Karlsruhe Institute of Technology

KASTEL

Towards Realising Post-Quantum Secure ElectionGuard

Master's Thesis of

Jonas K. Ludwig

at the Department of Informatics
KASTEL – Institute of Information Security and Dependability

Reviewer:	Prof. Jörn Müller-Quade
Second reviewer:	Prof. Thorsten Strufe
Advisor:	M.Sc. Felix Dörre
Second advisor:	M.Sc. Markus Raiber

15th June 2023 – 15th December 2023

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text and that I have complied with the KIT Statutes for Safeguarding Good Research Practice.

Karlsruhe, 15th December 2023

. . . . . .

(Jonas K. Ludwig)

Abstract

ElectionGuard is a toolkit used to enable end-to-end verification of elections. Since ElectionGuard uses exponential ElGamal, it is vulnerable to later vote decryption when faced with quantum adversaries. Therefore, to avoid influences on voter behaviour, a post-quantum adaption of ElectionGuard is needed to make the toolkits usage secure for the foreseeable future.

In this work, we present cryptographic components that can be used to construct a post-quantum version of ElectionGuard. We use the homomorphic encryption scheme BGV and the commitment scheme BDLOP in a lattice setting. Applying existing and new zero knowledge proofs, we construct the elements needed to show that votes are well-formed and later aggregated correctly. As a final step, we show how to verifiably decrypt the resulting tally.

In this paper, we lay the foundation for realising post-quantum secure ElectionGuard. We provide the components for post-quantum vote encryption, aggregation, tallying and verifiable decryption. Our work can be extended by distributed key generation and the components for end-to-end verification.

Acknowledgements

I could not have undertaken this journey without the support of my thesis supervisor Felix Dörre, who spent countless hours discussing ideas and proofs with me and therefore massively helped me write this thesis. I deeply appreciate the energy and effort you invested into this thesis and have learned a lot under your supervision.

Also, I would like to extend my sincere thanks to Prof. Jörn Müller-Quade for reviewing my thesis as well as helping me generate some ideas and improve my understanding when I was stuck. Additionally, I would like to mention Prof. Thorsten Strufe for being the second reviewer.

I am also grateful to Markus Raiber for helping out when Felix and I needed some more insights. Additionally, thanks should also go to Stefan Kühnlein for helping me get a grasp on the factorisation of polynomial rings; to Tjeran Silde from NTNU for helping me construct the adapted version of the proof of verifiable decryption and to Laurin Benz for helping me question and improve some of the basic concepts.

Moreover, I would like to thank my fellow students Alina, Anne, Charlotte, Florian, Lisa, Niels and Norman for proofreading my work, discussing concepts, helping me stay on track and just generally making my time as a student at KIT a lot of fun.

To close I would like to express my deepest gratitude to my family and especially my parents for supporting my studies as well as all my professional and personal endeavours and for helping me grow into the person I am today.

Contents

List of Figures

List of Tables

1. Introduction

In this work we present a post-quantum secure version of ElectionGuard, the "open source software development kit (SDK) that improves confidence and participation in elections"[1]. We first give an overview over how ElectionGuard works and then provide an explanation of the motivation behind making ElectionGuard post-quantum secure.

In the chapters of this work we initially provide a preliminaries section to set the mathematical framework. Then, we describe the different building blocks we use, i.e. the BGV encryption scheme, the BDLOP commitment scheme and various zero knowledge proofs. In the proceeding section we describe how to construct an election system from our building blocks and go into detail how to select parameters for exemplary instantiations and describe the compatibility with ElectionGuard as well as a security evaluation of our approach, concluding our work.

1.1. Overview over ElectionGuard

We first want to provide a short introduction to ElectionGuard and its underlying principles. ElectionGuard is mainly supported and developed by Microsoft and is in parts based on the PhD-thesis of Josh Benaloh [9]. The general principle is to encrypt votes as ciphertexts and to form the tally using homomorphic addition, which is a different approach than using mixnets like the Helios voting system does [1]. It is open-source and available on Github[2] and can be used to implement end-to-end verifiable voting systems or to add this functionality to other, already existing election systems. Note that we refer to specification v1.1 as available on the ElectionGuard website [10].

The basic principle of ElectionGuard is to use homomorphic encryption as a way to calculate the tally of elections. Specifically, ElectionGuard uses exponential ElGamal which is additively homomorphic. After an election has been set up including the key generation, voters can cast their votes by encrypting 0 (candidate not selected) or 1 (candidate selected) for each candidate. Then, non-interactive zero knowledge proofs are generated to show that the votes are in fact either a encryption of zero or one (and not any other value) and that the sum of all encrypted values is equal to the number of candidates the voter is allowed to cast votes for. In case a voter wants to cast less votes than he is allowed to, he casts the remaining votes into *placeholder* options instead. Once the vote is complete, the voter can submit it and receive a confirmation code which is used for end-to-end verifiability. Then the voter can choose to either finally cast their ballot or reject it to verify it and then start again.

[1] `https://www.electionguard.vote/`, Date accessed: 07.12.2023

[2] `https://github.com/microsoft/electionguard/`, Date accessed: 07.12.2023

Once the election time is up and the voters have cast their votes, the tally can be calculated as the additively homomorphic sum of all submitted votes for each of the candidates. Anyone can perform this operation since the votes are all published. Finally, the trustees decrypt the final tally and calculate a non-interactive zero knowledge proof of verifiable decryption ensuring that the decryption is correct.

These three steps 1) casting votes and ensuring their correctness, 2) tallying using the additive homomorphic property and 3) verifiable decryption are the centerpiece of ElectionGuard. Other features like end-to-end-verifiabilty or threshold encryption are essentially added "on-top". To keep the work in this thesis within bounds, we focus on the three basic steps and leave detailed consideration of other features for future work.

1.2. Motivation

Since ElectionGuard currently uses exponential ElGamal its security is based on the discrete logarithm assumption, i.e. that its hard to calculate discrete logarithms. However, quantum computers can solve discrete logarithms easily using Shor's Algorithm [21], voiding ElectionsGuard's underlying security assumption. The consequence is that attackers with sufficiently advanced quantum computers can break ElectionGuard and decrypt ballots without access to the election authorities secret keys. This would mean that ElectionGuard is entirely insecure.

The problem however, is even more severe than it appears at first. Since all cast votes are published to ensure end-to-end-verifiability, a public record of all voters choices - in an encrypted format - remains. Given the developments in quantum computers, it is plausible that Shor's algorithm can be run in the next decades and subsequently the decryption of the published votes is a plausible outcome. This yields a major problem even for elections run today: The fact that your vote can be become known might influence your election behaviour. This can reach from minor cases like "I don't want anyone to know that I voted for candidate XY" to more severe cases like coercion "If you don't vote for candidate XY I will harm you when I find out".

The implication of this problem is that we should switch to *post-quantum* cryptography for elections even before quantum computers are advanced enough. We need the secrecy guarantee to hold far longer than the election was running, because threats in the future impact coercion resistance right now. Hence, the aim of this work is to create to underlying framework for a new version of ElectionGuard that uses post-quantum encryption and can therefore guarantee long lasting security of the voters opinions.

There already exists some research on post-quantum e-voting schemes, however they cover other systems, e.g. Helios [17]. We are not aware of a post-quantum version of ElectionGuard so far.

2. Preliminaries

We first provide a definition of zero knowledge proofs. Then, we describe out mathematical setting in detail, i.e. our lattice, the normal distributions we use and some properties of polynomial rings. Afterwards we state the hardness assumptions we use for security.

