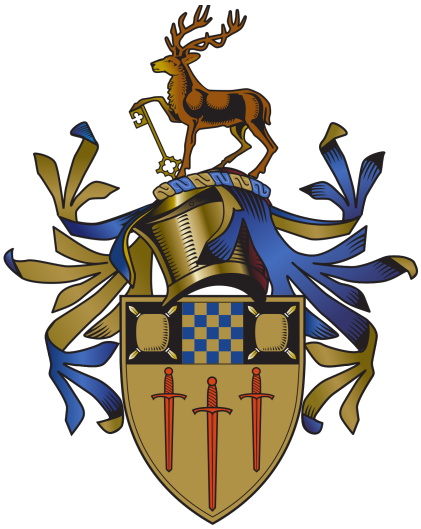
Trust and Privacy in Electronic Voting

by
Navid Abapour
October 28, 2024



UNIVERSITY OF
SURREY

Computer Science Research Centre
Faculty of Engineering and Physical Sciences
University of Surrey

Supervised by
Cătălin Drăgan
and
Ioana Boureanu

Confirmation Report

Presented as part of the requirements for the degree of
Doctor of Philosophy

Declaration

This thesis and the work to which it refers are the results of my own efforts. Any ideas, data, images or text resulting from the work of others (whether published or unpublished, and including any content generated by a deep learning/artificial intelligence tool) are fully identified as such within the work and attributed to their originator in the text, bibliography or in footnotes. This thesis has not been submitted in whole or in part for any other academic degree or professional qualification. I agree that the University has the right to submit my work to the plagiarism detection service TurnitinUK for originality checks. Whether or not drafts have been so-assessed, the University reserves the right to require an electronic version of the final document (as submitted) for assessment as above.

Navid Abapour
2024-10-28

Acknowledgement

I am truly thankful to Ghazal Afroozi Milani, PGR in Computer Science, for her faith in me and her cordial kindness to me during the past year. Also, I am greatly obliged to Samaneh Rashidi, PGR in Psychology, for her support, which paved the way for me to come to Guildford and start my studies more easily. I am highly indebted to my supervisors, Cătălin Drăgan, for his patience and belief in me and to Ioana Boureanu, for her rich and valuable guidance during this first year of my PhD journey.

Abstract

Electronic voting is considered the giant leap towards modern democracies: improving accessibility, security, and efficiency in elections because such have risen due to an upsurge of interest by the citizens in integrating technology into government affairs. Challenges to e-voting, related to trust, have to be resolved if concepts of cybersecurity and transparency are retained, along with voters' trust in the integrity and reliability of digitally conducted elections. To address the privacy and trust concerns in electronic voting, this project presents a rigorous study of building blocks of conventional electronic voting schemes with a cryptographic approach, along with state-of-art essential privacy properties of electronic voting systems. Also, we provide the first elegant formalization of Microsoft ElectionGuard, a software development kit for voting; this algorithmic formalization can be used for further security studies. Moreover, we check the possibility of whether ElectionGuard is ballot private, and whether it preserves participation privacy. To indicate future research avenues, underline the importance of both post-quantum cryptography and mechanisms for resolving disputes that support voter confidence and give assurance of election integrity. The goal of this project is to contribute to improving verifiable and privacy-preserving electronic voting system development securely concerning dispute resolution.

Contents

List of Figures

Chapter 1

Introduction

Nowadays, the importance of migrating actions to the electronic side is unquestionable. Voting is not an exception to this, and moving towards electronic voting is essential from many perspectives. It increases accessibility whereby people with disabilities are among other remotely located individuals who can participate with much more convenience than perhaps it is today [1]. Additionally, electronic voting improves efficiency in terms of speedily casting and tallying votes, avoiding long queues to pave the way for quicker results. Another advantage is increased accuracy: the elimination of human error while counting votes and in the handling of ballots through the use of electronic systems [2]. Most of the costs related to traditional paper-based elections can equally be reduced in an electronic vote: printing, distribution, and manual counting. Further, security could be furthered in electronic systems to prevent fraud, tampering, or double voting, which would ensure reliability. The latter involves improved auditing capabilities and clearly defined voting records to increase transparency and foster public trust in the electoral system [3].

However, according to previous cases, implementing electronic voting in the real world is not easy.

Immediately following the 2020 US presidential election, there were serious claims of some sort of electronic voting machine irregularities [4], [5]. Estonia, often marked as the leader in Internet voting, was once criticized when security researchers revealed that the system was even vulnerable to severe threats in October 2013 [6], [7]. In November 2019, major electronic voting equipment problems were reported to have occurred in Northampton County, Pennsylvania, which advocacy groups took to court [8]. During the 2021 parliamentary elections in Russia, the electronic elections were massively flawed, with credible reports of fraud, and critics of the ruling party were barred [9]. During the 2015 Argentina mayoral election, there were massive failures in the e-voting system: many polling stations reported voting delays due to malfunctioning machines, with other voters claiming that the machines were registering votes for candidates not chosen by the voters [10]. In 2015, during the implementation, issues were encountered with the iVote system used in New South Wales, Australia [7], [11]. In Venezuela's 2017 highly controversial election, the company that supplied the voting technology, Smartmatic, later reported that the turnout figures were manipulated [12].

Trust is perceived to be the underpinning of any democratic electoral process. Provided that one desires a voting system to be effective, then such a system should engender trust from all those involved: the voters, candidates, election officials, and observers. In securing trust, which is multifaceted, a voting system derives it from several critical features: confidentiality of the votes, integrity in the process of voting, transparency of the system, and provision for auditing and verification of results. In the event that this trust is lacking, the legitimacy of any election result cannot be guaranteed, and political instability with a crisis of confidence in democratic institutions will follow [13].

Many schemes have been proposed and implemented to fulfill the need for electronic voting, including ElectionGuard. The ElectionGuard is an open-source software framework developed by Microsoft that enables secure and verifiable electronic voting systems. It uses encryption to ensure votes are private and allows third parties to independently verify the integrity of election results without compromising voter privacy [14]. The main novelty regarding ElectionGuard lies in segregating the cryptographic tool from the traditional voting system. This will enable voters, candidates, observers, and independent organizations to verify election results. ElectionGuard has been used in real life, including in public elections: for instance, in the US alone, states such as Wisconsin, California, and Maryland, among others; international ones include France or Switzerland [15].

Therefore, this toolkit looks promising for various polling platforms. However, as far as we know, there is no algorithmic formalization of ElectionGuard that can help security researchers and cryptographers investigate the security proofs of this developmental gadget because this toolkit includes many details and cryptographic particularities.

Furthermore, no cryptographic proof of the privacy aspects of ElectionGuard has been provided so far. One reason for this is the lack of algorithmic formalization, and another reason is that ElectionGuard uses a threshold structure, and for the components of this structure, there is no threshold privacy definition and researchers usually avoid threshold property in the privacy definitions.

ElectionGuard is similar to the well-known online voting system, Helios [16], but there are significant differences that require additional proof.

To fill these gaps, in this report, we start by detailing the cryptographic building blocks, primitives, and hard mathematical problems as security assumptions. Then, we cover the state-of-art in electronic voting and widely accepted definitions of privacy and verifiability. Next, an abstracted version of ElectionGuard is proposed, which is an elegant formalization that makes security investigations easier, and a simplified version of ElectionGuard is presented, along with a discussion of the potential of ElectionGuard in satisfying the Ballot Privacy and Participation Privacy definitions. Eventually, we focus on the ideas that not only address trust issues in electronic voting but offer practical insights into the challenges and opportunities of integrating post-quantum cryptography into e-voting systems and finish the report by representing the direction of this PhD project, together with a rough plan for the subsequent phases of research.

Chapter 2

Preliminaries

In this section, we introduce the core cryptographic primitives (e.g, hash functions, encryption, zero-knowledge proofs) and highlight their security properties.

2.1 Cryptographic Primitives

Hash Functions. A cryptographic **hash function** $\mathsf{H} : \{0,1\}^* \to \mathbb{G}$ takes an arbitrarily long input bitstring and produces a fixed-size output bitstring. Hash functions are

- *deterministic*, so for any input $x \in \{0,1\}^*$, the output $y \leftarrow \mathsf{H}(x)$ is always the same.
- *pre-image resistance*, so given an output y it should be computationally infeasible to find any input $x \in \{0,1\}^*$ such that $\mathsf{H}(x) = y$. We formalize this as, the advantage of an efficient adversary $\mathcal{A}$ is:

$$\mathsf{Adv}^{\mathsf{pre\text{-}image}}_{\mathcal{A},\mathsf{H}}(\lambda) = \Pr[y \leftarrow\!\!\$\ \mathbb{G};\ x \leftarrow \mathcal{A}(y);\ \textbf{return}\ \mathsf{H}(x) = y] \leq \mathsf{negl}(\lambda)$$

- *collision resistance*, where it should be computationally infeasible to find any two distinct inputs $x_1, x_2 \in \{0,1\}^*$ such that $\mathsf{H}(x_1) = \mathsf{H}(x_2)$.We formalize this as, the advantage of an efficient adversary $\mathcal{A}$ is:

$$\mathsf{Adv}^{\mathsf{collision}}_{\mathcal{A},\mathsf{H}}(\lambda) = \Pr[(x_1, x_2) \leftarrow \mathcal{A};\ \textbf{return}\ \mathsf{H}(x_1) = \mathsf{H}(x_2)] \leq \mathsf{negl}(\lambda)$$

Remark 1 (Random Oracle Model)**.** Hash functions are often modelled as Random Oracles where for every unique input $x \in \{0,1\}^*$ an output is chosen uniformly at random $y \leftarrow\!\!\$\ \mathbb{G}$, and the deterministic property is maintained.

HMAC. In ElectionGuard [17] they instantiate the hash functions with keyed-hash message authentication code [18] $\mathsf{H} : \mathbb{Z}_p \times \{0,1\}^* \to \mathbb{G}$. More precisely, considering K as the key and M as the input message, we have

$$\mathsf{H}(K, M) = \mathsf{SHA256}\left((K \oplus opad) \| \mathsf{SHA256}\left((K \oplus ipad) \| M\right)\right),$$

where $opad =$ `0x5c5c5c`$\ldots$ (block size times) and $ipad =$ `0x363636`$\ldots$ (block size times).

As a security property, we rely on *existential Unforgeability under Chosen Message Attack* (EUF-CMA), that measures an adversary's capability to create forgeries for a new message where it han't seen valid MACs. We formalize this as,

$$\mathsf{Adv}^{\mathsf{euf}}_{\mathcal{A},\mathsf{H}}(\lambda) = \Pr\Big[k \leftarrow\!\$\ \mathbb{Z}_p;\ (m^*, \tau^*) \leftarrow \mathcal{A}^{\mathsf{O}}; \mathbf{return}\ m* \notin \mathcal{Q} \wedge \mathsf{H}(k, m^*) = \tau^*\Big]$$

where $\mathsf{O}(m) = [\mathcal{Q} \leftarrow m \cup \mathcal{Q};\ \mathbf{return}\ \mathsf{H}(k, m)]$.

Public Key Encryption. A public-key encryption scheme consists of three algorithms: $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}()$: As a probabilistic algorithm, the key generation takes no input and returns a key pair sk as the secret key and pk known as the public key.
- $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$: The encryption algorithm takes a public key pk and a message m and computes a ciphertext c.
- $d \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$: The deterministic algorithm takes a secret key sk and a ciphertext c and outputs either a decrypted message d, or expresses the ciphertext invalid.

It satisfies *correctness* if for any key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}$ we have with overwhelming probability that $m \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m))$.

We rely on *indistinguishability under chosen plaintext attack* (IND-CPA) to model an adversary's ability to distinguish between the encryption of two chosen plaintexts (called challenge step) without access to decryption oracles. This is illustrated in Figure 2.1. More formally,

$$\mathsf{Adv}^{\mathsf{cma}}_{\mathcal{A},\mathcal{E}}(\lambda) = |\Pr\Big[\mathsf{Exp}^{\mathsf{IND-CPA}}_{\mathcal{A},\mathcal{E}} = 1\Big] - 1/2|.$$

Similarly, we define *indistinguishability under chosen ciphertext attack* (IND-CCA2) where an adversary with access to a decryption oracle needs to distinguish between the encryption of two chosen plaintexts

$$\mathsf{Adv}^{\mathsf{cca}}_{\mathcal{A},\mathcal{E}}(\lambda) = |\Pr\Big[\mathsf{Exp}^{\mathsf{IND-CCA2}}_{\mathcal{A},\mathcal{E}} = 1\Big] - 1/2|.$$

If we restrict the capabilities of the adversary in IND-CCA2 we obtain two further weaker properties that are often used in e-voting:

- IND-CCA1, where the adversary can't submit decryption oracle queries after the challenge step
- IND-1-CCA, where the adversary is allowed to make a single decryption query after the challenge step. However, that single decryption query can handle multiple ciphertexts instead of the single 1 ciphertext considered by IND-CCA2 and IND-CCA1.

$\mathsf{Exp}^{\mathsf{IND-CPA}}_{\mathcal{E},\mathcal{A}}(\lambda)$

1 : $(\mathsf{pk},\mathsf{sk}) \leftarrow\$ \mathsf{KGen}()$
2 : $(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk})$
3 : $b \leftarrow\$ \{0,1\}$
4 : $c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$
5 : $b' \leftarrow \mathcal{A}(c^*)$
6 : **return** $b \stackrel{?}{=} b'$

$\mathsf{Exp}^{\mathsf{IND-CCA2}}_{\mathcal{E},\mathcal{A}^{\mathcal{O}_{\mathrm{dec}}}(\lambda)}$

1 : $(\mathsf{pk},\mathsf{sk}) \leftarrow\$ \mathsf{KGen}()$
2 : $(m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{dec}}}(\mathsf{pk})$
3 : $b \leftarrow\$ \{0,1\}$
4 : $c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$
5 : $b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{dec}}}(c^*)$
6 : **return** $b \stackrel{?}{=} b'$

$\mathcal{O}_{\mathrm{dec}}(c)$

1 : $m \leftarrow \perp$
2 : **if** $(c^* \neq c) \wedge (c^* \neq \perp)$ **then**
3 : $\quad m \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
4 : **return** m

$\mathsf{Exp}^{\mathsf{IND-1-CCA}}_{\mathcal{E},\mathcal{A}^{\mathcal{O}_{\mathrm{dec}}}(\lambda)}$

1 : $(\mathsf{pk},\mathsf{sk}) \leftarrow\$ \mathsf{KGen}()$
2 : $go \leftarrow \mathsf{true}$
3 : $(m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{dec}}}$
4 : $b \leftarrow\$ \{0,1\}$
5 : $c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$
6 : $b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{dec}'}}(c^*)$
7 : **return** $b \stackrel{?}{=} b'$

$\mathsf{Exp}^{\mathsf{IND-CCA1}}_{\mathcal{E},\mathcal{A}^{\mathcal{O}_{\mathrm{dec}}}(\lambda)}$

1 : $(\mathsf{pk},\mathsf{sk}) \leftarrow\$ \mathsf{KGen}()$
2 : $(m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{dec}}}$
3 : $b \leftarrow\$ \{0,1\}$
4 : $c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$
5 : $b' \leftarrow \mathcal{A}(c^*)$
6 : **return** $b \stackrel{?}{=} b'$

$\mathcal{O}_{\mathrm{dec}'}(cL)$

1 : **if** $c^* \notin cL \wedge go$ **then**
2 : $\quad go \leftarrow \mathsf{false}$
3 : $\quad$ **for** $c \in cL$ **do**
4 : $\quad\quad mL \leftarrow mL \cup \{\mathsf{Dec}(\mathsf{sk}, c)\}$
5 : **return** mL

Figure 2.1: The Experiments of Security Properties for a Public Key Encryption Scheme

ElGamal PKE. We illustrate in Figure 2.2 two variants of ElGamal, that slightly differ in how in how encryption/decruption is used. The version used by ElectionGuard is the 2nd one [19], as it is considered more efficient with respect to the number of exponentiation. The voting system Helios [16] relies on a different version of ElGamal, called signed ElGamal, where the $\beta = g^m \mathsf{pk})^r$.

It is well established that ElGamal is IND-CPA and not IND-CCA2, IND-CCA1, or IND-1-CCA. The reason is that ElGamal is malleable-meaning with access to an encryption oracle it is possible for an adversary to manipulate a ciphertext such that an obtained ciphertext is a valid ciphertext related to the underlying plaintext, without knowing the latter [20].

Lemma 1 ([21])**.** *If the DDH assumption holds in the group G, then the ElGamal encryption scheme is* IND−CPA *secure.*

It has been claimed that by Devillez et al. [22] that the Modifid ElGamal is IND−CPA, as the modifications have no impact on the security but simply reduce the cost of computing in some applications.

However, further investigation into this aspect will likely provide more insights, and we aim to tackle it in our future studies.

Encrypt-then-MAC. This technique is used to make information confidential and integral, the message is first encrypted by some symmetric encryption scheme, and after that, a Message Authentication Code on the ciphertext, resulting from that encryption, is

$\mathsf{ElGamal.KGen}(1^\lambda)$

1 : $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p^*$
2 : $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$
3 : **return** $(\mathsf{sk}, \mathsf{pk})$

$\mathsf{ElGamal.Enc}(\mathsf{pk}, m)$

1 : $r \leftarrow\!\!\$\ \mathbb{Z}_p^*$
2 : $\alpha \leftarrow g^r \mod p$
3 : $\beta \leftarrow m \cdot (\mathsf{pk})^r \mod p$
4 : $c \leftarrow (\alpha, \beta)$
5 : **return** c

$\mathsf{ElGamal.Dec}(\mathsf{sk}, c)$

1 : $K \leftarrow \alpha^{\mathsf{sk}} \mod p$
2 : $m \leftarrow \beta \cdot K^{-1} \mod p$
3 : **return** m

$\mathsf{ElGamal'.KGen}(1^\lambda)$

1 : $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p^*$
2 : $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$
3 : **return** $(\mathsf{sk}, \mathsf{pk})$

$\mathsf{ElGamal'.Enc}(\mathsf{pk}, m)$

1 : $r \leftarrow\!\!\$\ \mathbb{Z}_p^*$
2 : $\alpha \leftarrow g^r \mod p$
3 : $\beta \leftarrow (\mathsf{pk})^{r+m} \mod p$
4 : $c \leftarrow (\alpha, \beta)$
5 : **return** (c, r)

$\mathsf{ElGamal'.Dec}(\mathsf{sk}, c)$

1 : $K \leftarrow \alpha^{\mathsf{sk}} \mod p$
2 : $m \leftarrow \log_{\mathsf{pk}}(\beta \cdot K^{-1} \mod p)$
3 : **return** m

Figure 2.2: ElGamal Cryptosystem, and, Modified Version, Variant Exponential Form

computed. In schemes like these, there are two different keys: one used for encryption, the other for the MAC. This feeds into the MAC algorithm for verification purposes, thereby performing dual duty in ensuring integrity and authenticity of the ciphertext with a tag generated for that. The ciphertext-tag bundle that results is then sent. The receiver, at the decrypting end, shall verify the integrity of the ciphertext first through its MAC tag. If the latter would be valid, the ciphertext shall be decrypted in order to retrieve its source message [23].

$(k_{\text{enc}}, k_{\text{mac}}) \leftarrow \mathsf{KeyGen}(1^\lambda)$: The key generation algorithm is a probabilistic algorithm that takes as input a security parameter λ. It outputs two keys: k_{enc} for encryption and k_{mac} for the message authentication code (MAC). These keys are derived independently and used for encryption and message authentication, respectively.

$c \leftarrow \mathsf{Enc}(k_{\text{enc}}, m)$: The encryption algorithm takes as input the encryption key k_{enc} and a message m. It outputs a ciphertext $c = \mathsf{Enc}_{\text{Symmetric}}(k_{\text{enc}}, m)$, where $\mathsf{Enc}_{\text{Symmetric}}$ is a symmetric encryption algorithm (such as AES) applied to the message m.

$t \leftarrow \mathsf{MAC}(k_{\text{mac}}, c)$: The MAC generation algorithm takes as input the MAC key k_{mac} and the ciphertext c. It outputs a MAC tag $t = \mathsf{MAC}_{\text{HMAC}}(k_{\text{mac}}, c)$, where $\mathsf{MAC}_{\text{HMAC}}$ is a hash-based message authentication code (HMAC) applied to the ciphertext c.

$(c, t) \leftarrow \mathsf{Enc}'_5(k_{\text{enc}}, k_{\text{mac}}, m)$: The Encrypt-then-MAC construction involves encrypting the message m to obtain the ciphertext c, and then computing a MAC on the ciphertext c to obtain the tag t. The output is the tuple (c, t), representing the ciphertext and its corresponding MAC tag. In this report, we consider a case where

the MAC key and encryption key are generated in the procedure itself, so the function will be like $((c,t)) \leftarrow \mathsf{Enc}_5(\mathsf{pp}, m, k)$ and the encryption key and MAC key are derived inside the method.

$\perp/\top \leftarrow \mathsf{Verify}(k_{\text{mac}}, c, t)$: The verification algorithm takes as input the MAC key k_{mac}, the ciphertext c, and the tag t. It verifies whether $t = \mathsf{MAC}_{\text{HMAC}}(k_{\text{mac}}, c)$. If the verification is successful, it outputs $\top$ indicating the integrity of the ciphertext, otherwise it outputs $\perp$.

$m \leftarrow \mathsf{Dec}(k_{\text{enc}}, c)$: The decryption algorithm takes as input the encryption key k_{enc} and the ciphertext c. If the MAC verification is successful, it decrypts the ciphertext c to recover the original message $m = \mathsf{Dec}_{\text{Symmetric}}(k_{\text{enc}}, c)$, where $\mathsf{Dec}_{\text{Symmetric}}$ is the symmetric decryption algorithm.

$m/\perp \leftarrow \mathsf{Dec}'_5(k_{\text{enc}}, k_{\text{mac}}, c, t)$: The Decrypt-then-Verify construction involves first verifying the MAC tag t using the key k_{mac} and the ciphertext c. If the verification is successful ($\top$), the ciphertext c is decrypted using the decryption key k_{enc} to recover the message m. If the verification fails ($\perp$), the output is $\perp$, indicating either a tampered ciphertext or an invalid tag. In this report, we consider a case where the ciphertext and tag are entangled into each other, so the function will be like $m \leftarrow \mathsf{Dec}_5(\mathsf{pp}, \mathsf{sk}, (c,t))$.

Verifiable Secret Sharing. Splitting a piece of confidential information among multiple parties in a way that allows each party to verify the authenticity of their share without revealing the information and ensures that only a sufficient subset of parties can reconstruct the original secret can be achieved by the *verifiable secret sharing* scheme [24]. Formally, it consists of the algorithms $\mathsf{VSS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Rec}, \mathsf{Vf})$ where:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(t, n)$ defines the public parameters $\mathsf{pp} = (p, q, g, t, n, \mathsf{Idx})$ including the set of identities $\mathsf{Idx} = \{1, \ldots, n\}$.
- $(\{\mathsf{shr_i}\}_{i \in \mathsf{Idx}}, \mathsf{pubdata}) \leftarrow \mathsf{Share}(\mathsf{pp}, s)$ the sharing procedure of the secret s which is a probabilistic algorithm that produces shares $\mathsf{shr_i}$ for each user $i \in \mathsf{Idx}$ and a set of public data $\mathsf{pubdata}$ that will be used to verify shares.
- $s \leftarrow \mathsf{Rec}(\mathsf{pp}, \{\mathsf{shr_j}\}_{j \in J})$ that reconstructs the secret s from a subset of users $J \subseteq \mathsf{Idx}$ with $|J| \geq t$.
- $\{0,1\} \leftarrow \mathsf{Vf}(\mathsf{pp}, \mathsf{pubdata}, (i, \mathsf{shr_i}))$ which checks the validity of a share $\mathsf{shr_i}$.

We instantiate the discussed verifiable secret sharing scheme, with the construction from [25] that combines Shamir sharing and Feldmand commitments. We illustrate the actual construction in Figure 2.3.

