISBN 3540425802.

Rafaël del Pino, Vadim Lyubashevsky, Gregory Neven, and Gregor Seiler. Practical quantum-safe voting from lattices. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 15651581, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349468.

Stephanie Delaune, Steve Kremer, and Mark Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *19th IEEE Computer Security Foundations Workshop*, pages 28–42, 2006.

David Derler, Christian Hanser, and Daniel Slamanig. Revisiting cryptographic accumulators, additional properties and relations to other primitives. In *CT-RSA*, 2015a.

David Derler, Christian Hanser, and Daniel Slamanig. Revisiting cryptographic accumulators, additional properties and relations to other primitives. In *CT-RSA*, 2015b.

W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644654, September 2006. ISSN 0018-9448.

Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM04, page 21, USA, 2004. USENIX Association.

Shahar Dolev, Itamar Pitowsky, and Boaz Tamir. A quantum secret ballot. *arXiv preprint quant-ph/0602087*, 2006.

Frédéric Dupuis, Jesper Buus Nielsen, and Louis Salvail. Actively secure two-party evaluation of any quantum operation. Cryptology ePrint Archive, Report 2012/304, 2012. https://eprint.iacr.org/2012/304.

Cynthia Dwork. *Non-Malleability*, pages 849–852. Springer US, Boston, MA, 2011. ISBN 978-1-4419-5906-5.

A. Einstein, B. Podolsky, and N. Rosen. Can quantum-mechanical description of physical reality be considered complete? *Phys. Rev.*, 47:777–780, 1935.

Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In George Robert Blakley and David Chaum, editors, *Advances in Cryptology*, pages 10–18, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg. ISBN 978-3-540-39568-3.

Saghar Estehghari and Yvo Desmedt. Exploiting the client vulnerabilities in internet e-voting systems: Hacking helios 2.0 as an example. In *EVT/WOTE*, 2010.

Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security*, pages 436–454, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-662-45472-5.

Nelly Fazio and Antonio Nicolosi. Cryptographic accumulators: Definitions, constructions and applications. *Paper written for course at New York University: www. cs. nyu. edu/nicolosi/papers/accumulators. pdf*, 2002.

Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In Jennifer Seberry and Yuliang Zheng, editors, *Advances in Cryptology — AUSCRYPT '92*, pages 244–251. Springer Berlin Heidelberg, 1993. ISBN 978-3-540-47976-5.

Gina Gallegos-Garcia, Vincenzo Iovino, Alfredo Rial, Peter B. Roenne, and Peter Y. A. Ryan. (universal) unconditional verifiability in e-voting without trusted parties, 2016.

Juan A. Garay, Clinton Givens, Rafail Ostrovsky, and Pavel Raykov. Fast and unconditionally secure anonymous channel. PODC '14, page 313321, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450329446.

Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*, pages 281–310, 2015.

Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, 2013.

Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, Jan 2007.

Elizabeth Gibney. Quantum computer race intensifies as alternative technology gains steam, 2020. URL https://www.nature.com/articles/d41586-020-03237-w.

Oded Goldreich. *The Foundations of Modern Cryptography*, pages 1–37. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999. ISBN 978-3-662-12521-2.

Shafi Goldwasser and Yehuda Lindell. Secure multi-party computation without agreement. *J. Cryptol.*, 18(3):247287, July 2005. ISSN 0933-2790.

Michael T. Goodrich, Roberto Tamassia, and Jasminka Hasic. An efficient dynamic and distributed cryptographic accumulator. In *ISC*, 2002.

Jens Groth. Evaluating security of voting schemes in the universal composability framework. In *Applied Cryptography and Network Security*, pages 46–60. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-24852-1.

Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 253–280, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

Sean Hallgren, Adam Smith, and Fang Song. Classical cryptographic protocols in a quantum world. In Phillip Rogaway, editor, *Advances in Cryptology–CRYPTO 2011*, pages 411–428. Springer Berlin Heidelberg, 2011.

Mark Hillery, Mário Ziman, Vladimír Bužek, and Martina Bieliková. Towards quantum-based privacy and voting. In *Physics Letters A*, volume 349, pages 75–81. Elsevier, 2006.

Martin Hirt and Vassilis Zikas. Adaptively secure broadcast. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 466–485, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-13190-5.

Dmitri Horoshko and Sergei Kilin. Quantum anonymous voting with anonymity check. In *Physics Letters A*, volume 375, pages 1172–1175. Elsevier, 2011.

Wei Huang, Qiao-Yan Wen, Bin Liu, Qi Su, Su-Juan Qin, and Fei Gao. Quantum anonymous ranking. In *Physical Review A*, volume 89, page 032325. APS, 2014.

IBM. What is quantum computing?, 2019. URL https://www.ibm.com/quantum-computing/what-is-quantum-computing/.

Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Proc. of the ACM workshop on privacy in the electronic society*, pages 61–70, 2005.

Elham Kashefi, Luka Music, and Petros Wallden. The quantum cut-and-choose technique and quantum two-party computation. *arXiv preprint arXiv:1703.03754*, 2017.

Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *TCC*, pages 477–498, 2013.

A. Khisti, A. Tchamkerten, and G. W. Wornell. Secure broadcasting over fading channels. *IEEE Transactions on Information Theory*, 54(6):2453–2469, 2008. doi: 10.1109/TIT.2008.921861.

Aggelos Kiayias and Moti Yung. Self-tallying elections and perfect ballot secrecy. In David Naccache and Pascal Paillier, editors, *Public Key Cryptography*, volume 2274, pages 141–158. Springer Berlin Heidelberg, 2002.

Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. End-to-end verifiable elections in the standard model. In Elisabeth Oswald and Marc Fischlin, editors,

Advances in Cryptology - EUROCRYPT 2015, pages 468–498. Springer Berlin Heidelberg, 2015.

Aggelos Kiayias, Annabell Kuldmaa, Helger Lipmaa, Janno Siim, and Thomas Zacharias. On the security properties of e-voting bulletin boards. In Dario Catalano and Roberto De Prisco, editors, *Security and Cryptography for Networks*, pages 505–523, Cham, 2018. Springer International Publishing. ISBN 978-3-319-98113-0.

Anna Kobusiska, Jerzy Brzeziski, Micha Boro, ukasz Inatlewski, Micha Jabczyski, and Mateusz Maciejewski. A branch hash function as a method of message synchronization in anonymous p2p conversations. *International Journal of Applied Mathematics and Computer Science*, 26(2):479 – 493, 01 Jun. 2016.

Czesław Kościelny, Mirosław Kurkowski, and Marian Srebrny. *Foundations of Symmetric Cryptography*, pages 77–118. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-41386-5.

Jiangtao Li, Ninghui Li, and Rui Xue. Universal accumulators with efficient nonmembership proofs. In Jonathan Katz and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 253–269, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

Yannan Li, Willy Susilo, Guomin Yang, Yong Yu, Dongxi Liu, and Mohsen Guizani. A blockchain-based self-tallying voting scheme in decentralized iot, 2019.

Yuan Li and Guihua Zeng. Quantum anonymous voting systems based on entangled state. In *Optical review*, volume 15, pages 219–223. Springer, 2008.

Yehuda Lindell. Highly-efficient universally-composable commitments based on the ddh assumption. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 446–466, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-20465-4.

Jia Liu, Tibor Jager, Saqib A. Kakvi, and Bogdan Warinschi. How to build time-lock encryption. *Designs, Codes and Cryptography*, Feb 2018. ISSN 1573-7586.

Hoi-Kwong Lo and H.F. Chau. Why quantum bit commitment and ideal quantum coin tossing are impossible. In *Physica D: Nonlinear Phenomena*, volume 120, pages 177 – 187, 1998.

T. Lunghi, J. Kaniewski, F. Bussières, R. Houlmann, M. Tomamichel, S. Wehner, and H. Zbinden. Practical relativistic bit commitment. *Phys. Rev. Lett.*, 115: 030502.

Mohammad Mahmoody, Tal Moran, and Salil Vadhan. Time-lock puzzles in the random oracle model. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pages 39–50, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-22792-9.

Lenka Marekova. Zerovote: Self-tallying e-voting protocol in the uc. project-archive.inf.ed.ac.uk/ug4/2018-outstanding, 2018. https://project-archive.inf.ed.ac.uk/ug4/2018-outstanding.html.

Ueli Maurer and Renato Renner. Abstract cryptography. In *IN INNOVATIONS IN COMPUTER SCIENCE*. Tsinghua University Press, 2011.

Timothy C. May. Timed-release crypto, http://cypherpunks.venona.com/date/1993/02/msg001 1993.

Dominic Mayers. Unconditionally secure quantum bit commitment is impossible. *Phys. Rev. Lett.*, 78:3414–3417, 1997.

Will McCutcheon, Anna Pappa, BA Bell, A McMillan, André Chailloux, Tom Lawson, M Mafu, Damian Markham, Eleni Diamanti, and Iordanis Kerenidis. Experimental verification of multipartite entanglement in quantum networks. *Nat. Comm.*, 7:13251, 2016.

Ralph Charles Merkle. *Secrecy, Authentication, and Public Key Systems.* PhD thesis, Stanford, CA, USA, 1979. AAI8001972.

Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *IEEE S&P*, 2013.

Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006*, pages 373–392, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

David Naccache. *Standard Model*, pages 1253–1253. Springer US, Boston, MA, 2011. ISBN 978-1-4419-5906-5.

Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, http://bitcoin.org/bitcoin.pdf, 2008.

Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *CRYPTO*, 2002.

Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, New York, NY, USA, 2011.

Mehrdad Nojoumian and Douglas R. Stinson. Efficient sealed-bid auction protocols using verifiable secret sharing. In Xinyi Huang and Jianying Zhou, editors, *Information Security Practice and Experience*, pages 302–317, Cham, 2014. Springer International Publishing. ISBN 978-3-319-06320-1.

Tatsuaki Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Security Protocols*, 1998.