2.1. Zero Knowledge Proofs

A zero knowledge proof allows a *prover* $\mathcal{P}$ to convince a *verifier* $\mathcal{V}$ that a specific statement is true without revealing any other information than the one given in the statement. The *statements* can be expressed as $u \in \mathcal{L}$, where $\mathcal{L}$ is a language in NP. A *witness* w is a witness iff $(u, w) \in R$ with R being a polynomial time decidable binary relation associated with $\mathcal{L}$. Zero knowledge proofs need to fulfil three properties:

- **Completeness:** If the prover $\mathcal{P}$ runs his protocol, then for $x \in \mathcal{L}$, the verifier $\mathcal{V}$ accepts the common input x with probability 1. In other words, the prover can convince the verifier of $x \in \mathcal{L}$ [15].
- **Soundness:** The prover $\mathcal{P}$ can run any program and for every constant $c > 0$ and large enough $x \notin \mathcal{L}$ the verifier $\mathcal{V}$ the verifier rejects x with probability at least $1 - |x|^{-c}$. In other words, the prover cannot fool the verifier [15].
- **Statistically Honest Verifier Zero Knowledge:** There exists an expected ppt simulator $\mathcal{S}$ such that for $\forall x \in \mathcal{L}$ the distributions $\mathcal{S}(x)$ (simulated transcripts) and $\text{View}_{\mathcal{V}}(\langle \mathcal{P}, \mathcal{V} \rangle(x))$ (real transcripts) are statistically indistinguishable [8].

A *proof of knowledge* is a zero knowledge proof where an extractor can extract the witness w given two transcripts with the different challenges. Since w can be extracted in this case, we can assume that the prover also knows w (or can at least calculate it), i.e. that the prover *knows* the witness w. If a proof fulfils the proof of knowledge property, it also fulfils soundness. We transform our zero knowledge proof into non-interactive proofs with the *Fiat-Shamir heuristic* adapted to the quantum random oracle model [13, 18].

2.2. The Setting

We use a similar setting as Baum, Damgård, Lyubashevsky, Oechsner and Peikert [7] with some changes and additions as described in the following.

Define the ring $R = \mathbb{Z}[x]/\langle x^N + 1 \rangle$ with *ring dimension* N. We use an index to denote a modulus, for example $R_q = \mathbb{Z}_q[x]/\langle x^N + 1 \rangle$ denotes the set of polynomials with degree smaller than N and all coefficients from $\mathbb{Z}_q$. We center the coefficients around 0, i.e. for

$g \in R_q$ with $g = \sum_i \overline{g_i} x^i$ we represent each $\overline{g_i}$ with an $g_i \in \left\{-\frac{q-1}{2}, \frac{q-1}{2}\right\}$ such that $\overline{g_i} = g_i \mod q$. Different rings with varying moduli will be used throughout this work with equivalent notation and behaviour.

For all $f \in R$ write $f = \sum_i f_i x^i$ to define the norms of f:

$$\ell_1 : ||f||_1 = \sum_i |f_i|$$
$$\ell_2 : ||f||_2 = \sqrt{\sum_i |f_i|^2}$$
$$\ell_\infty : ||f||_\infty = \max_i |f_i|$$

There are a few inequalities to note here. For $f \in R_q$ it holds that

$$||f||_1 \leq \sqrt{N}||f||_2 \leq N||f||_\infty \text{ and } ||f||_\infty \leq ||f||_2 \leq ||f||_1.$$

Additionally, we can give a bound on the norm of a product due to the choice of the polynomial $x^N + 1$. For $f, g \in R_q$: [7]

$$\text{If } ||f||_\infty \leq \beta, ||g||_1 \leq \gamma \text{ then } ||f \cdot g||_\infty \leq \beta \cdot \gamma$$
$$\text{If } ||f||_2 \leq \beta, ||g||_2 \leq \gamma \text{ then } ||f \cdot g||_\infty \leq \beta \cdot \gamma$$

For positive integers α define $S_{R_q,\alpha} = \{f \in R_q \mid ||f||_\infty = \alpha\}$ to be used as the set of "small" polynomials over R_q (and other rings respectively). Also, define a challenge space $C_{R_q,\kappa} = \{c \in R_q \mid ||c||_\infty = 1, ||c||_1 = \kappa\}$ and a set of differences $\overline{C}_{R_q,\kappa} = \{c - c' \mid c \neq c' \in C_{R_q,\kappa}\}$. It holds that $\forall f \in \overline{C}_{R_q,\kappa} : ||f||_\infty \leq 2, ||f||_1 \leq 2\kappa$. We use this in combination with Theorem 1 (see Section 2.2.2) and by setting the parameters of the rings in a way that all non-zero elements of ℓ_∞-norm at most 2 will be invertible in the corresponding ring, which especially makes all elements in $\overline{C}_{R_q,\kappa}$ invertible. When choosing κ we aim to have the size of C be 2^λ which can be achieved by setting κ with $\binom{N}{\kappa} \cdot 2^\kappa > 2^\lambda$ [7]. For example, if $\lambda = 256, N = 8192$ we can set $\kappa := 24$.

We write $\leftarrow$ for storing the result of a randomised algorithm in a variable, e.g. $x \leftarrow \mathcal{A}$ and $\xleftarrow{\$}$ to signify drawing a uniformly random sample from a set and storing it in a variable, e.g. $x \xleftarrow{\$} \{-1, 0, 1\}$ or $x \xleftarrow{\$} S_{R_q,1}$. When storing the value of a deterministic operation, we write $:=$, for example $x := 5y + 3$

2.2.1. Normal Distributions

Just as the general setting, we use normal distributions in a similar notation as Baum et. al. [7] as described in the following.

For a continuous normal distribution over $\mathbb{R}^N$ centered at $\mathbf{v} \in \mathbb{R}^N$ with a standard deviation σ the probability density function is

$$\rho^N_{\mathbf{v},\sigma}(x) = \frac{1}{\sqrt{2\pi}\sigma} \cdot \exp\left(\frac{-||x - \mathbf{v}||_2^2}{2\sigma^2}\right)$$

For a *discrete normal distribution* over R^k centered at $\mathbf{v} \in R^k$ with a standard deviation σ we use the following distribution function for all $x \in R^k$. Note that omitting the subscript $\mathbf{v}$ means that it is zero and omitting the superscript k means that it is one.

$$\mathcal{N}^k_{\mathbf{v},\sigma}(x) = \rho^{k\cdot N}_{\mathbf{v},\sigma}(x)/\rho^{k\cdot N}_{\sigma}(R^k) \text{ with } \rho^{k\cdot N}_{\sigma}(R^k) = \sum_{x\in R^k} \rho^{k\cdot N}_{\sigma}(x)$$

Lemma 1 *Since we work with "small" vectors, we also need a tail-bound version of the normal distribution to ensure that we don't sample far outliers. We use the tail-bound from [6] (also see Remark 1 in [7] and Lemma 3.3 in [19]) where for any $\delta > 0$ and $\sigma > 0$*

$$Pr[||z||_2 > \delta\sigma\sqrt{kN} \mid z \leftarrow \mathcal{N}^k_\sigma] < \delta^{kN} \cdot \exp\left(\frac{kN}{2}(1-\delta^2)\right)$$

We set $\delta := 2$ which is sufficient for the tail-bound to hold with a probability that is overwhelming in λ since $N = \Omega(\lambda)$ [7].

We use the following lemma to show that a normal distribution centered around 0 and a normal distribution with another center adjusted with rejection-sampling are statistically indistinguishable in our security parameter.

Lemma 2 *Let $V = \{v \in R^k \mid ||v||_2 < T\} \subseteq R^k$ and $\sigma \in \mathbb{R}$ with $\sigma \in \omega(T\sqrt{\log(kN)})$ and let $h : V \to \mathbb{R}$ be a probability distribution. Then there exists $M \in O(1)$ such that the probability distribution between the following to algorithms $\mathcal{A}, \mathcal{S}$ lies within statistical distance $2^{-\sigma}/M$ for $\sigma \in \omega(T\sqrt{\log(kN)})$*

$\mathcal{A}$:

1. $v \leftarrow h$
2. $z \leftarrow \mathcal{N}^k_{v,\sigma}$
3. *Output* (v, z) *with probability* $\min\left(\frac{\mathcal{N}^k_\sigma(z)}{M\mathcal{N}^k_{v,\sigma}(z)}, 1\right)$

$\mathcal{S}$:

1. $v \leftarrow h$
2. $z \leftarrow \mathcal{N}^k_\sigma$
3. *Output* (v, z) *with probability* $1/M$

Note that $\mathcal{A}$ outputs something with probability at least $\frac{1-2^{-l}}{M}$ for $l \in \omega(\log(kN))$. By setting $\sigma = \alpha T$ we obtain $M = \exp(12/\alpha + 1/(2\alpha^2))$ and therefore the statistical distance of the outputs of $\mathcal{A}, \mathcal{S}$ is at most $2^{-100}/M$ and $\mathcal{A}$ outputs something with probability at least $(1 - 2^{-100})/M$. In practice we choose $kN \gg 128$ but even for $kN = 128$ we obtain $M \approx 4.5$ which just decreases for larger choices [7, 19].