Lemma 2 (Fairness and Completeness [25], [26])**.** *The described* VSS *satisfies Fairness and Completeness, meaning that no subset of participants (smaller than the threshold t)*

$\mathsf{Share}(\mathsf{pp}, s)$

1 : // size of Idx is n
2 : $(p, q, g, t, n, \mathsf{Idx}) \leftarrow \mathsf{pp}$
3 : $a_0 = s$
4 : **for** $j \in \{1, \ldots, t-1\}$ **do**
5 : $a_j \leftarrow\$ \mathbb{Z}_q$
6 : $A_j \leftarrow g^{a_j} \bmod p$
7 : $P(x) = a_t x^{t-1} + \cdots + a_1 x + a_0$
8 : **for** $\mathsf{idx} \in \mathsf{Idx}$ **do**
9 : $\mathsf{shr}_{\mathsf{idx}} \leftarrow P(\mathsf{idx})$
10 : $\mathsf{pubdata} \leftarrow (A_0, \ldots, A_{t-1})$
11 : **return** $((i, \mathsf{shr}_{\mathsf{idx}})_{i \in \mathsf{Idx}}, \mathsf{pubdata})$

$\mathsf{Rec}(\mathsf{pp}, (\mathsf{idx}, \mathsf{shr}_{\mathsf{idx}})_{\mathsf{idx} \in J})$

1 : $(p, q, g, t, n, \mathsf{Idx}) \leftarrow \mathsf{pp}$
2 : **for** $j \in J$ **do**
3 : $L_j \leftarrow \prod_{i \in J \setminus \{j\}} \frac{-i}{j-i} \bmod q$
4 : $\mathsf{sk}' \leftarrow \sum_{j \in J} \mathsf{shr}_{\mathsf{j}} \cdot L_j \bmod q$
5 : **return** sk';

$\mathsf{Vf}(\mathsf{pp}, \mathsf{pubdata}, (i, \mathsf{shr}_{\mathsf{i}}))$

1 : $(p, q, g, t, n, \mathsf{Idx}) \leftarrow \mathsf{pp}$
2 : $S_j = \prod_{\ell=0}^{t} A_\ell^{j^\ell} \bmod p$
3 : **return** $(g^{s_j} = S_j \wedge A_0 = \mathsf{pk})$

Figure 2.3: Verifiable Secret Sharing

can reconstruct the secret earlier than others, and the reconstruction process only succeeds when the threshold number of participants collaborate (Fairness) and if the protocol is followed honestly, any set of at least t valid shares can correctly reconstruct the secret sk *without errors (Completeness).*

Lemma 3 (Soundness and Dealer Commitment [26], [27])**.** *Let* $\mathsf{VSS} = (\mathsf{Share}, \mathsf{Rec}, \mathsf{Vf})$ *be the Shamir Verifiable Secret Sharing scheme as defined. The reconstructed secret* sk' *from any valid set of t or more shares is equal to the originally shared secret* sk *(Soundness), and the dealer is committed to a unique secret* sk *when the shares and commitments* $\{A_0, \ldots, A_{t-1}\}$ *are published. Any reconstruction from valid shares will always yield this unique secret* sk*, and the dealer cannot change or equivocate the secret after the sharing phase (Dealer Commitment).*

Lemma 4 (Unforgeability [25])**.** *Let* $\mathsf{VSS} = (\mathsf{Share}, \mathsf{Rec}, \mathsf{Vf})$ *be the Shamir Verifiable Secret Sharing scheme as defined. The scheme satisfies Unforgeability of Shares, meaning that no participant or adversary can forge a valid share* $\mathsf{shr}_{\mathsf{idx_i}}$ *for any identity* $\mathsf{idx_i} \in \mathsf{Idx}$ *that is consistent with the secret sharing process and the commitments* $\{A_0, A_1, \ldots, A_{t-1}\}$*, without access to the secret polynomial* $P(x)$*.*

Distributed Key Generation A Distributed Key Generation (DKG) protocol enables certain participants in the subset to jointly generate a secret or cryptographic key, and then the mechanisms protect properties like secrecy, robustness, and verifiability of the same. Protocols instantiate DKG; that is, there are actual instantiations for useful applications depending on security requirements: Shamir's Secret Sharing, Pedersen commitments, elliptic curve cryptography, etc [28].

Pedersen proposed an influential DKG protocol for discrete logarithm-based cryptosystems [29]. However, Gennaro et al. [28] demonstrated that Pedersen's DKG does not guarantee

a uniformly random distribution of generated keys and introduced a secure variant of it that we present here.

A Distributed Key Generation Protocol $\mathcal{DKG} = (\mathsf{Initialize}, \mathsf{ShareGen}, \mathsf{Distribute}, \mathsf{Reconstruct})$ as introduced by Gennaro et al., can be defined with the following algorithms:

$\{P_1, P_2, \ldots, P_n\} \leftarrow \mathsf{Initialize}(t, n)$: The initialization algorithm takes as input the total number of participants n and a threshold t. It sets up the participants $P = \{P_1, P_2, \ldots, P_n\}$ and determines that any subset of t participants can reconstruct the secret, while fewer participants cannot learn any information about it.

$\{s_i\}_{i=1}^n \leftarrow \mathsf{ShareGen}(\mathcal{G}, P, t)$: The share generation algorithm uses a verifiable secret sharing (VSS) scheme to generate shares. Each participant P_i selects a random polynomial $f_i(x)$ of degree $t-1$ with coefficients in $\mathbb{F}_p$, and evaluates it at $P_1, P_2, \ldots, P_n$. The participants exchange commitments to the coefficients of their polynomials and verify each other's shares using the distributed discrete logarithm scheme. Each participant computes their local share s_i from the sum of the received shares:

$$s_i = \sum_{j=1}^{n} f_j(i)$$

where $f_j(i)$ is the share received from participant P_j.

$\mathcal{S} \leftarrow \mathsf{Distribute}(\{s_i\}_{i=1}^n, P)$: The distribution algorithm ensures that the shares s_i are consistently distributed to all participants, with each participant holding their respective share. The public key $\mathsf{pk} = g^S$ is derived as the product of the commitments to each participant's polynomial, where g is a generator of the cyclic group $\mathbb{G}$, and S is the shared secret: $\mathsf{pk} = g^{\sum_{i=1}^{n} a_{i,0}}$. Here, $a_{i,0}$ is the constant term of participant P_i's polynomial $f_i(x)$, which corresponds to the secret component of the DKG protocol.

$S \leftarrow \mathsf{Reconstruct}(\{s_j\}_{j\in T}, \mathcal{R})$: The reconstruction algorithm allows any subset $T \subseteq P$ with $|T| \geq t$ participants to combine their shares $\{s_j\}_{j\in T}$ using a Lagrange interpolation-based reconstruction function $\mathcal{R}$. The secret S is reconstructed as $S = \sum_{j\in T} s_j \lambda_j \mod p$, where λ_j are the Lagrange coefficients $\lambda_j = \prod_{\substack{k\in T \\ k\neq j}} \frac{k}{k-j} \mod p$. This guarantees that the secret S is correctly reconstructed if $|T| \geq t$, and fewer than t participants learn nothing about the secret.

Lemma 5 (Correctness [28]). *Given a set of valid shares $\{s_j\}_{j\in T}$, where $|T| \geq t$, from a DKG scheme based on a threshold t, the secret S is uniquely and correctly reconstructed as $S = \sum_{j\in T} s_j \lambda_j \mod p$, where λ_j are the Lagrange interpolation coefficients.*

Lemma 6 (Secrecy [28]). *Let $T' \subset P$ such that $|T'| < t$. The participants in T' learn no information about the secret S as long as the discrete logarithm problem in the group $\mathbb{G}$ is hard.*

Threshold Cryptosystem A threshold cryptosystem is a general cryptographic system designed to share the power of decryption among members so that nobody can take full control of the resources. The secret key in this scheme is divided into parts such that only a threshold number of participants can cooperate in recovering the plaintext or the secret. This mechanism not only promises superior security but also better fault tolerance: even if some participants are compromised or unavailable, the system would still be working efficiently and securely. Threshold cryptosystems, indeed, have been used in very different settings that require much security and decentralization, such as secure voting systems, distributed databases, and blockchain protocols [30].

A Threshold Cryptosystem $\mathcal{E}' = (\mathsf{KeyGen}_4, \mathsf{Enc}_4, \mathsf{Share}_4, \mathsf{Combine}_4, \mathsf{Dec}_4)$ can be consisting of five algorithms defined as follows:

$(\mathsf{pk}, \{\mathsf{sk}_i\}_{i=1}^n) \leftarrow \mathsf{KeyGen}_4(1^\lambda, t, n)$: The key generation algorithm is a probabilistic algorithm that takes as input a security parameter λ, the number of participants n, and a threshold value $t \leq n$. It generates a public key pk and a set of secret key shares $\{\mathsf{sk}_i\}_{i=1}^n$, where each sk_i is distributed to the corresponding participant P_i. The secret can only be reconstructed by at least t participants. The public key pk is computed as a function of a jointly generated secret $S \in \mathbb{F}_p$ and is shared among all participants.

$c \leftarrow \mathsf{Enc}_4(\mathsf{pk}, m)$: The encryption algorithm is a probabilistic algorithm that takes the public key pk and a message $m \in \mathbb{F}_p$. It outputs a ciphertext c, often involving randomness $r \in \mathbb{F}_p$, ensuring semantic security. The ciphertext c may be structured as $c = (\alpha, \beta)$, where $\alpha = g^r \mod p, \quad \beta = m \cdot h^r \mod p$. Here, h is some element related to pk, and g is a generator of the finite group $\mathbb{F}_p$.

$\{\gamma_i\}_{i \in T} \leftarrow \mathsf{Share}_4(\{\mathsf{sk}_i\}_{i \in T}, c)$: The share decryption algorithm is a deterministic algorithm executed by each participant $P_i \in T$, where $T \subseteq P$ is a set of at least t participants. Each participant uses their secret share sk_i and the ciphertext $c = (\alpha, \beta)$ to compute their partial decryption γ_i, which may be represented as $\gamma_i = \alpha^{\mathsf{sk}_i} \mod p$. The partial decryptions $\{\gamma_i\}_{i \in T}$ will later be combined to recover the message m.

$m \leftarrow \mathsf{Combine}_4(\{\gamma_i\}_{i \in T}, c)$: The combine algorithm takes as input the set of partial decryptions $\{\gamma_i\}_{i \in T}$ from at least t participants and the ciphertext c. It uses Lagrange interpolation (or a similar threshold function) to compute the shared secret K and recover the original message m. For example, the Lagrange coefficients λ_j for each participant $j \in T$ are computed as $\lambda_j = \prod_{\substack{k \in T \\ k \neq j}} \frac{k}{k-j} \mod p$. The recovered shared secret K is then used to decrypt the message $m = \beta \cdot K^{-1} \mod p$, where the value $K = \prod_{j \in T} \gamma_j^{\lambda_j} \mod p$, is the reconstruction of the secret K from the partial decryptions.

$m \leftarrow \mathsf{Dec}_4(\{\mathsf{sk}_i\}_{i=1}^n, c)$: The decryption algorithm is a deterministic algorithm that takes the entire set of secret shares $\{\mathsf{sk}_i\}_{i=1}^n$ (or a subset of at least t shares) and

the ciphertext c. It reconstructs the shared secret S and decrypts the ciphertext to recover the original message m by combining the shares using Lagrange interpolation and recovering the secret as in the $\mathsf{Combine}_4$ step.

Any subset $T \subseteq P$ of at least t participants can correctly decrypt the message m by combining their shares using the algorithm $\mathsf{Combine}_4$. That is, for any T with $|T| \geq t$, the output m of the decryption procedure satisfies $m = \mathsf{Dec}_4(\{\mathsf{sk}_i\}_{i \in T}, c)$.

This cryptosystem is closely related to the Shamir's Secret Sharing Scheme combined with ElGamal Encryption in a threshold setting, so it will hold the same security properties of these two building blocks. This cryptosystem is inspired by the work of De Santis et al. (1994) [31], where they provide security proofs showing that the scheme is resistant to collusion and guarantees unconditional security. Additionally, the scheme supports homomorphic properties, enabling computations on shared inputs, which is crucial for applications in secure multi-party computation.

2.2 Proof of Systems

Proof systems are the cryptographic protocols where one party prover can convince another party verifier about the validity of a certain statement while disclosing nothing more than the fact that this statement is, indeed, valid [32].

$\mathsf{Exp}^{pok}_{\mathcal{B},\mathsf{V},\mathcal{R}}()$

1 : $(x_1, \pi_1, x_2, \pi_2) \leftarrow \mathcal{B}$
2 : $w_1 \leftarrow K(x_1, \pi_1)$
3 : $w_2 \leftarrow K(x_2, \pi_2)$
4 : $e \leftarrow \mathsf{V}(x_1, \pi_1) \wedge \mathsf{V}(x_2, \pi_2)$
5 : $rel \leftarrow \mathcal{R}(x_1, w_1) \wedge \mathcal{R}(x_2, w_2)$
6 : **return** $e \wedge \neg rel$

$\mathsf{Exp}^{\mathsf{zk},0}_{\mathcal{B},\mathsf{P},\mathcal{R}}(\lambda)$

1 : $(x, w, \mathsf{state}) \leftarrow \mathcal{B}(\lambda)$
2 : $\pi \leftarrow \perp$
3 : **if** $(\mathcal{R}(x, w))$ **then**
4 : $\quad \pi \leftarrow \mathsf{P}(x, w)$
5 : $\beta' \leftarrow \mathcal{B}(\mathsf{state}, \pi)$
6 : **return** β'

$\mathsf{Exp}^{\mathsf{zk},1}_{\mathcal{B},\mathsf{S},\mathcal{R}}(\lambda)$

1 : $(x, w, \mathsf{state}) \leftarrow \mathcal{B}(\lambda)$
2 : $\pi \leftarrow \perp$
3 : **if** $(\mathcal{R}(x, w))$ **then**
4 : $\quad \pi \leftarrow \mathsf{S}(x)$
5 : $\beta' \leftarrow \mathcal{B}(\mathsf{state}, \pi)$
6 : **return** β'

Figure 2.4: Experiments for Proof of knowledge and Zero-Knowledge.

Consider a binary (NP-relation) $\mathcal{R}$ over a pair of elements (x, w), where the element x is called a *statement* and w is called *witness*. A non-interactive proof system for $\mathcal{R}$ consists of a prover P and a verifier V algorithm which works on a common input x; the prover computes a proof $\pi \leftarrow \mathsf{P}(x, w)$ with the help of the witness and sends it to the verifier; the verifier then decides to whether to accept or reject the proof. A *proof system* is said to be

- *complete*, if for any $(x, w) \in \mathcal{R}$, we have that $\mathsf{V}(x, \mathsf{P}(x, w)) = \top$ with overwhelming probability.
- *soundness* if a prover cannot convince a verifier that a false statement is true.
- *proof of knowledge* [33] - if a prover can convince a verifier that a statement is true, then is possible to extract a witness for that statement. For any efficient adversary

$\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ the probability of winning the experiment in Figure 2.4 is negligible: $\Pr\left[\mathsf{Exp}^{pok}_{\mathcal{B},\mathsf{V},\mathcal{R}}() = 1\right] \leq \mathsf{negl}(\lambda)$.

- *zero-knowledge* It ensures that no information is leaked, beside the validity of the relation. Formally, we compare the execution of a zero-knowledge adversary $\mathcal{B}$ with that of a simulator S in the following two experiments. For any efficient adversary $\mathcal{B}$ the probability of winning the experiment in Figure 2.4 is negligible $\left|\Pr\left[\mathsf{Exp}^{\mathsf{zk},0}_{\mathcal{B},\mathsf{P},\mathcal{R}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{zk},1}_{\mathcal{B},\mathsf{S},\mathcal{R}}(\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda)$.

Schnorr Zero-knowledge System We introduce the Schnorr *non-interactive zero-knowledge proof* [34] that shows knowledge of the exponent without revealing it. This is formalized in Figure 2.5 as $\mathsf{ZKP}_1 = (\mathcal{R}'_1, \mathsf{P}'_1, \mathsf{V}'_1)$ for relation $\mathcal{R}'_1(\mathsf{a}, (\mathsf{pp}, \mathsf{A})) = [A = g^a \bmod p]$.

$\mathsf{P}'_1(\mathsf{a} : \mathsf{pp}, \mathsf{A})$	$\mathsf{V}'_1((\mathsf{pp}, \mathsf{A}, \pi)$
1 : $(p, q, g) \leftarrow \mathsf{pp}$	1 : $(p, q, g) \leftarrow \mathsf{pp}$
2 : $r \leftarrow\!\!\$\ \mathbb{Z}_q$	2 : $(c, z) \leftarrow \pi$
3 : $h \leftarrow g^r \bmod p$	3 : $h \leftarrow (g^z \cdot \mathsf{A}^{-c}) \bmod p$
4 : $c \leftarrow \mathsf{H}(\mathsf{A}, h)$	4 : **return** $(c = \mathsf{H}(\mathsf{A}, h))$
5 : $z \leftarrow (r + c \cdot \mathsf{a}) \bmod q$	
6 : $\pi \leftarrow (c, z)$	
7 : **return** π	

Figure 2.5: Schnorr Zero-knowledge Proof

It has been shown in [35] and [36], that the Schnorr protocol is *complete*, *sound*, and *zero-knowledge*.

Chaum-Pedersen Zero-knowledge System The Chaum-Pedersen proof [37] is based on the equality of discrete logarithms. For example, in the encryption of votes, it is to prove that encryption is a specific value (e.g., 0 or 1) without revealing the value used for encryption or the secret decryption key. Figure 2.6 is a model of this proof as $\mathsf{ZKP}_2 = (\mathcal{R}_2, \mathsf{P}'_2, \mathsf{V}'_2)$ for the relation $\mathcal{R}_2(\alpha, (\mathsf{pp}, \mathsf{A}, \mathsf{B})) = [\mathsf{A} = g^\alpha \bmod p \wedge \mathsf{B} = h^\alpha \bmod p]$.

In [36], [38] it was shown that the Chaum-Pedersen protocol is *complete*, *sound*, *zero-knowledge*.

Disjunctive Chaum-Pedersen There is a variant of Chaum-Pedersen zero-knowledge proof known as Witness-indistinguishable Chaum-Pedersen. In [39], how to transform a proof of knowledge into a witness-indistinguishable protocol has been presented. This is particularly useful in scenarios where the prover needs to prove knowledge of one out of several possible witnesses without revealing which one. It is applicable to a range of proofs for cardinal voting methods, where we need to prove that ciphertext is an encryption of an integer within a specified range. It has been formalized in Figure 2.7 as $\mathsf{ZKP}_3 =$

$\mathsf{P}'_2(\alpha : \mathsf{pp}, (\mathsf{A}, \mathsf{B}))$	$\mathsf{V}'_2((\mathsf{pp}, \mathsf{A}, \mathsf{B}), \pi)$
1 : $(p, q, g, h) \leftarrow \mathsf{pp}$	1 : $(c, z) \leftarrow \pi$
2 : $r \leftarrow\!\!\$\ \mathbb{Z}_q$	2 : $(p, q, g, h) \leftarrow \mathsf{pp}$
3 : $u \leftarrow g^r \bmod p$	3 : $u' \leftarrow (g^z \bmod p) \cdot (\mathsf{A}^{-c} \bmod p) \bmod p$
4 : $v \leftarrow h^r \bmod p$	4 : $v' \leftarrow (h^z \bmod p) \cdot (\mathsf{B}^{-c} \bmod p) \bmod p$
5 : $c \leftarrow \mathsf{H}(\mathsf{A}, \mathsf{B}, u, v) \bmod q$	5 : **return** $(c = \mathsf{H}(\mathsf{A}, \mathsf{B}, u', v') \bmod q)$
6 : $z \leftarrow (r + c \cdot \alpha) \bmod q$	
7 : $\pi \leftarrow (c, z)$	
8 : **return** π	

Figure 2.6: Chaum-Pedersen Zero-Knowledge Proof

$(\mathcal{R}_3, \mathsf{P}_3, \mathsf{V}_3)$ for the relation $\mathcal{R}_3(\alpha_1, \alpha_2, (\mathsf{pp}, \mathsf{A}_1, \mathsf{B}_1, \mathsf{A}_2, \mathsf{B}_2)) = [\mathsf{A}_1 = g^{\alpha_1} \bmod p \wedge \mathsf{B}_1 = h^{\alpha_1} \bmod p] \vee [\mathsf{A}_2 = g^{\alpha_2} \bmod p \wedge \mathsf{B}_2 = h^{\alpha_2} \bmod p]$ with

According to [36], this variant of Caum-Pedersen preserves properties of *completeness*, *soundness*, and *zero-knowledge*.

Fiat-Shamir Transformation The Fiat-Shamir transformation transforms an interactive proof system into a non-interactive one by replacing the verifier's challenge with a cryptographic hash [40]. Let $\Pi = (\mathcal{P}, \mathcal{V})$ be an interactive proof system where the prover $\mathcal{P}$ commits a value $\alpha = f(w, r)$, and the verifier challenges with $c \in \mathcal{C}$. In the Fiat-Shamir transformation, the challenge is computed non-interactively as $c = \mathsf{H}(x, \alpha)$, where $\mathsf{H} : \mathcal{X} \times \mathcal{A} \to \mathcal{C}$ is a cryptographic hash function, with H, $\mathcal{A}$, and $\mathcal{C}$ being the statement space, commitment space, and challenge space, respectively. The prover responds with $\beta = g(w, r, c)$, and the verifier accepts or rejects by verifying if $\mathcal{V}(x, \alpha, c, \beta) = \top$, where c is recomputed as $\mathsf{H}(x, \alpha)$.

This transformation makes the protocol non-interactive and secure in the random oracle model. If both parties follow the protocol honestly, the verifier always accepts with probability 1. Formally,

$$\Pr[\mathcal{V}(x, \alpha, c, \beta) = \top \mid R(x, w)] = 1.$$

If x is false, no cheating prover can convince the verifier, except with negligible probability ε, where

$$\Pr[\mathcal{V}(x, \alpha, c, \beta) = \top \mid \neg R(x, w)] \leq \varepsilon.$$

However, this transformation has a weakness [41] and should be used carefully, which is discussed in the Appendix 6.2.

$\mathsf{P}_3((\alpha, b) : (\mathsf{A}_1, \mathsf{B}_1, \mathsf{A}_2, \mathsf{B}_2))$

1 : $(p, q, g) \leftarrow \mathsf{pp}$
2 : where $\mathsf{A}_1 = g^{\alpha_1} \bmod p, \mathsf{B}_1 = h^{\alpha_1} \bmod p$
3 : and $\mathsf{A}_2 = g^{\alpha_2} \bmod p, \mathsf{B}_2 = h^{\alpha_2} \bmod p$
4 : $r_1, r_2 \leftarrow\!\!\$\ \mathbb{Z}_q$
5 : **if** $b = 0$ **then**
6 : $\quad u_1 \leftarrow g^{r_1} \bmod p$
7 : $\quad v_1 \leftarrow h^{r_1} \bmod p$
8 : $\quad c_2, z_2 \leftarrow\!\!\$\ \mathbb{Z}_q$
9 : $\quad u_2 \leftarrow (g^{z_2} \cdot \mathsf{A}_2^{-c_2}) \bmod p$
10 : $\quad v_2 \leftarrow (h^{z_2} \cdot \mathsf{B}_2^{-c_2}) \bmod p$
11 : $\quad c_1 \leftarrow \mathsf{H}(\mathsf{A}_1, \mathsf{B}_1, \mathsf{A}_2, \mathsf{B}_2, u_1, v_1, u_2, v_2) \bmod q$
12 : $\quad z_1 \leftarrow (r_1 + c_1 \cdot \alpha_1) \bmod q$
13 : **endif**
14 : **if** $b = 1$ **then**
15 : $\quad u_2 \leftarrow g^{r_2} \bmod p$
16 : $\quad v_2 \leftarrow h^{r_2} \bmod p$
17 : $\quad c_1, z_1 \leftarrow\!\!\$\ \mathbb{Z}_q$
18 : $\quad u_1 \leftarrow (g^{z_1} \cdot \mathsf{A}_1^{-c_1}) \bmod p$
19 : $\quad v_1 \leftarrow (h^{z_1} \cdot \mathsf{B}_1^{-c_1}) \bmod p$
20 : $\quad c_2 \leftarrow \mathsf{H}(\mathsf{A}_1, \mathsf{B}_1, \mathsf{A}_2, \mathsf{B}_2, u_1, v_1, u_2, v_2) \bmod q$
21 : $\quad z_2 \leftarrow (r_2 + c_2 \cdot \alpha_2) \bmod q$
22 : **endif**
23 : **return** (c_1, z_1, c_2, z_2)

$\mathsf{V}_3((\mathsf{A}_1, \mathsf{B}_1, \mathsf{A}_2, \mathsf{B}_2), (c_1, z_1, c_2, z_2))$

1 : $u_1 \leftarrow (g^{z_1} \cdot \mathsf{A}_1^{-c_1}) \bmod p$
2 : $v_1 \leftarrow (h^{z_1} \cdot \mathsf{B}_1^{-c_1}) \bmod p$
3 : $u_2 \leftarrow (g^{z_2} \cdot \mathsf{A}_2^{-c_2}) \bmod p$
4 : $v_2 \leftarrow (h^{z_2} \cdot \mathsf{B}_2^{-c_2}) \bmod p$
5 : $c' \leftarrow \mathsf{H}(\mathsf{A}_1, \mathsf{B}_1, \mathsf{A}_2, \mathsf{B}_2, u_1, v_1, u_2, v_2) \bmod q$
6 : **return** $(c_1 + c_2 = c' \bmod q)$

Figure 2.7: Disjunctive Chaum-Pedersen Zero-Knowledge Proof

Chapter 3

Literature Review

In this section, we provide an overview of e-voting syntax, entities, and properties.

3.1 Entities

The units and parties in an online voting scheme can be variable. Here, we explain the general entities that exist in most voting systems.

- **Server Administrator**: Performs all the general infrastructure maintenance, provides system availability, and is responsible for system security, without having privileges to access sensitive information such as votes.
- **Credential Authority (Election Authority)**: Issues Credentials to voters and other parties; authenticates voters before letting them vote. Only authenticates identities and issues public credit.
- **Trustees (Guardians)**: Perform the threshold decryption and count where all the trustees hold private pieces of the decryption key, and a quorum of trustees are, in fact, required to decrypt the result.
- **Voters**: Voters cast votes with weight 1. Each voter has a public key for authentication and a private key for signing their vote. Their identities are protected via anonymous credentials.
- **Bulletin Board**: The public-readable, append-only ledger to which each piece of relevant election information publishes-whether it be a list of ballots or list of credentials-publishes all types of public data, such as election data, public credentials, and the list of accepted ballots in support of accountability.

3.2 Mini-voting (Single-pass Voting)

In this section we recall *single-pass voting schemes*, the class of schemes that we use as basis for our analysis of voting schemes. We start with their syntax and then detail the

desired privacy and verifiability properties for such schemes. A *single-pass voting system* [42] is a tuple of algorithms

(Setup, Register, Vote, Valid, VerifyVote, Publish, Tally, Verify, Box, Count, Policy).

$\mathsf{Setup}(1^\lambda)$: Returns a pair of keys $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Register}(id)$: Creates a pair of signing keys $(\mathsf{upk}, \mathsf{usk})$ for the voter id.