Tatsuaki Okamoto, Koutarou Suzuki, and Yuuki Tokunaga. Quantum voting scheme based on conjugate coding. *NTT Technical Review*, 6(1):1–8, 2008.

Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Authenticated hash tables based on cryptographic accumulators. *Algorithmica*, 74 (2):664–712, Feb 2016.

Anna Pappa, André Chailloux, Stephanie Wehner, Eleni Diamanti, and Iordanis Kerenidis. Multipartite entanglement verification resistant against dishonest parties. *Physical Review Letters*, 108(26):260502, 2012.

Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg. ISBN 978-3-540-46766-3.

Christopher Portmann. Quantum authentication with key recycling. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, pages 339–368. Springer International Publishing, 2017.

Michael O Rabin. Digitalized signatures. *Foundations of secure computation*, pages 155–168, 1978.

Michael O. Rabin. How to exchange secrets with oblivious transfer, 2005. Harvard University Technical Report 81 talr@watson.ibm.com 12955 received 21 Jun 2005.

José Ramos. *Liquid Democracy and the Futures of Governance*, pages 173–191. Springer International Publishing, Cham, 2015. ISBN 978-3-319-22994-2.

R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Cambridge, MA, USA, 1996.

Eric Roberts. Arguments in favor, 2007. URL https://cs.stanford.edu/people/eroberts/cs201/projects/2006-07/electronic-voting/index_files/page0001.html.

Peter Y. A. Ryan and Steve A. Schneider. Prêt-à-voter with re-encryption mixes. In *11th European Symp. On Research In Computer Security (ESORICS'06)*, volume 4189, pages 313–326. Springer, 2006.

Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology — EUROCRYPT '95*, pages 393–403, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg. ISBN 978-3-540-49264-1.

Tomas Sander. Efficient accumulators without trapdoor extended abstract. In Vijay Varadharajan and Yi Mu, editors, *Information and Communication Security*, pages 252–262, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-540-47942-0.

P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Washington, DC, USA, 1994. IEEE Computer Society. ISBN 0-8186-6580-7.

Peter W Shor and John Preskill. Simple proof of security of the bb84 quantum key distribution protocol. In *Physical review letters*, volume 85, page 441. APS, 2000.

Alan Szepieniec and Bart Preneel. New techniques for electronic voting. Washington, D.C., 2015. USENIX Association. URL https://www.usenix.org/conference/jets15/workshop-program/presentation/szepieniec.

Andranik Tangian. *Representative Democracy*, pages 353–403. Springer International Publishing, Cham, 2020a. ISBN 978-3-030-39691-6.

Andranik Tangian. *Direct Democracy*, pages 263–315. Springer International Publishing, Cham, 2020b. ISBN 978-3-030-39691-6.

Kishore Thapliyal, Rishi Dutt Sharma, and Anirban Pathak. Protocols for quantum binary voting. In *International Journal of Quantum Information*, volume 15, page 1750007, 2017.

Aaron Toponce. Further investigation into scrypt and argon2 password hashing, 2016. URL https://pthree.org/2016/06/29/further-investigation-into-scrypt-and-argon2-password-hashing/.

Edward Tremel. Real-world performance of cryptographic accumulators. *Undergraduate Honors Thesis, Brown University*, 2013.

Dominique Unruh. Universally composable quantum multi-party computation. In *Advances in Cryptology (EUROCRYPT 2010)*, volume 6110, pages 486–505, 2010.

Dominique Unruh. Everlasting multi-party computation. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 380–397, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-40084-1.

Joan Alfina Vaccaro, Joseph Spring, and Anthony Chefles. Quantum protocols for anonymous voting and surveying. In *Physical Review A*, volume 75, page 012333. APS, 2007.

Qingle Wang, Chaohua Yu, Fei Gao, Haoyu Qi, and Qiaoyan Wen. Self-tallying quantum anonymous voting. In *Physical Review A*, volume 94, page 022333. APS, 2016.

Wikipedia. Electronic voting in estonia, 2020. URL https://en.wikipedia.org/wiki/Electronic_voting_in_Estonia.

Douglas Wikström. Universally composable dkg with linear number of exponentiations. In Carlo Blundo and Stelvio Cimato, editors, *Security in Communication Networks*, pages 263–277, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

Scott Wolchok, Eric Wustrow, Dawn Isabel, and J. Alex Halderman. Attacking the washington, D.C. internet voting system. In *FC*, 2012.

Peng Xue and Xin Zhang. A simple quantum voting scheme with multi-qubit entanglement. In *Scientific Reports*, volume 7, page 7586. Nature Publishing Group, 2017.

Xun Yi, Russell Paulet, and Elisa Bertino. *Homomorphic Encryption*, pages 27–46. Springer International Publishing, Cham, 2014. ISBN 978-3-319-12229-8.

Y. Zhang, J. Katz, and C. Papamanthou. An expressive (zero-knowledge) set accumulator. In *Euro S&P*, 2017.

Rui-Rui Zhou and Li Yang. Distributed quantum election scheme. *arXiv preprint arXiv:1304.0555*, 2013.