2.2.2. Polynomial ring properties

We use some properties of polynomial rings $R_q = \mathbb{Z}_q[x]/(x^N + 1)$ for our protocol, for example for embedding votes into polynomials.

Theorem 1 *[20, Corollary 1.2] Let $N \geq \omega > 1$ be powers of two and $q = 2\omega + 1 \mod 4\omega$ be a prime. Then the polynomial $x^N + 1$ factors as*

$$x^N + 1 = \prod_{j=1}^{\omega} (x^{N/\omega} - r_j) \mod q$$

for distinct $r_j \in \mathbb{Z}_q^*$ where $x^{N/\omega} - r_j$ are irreducible in the ring $\mathbb{Z}_q[x]$. Furthermore, any $y \in \mathbb{Z}_q[x]/(x^N + 1)$ that satisfies either

$$0 < ||y||_\infty < \frac{1}{\sqrt{\omega}} \cdot q^{1/\omega}$$

or

$$0 < ||y||_2 < q^{1/\omega}$$

has an inverse in $\mathbb{Z}_q[x]/(x^N + 1)$.

Since we want all challenge polynomials and their differences (i.e. elements $\overline{c} \in \overline{C}_{R_q,\kappa}$ which have $||\overline{c}||_\infty \leq 2$) to be invertible, we have to ensure that $2 < \frac{1}{\sqrt{\omega}} \cdot q^{1/\omega}$. For example, with $\omega = 8$ we get that $q > 2^{20}$ [20]. Note that q has to be chosen with $q = 2\omega + 1 \mod 4\omega$. Since ω represents the number of "slots" corresponding to the number of candidates of the election, this relation is crucial when choosing parameters and will be further discussed in Section 4.2.1 regarding the parameter selection.

Theorem 2 *In the case of Theorem 1 as stated above, all factors $x^{N/\omega} - r_j$ are relatively coprime to each other.*

With the Chinese remainder theorem it follows that

$$\mathbb{Z}_q[x]/(x^N + 1) \cong \mathbb{Z}_q[x]/(x^{N/\omega} - r_1) \times \mathbb{Z}_q[x]/(x^{N/\omega} - r_2) \times \cdots \times \mathbb{Z}_q[x]/(x^{N/\omega} - r_\omega)$$

are isomorph to each other. Calculations in of the rings can therefore be performed in the other and vice versa. Note that

$$\mathbb{Z}_q[x]/(x^{N/\omega} - r_1) \times \mathbb{Z}_q[x]/(x^{N/\omega} - r_2) \times \cdots \times \mathbb{Z}_q[x]/(x^{N/\omega} - r_\omega) \supseteq \mathbb{Z}_q \times \mathbb{Z}_q \times \cdots \times \mathbb{Z}_q = \mathbb{Z}_q^\omega$$

which we can use to store values from $\mathbb{Z}_q^\omega$ in R_q, i.e. for storing the votes. Multiplying or adding polynomials in $\mathbb{Z}_q[x]/(x^N + 1)$ corresponds to the respective component-wise operation in $\mathbb{Z}_q^\omega$.

The factors of $\mathbb{Z}_q[x]/(x^N+1)$ are coprime to each other because they are irreducible and therefore don't have any divisors.

Storing values from $\mathbb{Z}_q^\omega$ in $\mathbb{Z}_q[x]/(x^{N/\omega}-r_1)\times\mathbb{Z}_q[x]/(x^{N/\omega}-r_2)\times\cdots\times\mathbb{Z}_q[x]/(x^{N/\omega}-r_\omega)$ is achieved by setting the value in each of the components as the coefficient of x^0, i.e. the constant term.

2.3. Hardness Assumptions

Definition 1 *The search knapsack problem in the ℓ_2-norm (in short SKS²) is defined as follows: The SKS² problem is to find a short, non-zero vector x of ℓ_2-norm less than or equal to β in R_q^2 with $\begin{bmatrix} a & 1 \end{bmatrix} \cdot x = 0$ for a given uniformly random $a \xleftarrow{\$} R_q$. An algorithm $\mathcal{A}$ has advantage ϵ in solving $SKS^2_{N,q,\beta}$ if*

$$Pr\left[\begin{array}{c|c} \begin{bmatrix} a & 1 \end{bmatrix} \cdot x = 0 & a \xleftarrow{\$} R_q \\ \wedge ||x_i||_2 \leq \beta & 0 \neq x \in R_q^2 \leftarrow \mathcal{A}(a) \end{array} \right] \geq \epsilon$$

The SKS² problem corresponds to the Ring-SIS problem in its Hermite Normal form [22].

Lemma 3 *Let $1 < \omega < N$ be a power of 2. If q is a prime congruent to $2\,\omega+1 \mod 4\,\omega$ and*

$$\beta < q^{1/\omega}, \text{ and}$$
$$\beta < \sqrt{\frac{N}{2\pi e}} \cdot q^{n/k} \cdot 2^{-128/(k\cdot N)} - \sqrt{N}/2$$

then any (all-powerful) algorithm $\mathcal{A}$ has advantage at most 2^{-128} in solving $SKS^2_{n,k,\beta}$ [7].

Definition 2 *The decisional knapsack problem in the ℓ_∞-norm (in short DKS$^\infty$) is defined as follows: The DKS$^\infty$ problem is to distinguish the distribution $\begin{bmatrix} a & 1 \end{bmatrix} \cdot x$, for a short x, from the uniform distribution when given uniformly random $a \xleftarrow{\$} R_q$. An algorithm $\mathcal{A}$ has advantage ϵ in solving the $DKS^\infty_{N,q,\beta_\infty}$ problem if*

$$\left| Pr\left[b = 1 \mid a \xleftarrow{\$} R_q; x \xleftarrow{\$} S_{R_q,\beta_\infty}; b \leftarrow \mathcal{A}(a, \begin{bmatrix} a & 1 \end{bmatrix} \cdot x) \right] \right.$$
$$\left. - Pr\left[b = 1 \mid a \xleftarrow{\$} R_q; u \xleftarrow{\$} R_q; b \leftarrow \mathcal{A}(a, u) \right] \right| \geq \epsilon$$

The DKS$^\infty$ problem corresponds to the Ring-LWE problem when the number of samples is limited [22].

Lemma 4 *Let $1 < \omega < N$ be a power of 2. If q is a prime congruent to $2\,\omega+1 \mod 4\,\omega$ and*

$$q^{n/k} \cdot 2^{256/(k\cdot N)} \leq 2\beta \leq \frac{1}{\sqrt{\omega}} \cdot q^{1/\omega}$$

then any (all-powerful) algorithm $\mathcal{A}$ has advantage at most 2^{-128} in solving $DKS^\infty_{n,k,\beta}$ [7].

3. Building Blocks

3.1. Vote Encoding

We represent votes as vectors $\vec{v} \in \mathbb{Z}_t^\omega$ where each of the ω entries is used to represent a candidate. Setting an entry to 1 signifies casting a vote for the corresponding candidate and respectively setting it to 0 signifies not casting a vote for the corresponding candidate. Note that not all ω entries have to be used for candidates, instead some of these "slots" can remain empty in case there are less than ω candidates.

Depending on the election, a voter can cast votes for up to τ candidates (e.g 3 or 5). Since the number of votes cast needs to add up to τ, we use empty slots without corresponding candidates as filler and in case someone only wants to cast votes for ρ candidates (with $\rho < \tau$) the remaining $\tau - \rho$ votes will be filled in to the next empty slots each as a 1. This is similar to the *placeholder votes* in ElectionGuard in its specification [10]. Since a voter could potentially abstain from casting any of their τ votes, the number of empty slots needs to be at least τ so that they can cast all their votes as invalid.