$\mathsf{Vote}(id, v, \mathsf{pk}, \mathsf{usk})$: Constructs a ballot b for voter id that contains the encryption of their cast vote v with upk, and a signature over this encryption with the signing key usk. The verification key upk can trivially be computed from the signing key usk.

$\mathsf{Valid}(\mathsf{BB}, b, \mathsf{pk})$: Checks the validity of ballot b with respect to the ballot box BB.

$\mathsf{VerifyVote}(b, pbb)$: Evaluates if the ballot b was properly registered with respect to the public bulletin board pbb.

$\mathsf{Publish}(\mathsf{BB})$: Returns the public view of the ballot box BB called the public bulletin board.

$\mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$: Computes the result r of the election and a proof π of correct computation from BB.

$\mathsf{Verify}((\mathsf{pk}, pbb, r), \pi)$: Checks that π is a valid proof of correct computation for result r and public bulletin board pbb.

$\mathsf{Count}(L)$: Given a list of votes as input, a method of computing the result of the election is applied.

$\mathsf{Policy}(L)$: Applies a filtering policy that decides which vote is kept for each voter, given a list of voters and votes as input.

$\mathsf{Box}(\mathsf{BB}, b)$: Returns a ballot box BB'. If $\mathsf{Valid}(\mathsf{BB}, b, \mathsf{pk})$ holds then $\mathsf{BB}' = \mathsf{BB} \cup \{b\}$; otherwise $\mathsf{BB}' = \mathsf{BB}$.

3.3 Privacy

Vote privacy, is the idea that no one can obtain information about how individual voters have voted. For our analysis, we use the approach of Bernhard et al [42] who distinguish between three rather distinct aspects which impact the privacy of individual votes: ballot privacy, strong consistency and strong correctness. Together, the three notions imply security in a simulation-based sense. We will say a voting scheme is private, if it meets all three properties. In the following, we recall the intuition behind these definitions and explain how we adapt and extend them to the case of different voting systems.

We rely on the privacy defintion introduced by Cortier et. al. [43].

Ballot Privacy. Proving a voting scheme secure against ballot privacy (BPRIV) means that ballots do not leak information on votes: it should be impossible even for active adversaries who can submit arbitrary ballots, as formalized in the following game-based definition.

Definition 1 (Ballot Privacy[43])**.** A voting scheme $\mathcal{V}$ has ballot privacy if there exists a simulator Sim such that no efficient adversary $\mathcal{A}$ can distinguish between the games $\mathsf{Exp}^{\mathsf{BPRIV},0}_{\mathcal{A},\mathcal{V},\mathsf{Sim},I}(\lambda)$ and $\mathsf{Exp}^{\mathsf{BPRIV},1}_{\mathcal{A},\mathcal{V},\mathsf{Sim},I}(\lambda)$ defined in Figure3.1. That is, the expression

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

is negligible in λ, for any set of voters I.

An experiment is shown in Figure 3.1; in the experiments $\mathsf{Exp}^{\mathsf{bpriv},\beta}_{\mathcal{A},\mathcal{V},I}$, the adversary $\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2)$ has access to the set of oracles $\mathcal{O}=\{Ocast, Ovote, Otally, Oboard\}$. The adversary is allowed to call the *Otally* oracle at most once. In this model, the adversary tries to distinguish between two worlds. The adversary chooses votes for honest voters and is even allowed to cast ballots on behalf of corrupted users. The attacker obtains the bulletin board either from the real world $\beta=0$ or from an ideal world $\beta=1$, in which the votes of the honest parties are replaced by simulated ones. The adversary always has the actual count in mind, depending on BB_0. If $\beta=0$, the output is an honest proof of the correctness of the process, while if $\beta=1$, the proof is simulated by an efficient simulator that observes only the public part of the board. The challenge for the adversary is to distinguish whether the number of rounds went using $\beta=0$ or $\beta=1$.

It considers an adversary that attempts to distinguish between two worlds characterised by a hidden bit β. In the real world ($\beta=0$) honest votes are cast as chosen by the users, and the final tally is performed honestly by the authorities; in particular the tally may come with additional information (e.g. a ZK proof that the tally was performed honestly). In the ideal world ($\beta=1$) ballots submitted by honest parties contain fake votes. We emphasize that the tally is always performed on the real board (i.e. the BB_0): this captures the idea that the tally itself may reveal some information and that ballot privacy does not account for that information. Nonetheless, since the adversary always sees the "real" result, in the fake world (i.e. for $\beta=1$), security demands the existence of an efficient simulator (who only has access to the visible part of the real board) and needs to produce the additional information in a way that is not distinguishable by the adversary. Within this setup, if for some scheme the ballots leak information about the underlying votes then clearly no simulator would exist, and the scheme would be deemed insecure, as desired.

The security experiment starts by initializing various variables used throughout: the two boards BB_0 and BB_1, credentials for the users in the involved (those in I), and various bookkeeping lists. Next, the adversary decides on a list of users to corrupt (those in L) and obtains their credentials (maintained in list cL). The rest of the experiment models the execution of the protocol in the presence of the adversary. The capabilities of the adversary are modeled using the oracles in Figure 3.1, which we informally describe below:

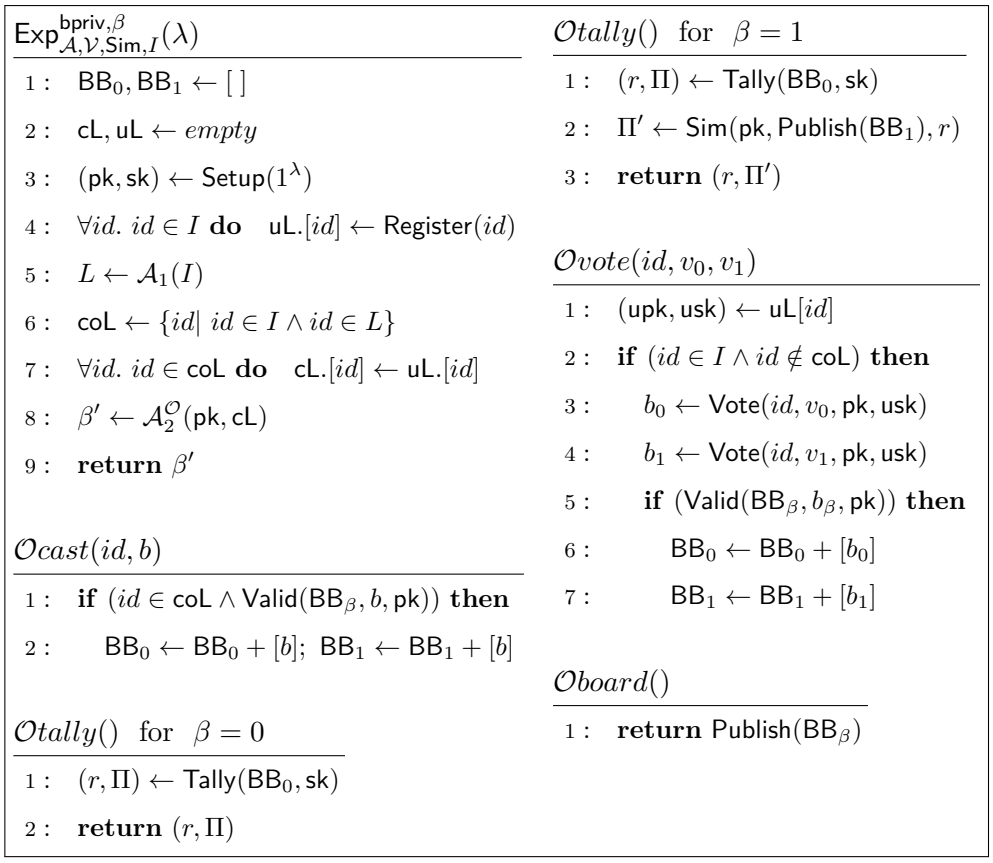


Figure 3.1: Ballot Privacy Experiment and Oracles

$\mathcal{O}vote(id, v_0, v_1)$: models voting by honest parties. The adversary decides on two votes for user id; the oracle uses the secret credential usk of id to create ballots b_0 and b_1 for v_0 and v_1. The oracle then checks if ballot b_β is valid with respect to board BB_β; if this is the case then b_0 is added to BB_0 and b_1 is added to BB_1.

$\mathcal{O}cast(id, b)$: models ballot submission by corrupt users. If ballot b submitted on behalf of user id is valid with respect to board BB_β (which is the board which the adversary can see) then b is added to both BB_0 and BB_1.

$\mathcal{O}board(\mathsf{BB})$: allows the adversary to access the public part of the bulletin board.

$\mathcal{O}tally()$: returns to the adversary the result of the tally. The oracle computes the result on board BB_0 and additional information that depends on β: if $\beta = 0$ this is the information that the tally authorities would normally return. If $\beta = 1$ it this information is provided by the simulator Sim.

This formalism constrains the ballot privacy adversary to corrupt users only before the voting process starts. This constraint is introduced since natural extensions of the existing ballot privacy definition to incorporate registration authorities are too strong. Indeed, a simple replay of a honest ballot as a cast ballot for the same voter (after corruption) shows that most voting protocols would be insecure under such extensions, although this scenario does not indicate a real weakness in the system.

Strong Consistency Informally, this notion demands that for any (adversarially produced) bulletin board for which each individual ballot is valid, the result provided by the Tally algorithm is the correct one.

The original definition associated voter identities with ballots; here, we use their public credentials instead.

Definition 2 (Strong Consistency)**.** A voting scheme $\mathcal{V}$ is *strongly consistent* if there exists:

- An extraction algorithm $\mathsf{Extract}(b, \mathsf{sk})$ that provides the upk with the value computed by decrypting of the ciphertext contained in the ballot b with the secret sk; and
- A ballot validation algorithm $\mathsf{ValidInd}(b, \mathsf{pk})$ that returns true iff the ballot b is "well-formed" with respect to some notion of well-formedness determined in advance by the election.

These algorithms must satisfy the following conditions:

- For any $(\mathsf{pk}, \mathsf{sk})$ obtained from $\mathsf{Setup}(\lambda)$, and any (id, v, usk) with upk trivially computed from usk, if $b \leftarrow \mathsf{Vote}(id, v, \mathsf{pk}, \mathsf{usk})$ then $\mathsf{Extract}(b, \mathsf{sk})$ returns (upk, v) with overwhelming probability.
- For any adversarially produced (BB, b), if $\mathsf{Valid}(\mathsf{BB}, b, \mathsf{pk})$ returns true, then $\mathsf{ValidInd}(b, \mathsf{pk})$ returns true as well.
- For any adversary $\mathcal{B}$ that returns a ballot box with ballots that satisfy $\mathsf{ValidInd}$, the experiment $\mathsf{Exp}^{\mathsf{consis}}_{\mathcal{B},\mathcal{V},I}(\lambda)$ specified in Figure 3.2 returns true with a probability negligible in the security parameter.

$\mathsf{Exp}^{\mathsf{consis}}_{\mathcal{B},\mathcal{V},I}(\lambda)$

1 : $\mathsf{uL} \leftarrow \mathsf{empty}$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
3 : $\forall id \in I$ **do** $\mathsf{uL}.[id] \leftarrow \mathsf{Register}(id)$
4 : $\mathsf{BB} \leftarrow \mathcal{B}(\mathsf{pk}, \mathsf{uL})$
5 : $e \leftarrow \forall b \in \mathsf{BB}.\ \mathsf{ValidInd}(b, \mathsf{pk})$
6 : $(r, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$
7 : **for** i **in** $1..|\mathsf{BB}|$ **do**
8 : $\quad dbb[i] \leftarrow \mathsf{Extract}(\mathsf{BB}[i], \mathsf{sk})$
9 : $r' \leftarrow \mathsf{Count} \circ \mathsf{Policy}(dbb)$
10 : **return** $(r \neq r' \wedge e)$

Figure 3.2: The Strong Consistency experiment

Strong Correctness Finally, strong correctness is the guarantee that an honestly created ballot will not be rejected by a ballot validation algorithm. The existing definition [42] demands that this property holds *even* with respect to boards that are created maliciously. While natural, this requirement is too strong for systems where there is a registration authority, and where validity of ballots depends on wether their associated credentials having been used already. For example, an adversary could easily win the game by returning a bulletin board which contains some identity id with a credential that does not belong to id and a ballot: any further vote by id with his own credential is rejected. Strong correctness can be incorporated in the security experiment if the condition that the board that the adversary returns satisfies some minimal well-formedness condition.

The formal definition uses the experiment in Figure 3.3. It considers an adversary $\mathcal{B}$ who after seeing the list of credentials (honestly generated by the registration authority), it choses a voter id, a vote v and outputs a bulletin board BB (lines 1-4 from the experiment). The experiment will check if an honestly created ballot by id for vote v will be accepted as valid with respect to BB. The minimal condition that BB needs to satisfy is that any ballot present on BB which involves either id or its credentials must involve both. The adversary wins if for such a board, the honest vote for id is invalid.

Definition 3 (Strong Correctness)**.** A voting scheme $\mathcal{V}$ is *strongly correct* if the advantage of any efficient adversary $\mathcal{B}$, defined by $\mathcal{A}^{\mathsf{corr}}_{\mathcal{B},\mathcal{V},I}(\lambda) = \Pr[\mathsf{Exp}^{\mathsf{corr}}_{\mathcal{B},\mathcal{V},I}(\lambda) = 1]$ (where $\mathsf{Exp}^{\mathsf{corr}}_{\mathcal{B},\mathcal{V},I}(\lambda)$ is defined in Figure 3.3) is negligible as a function of λ.

$\mathsf{Exp}^{\mathsf{corr}}_{\mathcal{B},\mathcal{V},I}(\lambda)$

1 : $\mathsf{uL} \leftarrow \mathsf{empty}$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
3 : $\forall id \in I$ **do** $\mathsf{uL}.[id] \leftarrow \mathsf{Register}(id)$
4 : $(id, v, \mathsf{BB}) \leftarrow \mathcal{B}(\mathsf{pk}, \mathsf{uL})$
5 : $e \leftarrow \mathsf{false}$
6 : **if** $(id \in I)$ **then**
7 : $\quad(\mathsf{usk}, \mathsf{upk}) \leftarrow \mathsf{uL}.[id]$
8 : $\quad e_1 \leftarrow \forall x.\forall y.\ (x, y, \star, \star) \in \mathsf{BB} \implies$
9 : $\quad\quad (x, y) = (id, \mathsf{upk}) \vee (x \neq id \wedge y \neq \mathsf{upk})$
10 : $\quad b \leftarrow \mathsf{Vote}(id, v, \mathsf{pk}, \mathsf{usk})$
11 : $\quad e_2 \leftarrow \mathsf{Valid}(\mathsf{BB}, b, \mathsf{pk})$
12 : $\quad e \leftarrow e_1 \wedge \neg e_2$
13 : **return** e

Figure 3.3: The Strong Correctness experiment

Ideal Functionality We describe an ideal voting functionality for a set of identities I and a result function $\rho : (I \times \mathbb{V})^* \rightarrow R$. This functionality has a simple behavior:

1) It first expects to receive one or more votes, both from honest and from corrupted voters. Every time a vote is received, the identity of the voter and the vote content are stored, and the functionality lets the adversary know who submitted a vote (but not the content of the vote, of course). This captures the idea that submitting a vote is a public action but could be relaxed in settings where voting would be private (though this usually has a cost in terms of eligibility verifiability.)

2) When it receives the order to tally, the functionality evaluates the ρ function on the sequence of pairs of identities and votes that it has received, and sends the result to the adversary.

Definition 4. The functionality $F_{\text{voting}}(\rho)$, interacting with an environment $\mathfrak{E}$ and an adversary S, proceeds as follows:
1) On input vote (id, v) from $\mathfrak{E}$ or S, store (id, v) and send ack (id) to S.
2) On input tally from S, return to both $\mathfrak{E}$ and S the result of the function ρ applied on the sequence of (id, v) pairs received, then halt.

This functionality has clear privacy properties: it reveals who votes, and the result of the election, but nothing more. Furthermore, the votes from the simulator are sent to the functionality in complete independence of the honest votes from the environment unless the environment itself tipped the simulator in the first place (which cannot be prevented by any voting system): the functionality does not give S any information related to the honest votes before it provides the election result. This seems to be the best we can hope for regarding the privacy of votes.

3.4 Other Notions of Privacy

Ballot Privacy with Malicious Box. Cortier et al. 2020 [44] introduced a stronger definition of ballot privacy where the adversary controls the ballot box. While ballots from honest voters are produced following the steps in the Vote algorithm, the adversary can manipulate them afterwards.

As seen from BPRIV definition the tally must not decrypt ballots from honest voters directly as that would ensure trivial wins for the adversary - as it will see v_0 in case left and v_1 in case right. The new definition introduces an algorithm Recover that simply replaces any ballots from honest voters that have not been changed by the adversary with the ballot from the left side. Thus, ensuring a consistent view of the tally in both worlds for ballots that the adversary has not manipulated.

Participation Privacy. To define participation privacy according to [45], we need dummy ballots, and in order to specify the dummy ballot casting algorithm for the posting trustee, we use two probability distributions $\mathbb{P}_d$ and $\mathbb{P}_t$. The first probability distribution $\mathbb{P}_d$ is used to sample a number of dummy ballots for each voter. This distribution therefore has a support $[x, y]$ with x, y as the minimal and maximal number of dummy ballots that the posting trustee is going to cast for each voter (i. e., $x \in \mathbb{N}_0$, $y \in \mathbb{N}_0 \cup \{\infty\}$). The

parameters x and y, as well as the exact $\mathbb{P}_d$ needs to be defined by the election authorities when setting up a corresponding system, i. e. their optimal trade-off between security and efficiency. The second probability distribution $\mathbb{P}_t$ is used to determine the time to cast each dummy ballot. Thus, this distribution has a support $[T_s, T_e]$ with T_s denoting the timestamp at the start of the voting phase and T_e the timestamp at the end of the voting phase. In order to obfuscate the ballots cast by voters, $\mathbb{P}_t$ should resemble the distribution of times at which the voters cast their ballots. For this, information from previous elections could be used. As the timestamp t denotes the time at which b is submitted to the bulletin board, we assume that it is chosen in $[T_s, T_e]$.

Adding other functionalities to the predefined voting scheme:

$\mathsf{Vote}'((id', \ \mathrm{sk}_{id'}), id, v, t)$: creates a ballot for voter $id \in I$ voting option v that is cast at a timestamp t. If $id = id'$, then a voter casting her own ballot. If $id' = \hat{id}$ (the posting trustee is casting a ballot on behalf of voter id) then $\mathrm{sk}_{id'}$ is not required but v must be 0 .

$\mathsf{VoteDummy}(id)$: is used by the posting trustee to cast dummy ballots for a given voter id. The posting trustee samples a random number $m \leftarrow \mathbb{P}_d$ and random timestamps $t_1, \ldots, t_m \leftarrow s\mathbb{P}_t$, and returns a set of ballots

$$\left(\mathsf{Vote}'\left((\hat{id}, 0), id, 0, t_1\right), \ldots, \mathsf{Vote}'\left((\hat{id}, 0), id, 0, t_m\right)\right).$$

The vote cast by the trustee on behalf of the voter id is represented by $\hat{id}$.

We consider the following experiment $\mathrm{Exp}^{\mathrm{ppriv},\,\beta}_{\mathcal{A},\mathcal{S},k}$ given the adversary $\mathcal{A} \in C_S$, so that C_S is a set of PPT adversaries, defined according the adversarial model for a particular scheme: There are two bulletin boards BB_0, BB_1 that are set up by the challenger. The adversary only sees the public output for one of these bulletin boards $\mathrm{BB}_\beta, \beta \leftarrow \{\{0,1\}$. Let Q_S be a set of oracle queries which the adversary has access to. Using these queries, the adversary fills both of the bulletin boards with additional content modeling the voting, so that BB_0 and BB_1 contain the same cast ballots except for the ballots for the voters id_0, id d_1; given a number of voting options $v_1, \ldots, v_{k'}$ chosen by the adversary, $k' \leq k$, for each $i = 0, 1$, the bulletin board BB_i contains the votes for $v_1, \ldots, v_{k'}$ on behalf of id_i and an abstention from the election is modeled for the voter id_{1-i}. The oracle computes the tally result R on BB_0. In case a voting scheme provides auxiliary output Π for the tally, the oracle returns (R, Π) in case $\beta = 0$, and simulates the auxiliary output $\Pi' = \mathrm{SimTally}\,(\mathrm{BB}_1, R)$, returning the tuple (R, Π') in case $\beta = 1$. The oracle further outputs the public content of BB_β to the adversary. The goal of the adversary is to guess whether the provided output corresponds to BB_0 or to BB_1, i.e. to guess β.

Assuming δ represents the adversarial advantage in distinguishing whether a particular voter participated or abstained and, k is the maximum number of ballots a voter could have cast in the election, the definition of (δ, k)-participation privacy is then as follows:

Definition 5 ((δ, k)-participation privacy)**.** The voting scheme $\mathcal{S}$ achieves (δ, k)-participation privacy given a subset of PPT adversaries C_S, if for any adversary $\mathcal{A} \in C_S, k \in \mathbb{N}$ and two

honest voter id_0, id_1 holds

$$\left|\Pr\left[\mathrm{Exp}^{\mathrm{ppriv},0}_{\mathcal{A},\mathcal{S},k} = 0\right] - \Pr\left[\mathrm{Exp}^{\mathrm{ppriv},1}_{\mathcal{A},\mathcal{S},k} = 0\right] - \delta\right|$$

is negligible in the security parameter.

We define C_S as a set of adversaries that are given access to the queries $Q_S = \{\mathcal{O}\mathrm{Cast}, \mathcal{O}\mathrm{VoteAbstain}, \mathcal{O}\mathrm{VoteLR}, \mathcal{O}\mathrm{Tally}\}$ in the experiment $\mathrm{Exp}^{\mathrm{ppriv}\beta}_{\mathcal{A},\mathcal{S},k}$. These queries are defined as Figure 3.4.

$\mathcal{O}\mathsf{VoteAbstain}(v_1, ..., v_{k'})$

1 : **if** $k' > k$ **then return** $\perp$
2 : $b_{0,1}, ..., b_{0,m_0} \leftarrow\$ \mathsf{VoteDummy}(id_0)$
3 : $b_{1,1}, ..., b_{1,m_1} \leftarrow\$ \mathsf{VoteDummy}(id_1)$
4 : **for** $j = 1, ..., k'$ **do**
5 : $\quad t'_j \leftarrow\$ \mathbb{P}_t$
6 : $\quad b'_{0,j} \leftarrow \mathsf{Vote}'((id_\beta, \mathsf{sk}_{id_0}), id_0, v_j, t'_j)$
7 : $\quad b'_{1,j} \leftarrow \mathsf{Vote}'((id_\beta, \mathsf{sk}_{id_1}), id_1, v_j, t'_j)$
8 : **endfor**
9 : $\mathsf{Append}(b_{0,1}, ..., b_{0,m_0}, \mathsf{BB}_0)$
10 : $\mathsf{Append}(b_{1,1}, ..., b_{1,m_1}, \mathsf{BB}_1)$
11 : $\mathsf{Append}(b'_{0,1}, ..., b'_{0,k'}, \mathsf{BB}_0)$
12 : $\mathsf{Append}(b'_{1,1}, ..., b'_{1,k'}, \mathsf{BB}_1)$

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

1 : **if** $\mathsf{Valid}(\mathsf{BB}_\beta, b)$ **then**
2 : $\quad \mathsf{Append}(b, \mathsf{BB}_0)$
3 : $\quad \mathsf{Append}(b, \mathsf{BB}_1)$
4 : **endif**

$\mathcal{O}\mathsf{VoteLR}(id, v_0, v_1, t)$

1 : $b_0 \leftarrow \mathsf{Vote}'((id_\beta, \mathsf{sk}_{id}), id, v_0, t)$
2 : $b_1 \leftarrow \mathsf{Vote}'((id_\beta, \mathsf{sk}_{id}), id, v_1, t)$
3 : **if** $\mathsf{Valid}(\mathsf{BB}_\beta, b_\beta) = 0$ **then**
4 : $\quad$ **return** $\perp$
5 : **endif**
6 : $\mathsf{Append}(b_0, \mathsf{BB}_0)$
7 : $\mathsf{Append}(b_1, \mathsf{BB}_1)$

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

1 : **if** $\beta = 0$ **then**
2 : $\quad$ **return** $\mathsf{Tally}(\mathsf{sk}, \mathsf{BB}_0)$
3 : **else**
4 : $\quad (R, \Pi) \leftarrow \mathsf{Tally}(\mathsf{sk}, \mathsf{BB}_0)$
5 : $\quad \Pi' \leftarrow \mathsf{SimTally}(\mathsf{BB}_1, R)$
6 : **endif**
7 : **return** (R, Π')

Figure 3.4: The Participation Privacy Game

3.5 Verifiability

At the very least, a verifiable voting scheme should allow voters to check that their vote contributed to the outcome of the election. This is usually done in an individual way: each user checks the presence of their ballot in the ballot box. In addition, it should also be possible for voters (or, better, for any third party) to check whether the published result of the election corresponds to the votes cast by voters (and recorded in the ballot box). Various definitions of verifiability have been proposed [46]. We consider the notion of *strong verifiability* introduced in [47]. It captures individual, universal, and eligibility verifiability and accounts for the fact that voters may not perform the verifiability checks.

Verifiability against dishonest ballot boxes and verifiability against dishonest registrars are the two types that have been well-studied.