We use Theorem 1 and Theorem 2 to embedd votes into elemtents of $R_t = \mathbb{Z}_t[x]/(x^N+1)$.

$$\text{encodeVote} : \mathbb{Z}_t^\omega \to R_t$$

$$\text{decodeVote} : R_t \to \mathbb{Z}_t^\omega$$

The encoding and decoding can be prepared by calculating e, factors and ω using the pseudocode for `prepareEncode` below. Note that this works by first factoring $x^N + 1$ into its ω factors (see Theorem 1) and then iterativly building a canonical basis $e = (e_1, ..., e_\omega)$. This basis has the property that

$$\forall i \in 1...\omega : e_i \equiv 1 \mod \text{factor}_i \wedge \forall i' \neq i : e_i \equiv 0 \mod \text{factor}_{i'}$$

The calculation of the canonical basis e and the factors of $x^N + 1$ can be done before an election by the election authority and be published along the other details of the election since it is unique for a given set of election parameters N, t. Voters or auditors can easily check the correctness by verifying that $\prod_{i=1}^{\omega} \text{factor}_i \stackrel{?}{=} x^N + 1$ and that e fulfils its property as mentioned above (i.e. each basis polynomial is one modulo one of the factors and zero modulo all other factors).

Using the pseudocode for `encodeVote` and `decodeVote` votes can then be transformed isomorphically between Z_t^ω and R_t. See Appendix A.1 for an example implementation in SageMath.

prepareEncode(N, t)

$R_t = \mathbb{Z}_t/(x^N + 1)$
factors = factor $(x^N + 1)$ under $\mathbb{Z}_t$
ω = len(factors)
e = [1 for i in 1...ω]
for i in 1...ω :
  A = list(factors)
  A.remove(factors[i])
  B = list(factors[i])
  while len(A) $\geq$ 1 :
    $c = A$.pop()
    $e[i] = e[i] - e[i] * \text{inversemod}(\prod_{b \in B} b, c) * \prod_{b \in B} b$
  B.append(c)
return e, factors, ω

encodeVote$(v \in \mathbb{Z}_t^\omega; e, \omega) \in R_t$

return $\sum_{i=1}^{\omega} v[i] \cdot e[i]$

decodeVote$(p \in R_t; \text{factors}, \omega) \in \mathbb{Z}_t^\omega$

return [p mod factors[i] for i in 1...ω]

3.2. BDLOP Commitments

We use the commitment scheme by Baum, Damgård, Lyubashevsky, Oechsner and Peikert, called in short *BDLOP*. [7] We use the BDLOP commitments in a ring $R_t = \mathbb{Z}_t[x]/(x^N + 1)$ with t prime, $t = 2\,\omega + 1 \mod 4\,\omega$ and N a power of two.

3.2.1. Commitment Scheme

The commitment scheme has three algorithms:

`BDLOP.KeyGen`(params) $\rightarrow (A_1, A_2)$ is used to create public parameters that can be used to commit to messages $m \in R_t^l$. It creates the public parameters $A_1 \in R_t^{n \times k}$ and $A_2 \in R_t^{l \times k}$ as follows:

$$A_1 = \begin{bmatrix} I_n & A_1' \end{bmatrix} \qquad \text{where } A_1' \xleftarrow{\$} R_t^{n \times (k-n)}$$
$$A_2 = \begin{bmatrix} 0^{l \times n} & I_l & A_2' \end{bmatrix} \quad \text{where } A_2' \xleftarrow{\$} R_t^{l \times (k-n-l)}$$

`BDLOP.Commit`$(m, \text{params}; r) \rightarrow$ com This is used to commit to a message $m \in R_t^l$ by choosing a random polynomial vector $r \xleftarrow{\$} S_{R_t,\beta}^k$ and calculating the commitment:

$$\text{com} := \begin{bmatrix} \text{com}_1 \\ \text{com}_2 \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \cdot r + \begin{bmatrix} 0^n \\ m \end{bmatrix}$$

`BDLOP.Open`$(\text{com}, m, r, f, \text{params}) \rightarrow \{0,1\}$ This is used to check whether an opening for a commitment is valid. The opening consists of a 3-tuple of $m \in R_t^l$, $r = \begin{bmatrix} r_1 \\ \dots \\ r_k \end{bmatrix} \in R_t^k$, and $f \in \overline{C}_{R_t,\kappa}$. This can be verified by checking

$$f \cdot \begin{bmatrix} \text{com}_1 \\ \text{com}_2 \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \cdot r + f \cdot \begin{bmatrix} 0^n \\ m \end{bmatrix}$$

and that $\forall i \in (1, \dots, k) : ||r_i||_2 \leq 4\sigma_C\sqrt{N}$.

For a proof of correctness of the commitment see [7].

3.2.2. Security of BDLOP

The BDLOP encryption scheme as described above is hiding if the $\text{DKS}^{\infty}_{N,t,\beta}$ problem is hard according to Lemma 4 and binding if the $\text{SKS}^{2}_{N,t,16\sigma_C\sqrt{\kappa N}}$ is hard [22] according to Lemma 3. Additionally, one can set the parameters in between both of these cases to gaining computational hardness for both the binding and hiding property. For a proof of security refer to [7].

3.2.3. Zero Knowledge Proofs

3.2.3.1. Proof of Linear Relation

One of the zero knowledge proofs we use in our work is the proof of linear relation of commitment values by Aranha, Baum, Gjøsteen, Silde and Tunge [4]. Let $[[x]], [[x']]$ be BDLOP commitments as described above to the values x and x' such that $x' = \alpha x + \beta$ for some $\alpha, \beta \in R_t$. Define

$$[[x]] = \text{Com}(x, r_x) = \begin{bmatrix} \text{com}_1 \\ \text{com}_2 \end{bmatrix}, \quad [[x']] = \text{Com}(x', r_{x'}) = \begin{bmatrix} \text{com}'_1 \\ \text{com}'_2 \end{bmatrix}$$

and use the proof Π_{Lin} as shown in Figure 3.1 to show the relation:

$$\mathcal{R}_{\text{Lin}} = \left\{ (u, w) \;\middle|\; \begin{array}{l} u = (\alpha, \beta, [[x]], [[x']]), w = (x, r_x, r_{x'}, f, f') : \\ \text{Open}([[x]], x, r_x, f) = \text{Open}([[x']], \alpha \cdot x + \beta, r_{x'}, f') = 1 \end{array} \right\}$$

The proof is based on rejection sampling and using Lemma 2 to mask the values of r_x and r'_x into a normal distribution. Then the verification equations (1) - (4) ensure that the commitment randomness is sufficiently small, that the commitments have been calculated honestly and lastly that the linear relation between x and x' holds. Refer to [4] for a proof of the zero knowledge properties.

Prover $\mathcal{P}$ | | Verifier $\mathcal{V}$

$y, y' \leftarrow \mathcal{N}_{\sigma_C}^k$

$t \coloneqq A_1 y, t' \coloneqq A_1 y'$

$u \coloneqq \alpha\langle A_2, y\rangle - \langle A_2, y'\rangle$ $\xrightarrow{\quad t, t', u \quad}$

$\xleftarrow{\quad d \quad}$ $d \xleftarrow{\$} C_{R_t,\kappa}$

$z \coloneqq y + d r_x$

$z' = y' + d r'_x$

Continue with probability:

$\mathcal{M} \coloneqq \{(r_x, z), (r'_x, z')\}$

$\prod_{(a,b)\in\mathcal{M}} \min\left(1, \frac{\mathcal{N}_{\sigma_C}^k(b)}{M \cdot \mathcal{N}_{da,\sigma_C}^k(b)}\right)$ $\xrightarrow{\quad z, z' \quad}$

Accept iff

(1): $\forall i \in [k] : ||z[i]||, ||z'[i]|| \leq 2\sigma_C\sqrt{N}$

(2): $A_1 z \stackrel{?}{=} t + d\,\mathrm{com}_1$

(3): $A_1 z' \stackrel{?}{=} t' + d\,\mathrm{com}'_1$

(4): $\alpha\langle A_2, z\rangle - \langle A_2, z'\rangle \stackrel{?}{=} (\alpha\,\mathrm{com}_2 + \beta - \mathrm{com}'_2)d + u$

Figure 3.1.: Protocol for proof Π_{Lin}

The proof Π_{Lin} shows the linear relation $x' = \alpha x + \beta$ for commitments $[[x]]$, $[[x']]$ to x, x' respectively.

3.2.3.2. Proof of Sum

In a similar fashion to the proof of linear relation, we also use a proof of sum by Baum, Damgård, Lyubashevsky, Oechsner and Peikert [7]. Given the prover has published three commitments

$$\begin{aligned} c &= \text{BDLOP.Commit}(x, r) \\ c' &= \text{BDLOP.Commit}(x', r') \\ c'' &= \text{BDLOP.Commit}(x'', r'') \end{aligned}$$

it shows that $x'' = \alpha_1 \cdot x + \alpha_2 \cdot x'$ with $\alpha_1, \alpha_2 \in R_t$ being public constants.

Use the proof Π_{Sum} as shown in Figure 3.2 to show the following relation $\mathcal{R}_{\text{Sum}}$. This proof is very similiar to the proof of linear relation Π_{Lin} as seen in section 3.2.3.1. For a detailed proof of correctness, soundness and the zero knowledge property of Π_{Sum} refer to [7, Section 4.4]

$$\mathcal{R}_{\text{Sum}} = \left\{ (u, w) \;\middle|\; \begin{array}{l} u = (\alpha_1, \alpha_2, c, c', c''), w = (r, r', r'', x, x', f, f', f'') : \\ \text{Open}(c, x, r, f) = \text{Open}(c', x', r', f') \\ = \text{Open}(c'', \alpha_1 \cdot x + \alpha_2 \cdot x', r'', f'') = 1 \end{array} \right\}$$

Prover $\mathcal{P}$		Verifier $\mathcal{V}$
$y, y', y'' \leftarrow \mathcal{N}_{\sigma_C}^k$		
$t \coloneqq A_1 y, t' \coloneqq A_1 y'$		
$t'' \coloneqq A_1 y''$		
$u \coloneqq \alpha_1 \cdot A_2 \cdot y + \alpha_2 \cdot A_2 \cdot y' - A_2 \cdot y''$	$\xrightarrow{t, t', t'', u}$	
	$\xleftarrow{d}$	$d \xleftarrow{\$} C_{R_t,\kappa}$
$z \coloneqq y + dr$		
$z' \coloneqq y' + dr'$		
$z'' \coloneqq y'' + dr''$		
Continue with probability:		
$\mathcal{M} \coloneqq \{(r, z), (r', z'), (r'', z'')\}$		
$\prod_{(a,b)\in\mathcal{M}} \min\left(1, \frac{\mathcal{N}_{\sigma_C}^k(b)}{M \cdot \mathcal{N}_{d\mathbf{a},\sigma_C}^k(b)}\right)$	$\xrightarrow{z, z', z''}$	
		Accept iff
		(1): $\forall i \in [k] : \|\|z[i]\|\| \leq 2\sigma_C\sqrt{N}$
		$\wedge\ \forall i \in [k] : \|\|z'[i]\|\| \leq 2\sigma_C\sqrt{N}$
		$\wedge\ \forall i \in [k] : \|\|z''[i]\|\| \leq 2\sigma_C\sqrt{N}$
		(2): $A_1 z \stackrel{?}{=} t + dc_1$
		(3): $A_1 z' \stackrel{?}{=} t' + dc_1'$
		(4): $A_1 z'' \stackrel{?}{=} t'' + dc_1''$
		(5): $\alpha_1 \cdot A_2 \cdot z + \alpha_2 \cdot A_2 \cdot z' - A_2 \cdot z'' \stackrel{?}{=} (\alpha_1 \cdot c_2 + \alpha_2 \cdot c_2' - c_2'') \cdot d + u$

Figure 3.2.: Protocol for proof Π_{Sum}

The proof Π_{Sum} shows the relation $x'' = \alpha_1 \cdot x + \alpha_2 \cdot x'$ for commitments c, c', c'' to x, x', x'' respectively.

3.2.3.3. Proof of Message Bound

We use a proof of bound for the message of a commitment adapted from the proof of knowledge of preimages by Baum et. al. [8, Figure 1] which is similar to the proof of opening of a BDLOP commitment [7, Figure 4]. We use this proof in the verifiable decryption to ensure that the accumulated noise stays within bounds.

Given a message m, calculate $\text{BDLOP.Commit}(m; r) \coloneqq \text{com} = \begin{bmatrix} \text{com}_1 \\ \text{com}_2 \end{bmatrix}$ and set

$$A \coloneqq \begin{bmatrix} A_1 \in R_t^{n \times k} & 0^n \\ A_2 \in R_t^{1 \times k} & 1 \end{bmatrix} \in R_t^{n+1 \times k+1}$$

$$s \coloneqq \begin{bmatrix} r_1 \\ \vdots \\ r_k \\ m \end{bmatrix} \in R_t^{k+1}, t \coloneqq \begin{bmatrix} \text{com}_1 \\ \text{com}_2 \end{bmatrix} \in R_t^{n+1}$$

and prove the relation

$$\mathcal{R}_{\text{Bound}} = \left\{ (u, w) \;\middle|\; \begin{array}{l} u = \left(A, t = \begin{bmatrix} \text{com}_1 \\ \text{com}_2 \end{bmatrix}\right), w = \left(s = \begin{bmatrix} r \\ m \end{bmatrix}\right) : \\ As = t \wedge \text{Open}(t, m, r, f) = 1 \wedge ||m||_2 < 2 \cdot B_{\hat{e}} \end{array} \right\}$$

using the protocol for Π_{Bound} given in Figure 3.3 for some bound $B_{\hat{e}} \coloneqq \sqrt{2N}\sigma_{\hat{e}}$. Note that the value $\sigma_{\hat{e}}$ and subsequently depends on the range of m, i.e. we have to choose how "thight" of a bound we want to prove. We use this proof in Section 3.3.2 for the verifiable decryption of BGV ciphertexts and further explain the choice of $\sigma_{\hat{e}}$ there.

The properties of the protocol follow directly from the properties of the proof of opening Π_{Open} of BDLOP commitments [7, Lemma 7], making this proof fulfil *completeness, special soundness* and *honest-verifier zero knowledge*. Note that value m in s is hidden from the verifier because the commitment is hiding. An extractor however can extract it in the same fashion an extractor for Π_{Open} extracts its message.

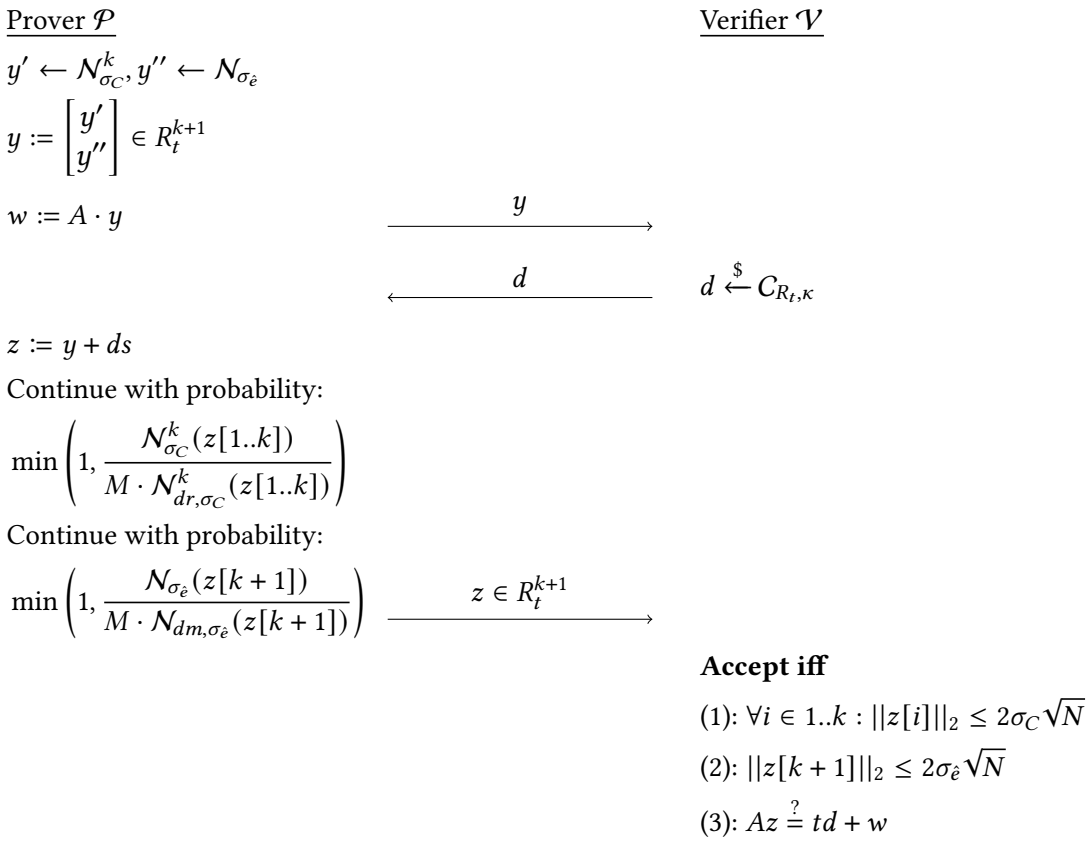

Figure 3.3.: Protocol for proof Π_{Bound}

The proof Π_{Bound} shows that the message contained in a given commitment is bound in the ℓ_2-norm by some value $2B_{\sigma_{\hat{e}}}$ and that the commitment is well-formed.