Verifiability against a dishonest ballot box ensures that an adversary controlling the ballot box can only manipulate the votes of corrupted voters and potentially exclude unchecked votes from honest voters, but it cannot introduce arbitrary votes beyond the number of corrupted voters. On the other hand, verifiability against a dishonest registrar focuses on the adversary controlling voter credentials during the registration phase, ensuring that the election result reflects all votes from honest voters who checked, a subset of votes from unchecked honest voters, and some arbitrary votes from corrupted voters. Both approaches maintain integrity in the election outcome but focus on different points of potential adversarial control—either the ballot box or the voter registration. Appendix section provides further details regarding verifiability.

3.6 Helios Voting System

Helios is an open-source, Web-based voting platform designed to provide verifiable, secure elections. It provides a means for voters to securely cast their votes over the Web, assured of privacy yet in a transparent manner. Homomorphic encryption is used in Helios, among other cryptographic mechanisms that keep votes secret while the count in the election is verifiable publicly. It is rather destined for applications where transparency and auditability are required: university elections, small organizations, or public referendums. In fact, the voters can check by themselves that their votes have been counted, without leaking the preferred option [16].

At a high level, Helios works in a couple of phases: during setup, generation of cryptographic keys is done by the election officials, and then voter registration is done with their labels. Then, the voters securely cast their vote - by means of some encryption scheme that keeps the vote private. Finally, the encrypted votes are posted to some public board, with the validity of every vote checked to ensure it obeys the election rules. Finally, the ciphertexts are decrypted, and the votes are counted. A ZKP is also given that illustrates the correctness of the tally while keeping the individual vote secret [48]. Figure 3.5 represents how Helios is working, where Helios[$\mathsf{Flabel}, \mathsf{ValidInd}, \mathsf{E}, \mathsf{ZKP}$] algorithms with $\mathsf{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ and $\mathsf{ZKP} = (\mathsf{P}, \mathsf{V})$.

- $\mathsf{Setup}(1^\lambda, \mathsf{aux}, \mathsf{Vo})$: The algorithm Setup initiates the election by generating keys and registering voters. $\mathsf{KGen}(1^\lambda)$ is the key generation function, where 1^λ is the security parameter. This is a unary representation of the security level λ, which controls the size and strength of cryptographic elements (e.g., key length). The function outputs a public key pk (used for encrypting votes) and a private key sk (used for decryption during tallying). The algorithm returns the $(\mathsf{pk}, \mathsf{sk})$, where pk and sk are the public and private keys.

- $\mathsf{Vote}(\mathsf{id}, \ell, v, \mathsf{pk})$: This procedure is responsible for casting a vote by encrypting the voter's choice. The voter with identity id encrypts their vote v using the public

key pk and the encryption function $\mathsf{Enc}(\mathsf{pk}, v)$. The encryption function Enc takes as input the public key pk, and the plaintext vote v. It outputs the ciphertext c, which is a secure, encrypted version of the vote. The algorithm then constructs a pair $\mathsf{b} = (\mathsf{id}, \mathsf{c})$, containing the voter's identity id, and the encrypted vote c, which is returned to be posted on the bulletin board for later validation.

– $\mathsf{Valid}(\mathsf{BB}, \mathsf{uL}, \mathsf{b}, \mathsf{pk})$: This method verifies the correctness and legitimacy of the vote b before it is counted. BB is the public bulletin board, a shared structure where all encrypted votes are posted. The function starts by extracting $(\mathsf{id}, \mathsf{c})$ from the tuple b. The algorithm then checks two conditions to ensure the vote is valid:

  I No copying: The condition $e_1 \leftarrow \forall \mathsf{id}'.(\mathsf{id}', \mathsf{c}) \notin \mathsf{BB}$ ensures that the same ciphertext is not cast twice.

  II Valid encryption: The function $\mathsf{ValidInd}(\mathsf{b}, \mathsf{pk})$ performs a cryptographic check on the encrypted vote c. It ensures that c is a valid encryption under the public key pk. This could involve verifying a zero-knowledge proof that the ciphertext c corresponds to a valid vote without revealing the plaintext.

  The vote is deemed valid only if all conditions hold, i.e., $e_1 \wedge e_2$.

– $\mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$: The Tally algorithm for Helios Y processes the encrypted votes posted on the bulletin board BB and computes the final election outcome. It starts by identifying valid ballots, denoted as *sbb*, by filtering the entries on BB that meet cryptographic validity checks, specifically using the function $\mathsf{ValidInd}(\mathsf{b}, \mathsf{pk})$ on each entry. Once the valid ballots are collected, the algorithm applies the function $\mathsf{Policy}(sbb)$ to produce a refined set of ballots fbb, which may include specific election rules such as removing duplicates or handling invalid entries. Following this, the encrypted votes in fbb are aggregated using the $\mathsf{Add}(fbb)$ function, yielding a combined ciphertext c. This ciphertext c is then decrypted with the private key sk through the decryption function $\mathsf{Dec}(c, \mathsf{sk})$, producing the final election result r. To ensure transparency and accountability, the algorithm publishes the bulletin board's final state $\mathsf{Publish}(\mathsf{BB})$ and generates a zero-knowledge proof Π. The proof Π confirms that the tallying process, including decryption and result generation, is consistent with the public bulletin board and the private key without revealing any individual vote data. The final output is the result r and the proof Π, ensuring the correctness and transparency of the election outcome.

Helios has been studied in numerous papers, and here we recall the privacy results from Bernhard et. al. 2017 [45].

Lemma 7 ([45])**.** *Helios satisfies strong correctness and strong consistency.*

Lemma 8 ([45])**.** *For any two distinct votes v_0 and v_1, and for any two honest voters id_0 and id_1, the Helios voting system ensures adversary $\mathcal{A}$ cannot distinguish between the election where id_0 votes for v_0 and id_1 votes for v_1, and the election where id_0 votes*

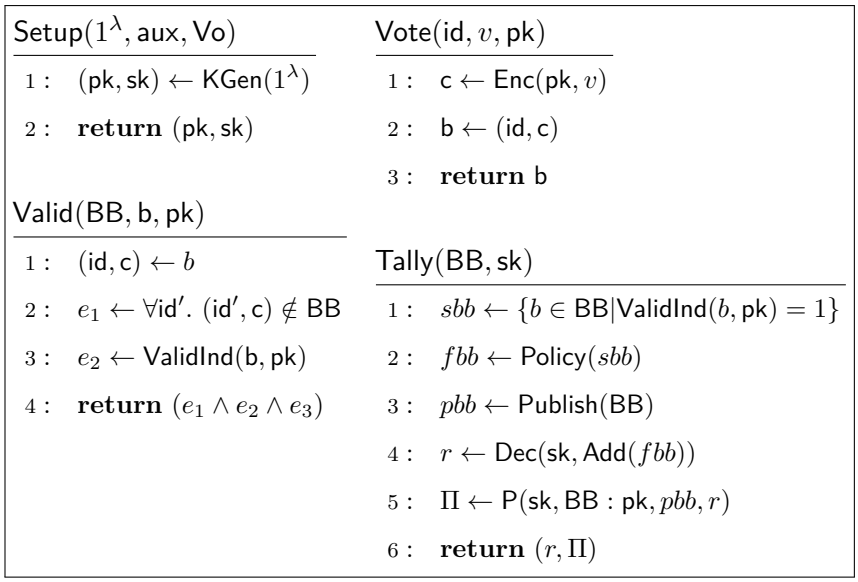


Figure 3.5: Helios Voting System

for v_1 and id_1 votes for v_0. The adversary's advantage in breaking ballot privacy is given by: $\mathrm{Adv}^{\mathsf{bpriv}}_{\mathcal{A},\mathcal{V}}(\lambda) = \left|\Pr\left[\mathsf{Exp}^{\mathsf{bpriv},0}_{\mathcal{A},\mathcal{V},I}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{bpriv},1}_{\mathcal{A},\mathcal{V},I}(\lambda) = 1\right]\right|$*, where* $\mathrm{Adv}^{\mathsf{bpriv}}_{\mathcal{A},\mathcal{V}}(\lambda)$ *is negligible as a function of the security parameter λ, meaning that the adversary's ability to distinguish between the two worlds is bounded by a negligible function. Therefore, Helios provides the guarantee that* $\mathrm{Adv}^{\mathsf{bpriv}}_{\mathcal{A},\mathcal{V}}(\lambda) \leq \varepsilon$*, where ε is a negligible function, and* $\mathsf{Exp}^{\mathsf{bpriv},\beta}_{\mathcal{A},\mathcal{V},I}(\lambda)$ *represents the adversary's interaction with the system under two different worlds: $\beta = 0$ where real votes are used, and $\beta = 1$ where simulated votes are used.*

Proof Sketch. The aim is to show under the assumptions of the ElGamal encryption scheme (which Helios uses and is IND−CPA secure), no adversary can distinguish between the encrypted votes with a probability significantly better than random guessing. Furthermore, using zero-knowledge proofs ensures that even if an adversary gains access to the cryptographic commitments or proofs on the bulletin board, they still cannot link a particular encrypted vote back to a specific voter. By proving that the adversary's advantage in breaking ballot privacy is negligible, we can conclude that Helios satisfies the ballot privacy property. □

Lemma 9 ([45])**.** *Considering* $\mathrm{Adv}^{\mathrm{ppriv}}_{\mathcal{A},\mathcal{V}}(k,\lambda) = \left|\Pr\left[\mathrm{Exp}^{\mathrm{ppriv},0}_{\mathcal{A},\mathcal{V},k} = 1\right] - \Pr\left[\mathrm{Exp}^{\mathrm{ppriv},1}_{\mathcal{A},\mathcal{V},k} = 1\right]\right|$ *as the adversary's advantage in distinguishing whether the voter id participated in the election or abstained, where* $\mathrm{Adv}^{\mathrm{ppriv}}_{\mathcal{A},\mathcal{V}}(k,\lambda)$ *is the advantage of the adversary in distinguishing between the two worlds where the voter either participates or abstains, and λ is the security parameter, Helios $\mathcal{V}$ provides participation privacy if for any adversary $\mathcal{A}$, the advantage is bounded by a negligible function* $\mathrm{Adv}^{\mathrm{ppriv}}_{\mathcal{A},\mathcal{V}}(k,\lambda) \leq \varepsilon$ *where ε is a negligible function λ.*

Proof Sketch. The participation privacy game would require an adversary to distinguish between two scenarios: one in which a voter abstains and dummy ballots are cast, and one in which the voter actively votes. If the adversary cannot tell the difference beyond a negligible probability, we can conclude that Helios ensures Participation Privacy, hiding whether a specific voter participated in the election. □

Chapter 4

ElectionGuard Formalization and Privacy

4.1 ElectionGuard

4.1.1 Overview and Procedure

ElectionGuard software development kit is designed to enhance the security and transparency of elections through integration with existing systems, adding cryptographic verification of end-to-end elections [17]. ElectionGuard works first through three essential stages: key generation, ballot encryption, and finally, tally decryption.

Starting with key generation, in the system, a group of entities called guardians generate a public key to be used for encrypting each vote cast in this election. Generation of the key was, by design, carried out in such a manner that no participant had any opportunity to take sole control over the process of encrypting/decrypting the votes, thereby spreading confidence among a number of guardians.

Along the entire chain of the voting process, ElectionGuard helps with the immediate encryption of every vote cast. In such a system, the content of every vote is encrypted in such a manner that, through homomorphic encryption, it can hold confidentiality and allow doing of mathematical operations like tallying on encrypted votes without decryption. This confirmation code one gets after voting is basically a cryptographic hash of the vote, encrypted. This allows the voter to check afterwards that their vote was recorded correctly, without leaking anything about the contents of their vote. ElectionGuard is designed to be agnostic with respect to different voting methods; it can be used with electronic ballot markers, hand-marked paper ballots, and even online voting.

After voting, the guardians shall then be reassembled for the final counting of votes. Homomorphic encryption will enable the addition of the votes together, tabulation securely while they are still encrypted, and guardians doing what is called distributed decryption in order to let the overall result be disclosed without exposing what's inside each individual vote. After this, there is a final publication of the overall result, where through the cryptographic proofs any third party could verify indeed whether the result of the tally

is accurate, with no vote manipulated or omitted, hence secure, private, with integrity assured in the election.

In the case of ElectionGuard, utilization of zero-knowledge proof would allow one verifying the correctness of the performed cryptographic operations, like vote encryption and tally decryption, without leaking sensitive information. On the other hand, this proof will still enable independent auditors or verifiers to proclaim the integrity of election results and thus give actual transparency and confidence in the electoral process. These make ElectionGuard a system where the accuracy of the results of an election can be independently checked without reliance on election software, hardware, or even administrators. It offers a sound method to provide the privacy of individual votes while ensuring transparency in the general process of elections [49].

In summary, the guardians of an election will each generate a public-secret key pair. The public keys will then be combined (as described in the following section) into a single election public key which is used to encrypt all selections made by voters in the election. At the conclusion of the election, each guardian will compute a verifiable partial decryption of each tally. These partial decryptions will then be combined to form full verifiable decryptions of the election tallies. To accommodate the possibility that one or more of the guardians will not be available at the conclusion of the election to form their partial decryptions, the guardians will cryptographically share their secret keys amongst each other during the key generation in a manner to be detailed in the next section. Each guardian will then compute a share of the secret decryption key, which it uses to form the partial decryptions. A pre-determined quorum value (k) out of the (n) guardians will be necessary to produce a full decryption. If the same set of n guardians supports multiple elections using the same threshold value k, the generated keys and key shares may be reused across several elections [19].

The next section shows how ElectionGuard operates; it presents the first formalization of ElectionGuard to make it more elegant for future research and proofs.

4.1.2 Components and Parameters

ElectionGuard uses the cryptographic hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{G}$, Hash-based Message Authentication Code $\mathsf{H} : \mathbb{Z}_p \times \{0,1\}^* \to \mathbb{G}$, the modified version of ElGamal on the key generation of cryptosystem $\mathsf{ElGamal}' = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$. This voting scheme utilizes the Encrypt-then-MAC $((c,t)) \leftarrow \mathsf{Enc}_5(\mathsf{pp}, m, k)$ and a technique for decrypting and verifying $m/\perp \leftarrow \mathsf{Dec}_5(\mathsf{pp}, \mathsf{sk}, (c,t))$; specific details of this technique are explained in Appendix. Moreover, it uses a verifiable secret-sharing scheme with $\mathsf{VSS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Rec}, \mathsf{Vf})$. This voting scheme is based on proof systems Schnorr $\mathsf{ZKP}_1 = (\mathcal{R}'_1, \mathsf{P}'_1, \mathsf{V}'_1)$ for proving shares of guardians in the setup phase and Chaum-Pedersen $\mathsf{ZKP}_2 = (\mathcal{R}_2, \mathsf{P}'_2, \mathsf{V}'_2)$ in the voting phase as well as counting the votes.

Like every other real-world voting system, ElectionGuard has many parameters apart from the cryptographic baselines. Here, we mentioned multiple of them starting with the public parameters pp which contains prime numbers p and q, and group generator g which

are getting used in different phases. To simplify in writing, every function that takes public parameters pp as input extracts the values p, q, and g, in the first step for further use as $(p, q, g) \leftarrow \mathsf{pp}$.

Throughout the election, three hash values are used as parameter base hash $\mathsf{H_P} \leftarrow \mathsf{H}(\mathtt{ver}; \mathtt{0x00}, p, q, g)$, election manifest hash $\mathsf{H_M} \leftarrow \mathsf{H}(\mathsf{H_P}; \mathtt{0x01}, \mathtt{manifest})$, and election base hash $\mathsf{H_B} \leftarrow \mathsf{H}(\mathsf{H_P}; \mathtt{0x02}, \mathsf{H_M}, n, k)$, where n indicates number of guardians and k shows quorum value (threshold) that describes the number of guardians necessary to decrypt tallies and produce election verification data. The values n and k are integers subject to the constraint that $1 \leq k \leq n$.

The `ver` denotes the version byte array that encodes the used version of ElectionGuard. This array has length 32 and contains the UTF-8 encoding of the string "v2.0.0" (this version oElectionGuard) followed by `0x00`-bytes. The `0x00`-byte at the beginning of the second argument of H for H_P is a domain separation byte written in hexadecimal notation.

`manifest` contains unique parameters for each election including contests, selectable options ballot styles, undervotes, overvotes, and optional data fields.

Additional information about the election, e.g. `ver`, `manifest`, etc. stored in aux represents the auxiliary data.

The notation U denotes the updated set of all n guardians participating currently in the election. Canvassing board members can often serve the role of election guardians. In each election, guardians are capable of operating the setup and tally phases in the presence of the election authority which generates and handles necessary information.

The generic version of ElectionGuard can be seen as:

$$\mathsf{EG} = (\mathsf{SetupEA}, \mathsf{SetupG}, \mathsf{Vote}, \mathsf{Valid}, \mathsf{Box}, \mathsf{VerifyVote}, \mathsf{Tally}, \mathsf{VerifyTally})$$

4.1.3 Setup Election Authority

The first step in this scheme is parameter generation by the authority of the election which is done by $\mathsf{SetupEA}(1^\lambda)$, which runs $\mathsf{ParGen}(1^\lambda)$ and generates the election hashes. The initialization of ElectionGuard is based on the value of q, a 256-bit prime order of a subgroup of $\mathbb{Z}_p^*$, and a 4096-bit prime modulus $p = 2q + 1$. As a cofactor of q, the value of r is considered to be $r = (p-1)/q$, along with a generator g such that $g = 2^r \mod p$ as Figure 4.1 shows. The output of $\mathsf{SetupEA}(1^\lambda)$ is returning the values of $p, q, g, \mathsf{H_P}, \mathsf{H_B}, \mathsf{H_M}$.

4.1.4 Setup Guardians

The setup phases aim to define how the joint election public key is henceforth to be used within the framework of the election for the encryption of votes-set up using a distributed key generation where a large number of guardians contribute partial pieces toward generating an election key in such a way that no one knows any full private key.

The function $\mathsf{SetupG}(1^\lambda)$ initiates the election by configuring the guardians. Initially, as shown in Figure 4.5, each guardian verifies the election parameters created by the election authority using $\mathsf{ParVer}([\mathsf{aux}], \mathsf{pp}, \mathsf{H_P}, \mathsf{H_B}, \mathsf{H_E})$. Subsequently, each guardian generates a

$\mathsf{ParGen}(1^\lambda)$

1 : $r \leftarrow 2$
2 : **do**
3 : **do**
4 : $q \leftarrow r + 1\ ,\ r \leftarrow r + 2$
5 : **while** $(\forall i \leq \sqrt{q},\ q \mod i = 0)$
6 : $p \leftarrow 2q + 1$
7 : **while** $(\forall j \leq \sqrt{p},\ p \mod j = 0)$
8 : **while** $(g^q \mod p \neq 1)$
9 : $g \leftarrow\$ [2, p-2]$
10 : **endwhile**
11 : **return** (p, q, g)

Figure 4.1: Parameter Instancing in ElectionGuard

private-public key pair, $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{sk}_i)$, along with a zero-knowledge proof π_i based on the Schnorr system, $\mathsf{ZKP_1}.\mathsf{P_1}(\mathsf{sk}_i, \mathsf{pp}, (\mathsf{pk}_i, [\mathsf{aux}]))$. This proof is designed to attest to the correctness of the public key. Once the keys and proofs are generated, each guardian exchanges their public key and proof with others. Verification of each received key and proof is conducted; for instance, G_ℓ verifies the received (pk_i, π_i) from G_i. If any public key or proof fails verification, a complaint is filed through $\mathsf{Complaint_1}(G_\ell, G_i, \mathsf{pk}_i, \pi_i)$, ensuring that potentially compromised keys or proofs are addressed. Next, guardians employ a secret sharing scheme to distribute their private keys. Each guardian G_i creates shares using $\mathsf{Share}(\mathsf{pp}, t, n, \mathsf{sk}_i)$, where t is the threshold needed for decryption, and n is the total number of guardians. Each share is encrypted for other guardians; for instance, G_i encrypts their share for G_ℓ as $E_\ell \leftarrow \mathsf{Enc_5}(\mathsf{pp}, \mathsf{shr}_i, \mathsf{pk}_\ell)$, and similarly G_ℓ encrypts their share for G_i as $E_i \leftarrow \mathsf{Enc_5}(\mathsf{pp}, \mathsf{shr}_\ell, \mathsf{pk}_i)$. The guardians exchange these encrypted shares and use their secret keys to decrypt the received shares, as G_ℓ decrypts E_ℓ to retrieve shr_i using $\mathsf{Dec_5}(\mathsf{pp}, \mathsf{sk}_\ell, E_\ell)$. Each decrypted share is then verified. If any decrypted share fails verification, a second type of complaint is issued, $\mathsf{Complaint_2}(G_\ell, G_i, \mathsf{pk}_i, E_\ell, (\ell, \mathsf{sk}_\ell))$, to report the discrepancy. Once all shares are verified as valid, the setup phase concludes, confirming that the guardians are configured correctly for the election.

Upon completion of guardians' setup, eventually, the election authority checks the validity of the election public key by making sure none of the conditions $(\mathsf{pk}_y \notin \mathbb{Z}_p^r)$, $(\mathsf{pk}_y \neq 1 \mod p)$, $(\mathsf{pk} = \prod_{y=0}^{t} \mathsf{pk}_y)$, and $(\mathsf{pk} \neq 1 \mod p)$ are holding, and generates extended base hash by $\mathsf{H_E} \leftarrow \mathsf{H}(\mathsf{H_B}; \mathtt{x012}, \mathsf{pk})$. As a recall, pk is the public key of the election, and pk_i is the guardian i's public key.

Parameter Verification

The ParVer validates the parameter hashes used in the ElectionGuard protocol. This method ensures the integrity and correctness of the parameter hashes by performing a series of checks as follows: It takes as input the auxiliary data [aux], the public parameters pp, and the hashes $\mathsf{H_P}$, $\mathsf{H_B}$, and $\mathsf{H_E}$, where the public parameters pp typically include the large prime modulus p, the small prime order q, and the generator g. The function first checks if $\mathsf{H_P}$ is correctly computed by verifying if $\mathsf{H_P} = \mathsf{H}(\texttt{ver}; \texttt{0x00}, p, q, g)$. If this check fails, it returns $\perp$. Next, it checks if $\mathsf{H_B}$ is correctly computed by verifying $\mathsf{H_B} = \mathsf{H}(\mathsf{H_P}; \texttt{0x01}, \texttt{manifest})$. Upon failing this check, it returns $\perp$. Finally, it checks if $\mathsf{H_E}$ is correctly computed by verifying if $\mathsf{H_E} = \mathsf{H}(\mathsf{H_P}; \texttt{0x02}, \mathsf{H_B}, n, k)$. If this check fails, it terminates the sub-procedure.

$\mathsf{ParVer}([\mathsf{aux}], \mathsf{pp}, \mathsf{H_P}, \mathsf{H_B}, \mathsf{H_E})$

1 : $e_1 \leftarrow (\mathsf{H_P} \neq \mathsf{H}(\texttt{ver}; \texttt{0x00}, p, q, g))$

2 : $e_2 \leftarrow (\mathsf{H_B} \neq \mathsf{H}(\mathsf{H_P}; \texttt{0x01}, \texttt{manifest}))$

3 : $e_3 \leftarrow (\mathsf{H_E} \neq \mathsf{H}(\mathsf{H_P}; \texttt{0x02}, \mathsf{H_B}, n, k))$

4 : **return** $(e_1 \wedge e_2 \wedge e_3)$

Figure 4.2: Parameter Validating

Key Generation

As Figure 4.4 shows, the KeyGen algorithm is used to generate a pair of public and secret keys for the election protocol. This method ensures that each participant has a valid key pair by performing the following steps: It takes as input the public parameters pp and a secret value sk_i belonging to the participant. The algorithm first randomly samples a secret key sk from the finite field $\mathbb{Z}_q$. Then, the public key pk is computed as $g^{\mathsf{sk}} \mod p$, where g is a generator of the cyclic group, and p is a large prime modulus. Finally, it outputs the key pair $(\mathsf{sk}, \mathsf{pk})$, where sk is kept private, and pk is shared publicly for further operations.

In this report, we handle this in a combined way, but the polynomial generation can also be formalized as a separate procedure as presented in Figure 4.3, the PolyGen algorithm generates a polynomial for each participant in the scheme. It ensures that the secret key of each guardian is embedded as the first coefficient of the polynomial. The algorithm proceeds as follows: It takes as input the public parameters pp. For each guardian, indexed by i, it generates $k-1$ random coefficients $a_{i,j}$ for the polynomial, where $j \in [1, k-1]$ and $a_{i,j}$ is sampled from the finite field $\mathbb{Z}_q$. The constant term $a_{i,0}$ is set to the guardian's secret key sk_i. The polynomial $P_i(x)$ is then constructed as $a_{i,0} + a_{i,1}x + a_{i,2}x^2 + \cdots + a_{i,k-1}x^{k-1}$, where x is the variable. The algorithm returns the polynomial $P_i(x)$.

$\mathsf{PolyGen}(\mathsf{pp}, \mathsf{sk}_i)$

1 : // The value of i indicates index of guardian
2 : **for** $j \in [1, k-1]$ **do**
3 : $a_{i,j} \leftarrow\!\!\$\ \mathbb{Z}_q$
4 : **endfor**
5 : $a_{i,0} \leftarrow \mathsf{sk}_i$
6 : $P_i(x) \leftarrow a_{i,0} + a_{i,1}x + a_{i,2}x^2 + \ldots + a_{i,k-1}x^{k-1}$
7 : **return** $(P_i(x))$

Figure 4.3: Polynomial Generation

$\mathsf{KeyGen}(\mathsf{pp})$

1 : $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_q$
2 : $\mathsf{pk} \leftarrow g^{\mathsf{sk}} \mod p$
3 : **return** $(\mathsf{sk}, \mathsf{pk})$

Figure 4.4: Key Generation

Complaints

During the Setup, if guardians get suspected of the honesty of each other, they can complain to the election authority. If a guardian, as a Plaintiff Plaint, realizes the invalidity of the information of another guardian, known as Respondent Respon, the plaintiff will use Complaint function to raise a complaint by submitting evidence, and respondent reveals a value related to evidence by publishing it.