3.3. BGV Encryption

We use the fully homomorphic encryption scheme by Brakerski, Gentry and Vaikuntanathan, called in short *BGV*. [12] The following definition and explanation of BGV is based on [16]. The BGV scheme is a levelled scheme, i.e. homomorphic multiplications decrease a level and once a certain bound has been reached one needs to perform a bootstrapping operation to decrease the accumulated noise, which grows quadratically with each multiplication. Since we however don't make use of homomorphic multiplications in our use-case of BGV, we omit die ciphertext levels for simplicity and do not provide the homomorphic multiplication operation. For a full, detailed explanation of BGV refer to [12, 16].

3.3.1. Encryption Scheme

We present the the encryption scheme by detailing the plaintext and ciphertext spaces used, descryping the operations used and definining the security.

3.3.1.1. Plaintext and Ciphertext Spaces

Let the plaintext space of BGV be $\mathcal{P} = R_t = \mathbb{Z}_t[x]/(x^N + 1)$. This denotes the set of polynomials with degree smaller than N and all coefficients from $\mathbb{Z}_t$. Note that this ring corresponds to the one we use for the BDLOP commitments. Call t the *plaintext modulus*; while N denotes the ring dimension. Define the ciphertext space as $C = R_q \times R_q$ with $R_q = \mathbb{Z}_q[x]/(x^N + 1)$ and call $q \in \mathbb{Z}$ the *ciphertext modulus*.

3.3.1.2. Encryption Scheme Operations

We use a discrete noise distribution $\chi := \mathcal{N}_{\sigma_\chi}$ with a standard deviation of σ_χ. Note that the distribution outputs values from $\mathbb{Z}[x]/(x^N + 1)$ with coefficients bound as described by the tail-bound from Lemma 1. See Section 4.2.1 for how to set the value of σ_e. We set params $:= (R_q, \chi, t, N)$ as the public parameters.

The BGV encryption scheme consists of the following operations:

`BGV.KeyGen`$(\text{params}) \longrightarrow (\text{sk}, \text{pk})$ is used to generate a key pair. First, draw a small polynomial as the secret key $s \xleftarrow{\$} S_{R_q,1}$ and set sk $:= s$. Then generate the public key by drawing a random element $a \xleftarrow{\$} R_q$ and noise $e \leftarrow \chi$. Then set pk $:= (\text{pk}_1, \text{pk}_2) = (a \cdot s + te, -a)$.

`BGV.Encrypt`$(m, \text{pk}; \text{params}) \longrightarrow \text{c} = (\text{c}_1, \text{c}_2)$ is used to encrypt a message $m \in \mathcal{P}$. First, draw noise $e_1, e_2 \leftarrow \chi$ and a "small" polynomial $u \xleftarrow{\$} S_{R_q,1}$. Then compute

$$\begin{bmatrix} \text{c}_1 \\ \text{c}_2 \end{bmatrix} = \begin{bmatrix} \text{pk}_1 \\ \text{pk}_2 \end{bmatrix} \cdot u + t \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} + \begin{bmatrix} m \\ 0 \end{bmatrix}$$

and return $\text{c} = (\text{c}_1, \text{c}_2)$.

$\texttt{BGV.Decrypt}(c, \text{sk}; \text{params}) \rightarrow m$ is used to decrypt a ciphertext $c \in C$ back to a message $m \in \mathcal{P}$. This is done by calculating

$$m = \text{c}_1 + \text{c}_2 \cdot \text{sk} \mod q \mod t$$

and then returning m.

We use the following homomorphic operation:

$\texttt{BGV.Add}(c, \hat{c}; \text{params}) \rightarrow \tilde{c}$ is used to form the sum of two ciphertexts which is a homomorphic operation for forming the sum of the contained messages. It can simply be calculated for $c = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}, \hat{c} = \begin{bmatrix} \hat{c}_1 \\ \hat{c}_2 \end{bmatrix}$ as $\tilde{c} = \begin{bmatrix} \tilde{c}_1 \\ \tilde{c}_2 \end{bmatrix} := \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + \begin{bmatrix} \hat{c}_1 \\ \hat{c}_2 \end{bmatrix}$

which results in the noise e_1, e_2 and u growing linearly with the number of chiphertext additions. Refer to Section 3.3.2 for a calculation of the accumulated noise generated.

The BGV scheme is correct if $|| \text{c}_1 + \text{sk}\, \text{c}_2 ||_\infty < q/2$.

3.3.1.3. Security of BGV

The BGV encryption scheme as described above is CPA-secure if the $\text{DKS}^{\infty}_{N,q,\beta}$ problem is hard for some $\beta = \beta(N, q, p, \beta_\infty)$ [22]. For a proof of security refer to [12].

3.3.2. Verifiable Decryption

We use a proof of decryption to show that the final tally of the election has been decrypted correctly. We obtain the ciphertext $c = (c_1, c_2)$ as the homomorphic sum of all cast votes and then want to proof that $\text{BGV.Decrypt}(c, \text{sk}, \text{params}) = v$, where v is the tally. Our proof of decryption for a single ciphertext is very similar to the proof by Silde [22], which can be used to verifiably open multiple ciphertexts at once.

For the proof we need a public commitment to the secret key of the ciphertext, i.e. $[[\text{sk}]] := \text{BDLOP.Commit}(\text{sk}; r_{\text{sk}})$. This can be calculated as part of the key generation and published on the bulletin board alongside a proof of correctness.

To prove the correctness of the decryption, the prover first decrypts the ciphertext $\text{BGV.Decrypt}(c, \text{sk}, \text{params}) := v$ and then calculats the accumulated noise $\hat{e} := e \cdot u + e_1 + \text{sk} \cdot e_2$ and commits to it: $\text{BDLOP.Commit}(\hat{e}) = \text{com}_{\hat{e}}$.

Then we prove the relation

$$\mathcal{R}_{\text{VerDec}} = \left\{ (\hat{u}, w) \;\middle|\; \begin{array}{l} \hat{u} = (\text{pk} = (\text{pk}_1, \text{pk}_2), [[\text{sk}]], \text{c} = (\text{c}_1, \text{c}_2), v), w = (\text{sk}, \hat{e}) : \\ t \cdot \hat{e} = \text{c}_1 - v + \text{c}_2 \cdot \text{sk} \wedge ||\hat{e}||_\infty < q/2t \end{array} \right\}$$

using the proof Π_{VerDec} as given in Figure 3.4.

The proof of verifiable decryption has two parts. First, it shows that the accumulated noise $\hat{e}$ is actually the noise contained in the ciphertext to guarantee that it has been calculated correctly. This is accomplished by using Π_{Lin} to show the according linear equation. Then, we use the proof Π_{Bound} to show that the noise $\hat{e}$ is within certain bounds that guarantee that the ciphertext is still correct and did not "overflow".

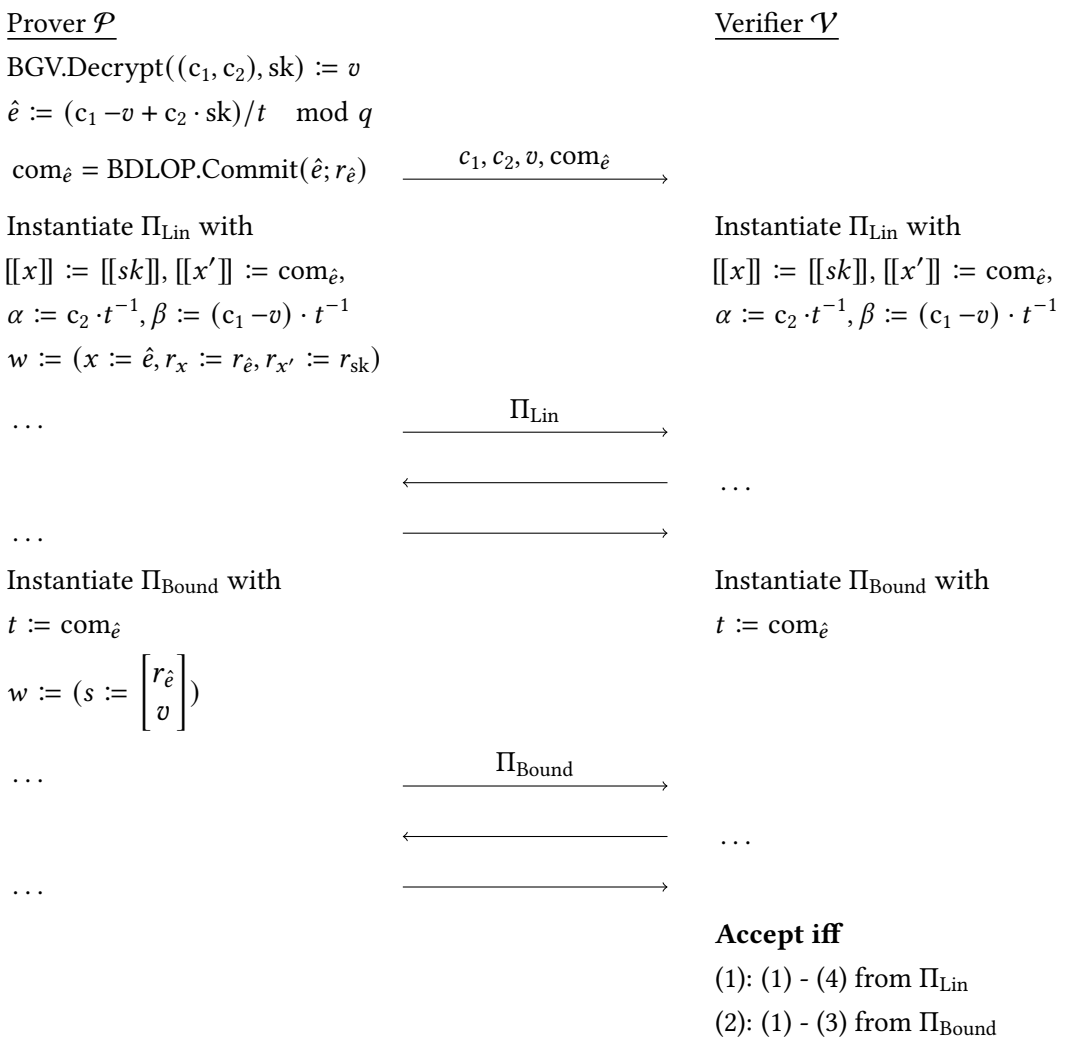

Figure 3.4.: Protocol for proof Π_{VerDec}

The proof Π_{VerDec} shows that a ciphertext (c_1, c_2) has been honestly decrypted to the value v by calculating the accumulated noise $\hat{e}$ and first proving a linear relation to show that $\hat{e}$ has been calculated correctly and then proving a bound on $\hat{e}$ that ensures that the ciphertext is valid.

The verifiable decryption protocol Π_{VerDec} is *complete* and fulfils *soundness* and *honest-verifier zero knowledge* for the relation $\mathcal{R}_{\text{VerDec}}$ if $B_{\hat{e}} < q/(4t)$. We prove these three properties in the following. Note that this proof is very similar to the proof by Silde [22].

Completeness The proof Π_{VerDec} is *complete*, if the BGV encryption scheme is correct, i.e. if $|| \, \mathrm{c}_1 + \mathrm{sk} \cdot \mathrm{c}_2 \, || < q/2$ and if additionally the two proofs Π_{Lin} and Π_{Bound} are complete, which is the case. During parameter selection we therefore need to ensure $|| \, \mathrm{c}_1 + \mathrm{sk} \cdot \mathrm{c}_2 \, || < q/2$. Then, the proof Π_{Bound} gives us a bound on the accumulated noise $||\hat{e}||_2 \leq 2B_{\hat{e}}$. Subsequently, if $B_{\hat{e}} < q/(4t)$ we get $||\hat{e}||_\infty < q/(2t)$ and the decryption is correct. Hence, we also need to ensure that $||\hat{e}||_\infty < q/(2t)$ during parameter selection to make our protocol fulfil completeness.

We now first calculate a bound for $|| \, \mathrm{c}_1 + \mathrm{sk} \cdot \mathrm{c}_2 \, ||_\infty$ and then a bound for $||\hat{e}||_\infty$, which we both use for parameter selection.

To begin, note that for a honestly, freshly generated ciphertext the following bound holds:

$$\begin{aligned} ||c_1 + \mathrm{sk} \cdot c_2||_\infty &= || \, \mathrm{pk}_1 \cdot u + te_1 + m + s \cdot (\mathrm{pk}_2 \cdot u + te_2)||_\infty \\ &= ||t \cdot (e \cdot u + e_1 + s \cdot e_2) + m||_\infty \\ &\leq t \cdot \left(2 \cdot 2\sigma_e N + 2\sigma_e \sqrt{N}\right) \end{aligned}$$

with $||e||_2, ||e_1||_2, ||e_2||_2 \leq 2\sigma_e\sqrt{N}$ as dictated by the tail-bound from Lemma 1 and $||s||_\infty, ||u||_\infty = 1$ which have been sampled from $S_{R_q,1}$.

Since the ciphertext we want to verifiability decrypt is the result of the homomorphic sum of ψ many ciphertexts, we need to provide a similar bound there. For $i \in [1..\psi]$:

$$\mathrm{c}_i \coloneqq \begin{bmatrix} \mathrm{c}_{1,i} \\ \mathrm{c}_{2,i} \end{bmatrix} = \begin{bmatrix} \mathrm{pk}_1 \\ \mathrm{pk}_2 \end{bmatrix} \cdot u_i + t \cdot \begin{bmatrix} e_{1,i} \\ e_{2,i} \end{bmatrix} + \begin{bmatrix} m_i \\ 0 \end{bmatrix}$$

the sum is

$$\sum_i \mathrm{c}_i = \begin{bmatrix} \widehat{\mathrm{c}_{1,i}} \\ \widehat{\mathrm{c}_{2,i}} \end{bmatrix} \coloneqq \begin{bmatrix} \sum_i \mathrm{c}_{1,i} \\ \sum_i \mathrm{c}_{2,i} \end{bmatrix} = \begin{bmatrix} \mathrm{pk}_1 \\ \mathrm{pk}_2 \end{bmatrix} \cdot \left(\sum_i u_i\right) + t \cdot \begin{bmatrix} \sum_i e_{1,i} \\ \sum_i e_{2,i} \end{bmatrix} + \begin{bmatrix} \sum_i m_i \\ 0 \end{bmatrix}$$

with the following bound

$$\begin{aligned} ||\widehat{\mathrm{c}_{1,i}} + \mathrm{sk} \cdot \widehat{\mathrm{c}_{2,i}}||_\infty &= \left\| t \cdot \left(e \cdot \left(\sum_i u_i\right) + \left(\sum_i e_{1,i}\right) + s \cdot \left(\sum_i e_{2,i}\right)\right) + \left(\sum_i m_i\right) \right\|_\infty \\ &\leq t \cdot (2 \cdot 2\sigma_e N\psi + 2\psi\sigma_e\sqrt{N}) \\ &= 2\psi\sigma_e(2N + \sqrt{N}) \end{aligned}$$

which results from

$$
\begin{aligned}
||e||_2 &\leq 2\sigma_e\sqrt{N} \\
||s||_\infty &= 1 \\
\left\|\sum_i u_i\right\|_\infty &\leq \psi \\
\left\|\sum_i e_{1,i}\right\|_2 &\leq 2\psi\sigma_e\sqrt{N} \\
\left\|\sum_i e_{2,i}\right\|_2 &\leq 2\psi\sigma_e\sqrt{N}
\end{aligned}
$$