- $\mathsf{Complaint}_1(\mathsf{Plaint}, \mathsf{Respon}, \mathsf{pk}, \pi_i)$: addresses a situation where Plaint, raises a complaint against Respon, regarding the validity of commitments (including public key) and the associated zero-knowledge proof. Upon using this function, Respon should reveal the secret polynomial. The election authority is going to check by applying the revealed values on $\mathsf{ZKP}_1.\mathsf{V}_1(\mathsf{pp}, (K_{i,j}, [\mathsf{aux}]), \pi_i)$. If the Respondent Respon fails to provide valid values, then he is removed from the set of guardians G, the total number of guardians n is decremented by 1, and the Setup gets restarted with updated guardians. But, if Respon succeeds in providing valid values, then Plaint is removed from the set of guardians G, the total number of guardians n is decremented by 1, and the Setup gets restarted with updated guardians.

- $\mathsf{Complaint}_2(\mathsf{Plaint}, \mathsf{pk}_i, \{A_0, \ldots, A_{t-1}\}, (i, s_\ell))$: deals a status where Plaint, raises a complaint against Respon, regarding the validity of decryption share. This function takes as input the Plaintiff Plaint, the Respondent Respon, the public key pk_i of the respondent, the set of commitments $\{A_0, \ldots, A_{t-1}\}$, extracted from E_ℓ, and the decryption share (i, s_ℓ). When this function gets called, Respon should reveal the secret polynomial and the value used to encrypt it. The election authority is going to check by applying the revealed values on $\mathsf{Vf}(\mathsf{pp}, t, n, \mathsf{pk}_i, \mathsf{shr}_i(\ell, s_\ell))$. If the Respondent Respon fails to provide valid values, then he is removed from the set of guardians G, the total number of guardians n is decremented by 1, and the Setup gets restarted with updated guardians. But, if Respon succeeds in providing valid values, then Plaint is removed from the set of guardians G, the total number of guardians n is decremented by 1, and the Setup gets restarted with updated guardians.

Therefore, complaint sub-procedures ensure that guardians are held accountable for the correctness of their contributions in the Setup.

Figure 4.5: Setup Phase in Generic ElectionGuard

	$\mathbf{G}_i$		$\mathbf{G}_\ell$
1	$\mathsf{ParVer}([\mathsf{aux}], \mathsf{pp}, \mathsf{H_P}, \mathsf{H_B}, \mathsf{H_E})$		$\mathsf{ParVer}([\mathsf{aux}], \mathsf{pp}, \mathsf{H_P}, \mathsf{H_B}, \mathsf{H_E})$
2	$(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{sk}_i)$		$(\mathsf{sk}_\ell, \mathsf{pk}_\ell) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{sk}_\ell)$
3	$\pi_i \leftarrow \mathsf{ZKP_1.P_1}(\mathsf{sk}_i, \mathsf{pp}, (\mathsf{pk}_i, [\mathsf{aux}]))$		$\pi_\ell \leftarrow \mathsf{ZKP_1.P_1}(\mathsf{sk}_\ell, \mathsf{pp}, (\mathsf{pk}_\ell), [\mathsf{aux}]))$
4		$\mathsf{pk}_i, \overbrace{(c_i, z_i)}^{\pi_i} \longrightarrow$	
5			**if** $(\mathsf{pk}_i \notin \mathbb{Z}_p^r) \vee (c_i \notin \mathbb{Z}_q)$ **then** $\perp$
6			**if** $(\neg \mathsf{ZKP_1.V_1}(\mathsf{pp}, (\mathsf{pk}_i, [\mathsf{aux}]), \pi_i))$ **then**
7			$\mathsf{Complaint}_1(G_\ell, G_i, \mathsf{pk}_i, \pi_i)$
8		$\mathsf{pk}_\ell, \overbrace{(c_\ell, z_\ell)}^{\pi_\ell} \longleftarrow$	
9	**if** $(\mathsf{pk}_\ell \notin \mathbb{Z}_p^r) \vee (c_\ell \notin \mathbb{Z}_q)$ **then** $\perp$		
10	**if** $(\neg \mathsf{ZKP_1.V_1}(\mathsf{pp}, (\mathsf{pk}_\ell, [\mathsf{aux}]), \pi_\ell))$ **then**		
11	$\mathsf{Complaint}_1(G_i, G_\ell, \mathsf{pk}_\ell, \pi_\ell)$		
12	$\mathsf{shr}_i \leftarrow \mathsf{Share}(\mathsf{pp}, t, n, \mathsf{sk}_i)$		$\mathsf{shr}_\ell \leftarrow \mathsf{Share}(\mathsf{pp}, t, n, \mathsf{sk}_\ell)$
13	$\forall j \in U:\ E_\ell \leftarrow \mathsf{Enc_5}(\mathsf{pp}, \mathsf{shr}_i, \mathsf{pk}_\ell)$		$\forall j \in U:\ E_i \leftarrow \mathsf{Enc_5}(\mathsf{pp}, \mathsf{shr}_\ell, \mathsf{pk}_i)$
14		$E_\ell \longrightarrow$	
15			$\forall j \in U. \ell \neq j:\ \mathsf{shr}_i \leftarrow \mathsf{Dec_5}(\mathsf{pp}, \mathsf{sk}_\ell, E_\ell)$
16			**if** $(\neg \mathsf{Vf}(\mathsf{pp}, t, n, \mathsf{pk}_i, (\mathsf{shr}_i, (\ell, \mathsf{sk}_\ell))))$ **then**
17			$\mathsf{Complaint}_2(G_\ell, G_i, \mathsf{pk}_i, E_\ell, (\ell, \mathsf{sk}_\ell))$
18		$E_i \longleftarrow$	
19	$\forall j \in U. i \neq j:\ \mathsf{shr}_\ell \leftarrow \mathsf{Dec_5}(\mathsf{pp}, \mathsf{sk}_i, E_i)$		
20	**if** $(\neg \mathsf{Vf}(\mathsf{pp}, t, n, \mathsf{pk}_\ell, (\mathsf{shr}_\ell, (i, \mathsf{sk}_i))))$ **then**		
21	$\mathsf{Complaint}_2(G_i, G_\ell, \mathsf{pk}_\ell, E_i, (i, \mathsf{sk}_i))$		

The described key generation process makes it impossible for any guardian to access the whole private key; that is just about the most solid security guarantee. Each guardian contributes to integrity via zero-knowledge proof, proving to know the secret share of the key without revealing it. Thus, this generates distributed and verifiable keys, that are secure, auditable, and resistant to insider attacks.

4.1.5 Vote

In the voting algorithm, a voter's choice for a particular candidate in a contest is encrypted, and proof is generated to ensure the integrity and confidentiality of the vote.

The algorithm Vote takes three inputs: the voter's identifier id, their choice vote, and the public key pk for the election. The encryption is performed using the ElGamal′.Enc algorithm, which takes the voter's choice vote and the public key pk to generate the

ciphertext $\mathbb{c}$ along with a random nonce ξ.

To ensure that the vote has been correctly encrypted, the algorithm computes a zero-knowledge proof π_1 using the $\mathsf{ZKP}_2.\mathsf{P}_2$ protocol. This proof confirms that the ciphertext $\mathbb{c}$ represents a valid vote, typically verifying whether the vote corresponds to a binary choice, like voting for one of two candidates.

In addition to the binary proof, the function also supports the verification of more complex election types where multiple candidates can be chosen. A proof, π_2, can generated using the $\mathsf{ZKP}_5.\mathsf{P}_5$ protocol. This range proof ensures that the encrypted vote falls within a valid range (e.g., 0 to R, where R might represent the maximum number of candidates). The proof relies on the secret nonce ξ and an auxiliary parameter ℓ to perform the range verification. Details of this type of range proof are explained in the Appendix.

The function concludes by bundling the voter's identifier id, the ciphertext $\mathbb{c}$, and the first proof π_1 into a package B. This package B is the ballot that represents the encrypted vote and can be submitted for further processing or storage in the ballot box.

$\mathsf{Vote}(id, \mathsf{vote}, \mathsf{pk})$

1 : $(\mathbb{c}, \xi) \leftarrow \mathsf{ElGamal}'.\mathsf{Enc}(\mathsf{pk}, \mathsf{vote})$

2 : $\pi_1 \leftarrow \mathsf{ZKP}_2.\mathsf{P}_2(\xi, \mathsf{pp}, \mathbb{c})$

3 : // The below range proof is for score election [50]

4 : // $\pi_2 \leftarrow \mathsf{ZKP}_5.\mathsf{P}_5((\xi, \ell), \mathsf{pp}, \mathbb{c}, [\mathsf{aux}])$

5 : $B \leftarrow (id, \mathbb{c}, \pi_1)$

6 : **return** B

Figure 4.6: Voting in Generic ElectionGuard

4.1.6 Verification of the Vote

The VerifyVote algorithm is designed to verify whether a particular ballot B has been correctly posted on the public bulletin board *pbb*. The algorithm works by first computing the hash of the ballot B, denoted as $\mathsf{H}(B)$. This hash serves as a unique identifier for the ballot, ensuring that any alteration to B would result in a different hash, thus preserving the integrity of the vote. The algorithm then checks whether this computed hash exists in the public bulletin board *pbb*. The public bulletin board is assumed to be a trusted and publicly accessible ledger that stores the hash of every valid ballot. If the hash of ballot B is found in *pbb*, the algorithm returns a positive result, confirming that the vote has been recorded. Otherwise, it returns a negative result, indicating that the ballot is either missing or invalid as $\mathsf{VerifyVote}(B, pbb)$: **return** $\mathsf{H}(B) \in pbb$.

In short, VerifyVote provides a mechanism for verifying that a voter's ballot has been properly included in the public election record.

4.1.7 Tally

The function $\mathsf{TallyVotes}(1^\lambda)$ performs the process of tallying and counting votes in the election. As depicted in Figure 4.7, each guardian starts by validating the bulletin board BB by checking if it is correctly formed using $\mathsf{Valid}(\mathsf{BB})$, returning $\perp$ in case of invalid input. Once validated, each guardian retrieves the finalized bulletin board FBB via $\mathsf{Policy}(\mathsf{BB})$. The guardians then compute the sum of the encrypted votes (A, B) using the homomorphic addition function $\mathsf{ElGamal'.Add}(\mathsf{FBB})$.

Each guardian proceeds to partially decrypt the vote tally by invoking $\mathsf{ElGamal'.Dec}(\mathsf{sk}_i, (\mathsf{A}, \mathsf{B}))$ using their secret key sk_i, yielding the partial result res_i. This partial result is broadcast to all other guardians. After receiving the partial results from other guardians, each guardian computes the Lagrange coefficient w_j by $\prod_{\ell \in U \setminus \{j\}} \frac{\ell}{\ell - j} \mod q$ for all participants $j \in U$, which enables them to combine the partial decryptions into the final decryption result. This is done by computing $\mathsf{result} \leftarrow \prod_{j \in U} \mathsf{res_j}^{w_j} \mod p$, effectively aggregating the contributions from all guardians.

Next, each guardian samples a random value $r_i \in \mathbb{Z}_q$ and computes the values $u_i \leftarrow g^{r_i} \mod p$ and $v_i \leftarrow \mathsf{A}^{r_i} \mod p$. These values are broadcast to all other guardians. The guardians aggregate the values received from other guardians to compute $u \leftarrow \prod_{j \in U} u_j \mod p$ and $v \leftarrow \prod_{j \in U} v_j \mod p$. A challenge c is computed using the hash function H over several parameters, including the public key, the ciphertext (A, B), and the aggregated values u, v. The challenge is then distributed among the guardians by computing $(c \cdot w_j) \mod q$ for each guardian.

The set U contains all guardians that were accepted after the setup phase. The value $A_{j,k} = g^{a_{j,k}}$ is a commitment to $a_{j,k}$. The value $a_{j,k}$ is the coefficient to the polynomial of degree t in the setup phase where secret key sk_j is shared.

The guardians compute a share of the secret using Shamir's secret sharing, where $\mathsf{share}_i = \sum_{j \in U} \mathsf{shr_{j,i}} \mod q$. Each guardian then computes the response $z_i \leftarrow (r_i - c_i \cdot \mathsf{share}_i) \mod q$. These responses are exchanged among the guardians and used to recompute the values (u'_j, v'_j) for verification, ensuring that the computations are consistent with the original broadcast values. If any guardian detects a mismatch, they abort the process by returning $\perp$.

Finally, if all values are verified, the responses are aggregated as $z \leftarrow \sum_{j \in U} z_j \mod q$, and the proof $\pi \leftarrow (c, z)$ is formed. The function returns the final result of the tally along with the proof (result, π), certifying the correctness of the tally computation.

4.1.8 Verification of Tally

The $\mathsf{VerifyTally}$ algorithm is responsible for verifying the correctness of the tally results by checking the aggregated result r and its accompanying proof π against the public key pk and the public bulletin board *pbb*. The algorithm takes three inputs: the public key pk, the public bulletin board *pbb*, and the tally result T. It first extracts the aggregated result r and the zero-knowledge proof π from the tally T. The core of the verification process

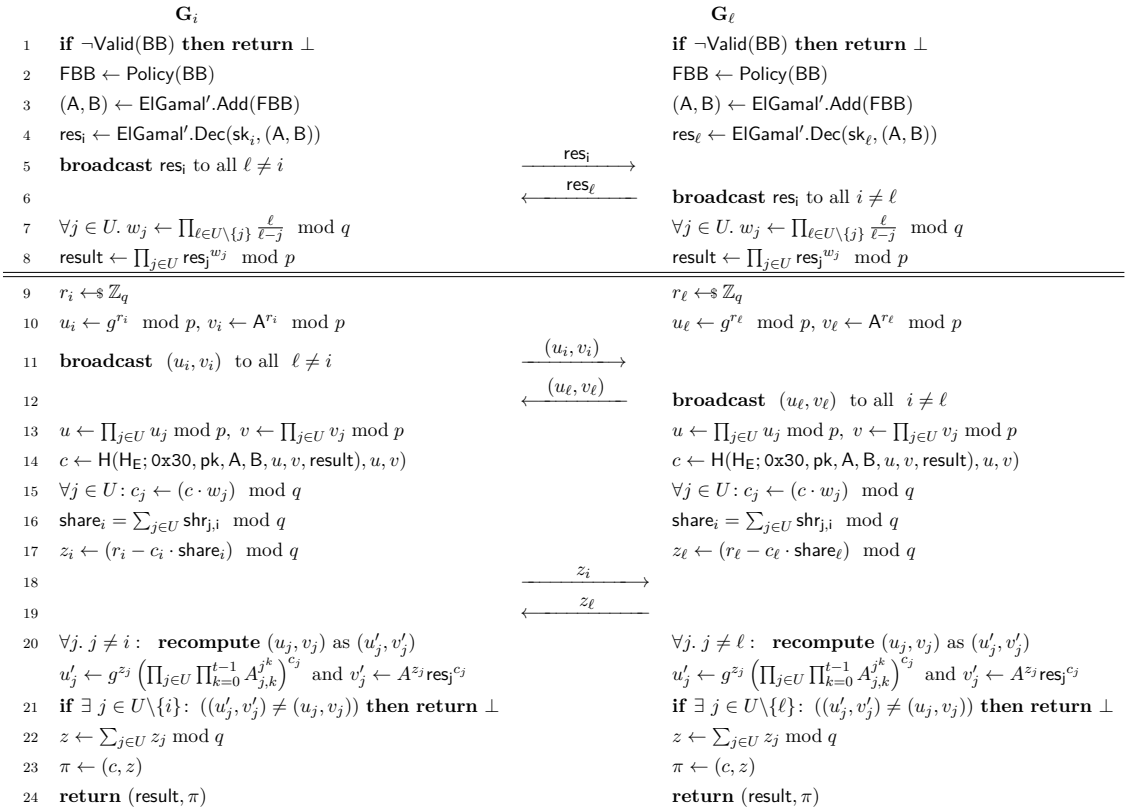


Figure 4.7: Tally and Counting Votes in ElectionGuard

$\mathsf{VerifyTally}(\mathsf{pk}, pbb, T)$

1 : $(r, \pi) \leftarrow T$

2 : **return** $\mathsf{V}((\mathsf{pk}, pbb, r), \pi)$

Figure 4.8: Verification of the Tally in Generic ElectionGuard

is performed by the function $\mathsf{V}((\mathsf{pk}, pbb, r), \pi)$, which checks the correctness of the tally r using the proof π and ensures that it corresponds to the public data on the bulletin board pbb.

If the verification is successful, the algorithm returns a positive result, indicating that the tally is valid and consistent with the underlying votes. Otherwise, it returns a failure, signaling that the tally or the proof does not hold.

4.2 ElectionGuardSimple

This section presents the ElectionGuard system with a single guardian. In the simplified version of the ElectionGuard protocol, named ElectionGuardSimple shown by EG', the election process is handled with only one guardian, simplifying certain cryptographic operations but preserving the core functionalities of setup, voting, tallying, and verification.

As Figure 4.9 shows, ElectionGuardSimple can be seen as:

$$\mathsf{EG}' = (\mathsf{Setup}, \mathsf{Vote}, \mathsf{VerifyVote}, \mathsf{Tally}, \mathsf{VerifyTally})$$

- The $\mathsf{Setup}(1^\lambda)$ algorithm initializes the election parameters. First, the algorithm performs a parameter verification check ParVer to ensure that the public parameters pp and cryptographic hash functions $\mathsf{H_P}$, $\mathsf{H_B}$, and $\mathsf{H_M}$ are valid. If the check fails, the setup process terminates. Otherwise, the algorithm proceeds to generate a secret key sk and a corresponding public key pk using the key generation function $\mathsf{KGen}(1^\lambda)$. Finally, the guardian returns the key pair $(\mathsf{sk}, \mathsf{pk})$, which will be used for encryption and decryption during the election.

- The $\mathsf{Vote}(id, \sigma, \mathsf{pk})$ algorithm handles the encryption of a voter's choice for one of the contests. The voter's selection σ, which could represent a single or multiple votes, is encrypted using the public key pk with a random nonce, producing the ciphertext $\mathbb{c}$ along with random values ξ and ξ_B. The encryption process also generates two zero-knowledge proofs: π_1 to prove that the encrypted value is either zero or one, and π_2 to prove that the encrypted value is within a valid range (between 0 and R). The ballot B is then formed by combining the voter's ID, the ciphertext $\mathbb{c}$, and the two proofs π_1 and π_2. The ballot B can then be published on the public bulletin board.

- The $\mathsf{VerifyVote}(B, pbb)$ algorithm is used to verify the integrity of a submitted vote. This verification process ensures that the hash of the ballot B is present in the public bulletin board pbb, confirming that the vote was recorded correctly. The algorithm returns a positive result if the ballot hash is found, indicating a valid vote.

- $\mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$: The Tally algorithm processes the encrypted votes in the ballot box BB and calculates the final result of the election. First, it checks the validity of the ballots in BB by invoking $\mathsf{Valid}(\mathsf{BB})$. If the ballots are valid, the algorithm proceeds by applying the $\mathsf{Policy}(\mathsf{BB})$ function, which filters or refines the ballots according to specific election rules, resulting in the set fbb of valid ballots. The algorithm then aggregates the votes by summing the ciphertexts in fbb using the function $\mathsf{Add}(fbb)$, which produces a combined ciphertext. This ciphertext is decrypted using the private key sk with the function $\mathsf{Dec}(\mathsf{sk}, \mathsf{Add}(fbb))$, yielding the final election result r. To ensure transparency and verify the integrity of the tallying process, the algorithm generates a zero-knowledge proof Π through $\mathsf{P}(\mathsf{sk} : \mathsf{pk}, \mathsf{BB}, r)$. This proof Π confirms that the decryption and result r are consistent with the inputs (the private key, public ballot box, and the result) without disclosing any sensitive information. The Tally algorithm then outputs the final result r and the proof Π, assuring both the correctness and privacy of the tallying procedure.

- The $\mathsf{VerifyTally}(\mathsf{pk}, pbb, T)$ algorithm verifies the correctness of the tally result r and the accompanying proof Π. The verification function V ensures that the decrypted

tally is consistent with the public key pk, the data published on the public bulletin board pbb, and the proof Π. If the verification succeeds, the tally is deemed valid.

$\mathsf{Setup}(1^\lambda)$

1 : **if** $(\neg\mathsf{ParVer}([\mathsf{aux}], \mathsf{pp}, \mathsf{H_P}, \mathsf{H_B}, \mathsf{H_M}))$ **then** $\perp$
2 : $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$
3 : **return** $(\mathsf{sk}, \mathsf{pk})$

$\mathsf{Vote}(id, \mathsf{vote}, \mathsf{pk})$

1 : $(\mathbb{c}, \xi) \leftarrow \mathsf{ElGamal}'.\mathsf{Enc}(\mathsf{pk}, \mathsf{vote})$
2 : $\pi \leftarrow \mathsf{ZKP}_2.\mathsf{P}_2(\xi, \mathsf{pp}, \mathbb{c})$
3 : $B \leftarrow (id, \mathbb{c}, \pi)$
4 : **return** B

$\mathsf{VerifyVote}(B, pbb)$

1 : **return** $\mathsf{H}(B) \in pbb$

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

1 : **if** $\mathsf{Valid}(\mathsf{BB})$ **then**
2 : $\quad fbb \leftarrow \mathsf{Policy}(\mathsf{BB})$
3 : $r \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{Add}(fbb))$
4 : $\Pi \leftarrow \mathsf{P}(\mathsf{sk} : \mathsf{pk}, \mathsf{BB}, r)$
5 : **return** (r, Π)

$\mathsf{VerifyTally}(\mathsf{pk}, pbb, T)$

1 : $(r, \pi) \leftarrow T$
2 : **return** $\mathsf{V}((\mathsf{pk}, pbb, r), \pi)$

Figure 4.9: Algorithms Defining ElectionGuardSimple

4.3 Reduction from Generic ElectionGuard to ElectionGuard Simple

An adversary's success in attacking Generic ElectionGuard with an independent number of guardians (for any n) is equivalent to finding an attack on ElectionGuardSimple. To demonstrate that the reduction from Generic ElectionGuard to ElectionGuardSimple preserves cryptographic properties for any number n of guardians ($1 < t < n$), we will show that any successful attack against Generic ElectionGuard with n independent guardians implies a successful attack on ElectionGuardSimple with a single guardian.

In Generic ElectionGuard, each guardian G_i generates a key pair $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ where $\mathsf{sk}_i \in \mathbb{Z}_q$ is the private key and $\mathsf{pk}_i = g^{\mathsf{sk}_i} \bmod p$ is the public key. The combined public key for the election is then $\mathsf{pk} = \prod_{i=1}^{n} \mathsf{pk}_i \bmod p$. For ElectionGuardSimple (i.e., $n = 1$), the public key is simply: $\mathsf{pk}' = \mathsf{pk}_1 = g^{\mathsf{sk}_1} \bmod p$. Thus, for any n, the combined public key pk in Generic ElectionGuard is equivalent to the public key pk' in ElectionGuardSimple when the single guardian's secret key corresponds to the combined secret key. Therefore, an adversary who can attack the public key pk in Generic ElectionGuard for any n can equivalently attack pk' in ElectionGuardSimple.

In Generic ElectionGuard, encryption is performed with the combined public key pk and generates ciphertext $\mathbb{c} = (\alpha, \beta)$ for a vote σ as $\alpha = g^\xi \bmod p, \quad \beta = \mathsf{pk}^\xi \cdot h(\sigma) \bmod p$, where $\xi \in \mathbb{Z}_q$ is a random nonce and $h(\sigma)$ encodes the vote. In ElectionGuardSimple, where $\mathsf{pk}' = g^{\mathsf{sk}_1}$, the encryption reduces to $\alpha' = g^\xi \bmod p, \quad \beta' = \mathsf{pk}'^\xi \cdot h(\sigma) \bmod$

p. Since pk in Generic ElectionGuard with n guardians functions identically to pk' in ElectionGuardSimple, the ciphertexts $\mathbb{c}$ in both systems are indistinguishable, and any adversary's advantage in distinguishing or attacking ciphertexts in Generic ElectionGuard for any n implies an equivalent advantage in ElectionGuardSimple.

In Generic ElectionGuard, decryption relies on threshold decryption with each guardian computing a partial decryption share $M_i = \beta^{\mathsf{sk}_i} \bmod p$. The final decryption is the aggregate of partial decryptions $M = \prod_{i=1}^{t} M_i^{\lambda_i} \bmod p$, where λ_i are the Lagrange coefficients. In ElectionGuardSimple (i.e., $n = 1$), decryption is performed as $M = \beta^{\mathsf{sk}_1} \bmod p$. For any n, if an adversary can compromise the threshold decryption in Generic ElectionGuard, they would equivalently compromise the single decryption operation in ElectionGuardSimple, demonstrating a reduction.

In both protocols, zero-knowledge proofs ensure the validity of encryption and decryption without revealing sensitive information. In Generic ElectionGuard, each encryption includes a proof π_1 using ZKP_2 confirming that the ciphertext encodes a valid vote. This proof structure remains identical in ElectionGuardSimple since it depends only on the ciphertext and not the number of guardians. Similarly, decryption correctness is assured in Generic ElectionGuard with the joint proof $\Pi = \mathsf{ZKP}_{\mathsf{joint}}(\mathsf{pk}, \mathbb{c}, M)$. In ElectionGuardSimple, this reduces to $\Pi' = \mathsf{ZKP}_{\mathsf{joint}}(\mathsf{pk}', \mathbb{c}', M')$. Since $\mathsf{pk} = \mathsf{pk}'$ and $M = M'$ for any n, the zero-knowledge proof for decryption is identical in Generic ElectionGuard and ElectionGuardSimple. Thus, any attack on the zero-knowledge proofs in Generic ElectionGuard for any n translates directly to an attack on ElectionGuardSimple.

Given the structural similarity between Generic ElectionGuard and ElectionGuardSimple, the following properties are preserved under reduction for any n: Encryption and decryption produce correct results in both schemes as they rely on the same ElGamal encryption (Correctness). The zero-knowledge proofs validate all encryptions and decryptions correctly in both systems (Completeness). Since the encryption and decryption proofs are identical, the soundness of each protocol holds for any n (Soundness). The zero-knowledge properties of the proofs are maintained, ensuring that no sensitive information is revealed (Zero-Knowledge). In Generic ElectionGuard, threshold decryption is effectively reduced to standard ElGamal decryption in ElectionGuardSimple, preserving privacy (Threshold Privacy).

Thus, it is tangible that any adversarial advantage against Generic ElectionGuard for any n implies an equivalent advantage against ElectionGuardSimple, preserving all cryptographic properties.

As a short note, in the ElectionGuard 1.1 [51], every guardian generates the tally proof separately which looks very similar to Helios and EG'. So, the tally process in ElectionGuard 1.1 is equivalent to the tally in EG'. All in all, with all the tiny differences that ElectionGuardSimple has with Helios (e.g. values $\mathsf{H_P}$, $\mathsf{H_B}$, $\mathsf{H_E}$, etc.), no changes are being made on similar security properties that they have.

4.4 Privacy for ElectionGuardSimple

Considering EG' operates the same as Helios, every proof for Helios also holds for EG'. However, in the following, we provide straighter proofs than similar ones.

Proof of Ballot Privacy

Theorem 10. *The voting scheme ElectionGuardSimple* EG' *is ballot-private according to BPRIV.*

Proof Sketch. To show EG' satisfies ballot privacy using a game-based approach, there is a need to show that the adversary's advantage in distinguishing between successive games is negligible, where each game represents a slight modification of the previous ones.

$\mathcal{G}_0$: This game corresponds to $\mathrm{EXP}_{\mathrm{BPRIV},0}(\mathcal{A})$ where the adversary $\mathcal{A}$ interacts with EG' system using the zero-knowledge proofs in the tallying process.

$\mathcal{G}_1$: The game is based on $\mathcal{G}_0$ but modifies the tallying process. Instead of using the original proof Π, it uses a simulated proof Π^* from $\mathsf{Sim}(\mathsf{BB}_0, r)$. So, game $\mathcal{G}_1$ will have $\Pi' \leftarrow \mathsf{Sim}(\mathsf{pk}, \mathsf{Publish}(\mathsf{BB}_1), r)$, and the output is **return** (r, Π').

In $\mathcal{G}_0$, adversary $\mathcal{A}$ observes real proofs Π produced by the ZKP. In $\mathcal{G}_1$, $\mathcal{A}$ sees simulated proofs Π^* generated by the simulator Sim. We construct an adversary $\mathcal{B}(\mathcal{A})$ against the zero-knowledge property of ZKP: $\mathcal{B}(\mathcal{A})$ simulates the experiments of $\mathcal{G}_0$ and $\mathcal{G}_1$ for $\mathcal{A}$ and presents $\mathcal{A}$ with either a real or a simulated proof depending on a hidden bit b. If $\mathcal{A}$ can distinguish the proofs with non-negligible advantage, $\mathcal{B}(\mathcal{A})$ uses $\mathcal{A}$'s output to guess b correctly with non-negligible advantage.

However, this contradicts the zero-knowledge property of ZKP, which asserts that simulated proofs are indistinguishable from real proofs. Hence, the advantage of $\mathcal{A}$ in distinguishing between $\mathcal{G}_0$ and $\mathcal{G}_1$ is bounded by the advantage of constructing an adversary $\mathcal{B}(\mathcal{A})$ that can distinguish between a real and a simulated zero-knowledge proof, and Thus, $|Pr[\mathcal{G}_0(\mathcal{A}) = 1] - Pr[\mathcal{G}_1(\mathcal{A}) = 1]|$ must be negligible, bounded by $\mathrm{Adv}^{\mathsf{ZKP}}_{\mathcal{B}(\mathcal{A})}$.

$$|\Pr[\mathcal{G}_0 = 1] - \Pr[\mathcal{G}_1 = 1]| \leq \mathrm{Adv}^{\mathsf{ZKP}}_{\mathcal{B}(\mathcal{A})} + \mathrm{negl}_1(\lambda)$$

$\mathcal{G}_2$: Modified version of $\mathcal{G}_1$ to use simulated ballots instead of real ballots $c_\beta = \mathsf{SimEnc}(\mathsf{pk}, v_\beta)$. Only $\mathcal{O}vote$ is getting changed.

Considering the semantic security of the encryption scheme, $\mathsf{IND-CCA}$, the simulated ballots c_β are indistinguishable from the actual ballots. For any adversary $\mathcal{A}_{\mathsf{IND\text{–}CCA}}$ attacking the $\mathsf{IND-CCA}$ security of the encryption scheme, there exists a negligible function $\mathrm{negl}_2(\lambda)$ such that:

$$|\Pr[\mathcal{G}_1 = 1] - \Pr[\mathcal{G}_2 = 1]| \leq \mathrm{Adv}_{\mathcal{A}_{\mathsf{IND\text{–}CCA}}} + \mathrm{negl}_2(\lambda)$$

$\mathcal{G}_3$: In this game, simulated ballots are getting used in the $\mathcal{O}cast$ oracle in such a way that $c' = \mathsf{SimVote}(id, b, \mathsf{pk})$, and if $\mathsf{Valid}(BB, b')$, then $BB \leftarrow BB + \{b'\}$. All oracles are same as $\mathcal{G}_2$ except $\mathcal{O}cast$.

By the $\mathsf{IND-CCA}$ property of the encryption scheme, the simulated ballots in the $\mathcal{O}cast$ are indistinguishable from real ballots, and there exists a negligible function $\text{negl}_3(\lambda)$ such that:

$$|\Pr[\mathcal{G}_2 = 1] - \Pr[\mathcal{G}_3 = 1]| \leq \text{Adv}_{\mathcal{A}_{\mathsf{IND-CCA}}} + \text{negl}_3(\lambda)$$

$\mathcal{G}_4$: Here, the $\mathcal{O}publish$ oracle of $\mathcal{G}_3$ gets modified to return $\mathsf{Publish}(BB_1)$, and $\mathcal{O}tally$ is same as $\mathcal{G}_1$ so that simulation does not affect the tallying process, which only relies on the validity and aggregation of already cast votes.

Since BB_1 is constructed in the same manner as BB_0, the published bulletin boards are indistinguishable, and there exists a negligible function $\text{negl}_4(\lambda)$ such that:

$$|\Pr[\mathcal{G}_3 = 1] - \Pr[\mathcal{G}_4 = 1]| \leq \text{negl}_4(\lambda)$$

Combining the game arguments, we have:

$$\left|\Pr[\text{Exp}^{\mathsf{bpriv},0}_{\mathcal{A},\mathsf{EG}'}(\lambda) = 1] - \Pr[\text{Exp}^{\mathsf{bpriv},1}_{\mathcal{A},\mathsf{EG}'}(\lambda) = 1]\right| \leq \sum_{i=1}^{4} \text{negl}_i(\lambda)$$

Since the sum of negligible functions is also negligible:

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

Thus, the EG' voting system satisfies ballot privacy under the BPRIV definition. □

Proof of Participation Privacy

Theorem 11. *The voting scheme ElectionGuardSimple* EG' *is participant-private according to PPRIV.*

Proof Sketch. Let G_0 be the real interaction where the voter *id* participates and submits an encrypted vote σ using the encryption algorithm $\mathsf{Enc}(\mathsf{pk}, \sigma)$, and the proofs π_1, π_2 are generated as part of the Vote process. The bulletin board BB_0 contains this real vote, the proofs, and the hash of the ballot. Let $B = (id, \mathbb{c}, \pi_1, \pi_2)$, $\mathbb{c} = \mathsf{Enc}(\mathsf{pk}, \sigma)$, where $\pi_1 = \mathsf{ZKP_2}.\mathsf{P_2}(\xi, \mathbb{c})$ is a zero-knowledge proof that the encrypted vote $\mathbb{c}$ is a valid vote, and $\pi_2 = \mathsf{ZKP_5}.\mathsf{P_5}(\xi, \ell)$ proves the vote lies within a valid range. The adversary $\mathcal{A}$ receives the public parameters, the bulletin board $\mathsf{BB_0}$, and the tally. The adversary's goal is to distinguish whether voter *id* has participated or abstained.

In G_1, we replace the real encrypted vote $\mathsf{Enc}(\mathsf{pk}, \sigma)$ with an encryption of a null vote $\sigma' = 0$ (i.e., a dummy vote). Let the new encryption be $\mathbb{c}' = \mathsf{Enc}(\mathsf{pk}, 0)$, where the ciphertext $\mathbb{c}'$ is indistinguishable from $\mathbb{c}$ due to the IND-CPA security of the encryption scheme $\mathcal{E}$. The zero-knowledge proofs π_1 and π_2 are simulated for $\mathbb{c}'$, but their indistinguishability follows from the Honest-Verifier Zero-Knowledge (HVZK) property of the proofs. By the IND-CPA security of the encryption scheme $\mathcal{E}$, we have that

$$|\Pr[\mathcal{A}(G_0) = 1] - \Pr[\mathcal{A}(G_1) = 1]| \leq \epsilon_{CPA},$$

where ϵ_{CPA} is negligible.

In G_2, we replace the real zero-knowledge proofs π_1 and π_2 with simulated proofs π_1^* and π_2^*. Due to the HVZK property of $\mathsf{ZKP_2}$ and $\mathsf{ZKP_5}$, the simulated proofs are computationally indistinguishable from the real proofs. Therefore, the adversary cannot distinguish between the real and simulated proofs. Thus, we have

$$|\Pr[\mathcal{A}(G_1) = 1] - \Pr[\mathcal{A}(G_2) = 1]| \leq \epsilon_{ZK},$$

where ϵ_{ZK} is negligible due to the zero-knowledge property of the proofs.

In G_3, we replace the hash of the ballot $\mathsf{H}(B)$ with a simulated hash $\mathsf{H}^*(B)$, where the hash is computed as $\mathsf{H}(B) = \mathcal{H}(id, \mathbb{c}, \pi_1, \pi_2)$. Due to the collision-resistance of the hash function $\mathcal{H}$, the adversary cannot distinguish between the real and simulated hash values. Formally, we rely on the assumption that the hash function $\mathcal{H}$ is collision-resistant, which ensures that: $|\Pr[\mathcal{A}(G_2) = 1] - \Pr[\mathcal{A}(G_3) = 1]| \leq \epsilon_{CR}$, where ϵ_{CR} is negligible.

Finally, in G_4, we simulate the case where the voter id abstains. In this case, no real vote or dummy vote is cast on behalf of id. The bulletin board contains only the votes of other participants and dummy ballots. By the indistinguishability of the encryption scheme (IND-CPA), the HVZK property of the zero-knowledge proofs, and the collision resistance of the hash function, the adversary $\mathcal{A}$ cannot distinguish between Game 3 and Game 4, i.e., between the case where a dummy vote is cast and the case where the voter abstains. Thus, we have $|\Pr[\mathcal{A}(G_3) = 1] - \Pr[\mathcal{A}(G_4) = 1]| \leq \epsilon_{DDH}$, where ϵ_{DDH} is negligible due to the Decisional Diffie-Hellman assumption in the group $\mathcal{C}$.

By combining the results of the transitions between each pair of games, we conclude that the adversary's advantage in distinguishing between participation and abstention is bounded by:

$$\epsilon = \epsilon_{CPA} + \epsilon_{ZK} + \epsilon_{CR} + \epsilon_{DDH},$$

which is negligible in the security parameter λ. Therefore, the EG' voting scheme satisfies δ-participation privacy with δ being negligible in λ. □

Strong Correctness

To investigate that ElectionGuardSimple is strongly correct, we show that any adversary $\mathcal{B}$ has a negligible advantage in producing a ballot box BB with a ballot b that passes the validity checks but is incorrect.

We proceed as follows: ElectionGuardSimple initializes by generating a public-private key pair $(\mathsf{pk}, \mathsf{sk})$ via $\mathsf{Setup}(1^\lambda)$. Each user id in the set I is registered, resulting in public-private credential pairs $(\mathsf{usk}, \mathsf{upk})$ associated with each user id. The adversary $\mathcal{B}$ receives $(\mathsf{pk}, \mathsf{uL})$, where uL is a list of registered users, and produces a tuple (id, v, BB), where id is a registered user's identifier, v is the vote, and BB is a ballot box containing potentially modified ballots. The adversary must ensure that every ballot $(x, y, \star, \star) \in \mathsf{BB}$ either matches the identifier id and public key upk of the registered user or is entirely unrelated to (id, upk) (Condition e_1). The ballot b generated by $\mathsf{Vote}(id, v, \mathsf{pk}, \mathsf{usk})$ for id with the

public key pk and private credential usk must pass the validation $\mathsf{Valid}(\mathsf{BB}, b, \mathsf{pk})$ within the context of the ballot box BB (Condition e_2).

Since ElectionGuardSimple uses ElGamal encryption and includes zero-knowledge proofs that ensure validity and correct structure for every encrypted vote, any ballot b that satisfies $\mathsf{Valid}(\mathsf{BB}, b, \mathsf{pk})$ and is produced by $\mathsf{Vote}(id, v, \mathsf{pk}, \mathsf{usk})$ will represent the correct vote v.

As a result, the probability that the adversary can produce a ballot box BB that includes an incorrect but seemingly valid ballot b is negligible, given that:

$$\mathcal{A}^{\mathsf{corr}}_{\mathcal{B},\mathcal{V},I}(\lambda) = \Pr\left[\mathsf{Exp}^{\mathsf{corr}}_{\mathcal{B},\mathcal{V},I}(\lambda) = 1\right]$$

is negligible in λ. Thus, ElectionGuardSimple satisfies strong correctness as the adversary's probability of success is negligible.

Strong Consistency

To check that ElectionGuardSimple is strongly consistent, we need to show that for any valid ballot box produced by an adversary, the result provided by the Tally algorithm matches the correct result.

> Extraction algorithm $\mathsf{Extract}(b, \mathsf{sk})$: Given a ballot $b = (\alpha, \beta)$, the extraction algorithm decrypts b using the secret key sk to obtain the vote v. Since ElectionGuardSimple uses ElGamal encryption, decrypting with sk correctly recovers v by $v = h^{-1}\left(\frac{\beta}{\alpha^{\mathsf{sk}} \bmod p}\right)$. This ensures that $\mathsf{Extract}(b, \mathsf{sk})$ returns (upk, v) with overwhelming probability.
>
> Ballot Validation Algorithm $\mathsf{ValidInd}(b, \mathsf{pk})$: This algorithm verifies that b is well-formed, checking that: - The elements (α, β) are within $\mathbb{Z}_p$. - b includes a zero-knowledge proof showing $v \in \{0, 1\}$. Thus, if $\mathsf{Valid}(\mathsf{BB}, b, \mathsf{pk})$ returns true for a bulletin board BB, then each ballot b individually satisfies $\mathsf{ValidInd}(b, \mathsf{pk})$.
>
> Strong Consistency Experiment: For any adversarially generated valid bulletin board BB, the tally result r from $\mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$ will match r', the result derived by individually decrypting each ballot and applying $\mathsf{Count} \circ \mathsf{Policy}$ on the decrypted results.

Since the ElGamal encryption and decryption are correct, $\mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$ is guaranteed to yield the correct result whenever each ballot is valid. Therefore, the probability of $r \neq r'$ given that each ballot satisfies $\mathsf{ValidInd}$ is negligible, proving that ElectionGuardSimple is strongly consistent.

4.5 ElectionGuard Versions

It is worth noting that this research is done based on the ElectionGuard Official Specification version 2.0; after we finished the formalization, the paper for the newer version was published, which preserved the security properties (as well as homomorphic encryption) of the previous version. The ElectionGuard paper and its recently published documentation

exhibit several key differences in terminology, equations, and structural processes, as outlined in the table 4.10.

Table 4.10: ElectionGuard Paper vs Documentation

Paper [52]	**Documentation** [19]
$\xi_{i,j} = H_q\left(\mathrm{H}_I; \texttt{0x21}, i, j, \xi_B\right)$ Second Paragraph, Page 9	$\xi_{i,j} = H\left(\mathrm{H}_E; \texttt{0x20}, \xi_B, \mathrm{ind}_c\left(\Lambda_i\right), \mathrm{ind}_o\left(\lambda_j\right)\right)$ Equation 25, Page 26
$\mathrm{H}_B = H(\mathrm{H}_P; \texttt{00x01}, \texttt{manifest})$ First Paragraph, Page 9	$\mathrm{H}_B = H\left(\mathrm{H}_P; \texttt{0x02}, \mathrm{H}_M, n, k\right)$ Equation 7, Page 19
$\mathrm{H}_I = H\left(\mathrm{H}_E; \texttt{0x20}, \mathrm{id}_B\right)$ Third Paragraph, Page 9	Undefined in Documentation
$\hat{K}$ in Key Generation First Paragraph, Page 10	Undefined in Documentation
$\mathrm{H}_E = H\left(\mathrm{H}_B; \texttt{0x14}, K, \hat{K}\right)$ Second Paragraph, Page 9	$\mathrm{H}_E = H\left(\mathrm{H}_B; \texttt{0x12}, k\right)$ Equation 23, Page 25
Key Generation Process: 1) random polynomial, 2) publishing commitment, 3) choosing a secret key, 4) proof of commitment	Key Generation Process: 1) choosing a secret key, 2) publishing commitment, 3) proof of commitment, 4) random polynomial
In the Key Generation section, they did not write the module parts at all. e.g., every time, instead of writing "$x \mod p$", they wrote just "x"! In the last paragraph of page 8, they say this is just for Hashes, but then they waive modules from everything.	Modules considered properly and precisely
$g^{P_i(j)} = \prod_{\ell=0}^{k-1} \left(K_{i,\ell}\right)^{j^\ell}$ Last Line, Page 9	$g^{P_i(\ell)} = \prod_{j=0}^{k-1} \left(K_{i,j}\right)^{\ell^j}$ Equation 21, Page 24
$\chi_l = H\left(\mathrm{H}_I; \texttt{0x28}, \mathrm{ind}_c\left(\Lambda_l\right), \alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_m, \beta_m\right)$ Last Line, Page 13	$\chi_l = H\left(\mathrm{H}_E; \texttt{0x23}, \mathrm{ind}_c\left(\Lambda_l\right), K, \alpha_1, \beta_1, \alpha_2, \beta_2 \ldots, \alpha_m, \beta_m\right)$ Equation 57, Page 35
Ballot Confirmation Code: $\mathrm{H}_C = H\left(\mathrm{H}_I; \texttt{0x29}, \chi_1, \chi_2, \ldots, \chi_{m_B}, \mathrm{B}_C\right)$ First Paragraph, Page 14	Ballot Confirmation Code: $\mathrm{H}(B) = H\left(\mathrm{H}_E; \texttt{0x24}, \chi_1, \chi_2, \ldots, \chi_{m_B}, \mathrm{B}_{\mathrm{aux}}\right)$ Equation 58, Page 35
Challenge of Decryption's Proof: $d_i = H\left(\mathrm{H}_E; \texttt{0x30}, i, A, B, a_i, b_i, M, U\right)$ Last Line, Page 15	Challenge of Decryption's Proof: $c = H\left(\mathrm{H}_E; \texttt{0x30}, K, A, B, a, b, M\right)$ Equation 71, Page 39

Chapter 5

Future Work

5.1 Sub-tasks

Here, we bring up an open problem that we might address in the future works as a sub-task.
Open Problem 1. Are there any general approaches to taking public-key cryptosystems (PKE) and making them into threshold public-key cryptosystems (TPKE)?

Generating a single public key from multiple public keys seems to be obtained only by distributed-key generation (DKG) - and often by verifiable secret sharing schemes (VSS). The decryption of ciphertexts in TPKE involves a "clever" use of all distributed secret keys - so, one will never reconstruct the master secret key. This intuitively means that some homomorphic property needs to happen when one combines all (decryptions done by distributed keys) [53]. Thus, is there a general approach that shows any (homomorphic) PKE + any DKG (or VSS) always produces a TPKE? What does IND-CPA for threshold look like? To refine this statement, we also need to include security properties:

- Initially, can we, for any IND-CPA (homomorphic) PKE + any Secure DKG (or VSS), always produce an IND-CPA TPKE in a somewhat efficient way?
- Secondly, can we, for any IND-CPA (homomorphic) PKE + cryptographic component (e.g. Threshold Decryption), always produce an IND-CPA TPKE efficiently?

The challenge below also remains unresolved and will be one of the secondary focus of our subsequent research.
Open Problem 2. Is it possible to prove that $\mathsf{ElGamal}'+$ ***Disjunctive Chaum-Pedersen*** *is secure under* $\mathsf{IND-CCA2}$ *or* $\mathsf{IND-1-CCA}$*?*

5.2 Dispute Resolution

It is assumed that solving disputes within an e-voting system-especially one employing cryptographic protocols-forms part of integrity, transparency, and reliability in the electoral process. Disputes in cryptographically secured voting systems might concern legitimacy with respect to votes, or identity with respect to voters, or system malfunction; mechanisms

dealing with such conflicts must, therefore, be set up. Cryptographic techniques, including zero-knowledge proofs, digital signatures, and secure multi-party computation, guarantee the confidentiality of the voters with verifiable elections. Even robust systems may suffer from a multitude of potential errors and malicious attacks. Dispute resolution mechanisms would resolve issues arising, for instance, when vote tampering or malfunctioning within the system is suspected, so that definite procedures can be followed to re-establish confidence in the integrity of the electoral process [54].

Besides, dispute resolution mechanisms ensure that the e-voting systems have transparency and fairness. If the cryptography evidence is utilized in dispute solving, then the evidence has to be verifiable by an independent third party not at the expense of voter secrecy. This becomes important upon the disputed outcome of elections, for then the auditing and re-counting can be done quite transparently. A well-designed and properly executed dispute resolution mechanism provides both voters and election officials with a forum in which to be heard regarding any issues raised, and it allows the ascertained election result to be checked, thereby helping avoid suspicion or election fraud. Public trust in the overall structure of dispute resolution will also build public trust in the electoral process as a whole, which is so crucial for the acceptance of e-voting [55].

Confidence is instilled in voters when they are certain that any problem would be resolved through robust and impartial mechanisms. Integrating human elements and procedural mechanisms for dispute resolution have to be considered incomplete where cryptography plays a foundational role in ensuring security aspects of a voting system. Efficient mechanisms of dispute resolution cannot but support the security and accountability of the voting system in the interest of much-desired enduring sustainability and legitimacy of electronic voting as a channel of democratic engagement [13].

Definition 6 (Dispute Resolution [54])**.** A voting system is considered to have dispute resolution if, in the event of a disagreement between two participants about whether they acted honestly, a third party is able to settle the dispute accurately.

Definition 7 (Dispute-freeness [55])**.** A dispute-free voting system incorporates prevention mechanisms that effectively eliminate conflicts among active participants. Additionally, any third party can verify if any participant has acted dishonestly or attempted to cheat.

We plan to address the following four problems in dispute resolution in electronic voting with the cryptographic approach.

The challenge for methods of dispute resolution is thereby multidimensional, given the range of possible errors-which may be software bugs, hardware faults, human mistakes, or cryptographic attacks. A system that could handle constructively any imaginable error would contribute significantly to the confidence and integrity so essential for e-voting. Such a system does, however, have to balance technical limitations with security and openness as well as strive for usability and efficiency.

Open Problem 3. Is it possible for a given system to have effective dispute resolution for all types of potential errors?

The bottom line is that the investigation of various definitions and approaches is important, as the choice of approach may prove unsuited or rigid in view of new emerging threats and vulnerabilities. Various contexts-electoral contexts, legal frameworks, or technological environments-could require tailored solutions. Plausible options might enlarge the dispute resolution space, either by applying new cryptographic schemes or by introducing new consensus algorithms or hybrid approaches that mix legal and technical elements. Basically, it means that searching and using such alternatives will enhance elasticity and resilience toward effective and fair dispute resolution in an e-voting system.

Open Problem 4. Are there alternative reasonable definitions and mechanisms for resolving disputes?

A mechanism that would provide sufficient evidence to address all the disputes that may arise would, therefore, be highly desirable to ensure the integrity of the electoral process. If the system allows cryptographic validation, verifiable records, or audit trails to sort out disputes, it would go a long way in developing confidence and clarity in the electoral process. The evidence for such comprehensiveness has to be tamper-resistant but also understandable to the participants: the voter, the auditor, and the electoral authorities. An important question indeed, since one has to balance between the provision of adequate evidence and the provision not impinging on security, privacy, or even system efficiency.

Open Problem 5. Can a system provide comprehensive dispute resolution where every conflict can be resolved using evidence generated by the election system?

Central issues of completely digital election systems concern either dispute resolution or dispute freeness without paper ballots. The general observation is that reliance on physical artifacts adds a degree of logistical complexity and potential vulnerabilities. Cryptographic approaches allow possibilities w.r.t. dispute resolution based on digital evidence; this is possible only depending on the strength of cryptography, secure systems design, and legal acceptance of digital proofs. Where possible, fully digital dispute resolution may enable a smoother voting experience and more access. Otherwise, without the appearance of security that paper records provide, such systems continue to be hard to adopt for widespread use and acceptance.

Open Problem 6. Is it feasible to achieve dispute resolution or dispute-freeness without relying on paper ballots and physical procedures?

5.3 Post-quantum End-to-End Verifiable Online Voting Platform

All currently widely applied cryptographic algorithms underlying present-day e-voting systems are essentially based on the hardness of some well-known mathematical problems, which include integer factorization [56] and discrete logarithms [57]. These are computationally infeasible to solve using classical computers, and therefore, they provide a strong basis for secure encryption and digital signatures.

This is threatened by the advent of quantum computing, as enhanced computational power poses a great threat to the security of these cryptographic techniques. Quantum

computers can resolve some mathematical problems exponentially faster than classical computers if the principles of quantum mechanics are used in their formulation. In this context, Shor's algorithm can efficiently factor large integers and compute discrete logarithms in that it makes many of the cryptographic techniques that are currently in use vulnerable against quantum attacks [58].

Researchers have been working on post-quantum cryptographic algorithms that would be secure not only against classical but also against quantum attacks. Such algorithms are based on various mathematical problems hypothesized to be resistant to quantum computers, like lattice-based, hash-based, and code-based cryptography [59]. As post-quantum cryptography moves from theoretical studies to practical applications, in particular, it becomes one of the most critical areas of study in securing e-voting systems in the post-quantum world.

The place of post-quantum cryptography in electronic voting opens new opportunities and challenges for building trust in the voting process. On the one hand, post-quantum cryptographic algorithms could provide a way to ensure the long-term security of votes against both classical and quantum attacks, thus ensuring confidentiality and integrity well into the future [60]. This is particularly significant for the integrity of election results over time, as archived votes might be subject to retrospective attacks once quantum computers are powerful enough.

On the other hand, post-quantum cryptography embeds additional complexity into the e-voting system, which may further complicate existing challenges to transparency, verifiability, and building public trust. Post-quantum algorithms are usually much more complex, less understood, and hence harder to analyze with respect to their security properties by all stakeholders involved; voters should be able to trust the system. Moreover, changes driven by post-quantum cryptography in existing e-voting infrastructures can be huge, thus putting a question mark on compatibility, usability, and the likelihood of the appearance of new vulnerabilities [61].

The junction of post-quantum cryptography and trust in electronic voting is systematically investigated in the dissertation, hence striving to contribute to developing secure, trustworthy, and future-proof e-voting systems able to resist upcoming challenges.

Quantum computers have not been built yet! Why do we care about their threat to e-voting?

The possibility of a quantum computer hitting the streets within the next coming years has heightened concerns from security experts regarding the security of encrypted data in different platforms [62], especially electronic voting systems, since quantum computers harness the power of quantum mechanics to perform their computations. As previously mentioned, it is supposed that this enhanced processing power will allow quantum computers to crack intricate mathematical problems, like factoring large prime numbers that form the basis of most current cryptographic systems [63]. The real question is not whether quantum computers will be available in the future, but their consequences for encrypted

data created today.

While these encrypted votes might be secure against decryption in current technology, they would become easily decipherable once quantum computers are operational. It is a huge risk to voter privacy and fair electoral processes. An entity storing encrypted voting data today could, at a future date—by which time quantum computers are very likely to be available—decrypt the information and infer how everybody voted. This could have far-reaching impacts in terms of reduced trust by members of the public in the conduct of elections and the democratic process more generally.

Future decryption also opens up a host of ethical and legal problems: for example, the legal frameworks that are currently in place preserve the secrecy of ballots, but may not be able to withstand the retroactive privacy violations that are enabled by quantum decryption. Consecutively, this issue further raises questions regarding the obligation of governments and organizations to adequately protect data against future technological breakthroughs.

This then poses a growing call to the cybersecurity community to develop and subsequently adopt quantum-resistant cryptography algorithms that can handle quantum computers, so that encrypted data remains secure in post-quantum times. Transitioning to quantum-resistant cryptography, however, is a very complex process, apart from being resource-intensive, with profound changes to be effected in existing infrastructure and protocols.

Why do we need a new voting system? Can not we just replace the classical cryptographic building blocks of existing systems with post-quantum ones?

Because:

- Helios and ElectionGuard have not explicitly proved they have Everlasting Privacy.
- None of the Helios and ElectionGuard provide strong Coercion Resistance proof. A variant of Helios (Helios++) uses multi-account features to counter coercion, allowing voters to use fake accounts if coerced, but it has not been proven.
- None of the Helios and ElectionGuard have been proven to be Strongly software-independent.
- Helios is not end-to-end verifiable.
- None of the Helios and ElectionGuard provide solutions for dispute resolution. ElectionGuard addresses minor complaints during the initialization of the election but does not provide any formal and robust answers.
- None of the Helios and ElectionGuard are post-quantum.
- Despite of all new components, ElectionGuard can basically be considered Cramer et al. [64] with some changes.
- Even though ElectionGuard is a software development kit, it is not compatible with all of the existing modes of casting votes.

- The tally decryption depends on a central core known as the Administrator in all these three works. If this individual is dishonest, then the election will fail and be a waste of time. Moreover, none of these three works explicitly described the characteristics of this Administrator and who it should be strictly.
- In the ElectionGuard, one of the issues is not that corrupted guardians will not decrypt ballots they are supposed to decrypt, but rather they may want to decrypt individual votes that they should not decrypt. This then enables a quorum less than n to accommodate emergency situations on behalf of one or few guardians, while still keeping it high enough to protect from curious guardians.
- In the voting systems that use the ElectionGuard kit, if the voters can put more than one vote on a candidate, the size of the discrete logarithm will not be a small number, and it will be a slightly larger value. ElectionGuard also proposed precomputed tables for computing discrete logarithms, but they have their own issues e.g. susceptibility to attacks and key exposure, space complexity, lack of generality, and efficiency trade-offs.
- ElectionGuard depends on the guardians, who are expected to protect the secrecy of votes and does not go any further to specify verification times by the guardians. The number of guardians can do little if in the election process, there is a compromise of a ballot marking device, which leaks encryption nonces, or a flawed pseudo-random number generator is used in the process.
- ElectionGuard simply assumes that the SHA-256 hash function is collision-resistant, meaning it cannot generate two different votes or ballots that correspond to the same hash value. Apart from this simple trust, voters and observers are free to outsource their trust to one or more verifiers of their choice, or they can create their own verifying tools that enable them to independently verify the integrity of the result in an election which causes serious trust issues in the whole system.
- In ElectionGuard voters obtain confirmation codes associated with their personally encrypted ballots, which sets up the opportunity for coercion. Any entity that may decrypt ballots could, therefore, coerce voters to provide their confirmation codes. Moreover, in the event of voters publishing or making their confirmation codes public in a non-private manner, chances for privacy breaking through unauthorized decryption will arise.
- The whole secrecy of ballots in ElectionGuard is entangled into a single value known as ballot nonce. These values must be carefully protected because they enable the decryption of the ballot which is encrypted. Apart from the fact that it is a serious risk, no solution has been provided to preserve these values.
- ElectionGuard does not support ranked-choice voting.

So, we will work on a new online voting system fill the above gaps as much as possible.

Chapter 6

Discussion and Progress

This PhD's initial focus has been checking the cryptographic elements and assumptions, which are the Achilles' heel for voting systems. Then, the structure of two systems was analyzed: Helios [65] and Belenios [66]; these are online voting platforms widely studied by security researchers and the community because of their end-to-end cryptographic construction. Next, we used provable security to study the latest definitions of ballot secrecy and participation privacy. Afterward, ElectionGuard [17], a software development kit developed by Microsoft, was our case study for privacy, primarily since ElectionGuard has been used in multiple real-world elections in the United States [52]. However, ElectionGuard has not yet been formalized, so we started to formalize it, where the lack of description and vagueness of several concepts in its version two's documentation [19] were our challenges, and this intensified by publishing a paper [52] on June 2024, where various notions and even procedures were differing than the ones on documentation of version 2.0, and Microsoft published the documentation of version 2.1 [67] on August 2024, which was more aligned with the ElectionGuard paper [52] but their time inconsistency was apparent which caused us some challenges during this research. After formalizing ElectionGuard, the next step was to derive an abstract version that is more suitable for algebraic proofs and be an elegant resource for the later use of cryptographic investigations. Moreover, we proved that ElectionGuard is ballot private. Furthermore, we proved it is satisfying Participation Privacy.

Our next steps will be to study connecting verifiability with ballot privacy, link verifiability with accountability and redress accountability by more stringent definitions, advance with the open problems of dispute-freeness and dispute resolution, and provide an online post-quantum end-to-end verifiable voting platform.

6.1 Gantt Chart

The Gantt chart in Figure 6.1 outlines a comprehensive timeline for this PhD project spanning four years (2023-2027). In the first year (2023-2024), the focus was on reviewing cryptographic primitives, analyzing real-world voting systems, and investigating ballot privacy experiments, which are essential foundations for understanding election security.

A study on Microsoft ElectionGuard, a widely used verifiable election technology, is conducted toward the latter part of the year. The second year (2024-2025) starts with a confirmation viva, followed by an in-depth literature review on verifiability, linking it with privacy and accountability. These steps pave the way for establishing a secure and accountable voting system.

Year three (2025-2026) focuses on more advanced topics such as privacy-preserving dispute resolution and lattice-based cryptography, which are key in designing secure post-quantum e-voting schemes. The final year (2026-2027) is dedicated to implementing an online e-voting system and completing the PhD thesis. The timeline includes substantial time for thesis writing, viva preparation, and report submission after the viva, ensuring that the research outcomes are thoroughly documented and defended. So, this project emphasizes a methodical approach to developing a secure, privacy-preserving e-voting system, from foundational research to implementation and reporting.

6.2 Conclusion

This report identified the building of trust and privacy in the mechanisms of electronic voting as paramount. ElectionGuard is a feasible solution where cryptography permits better election security with more transparency. Analyses do, however, find weaknesses in its proof guarantees, especially concerning the confidentiality of participation and possible vulnerabilities under improvements in quantum computing. The formalization of ElectionGuard presented here provides a starting point for security researchers to apply additional cryptographic analysis with higher confidence in its privacy and verifiability properties. Meanwhile, taking into account post-quantum cryptography also brings up several potentials and challenges of its own within the process of future-proofing e-voting systems.

However, further research is required in cryptographic techniques that guarantee privacy and dispute resolution mechanisms in order for the public to trust electronic voting. Quantum computers are a threat to currently used cryptographic systems; hence, the application of post-quantum algorithms would be needed in any e-voting system. Whichever of those such schemes balances the much-needed security against usability and transparency would see such systems get widespread usage and confidence among the electors. Their successful application would be fundamental in the guaranteeing of the future and validity of the mechanisms of electronic voting within the democratic setup.

All in all, from the point of research objective, given the critical importance of trust in the electoral process and the looming threat posed by quantum computing, this project aims to clarify trust in electronic voting by investigating ballot and participation privacy in real-world systems, extending verifiability by highlighting the link with ballot privacy (In e-voting, privacy can not exist in the absence of individual verifiability [68].), improving accountability by proposing more robust definition(s), addressing privacy-preserving dispute resolution, and presenting a practical remote voting system for the post-quantum era.

Figure 6.1: Gantt Chart of PhD Progress

Bibliography

[1] *Securing the Vote: Protecting American Democracy*. National Academies Press, **august** 2018, ISBN: 9780309476478. DOI: 10.17226/25120. **url**: http://dx.doi.org/10.17226/25120.

[2] P. Wolf, R. Nackerdien **and** D. Tuccinardi, *Introducing Electronic Voting: Essential Considerations* (Policy Paper). International Institute for Democracy **and** Electoral Assistance (International IDEA), 2011, ISBN: 9789186565428. **url**: https://books.google.co.uk/books?id=hPjVDwAAQBAJ.

[3] C. Lambrinoudakis, D. Gritzalis, V. Tsoumas, M. Karyda **and** S. Ikonomopoulos, "Secure Electronic Voting: the Current Landscape," **in***Secure Electronic Voting* D. A. Gritzalis, **editor**. Boston, MA: Springer US, 2003, **pages** 101–122, ISBN: 978-1-4615-0239-5. DOI: 10.1007/978-1-4615-0239-5_7. **url**: https://doi.org/10.1007/978-1-4615-0239-5_7.

[4] *EXPLAINER: Voting systems reliable, despite conspiracies — apnews.com*, https://apnews.com/article/2022-midterm-elections-technology-voting-donald-trump-campaigns-46c9cf208687636b8eaa1864c35a [Accessed 17-08-2024].

[5] https://www.facebook.com/bbcnews, *US election 2020: Is Trump right about Dominion machines? — bbc.co.uk*, https://www.bbc.co.uk/news/election-us-2020-54959962, [Accessed 17-08-2024].

[6] https://www.theguardian.com/profile/charlesarthur, *Estonian e-voting shouldn't be used in European elections, say security experts — theguardian.com*, https://www.theguardian.com/technology/2014/may/12/estonian-e-voting-security-warning-european-elections-research, [Accessed 17-08-2024].

[7] F. Hao **and** P. Ryan, *Real-world electronic voting : design, analysis and deployment*. Boca Raton, FL: Taylor & Francis Group, LLC CRC Press is an imprint of Taylor & Francis Group, an Informa Business, 2017, ISBN: 9781498714716.

[8] https://www.theguardian.com/profile/peter-stone, *Cyber attacks and electronic voting errors threaten 2020 outcome, experts warn — theguardian.com*, https://www.theguardian.com/us-news/2020/jan/02/elections-2020-cyber-attacks-democrats-experts, [Accessed 17-08-2024].

[9] https://www.facebook.com/bbcnews, *Russia election: Putin's party wins election marred by fraud claims — bbc.co.uk*, https://www.bbc.co.uk/news/world-europe-58614227, [Accessed 17-08-2024].

[10] F. Cantú **and** S. M. Saiegh, *Was Argentina's election stolen? Here's how you can tell.* **url**: https://www.washingtonpost.com/news/monkey-cage/wp/2015/11/06/was-argentinas-election-stolen-heres-how-you-can-tell/.

[11] h.-c. https://www.theguardian.com/profile/michael-safi https://www.theguardian.com/profile/gabrielle-chan, *NSW election result could be challenged over iVote security flaw — theguardian.com*, https://www.theguardian.com/australia-news/2015/mar/23/nsw-election-result-could-be-challenged-over-ivote-security-flaw, [Accessed 17-08-2024].

[12] Reuters, "Venezuela stands by election count despite fraud allegation," *Reuters*, 2017, Accessed: 2024-08-17. **url**: https://www.reuters.com/article/world/venezuela-stands-by-election-count-despite-fraud-allegation-idUSKBN1AI1WR/.

[13] J. Benaloh, M. Bernhard, J. A. Halderman **andothers**, "Public Evidence from Secret Ballots," *CoRR*, **jourvol** abs/1707.08619, 2017. arXiv: 1707.08619. **url**: http://arxiv.org/abs/1707.08619.

[14] *ElectionGuard - Official — electionguard.vote*, https://www.electionguard.vote/spec/, [Accessed 20-07-2024].

[15] J. Benaloh, M. Naehrig, O. Pereira **and** D. S. Wallach, *ElectionGuard: a Cryptographic Toolkit to Enable Verifiable Elections*, Cryptology ePrint Archive, Paper 2024/955, https://eprint.iacr.org/2024/955, 2024. **url**: https://eprint.iacr.org/2024/955.

[16] *Helios Voting — vote.heliosvoting.org*, https://vote.heliosvoting.org/, [Accessed 28-09-2024].

[17] *ElectionGuard - What is ElectionGuard? — electionguard.vote*, https://www.electionguard.vote/, [Accessed 18-08-2024].

[18] M. Bellare, R. Canetti **and** H. Krawczyk, "Keying Hash Functions for Message Authentication," **in***Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology* **jourser** CRYPTO '96, Berlin, Heidelberg: Springer-Verlag, 1996, **pages** 1–15, ISBN: 3540615121.

[19] *github.com*, https://github.com/microsoft/electionguard/releases/download/v2.0/EG_Spec_2_0.pdf, [Accessed 18-08-2024].

[20] D. Bernhard, M. Fischlin **and** B. Warinschi, *On the Hardness of Proving CCA-security of Signed ElGamal*, Cryptology ePrint Archive, Paper 2015/649, 2015. **url**: https://eprint.iacr.org/2015/649.

[21] S. Schäge, "New Limits of Provable Security and Applications to ElGamal Encryption," **in***Advances in Cryptology – EUROCRYPT 2024: 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26–30, 2024, Proceedings, Part IV* Zurich, Switzerland: Springer-Verlag, 2024, **pages** 255–285, ISBN: 978-3-031-58736-8. DOI: 10.1007/978-3-031-58737-5_10. **url**: https://doi.org/10.1007/978-3-031-58737-5_10.

[22] H. Devillez, O. Pereira **and** T. Peters, "How to Verifiably Encrypt Many Bits for an Election?" **in***Computer Security – ESORICS 2022* V. Atluri, R. D. Pietro, C. D. Jensen **and** W. Meng, **editors**, **jourser** Lecture Notes in Computer Science, **volume** 13555, Springer, Cham, 2022, **pages** 653–671. DOI: 10.1007/978-3-031-17146-8_32.

[23] M. Bellare **and** C. Namprempre, *Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm*, Cryptology ePrint Archive, Paper 2000/025, 2000. **url**: https://eprint.iacr.org/2000/025.

[24] T. Rabin **and** M. Ben-Or, "Verifiable secret sharing and multiparty protocols with honest majority," **in***Proceedings of the twenty-first annual ACM symposium on Theory of computing - STOC '89* **jourser** STOC '89, ACM Press, 1989. DOI: 10.1145/73007.73014. **url**: http://dx.doi.org/10.1145/73007.73014.

[25] S. Das, Z. Xiang, A. Tomescu, A. Spiegelman, B. Pinkas **and** L. Ren, *Verifiable Secret Sharing Simplified*, Cryptology ePrint Archive, Paper 2023/1196, 2023. **url**: https://eprint.iacr.org/2023/1196.

[26] V. Shoup **and** N. P. Smart, "Lightweight Asynchronous Verifiable Secret Sharing with Optimal Resilience," *Journal of Cryptology*, **jourvol** 37, **number** 3, **page** 27, 2024. DOI: 10.1007/s00145-024-09505-6. **url**: https://doi.org/10.1007/s00145-024-09505-6.

[27] R. Gennaro, M. O. Rabin **and** T. Rabin, "Simplified VSS and fast-track multiparty computations with applications to threshold cryptography," **in***Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing* **jourser** PODC '98, Puerto Vallarta, Mexico: Association for Computing Machinery, 1998, **pages** 101–111, ISBN: 0897919777. DOI: 10.1145/277697.277716. **url**: https://doi.org/10.1145/277697.277716.

[28] R. Gennaro, S. Jarecki, H. Krawczyk **and** T. Rabin, "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems," *Journal of Cryptology*, **jourvol** 20, **pages** 51–83, 1999. **url**: https://api.semanticscholar.org/CorpusID:3331212.

[29] T. P. Pedersen, "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing," **in***Advances in Cryptology — CRYPTO '91* J. Feigenbaum, **editor**, Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, **pages** 129–140, ISBN: 978-3-540-46766-3.

[30] V. Cortier, D. Galindo, S. Glondu **and** M. Izabachène, "Distributed ElGamal à la Pedersen: Application to Helios," **in***Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society* **jourser** WPES '13, Berlin, Germany: Association for Computing Machinery, 2013, **pages** 131–142, ISBN: 9781450324854. DOI: 10.1145/2517840.2517852. **url**: https://doi.org/10.1145/2517840.2517852.

[31] A. De Santis, Y. Desmedt, Y. Frankel **and** M. Yung, "How to share a function securely," **in***Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing* **jourser** STOC '94, Montreal, Quebec, Canada: Association for Computing Machinery, 1994, **pages** 522–533, ISBN: 0897916638. DOI: 10.1145/195058.195405. **url**: https://doi.org/10.1145/195058.195405.

[32] S. Goldwasser, S. Micali **and** C. Rackoff, "The knowledge complexity of interactive proof-systems," **in***Symposium on the Theory of Computing* 1985. **url**: https://api.semanticscholar.org/CorpusID:209402113.

[33] M. Bellare **and** O. Goldreich, "On Defining Proofs of Knowledge," **in***Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology* **jourser** CRYPTO '92, Berlin, Heidelberg: Springer-Verlag, 1992, **pages** 390–420, ISBN: 3540573402.

[34] F. Hao, *Schnorr Non-interactive Zero-Knowledge Proof*, RFC 8235, **september** 2017. DOI: 10.17487/RFC8235. **url**: https://www.rfc-editor.org/info/rfc8235.

[35] K. Baghery, *CO6GC: Introduction to Zero-Knowledge Proofs (Part 1)*, https://www.esat.kuleuven.be/cosic/blog/co6gc-introduction-to-zero-knowledge-proofs-1/, Accessed: 2024-10-09, 2020.

[36] D. Venturi **and** A. Villani, *Zero-Knowledge Proofs and Applications*, https://danieleventuri.altervista.org/files/zero-knowledge.pdf, Accessed: 2024-10-09, 2015.

[37] D. Chaum **and** T. P. Pedersen, "Wallet Databases with Observers," **in***Advances in Cryptology — CRYPTO' 92* E. F. Brickell, **editor**, Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, **pages** 89–105, ISBN: 978-3-540-48071-6.

[38] K. L. Dakshita Khurana **and** V. Jagtap, *Zero-Knowledge II*, https://courses.grainger.illinois.edu/cs498ac3/fa2020/Files/Lecture_15_Scribe.pdf, Lecture notes from CS 498AC3, University of Illinois at Urbana-Champaign, 2020.

[39] R. Cramer, I. Damgård **and** B. Schoenmakers, "Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols," **in***Advances in Cryptology — CRYPTO '94* Y. G. Desmedt, **editor**, Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, **pages** 174–187, ISBN: 978-3-540-48658-9.

[40] A. Fiat **and** A. Shamir, "How to prove yourself: practical solutions to identification and signature problems," **in***Proceedings on Advances in Cryptology—CRYPTO '86* Santa Barbara, California, USA: Springer-Verlag, 1987, **pages** 186–194, ISBN: 0387180478.

[41] D. Bernhard, O. Pereira **and** B. Warinschi, *How not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios*, Cryptology ePrint Archive, Paper 2016/771, 2016. **url**: https://eprint.iacr.org/2016/771.

[42] D. Bernhard, V. Cortier, D. Galindo, O. Pereira **and** B. Warinschi, *A comprehensive analysis of game-based ballot privacy definitions*, Cryptology ePrint Archive, Paper 2015/255, 2015. **url**: https://eprint.iacr.org/2015/255.

[43] D. Bernhard, V. Cortier, D. Galindo, O. Pereira **and** B. Warinschi, "SoK: A Comprehensive Analysis of Game-Based Ballot Privacy Definitions," **in***2015 IEEE Symposium on Security and Privacy* 2015, **pages** 499–516. DOI: 10.1109/SP.2015.37.

[44] V. Cortier, J. Lallemand **and** B. Warinschi, *Fifty Shades of Ballot Privacy: Privacy against a Malicious Board*, Cryptology ePrint Archive, Paper 2020/127, 2020. **url**: https://eprint.iacr.org/2020/127.

[45] D. Bernhard, O. Kulyk **and** M. Volkamer, "Security Proofs for Participation Privacy, Receipt-Freeness and Ballot Privacy for the Helios Voting Scheme," **in***Proceedings of the 12th International Conference on Availability, Reliability and Security* **jourser** ARES '17, Reggio Calabria, Italy: Association for Computing Machinery, 2017, ISBN: 9781450352574. DOI: 10.1145/3098954.3098990. **url**: https://doi.org/10.1145/3098954.3098990.

[46] V. Cortier, D. Galindo, R. Küsters, J. Müller **and** T. Truderung, "SoK: Verifiability Notions for E-Voting Protocols," **in***36th IEEE Symposium on Security and Privacy (S&P'16)* San Jose, USA, **may** 2016, **pages** 779–798.

[47] V. Cortier, D. Galindo, S. Glondu **and** M. Izabachène, "Election Verifiability for Helios under Weaker Trust Assumptions," **in***Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II* M. Kutylowski **and** J. Vaidya, **editors**, **jourser** Lecture Notes in Computer Science, **volume** 8713, Springer, 2014, **pages** 327–344. DOI: 10.1007/978-3-319-11212-1_19. **url**: http://dx.doi.org/10.1007/978-3-319-11212-1_19.

[48] B. Adida, "Helios: web-based open-audit voting," **in***Proceedings of the 17th Conference on Security Symposium* **jourser** SS'08, San Jose, CA: USENIX Association, 2008, **pages** 335–348.

[49] *github.com*, https://github.com/microsoft/electionguard/releases/download/v2.1/EG_Spec_2_1.pdf, [Accessed 03-10-2024].

[50] W. D. Smith, "Correction: The case for score voting," *Constitutional Political Economy*, **jourvol** 35, **number** 3, **pages** 447–448, **december** 2023, ISSN: 1572-9966. DOI: 10.1007/s10602-023-09425-w. **url**: http://dx.doi.org/10.1007/s10602-023-09425-w.

[51] *github.com*, https://github.com/microsoft/electionguard/releases/download/v1.1/EG_spec_v1_1.pdf, [Accessed 27-10-2024].

[52] J. Benaloh, M. Naehrig, O. Pereira **and** D. S. Wallach, *ElectionGuard: a Cryptographic Toolkit to Enable Verifiable Elections*, Cryptology ePrint Archive, Paper 2024/955, https://eprint.iacr.org/2024/955, 2024. **url**: https://eprint.iacr.org/2024/955.

[53] A. Kate, Y. Huang **and** I. Goldberg, *Distributed Key Generation in the Wild*, Cryptology ePrint Archive, Paper 2012/377, 2012. **url**: https://eprint.iacr.org/2012/377.

[54] T. Kaczmarek, J. Wittrock, R. Carback **andothers**, "Dispute Resolution in Accessible Voting Systems: The Design and Use of Audiotegrity," **in***E-Voting and Identify* J. Heather, S. Schneider **and** V. Teague, **editors**, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, **pages** 127–141, ISBN: 978-3-642-39185-9.

[55] A. Kiayias **and** M. Yung, "Self-tallying Elections and Perfect Ballot Secrecy," **in***Public Key Cryptography* D. Naccache **and** P. Paillier, **editors**, Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, **pages** 141–158, ISBN: 978-3-540-45664-3.

[56] R. L. Rivest, A. Shamir **and** L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, **jourvol** 21, **number** 2, **pages** 120–126, **february** 1978, ISSN: 0001-0782. DOI: 10.1145/359340.359342. **url**: https://doi.org/10.1145/359340.359342.

[57] W. Diffie **and** M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, **jourvol** 22, **number** 6, **pages** 644–654, 1976. DOI: 10.1109/TIT.1976.1055638.

[58] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM Journal on Computing*, **jourvol** 26, **number** 5, **pages** 1484–1509, **october** 1997, ISSN: 1095-7111. DOI: 10.1137/s0097539795293172. **url**: http://dx.doi.org/10.1137/S0097539795293172.

[59] D. J. Bernstein, "Introduction to post-quantum cryptography," **in***Post-Quantum Cryptography* D. J. Bernstein, J. Buchmann **and** E. Dahmen, **editors**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, **pages** 1–14, ISBN: 978-3-540-88702-7. DOI: 10.1007/978-3-540-88702-7_1. **url**: https://doi.org/10.1007/978-3-540-88702-7_1.

[60] G. Kaim, S. Canard, A. Roux-Langlois **and** J. Traoré, "Post-quantum Online Voting Scheme," **in***FC 2021 - Financial Cryptography and Data Security. International Workshops* **volume** Lecture Notes in Computer Science, Virtual event, France, **march** 2021, **pages** 290–305. DOI: 10.1007/978-3-662-63958-0_25. **url**: https://hal.science/hal-03355875.

[61] X. Boyen, T. Haines **and** J. Müller, "Epoque: Practical End-to-End Verifiable Post-Quantum-Secure E-Voting," **in***2021 IEEE European Symposium on Security and Privacy (EuroS&P)* 2021, **pages** 272–291. DOI: 10.1109/EuroSP51992.2021.00027.

[62] *Fast tracking quantum-computing tech — nature.com*, https://www.nature.com/articles/d42473-023-00091-y, [Accessed 17-08-2024].

[63] P. Sharma, V. Gupta **and** S. K. Sood, "Evolution of Quantum Cryptography in Response to the Computational Power of Quantum Computers: An Archival View," *Archives of Computational Methods in Engineering*, 2024. DOI: 10.1007/s11831-024-10122-6. **url**: https://doi.org/10.1007/s11831-024-10122-6.

[64] R. Cramer, R. Gennaro **and** B. Schoenmakers, "A Secure and Optimally Efficient Multi-Authority Election Scheme," **in***Advances in Cryptology — EUROCRYPT '97* W. Fumy, **editor**, Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, **pages** 103–118, ISBN: 978-3-540-69053-5.

[65] *Helios Voting — vote.heliosvoting.org*, https://vote.heliosvoting.org/, [Accessed 18-08-2024].

[66] *Belenios: Verifiable online voting system — belenios.org*, https://www.belenios.org/, [Accessed 18-08-2024].

[67] *github.com*, https://github.com/microsoft/electionguard/releases/download/v2.1/EG_Spec_2_1.pdf, [Accessed 18-08-2024].

[68] V. Cortier **and** J. Lallemand, "Voting: You Can't Have Privacy without Individual Verifiability," **jourser** CCS '18, Toronto, Canada: Association for Computing Machinery, 2018, **pages** 53–66, ISBN: 9781450356930. DOI: 10.1145/3243734.3243762. **url**: https://doi.org/10.1145/3243734.3243762.

Appendix

Fiat-Shamir Weakness

Let $\Sigma = (\mathsf{Prove}_\Sigma, \mathsf{Verify}_\Sigma)$ be a sigma protocol, $\mathcal{H}$ a hash function, and Y be the statement. The weak Fiat-Shamir transformation of Σ is the proof system $\mathrm{wFS}_\mathcal{H}(\Sigma) = (\mathsf{Prove}, \mathsf{Verify})$ defined as follows:

$\mathsf{Prove}(w, Y)$: Run $\mathsf{Prove}_\Sigma(w, Y)$ to obtain commitment A. Compute $c \leftarrow \mathcal{H}(A)$. Complete the run of Prover_Σ with c as input to get the response f. Output the pair (c, f).

$\mathsf{Verify}(Y, c, f)$: Compute A from (Y, c, f), then run $\mathsf{Verify}_\Sigma(Y, A, c, f)$.

Bernhard et al., 2012 [41] proposed the strong Fiat-Shamir transformation of Σ, i.e., sFS $(\Sigma) = (\mathsf{Prove}, \mathsf{Verify})$, which is obtained as above with the difference that c is computed by $c \leftarrow \mathcal{H}(Y, A)$, to avoid disclosing statement Y.

Although, we found some computational errors in this work[41]; as an example, on page 7 it has been mentioned that $\log_R(T) = x + \frac{ar-b}{c}$ which looks incorrect. Based on the following assumptions, which are abstracted from the above paper

$$x = \log_G(X), \quad (a, b) \leftarrow \mathbb{Z}_q^2, \quad A = G^a \quad B = G^b, \quad c = \mathcal{H}(A, B), \quad f = a + cx, \quad T^c = \frac{R^f}{B}$$

$$T = \left(\frac{R^f}{B}\right)^{\frac{1}{c}} = \log_R(T) = x + \frac{ar - b}{c},$$

the red colored equation should be $\log_R(T) = x + \frac{a}{c} - \frac{b}{rc}$, according to the below procedure.

$$\begin{aligned}
\log_R(T) &= \log_R\left(\left(\frac{R^f}{B}\right)^{\frac{1}{c}}\right) \\
&= \frac{1}{c}\log_R\left(\frac{R^f}{B}\right) \\
&= \frac{1}{c}(\log_R(R^f) - \log_R(B)) \\
&= \frac{1}{c}(f - \log_R(B)) \\
&= \frac{1}{c}(f - \log_{G^r}(G^b)) \\
&= \frac{1}{c}\left(f - \frac{\log_G(G^b)}{\log_G(G^r)}\right) \\
&= \frac{1}{c}\left(a + cx - \frac{b}{r}\right) \\
&= x + \frac{a}{c} - \frac{b}{rc}
\end{aligned}$$

Still, the logical approach and general idea of Strong Fiat-Shamir is correct and working.

Secure Shares Transmission

Here, we bring a detailed original version of the Encrypt-then-MAC processes that ElectionGuard uses. The encryption algorithm $\mathsf{Enc}_5(\mathsf{pp}, P_i(\ell), \mathsf{pk}_\ell)$ performs the following steps: It samples a random nonce $\xi_{i,\ell}$ from $\mathbb{Z}_q$. Using the nonce $\xi_{i,\ell}$, the generator g, the public key pk, and the message M, the ciphertext components $(\alpha_{i,\ell}, \beta_{i,\ell})$ are computed: $(\alpha_{i,\ell}, \beta_{i,\ell}) = \left(g^{\xi_{i,\ell}} \bmod p, \mathsf{pk}_\ell^{\xi_{i,\ell}} \bmod p\right)$. Also, the secret key $\mathsf{sk}_{i,\ell}$ is generated by hashing the parameter base hash $\mathsf{H_P}$, a constant delimiter $\texttt{0x11}$, sender guardian i, receiver guardian ℓ and its public key pk_ℓ, and the encryption components $\alpha_{i,\ell}$ and $\beta_{i,\ell}$. A MAC key sk_0 gets computed by $\text{HMAC}(\mathsf{sk}_{i,\ell}, \texttt{0x01}||\texttt{Label}||\texttt{0x00}||\texttt{Context}||\texttt{0x0200})$ and an encryption key sk_1 gets obtained in the same way but with different delimiter $\texttt{0x02}$. The output ciphertext is in three parts of $C_{i,\ell,0}{=}g^{\xi_{i,\ell}} \bmod p$, $C_{i,\ell,1}{=}\text{b}\left(P_i(\ell), 32\right) \oplus \mathsf{sk}_1$, $C_{i,\ell,2}{=}\text{HMAC}\left(\mathsf{sk}_0, \text{b}\left(C_{i,\ell,0}, 512\right) \| C_{i,\ell,1}\right)$, and the algorithm returns these ciphertext components. The decryption and verification algorithm $\mathsf{Dec}_5(\mathsf{pp}, s_\ell, E_\ell(P_i(\ell)))$ performs the following steps: encryption components $\beta_{i,\ell}$ and $\alpha_{i,\ell}$ get computed using guardian ℓ's secret share by $\beta_{i,\ell}{=}\left(C_{i,\ell,0}\right)^{s_\ell} \bmod p$, and $\alpha_{i,\ell} \leftarrow C_{i,\ell,0}$. By having access to $\beta_{i,\ell}$ and $\alpha_{i,\ell}$, secret key $\mathsf{sk}_{i,\ell}$, MAC key sk_0, and encryption key sk_1 can get obtained similar to the procedure of $\mathsf{Enc}_5(\mathsf{pp}, P_i(\ell), \mathsf{pk}_\ell)$. Dec_5 recomputes $C_{i,\ell,2}$ and if it does not equal to the value of $\text{HMAC}\left(\mathsf{sk}_0, \text{b}\left(C_{i,\ell,0}, 512\right) \| C_{i,\ell,1}\right)$, the sub-procedure returns an error $\perp$. Otherwise, it returns the decrypted message $P_i(\ell)$ and its verification result based on equality of two elements $\prod_{j=0}^{k-1}\left(\mathsf{pk}_{i,j}\right)^{\ell^j} \bmod p$ and $g^{P_i(\ell)} \bmod p$, which confirms validity of guardian i's commitments $\mathsf{pk}_{i,0}, \mathsf{pk}_{i,1}, \ldots, \mathsf{pk}_{i,k-1}$ to its coefficients.

$\mathsf{Enc}_5(\mathsf{pp}, P_i(\ell), \mathsf{pk}_\ell)$

1 : $\xi_{i,\ell} \leftarrow\!\$\ \mathbb{Z}_q; (\alpha_{i,\ell}, \beta_{i,\ell}) = \left(g^{\xi_{i,\ell}} \bmod p, \mathsf{pk}_\ell^{\xi_{i,\ell}} \bmod p\right)$

2 : $\mathsf{sk}_{i,\ell} \leftarrow H\left(\text{H}_P; \texttt{0x11}, i, \ell, \mathsf{pk}_\ell, \alpha_{i,\ell}, \beta_{i,\ell}\right)$

3 : $\mathsf{sk}_0 \leftarrow \text{HMAC}(\mathsf{sk}_{i,\ell}, \texttt{0x01}||\texttt{Label}||\texttt{0x00}||\texttt{Context}||\texttt{0x0200})$

4 : $\mathsf{sk}_1 \leftarrow \text{HMAC}(\mathsf{sk}_{i,\ell}, \texttt{0x02}||\texttt{Label}||\texttt{0x00}||\texttt{Context}||\texttt{0x0200})$

5 : $C_{i,\ell,0} \leftarrow g^{\xi_{i,\ell}} \bmod p$

6 : $C_{i,\ell,1} \leftarrow \text{b}\left(P_i(\ell), 32\right) \oplus \mathsf{sk}_1$

7 : $C_{i,\ell,2} \leftarrow \text{HMAC}\left(\mathsf{sk}_0, \text{b}\left(C_{i,\ell,0}, 512\right) \| C_{i,\ell,1}\right)$

8 : **return** $\left(C_{i,\ell,0}, C_{i,\ell,1}, C_{i,\ell,2}\right)$

$\mathsf{Dec}_5(\mathsf{pp}, s_\ell, E_\ell(P_i(\ell)))$

1 : $\beta_{i,\ell} \leftarrow \left(C_{i,\ell,0}\right)^{s_\ell} \bmod p; \alpha_{i,\ell} \leftarrow C_{i,\ell,0}$

2 : $\mathsf{sk}_{i,\ell} \leftarrow H\left(\text{H}_P; \texttt{0x11}, i, \ell, \mathsf{pk}_\ell, \alpha_{i,\ell}, \beta_{i,\ell}\right)$

3 : $\mathsf{sk}_0 \leftarrow \text{HMAC}(\mathsf{sk}_{i,\ell}, \texttt{0x01}||\texttt{Label}||\texttt{0x00}||\texttt{Context}||\texttt{0x0200})$

4 : $\mathsf{sk}_1 \leftarrow \text{HMAC}(\mathsf{sk}_{i,\ell}, \texttt{0x02}||\texttt{Label}||\texttt{0x00}||\texttt{Context}||\texttt{0x0200})$

5 : **if** $C_{i,\ell,2} \neq \text{HMAC}\left(\mathsf{sk}_0, \text{b}\left(C_{i,\ell,0}, 512\right) \| C_{i,\ell,1}\right)$ **then** $\perp$

6 : $\text{b}\left(P_i(\ell), 32\right) \leftarrow C_{i,\ell,1} \oplus \mathsf{sk}_1$

7 : **return** $\left(g^{P_i(\ell)} \bmod p = \prod_{j=0}^{k-1}\left(\mathsf{pk}_{i,j}\right)^{\ell^j} \bmod p, P_i(\ell)\right)$

Figure 6.2: Encrypt-then-MAC in ElectionGuard

Range Proof

This proof demonstrates that (α, β) is an encryption of an integer within the range $\{0, 1, \ldots, R\}$. The proof requires knowledge of the encryption nonce ξ, where (α, β) represents the encryption of the value ℓ.

The proof proceeds using a disjunctive Chaum-Pedersen range proof to assert that (α, β) encrypts an integer within the interval $[0, R]$. Specifically, for each integer j in the range $0 \leq j \leq R$, a random value $u_j \in \mathbb{Z}_q$ is chosen. The commitment for the specific value $j = \ell$ is generated using only u_ℓ as $(a_\ell, b_\ell) = (g^{u_\ell} \bmod p, K^{u_\ell} \bmod p)$.

For every $j \neq \ell$, random challenge values $c_j \in \mathbb{Z}_q$ are selected, and the commitments are calculated as $(a_j, b_j) = (g^{u_j} \bmod p, K^{t_j} \bmod p)$, where $t_j = (u_j + (\ell - j)c_j) \bmod q$. The commitments (a_j, b_j) for all j are then combined with the ciphertext and the election's extended base hash H_E to produce a pseudo-random challenge c as:

$$c = H\left(\mathrm{H}_E; 0 \times 21, K, \alpha, \beta, a_0, b_0, a_1, b_1, \ldots, a_R, b_R\right).$$

The challenge value for the correct index, c_ℓ, is computed by subtracting the sum of the other challenge values from the total challenge c:

$$c_\ell = \left(c - \sum_{j \neq \ell} c_j\right) \bmod q = (c - (c_0 + c_1 + \cdots + c_{\ell-1} + c_{\ell+1} + \cdots + c_R)) \bmod q$$

Finally, for each j in the range $0 \leq j \leq R$, a response value v_j is calculated as:

$$v_j = (u_j - c_j \xi) \bmod q$$

The proof consists of the challenge values $c_0, c_1, \ldots, c_R$ and the response values $v_0, v_1, \ldots, v_R$. This method enables the proof of an encrypted value being within a specific range without requiring access to the decryption key. The only information necessary is the encryption nonce ξ used for generating the ciphertext.

In the special case where $R = 1$, the proof demonstrates that the ciphertext encrypts either a 0 or a 1. When $\ell = 0$, it corresponds to the proof for the unselected option (encryption of 0), while $\ell = 1$ corresponds to the proof for the selected option (encryption of 1) [52].

Verifiability Against Dishonest Ballot Box

The security game corresponding to a dishonest ballot box and an honest registrar is depicted in Figure 6.3. After an honest registration phase, the adversary entirely controls the ballot box and has access to two oracles:

- $\mathsf{O}_{vote}(id, v)$ to require that some honest voter id votes for v. The fact that voter id voted is stored in Hvote.
- $\mathsf{O}_{corrupt}(id)$ to corrupt voter id and retrieve her secret credential. Previous votes from id are removed from Hvote.

Then the adversary publishes the final ballot box and voters may check whether their ballot has been included in the ballot box (through oracle $\mathsf{O}_{check}(id)$). If the test is successful, the corresponding voter (and vote) is stored in $\mathsf{Checked}$. The game $\mathsf{Exp}^{dbb}_{\mathcal{A},\mathcal{V},I}(\lambda)$ guarantees that the election result corresponds to:

- all the votes from the honest voters who checked (set C);
- some of the votes from the honest voters who did not check (the adversary may always remove such votes). This corresponds to the set $L \subseteq H$;
- at most k other (valid) votes, where k is the number of corrupted voters (set D). This last condition controls ballot stuffing: any adversary may be able to chose as many votes as the number of corrupted voters but not more. In particular, he cannot add arbitrary votes to the results.

Formally, a voting scheme $\mathcal{V}$ is *verifiable against dishonest ballot box*, if for any set I of voters and for any efficient adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ the probability

$$\Pr\left[\mathsf{Exp}^{dbb}_{\mathcal{A},\mathcal{V},I}() = 1\right]$$

to win the experiment depicted in Figure 6.3 is negligible, which in this experiment $\mathsf{Exp}^{dbb}_{\mathcal{A},\mathcal{V},I}$, the adversary $\mathcal{A}_1$ has access to the set of oracles $\mathcal{O} = \{Ocorrupt, Ovote\}$, and $\mathcal{A}_2$ has access to $Ocheck$.

Verifiability Against Dishonest Registrar

The security game corresponding to a honest ballot box and a dishonest registrar is depicted in Figure 6.4. It is very similar to the previous game except that the ballot box now behaves honestly and the adversary choses the credentials of the voters. Again, the result has to contain all the votes from honest voters who checked, a subset of the votes of voters who did not check and at most k arbitrary valid votes from the k dishonest voters. A voting scheme $\mathcal{V}$ is *verifiable against dishonest registration*, if for any efficient adversary $\mathcal{A}$ the probability

$$\Pr\left[\mathsf{Exp}^{dreg}_{\mathcal{A},\mathcal{V},I}() = 1\right]$$

to win the experiment depicted in Figure 6.4 is negligible, which in experiment $\mathsf{Exp}^{dreg}_{\mathcal{A},\mathcal{V}}$, the adversary $\mathcal{A}_2$ has access to the set of oracles $\mathcal{O} = \{Ocorrupt, Ovote, Ocast, Oboard\}$, and adversary $\mathcal{A}_3$ has access to $Ocheck$. $Ocheck$ and $Ocorrupt$ are those shown in Figure 6.3.

$\mathsf{Exp}^{dbb}_{\mathcal{A},\mathcal{V},I}(\lambda)$

1 : $\mathsf{BB}, \mathsf{Hvote}, \mathsf{Checked}, \mathsf{coL} \leftarrow [\,]$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
3 : // for all voters we assign credentials, and publish the public credentials
4 : $\forall id \in I$ **do**
5 : $(\mathsf{usk}, \mathsf{upk}) \leftarrow \mathsf{Register}(id)$
6 : $\mathsf{uL}[id] \leftarrow (\mathsf{usk}, \mathsf{upk})$
7 : $L \leftarrow L \cup \{\mathsf{upk}\}$
8 : // the adversary commits to a ballot box
9 : $bb, \mathsf{state} \leftarrow \mathcal{A}_1^{Ocorrupt,Ovote}(\mathsf{sk}, L)$
10 : // keep ballots that use exactly one of the assigned credentials
11 : $\mathsf{BB} \leftarrow \{(id, \mathsf{upk}, c, s) | (id, \mathsf{upk}, c, s) \in \mathsf{Policy}\ bb \wedge \mathsf{upk} \in L\}$
12 : $(r, \pi) \leftarrow \mathcal{A}_2^{Ocheck}(\mathsf{state})$
13 : $e_1 \leftarrow \mathsf{Verify}((\mathsf{pk}, \mathsf{Publish\ BB}, r), \pi)$
14 : // C contains votes from voters that checked, and H contains votes from honest voters that did not check
15 : $C \leftarrow \{v|\ (id, \mathsf{upk}, v, c, s) \in \mathsf{Checked}\}$
16 : $H \leftarrow \{v|\ (id, \mathsf{upk}, v, c, s) \in (\mathsf{Hvote} - \mathsf{Checked})\}$
17 : $e_2 \leftarrow \exists D.\ \exists L.\ L \subseteq H \wedge |D| \leq |\mathsf{coL}| \wedge r = \mathsf{Count}\Big(C \cup L \cup D\Big)$
18 : **return** $e_1 \wedge \neg e_2$

Oracle $Ocheck(id)$

1 : // Over the intended ballot, decided using the voting policy
2 : **for** $(id, \mathsf{upk}, v, c, s) \in \mathsf{Policy}(\mathsf{Hvote})$ **do**
3 : // test its membership w.r.t the ballot box.
4 : **if** $\mathsf{VerifyVote}((id, \mathsf{upk}, c, s), \mathsf{Publish\ BB}) \wedge (id, \star, \star, \star, \star) \notin \mathsf{Checked}$ **then**
5 : $\mathsf{Checked} \leftarrow \mathsf{Checked} + (id, \mathsf{upk}, v, c, s)$

Oracle $Ocorrupt(id)$

1 : $\mathsf{usk} \leftarrow \perp$
2 : // if it is a valid voter return the credential
3 : **if** $(id \in I \wedge id \notin \mathsf{coL})$ **then**
4 : $\mathsf{coL} \leftarrow \mathsf{coL} + [id]$
5 : // remove all honest ballots created by that voter
6 : $\mathsf{Hvote} \leftarrow \{(id', \mathsf{upk}, v, c, s) \in \mathsf{Hvote}|\ id' \neq id\}$
7 : $(\mathsf{usk}, \mathsf{upk}) \leftarrow \mathsf{uL}.[id]$
8 : **return** usk

Oracle $Ovote(id, v)$

1 : $(\mathsf{usk}, \mathsf{upk}) \leftarrow \mathsf{uL}.[id]$
2 : $b \leftarrow \perp$
3 : **if** $(id \in I \wedge id \notin \mathsf{coL})$ **then**
4 : // create a ballot and store it in Hvote
5 : $(id', \mathsf{upk}', c', s) \leftarrow \mathsf{Vote}(id, v, \mathsf{upk}, \mathsf{pk}, \mathsf{usk})$
6 : $\mathsf{Hvote} \leftarrow \mathsf{Hvote} + [(id, \mathsf{upk}', v, c', s)]$
7 : $b \leftarrow (id, \mathsf{upk}', c', s)$
8 : **return** b

Figure 6.3: The Verifiability Experiment Against a Dishonest Ballot Box

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

1 : $\mathsf{BB}, \mathsf{Hvote}, \mathsf{coL} \leftarrow [\,]$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
3 : // adversary gives the list of voters and their crdentials
4 : $(\mathsf{uL}, \mathsf{state}) \leftarrow \mathcal{A}_1(\mathsf{sk})$
5 : $I \leftarrow \{id | \mathsf{uL}.[id] \neq \perp\}$
6 : // the voting process is held- ballots are added to the ballot box
7 : $\mathsf{state} \leftarrow \mathcal{A}_2^{Ocorrupt,Ovote,Ocast,Oboard}(\mathsf{state})$
8 : // at the end of the election, a verification phase is run
9 : $(r, \Pi) \leftarrow \mathcal{A}_3^{Ocheck}(\mathsf{state})$
10 : $e_1 \leftarrow \mathsf{Verify}((\mathsf{pk}, \mathsf{Publish\ BB}, r), \Pi)$
11 : $C \leftarrow \{v |\ (id, \mathsf{upk}, v, c, s) \in \mathsf{Checked}\}$
12 : $H \leftarrow \{v |\ (id, \mathsf{upk}, v, c, s) \in (\mathsf{Hvote} - \mathsf{Checked})\}$
13 : $e_2 \leftarrow \exists D.\ \exists L.\ L \subseteq H \wedge |D| \leq |\mathsf{coL}| \wedge\ r = \mathsf{Count}\Big(C \cup L \cup D\Big)$
14 : **return** $e_1 \wedge e_2$

Oracle $Ovote(id, v)$

1 : $(\mathsf{upk}, \mathsf{usk}) \leftarrow \mathsf{uL}.[id]$
2 : $b \leftarrow \perp$
3 : **if** $(id \in I \wedge id \notin \mathsf{coL})$ **then**
4 : $(id', \mathsf{upk}', c', s') \leftarrow \mathsf{Vote}(id, v, \mathsf{pk}, \mathsf{usk})$
5 : $\mathsf{Hvote} \leftarrow \mathsf{Hvote} + [(id, \mathsf{upk}', v, c', s')]$
6 : $b \leftarrow (id, \mathsf{upk}', c', s')$
7 : **if** $\mathsf{Valid}(\mathsf{BB}, \mathsf{uL}, b, \mathsf{pk})$ **then**
8 : $\mathsf{BB} \leftarrow \mathsf{BB} + [b]$
9 : **return** b

Oracle $Ocast(id, \mathsf{upk}, c, s)$

1 : **if** $(id \in I \wedge id \in \mathsf{coL} \wedge \mathsf{Valid}(\mathsf{BB}, (id, \mathsf{upk}, c, s), \mathsf{pk}))$ **then**
2 : $\mathsf{BB} \leftarrow \mathsf{BB} + [(id, \mathsf{upk}, c, s)]$

Oracle $Oboard()$

1 : **return** Publish BB

Figure 6.4: The Verifiability Experiment Against a Dishonest Registration