We now write (c_1, c_2) for the ciphertext that is the result of homomorphic addition for simplicity. Using the result from above we can calculate a bound for $||\hat{e}||_\infty$ using the relation that $B_{\hat{e}} = \sqrt{2N}\sigma_{\hat{e}}$ (from the tail-bound in Lemma 1) and that we can choose $\sigma_{\hat{e}} = 0.675||s'd'||_2$ where d' is the challenge from the protocol Π_{Bound} and s' corresponds to the last part of the witness used in the proof Π_{Lin}, i.e. the accumulated noise and its bound is $||c_1 + \text{sk}\, c_2||_\infty \leq 2\psi\sigma_e(2N + \sqrt{N})$. This last value results from the rejection sampling lemma (see Lemma 2) and is comparable to the argumentation in [22].

$$
\begin{aligned}
||\hat{e}||_\infty &\leq 2B_{\hat{e}} \\
&= \sqrt{8N}\sigma_{\hat{e}} \\
&\leq \sqrt{8N} \cdot 0.675||s' \cdot d'||_2 \\
&\leq 2\sqrt{N}||s'||_1||d'||_\infty \\
&\leq 2\sqrt{N} \cdot (2\psi\sigma_e(2N + \sqrt{N}) \cdot N) \\
&= 4\psi\sigma_e N^2(2\sqrt{N} + 1)
\end{aligned}
$$

With these two results we can set our parameters such that $||\widehat{c_{1,i}} + \text{sk} \cdot \widehat{c_{2,i}}||_\infty \leq 2\psi\sigma_e(2N + \sqrt{N}) \leq q/2$ and $||\hat{e}||_\infty \leq 2B_{\hat{e}} \leq 4\psi\sigma_e N^2(2\sqrt{N} + 1) < (q/2t)$ with $B_{\hat{e}} = \sqrt{2N}\sigma_{\hat{e}}$ to gain completeness.

Soundness The soundness property follows directly from the soundness of Π_{Lin} and Π_{Bound}. Using rewinding, we can extract the secret key sk or the noise $\hat{e}$ (which subsequently reveals the secret key since we can then calculate the linear equation we prove with Π_{Lin}). Alternatively, we gain some short vectors breaking the underlying SKS^2 problem for the given parameters. Again, this argumentation is equal to the one in [22].

Honest-Verifier Zero Knowledge As before, this property follows from the protocols Π_{Lin} and Π_{Bound}, which are both honest-verifier zero knowledge. Given a message v we can simulate Π_{VerDec} by sampling a uniformly random value $\hat{e}$, committing to it as $\text{com}_{\hat{e}}$ and then simulating the proofs Π_{Lin} and Π_{Bound}. Since we can simulate a transcript for Π_{VerDec}, it is *honest-verifier zero knowledge* [22].

this page is intentionally left blank

3.4. Zero Knowledge Proofs

3.4.1. Commitment Ciphertext Equality

One of the further building blocks required is a proof to show that a BGV ciphertext and a BDLOP commitment contain the same message. This will be used to show that a commitment contains the same vote as a corresponding ciphertext to allow use of both in further steps of the voting procedure.

Prepare $(\text{sk}, \text{pk}) \leftarrow \text{BGV.KeyGen}(\text{params})$ and $(A_1, A_2) \leftarrow \text{BDLOP.KeyGen}()$. Then, for a message $m \in R_t$ calculate $(\text{c}_1 \in R_q, \text{c}_2 \in R_q) \leftarrow \text{BGV.Encrypt}(m, \text{pk}, \text{params}; e_1, e_2, u)$ with noise e_1, e_2 and encryption randomness u and calculate the commitment $\text{com} = \begin{bmatrix} \text{com}_1 \in R_t \\ \text{com}_2 \in R_t \end{bmatrix} = \text{BDLOP.Commit}(m, A_1, A_2; r)$ with randomness r.

It is important to note here that the commitments are calculated in the ring R_t, while the ciphertext resides in R_q, with $q \gg t$. We will make note of this throughout our argumentation to show that our proof is still correct.

We want to show the relation

$$\mathcal{R}_{\text{ComEqCiph}} = \left\{ (u, w) \;\middle|\; \begin{array}{l} u = (\text{c}_1, \text{c}_2, \text{com} = (\text{com}_1, \text{com}_2), A_1, A_2, \text{pk}), \\ w = (r, m, u, e_1, e_2, f) : \\ \text{BDLOP.Commit}(\text{com}, m, f; r) = 1 \\ \wedge (\text{c}_1, \text{c}_2) = \text{BGV.Encrypt}(m, \text{pk}, \text{params}; e_1, e_2, u) \end{array} \right\}$$

to show that the commitment and ciphertext have been calculated correctly and, more important, that the commitment and the ciphertext contain the same message m.

Notation: All components and calculations are assumed to be over the ring R_q unless explicitly marked otherwise. We use a gray colorbox to show calculations in R_t: For example, A_1 shows that A_1 is defined over the ring R_t.

Writing the commitment and encryption calculation encryptions in matrix form yields the following notation. The notation in matrix form can now be used to formulate the proof $\Pi_{\text{ComEqCiph}}$ as seen in Figure 3.5.

$$\begin{pmatrix} A_1 & & & & \\ A_2 & 1 & & & \\ & 1 & \text{pk}_1 & t & \\ & & \text{pk}_2 & & t \end{pmatrix} \cdot \begin{pmatrix} r \\ m \\ u \\ e_1 \\ e_2 \end{pmatrix} = \begin{pmatrix} \text{com}_1 \\ \text{com}_2 \\ \text{c}_1 \\ \text{c}_2 \end{pmatrix} \tag{3.1}$$

We use this unconventional notation with mixed polynomial rings since we have to show the correctness of the commitment in R_t and of the ciphertext in R_q. The basic idea is that we can calculate with variables $x \in R_t$ in R_q since $q \gg t$ and consequently the conversion to R_q is possible without information loss by simply interpreting x as a polynomial in R_q. Additionally, if we show that $||x||_\infty \leq q/2$ is valid at all times we can

guarantee that the modulus q did not affect x and hence we can revert back to R_t by switching all coefficients to modulus t and then receive a correct result for calculations over R_t.

We call the big matrix $\mathcal{M}$:

$$\mathcal{M} := \begin{pmatrix} A_1 & & & \\ A_2 & 1 & & \\ & 1 & \mathrm{pk}_1 & t \\ & & \mathrm{pk}_2 & & t \end{pmatrix}$$

At the end of the protocol, the verifier needs to check multiple equations (conditions) before accepting. The first condition is derived from the matrix notation above and is used to verify that a) both the commitment and the ciphertext contain the same message m and b) the prover knows the randomness used to calculate both. The second and third condition is used to ensure that a valid value has been chosen for r, m, u, e_1, e_2 during committing to the message and encrypting it.

Prover $\mathcal{P}$ | Verifier $\mathcal{V}$

$\hat{r} \leftarrow \mathcal{N}_{\sigma_C}^k, \hat{u} \leftarrow \mathcal{N}_{\sigma_C}$

$\hat{e_1}, \hat{e_2} \leftarrow \mathcal{N}_{\sigma_e}, \hat{m} \leftarrow \mathcal{N}_{\sigma_m}$

$$\begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix} = \mathcal{M} \cdot \begin{pmatrix} \hat{r} \\ \hat{m} \\ \hat{u} \\ \hat{e_1} \\ \hat{e_2} \end{pmatrix}$$

$z_1, z_2, z_3, z_4 \longrightarrow$

$\longleftarrow$ d

$d \xleftarrow{\$} C_{R_q,\kappa}$

$\tilde{r} := \hat{r} + dr$

$\tilde{m} := \hat{m} + dm$

$\tilde{u} := \hat{u} + du$

$\tilde{e_1} := \hat{e_1} + de_1$

$\tilde{e_2} := \hat{e_2} + de_2$

Continue with probability:

$$\min\left(1, \frac{\mathcal{N}_{\sigma_C}^k(\tilde{r})}{M \cdot \mathcal{N}_{dr,\sigma_C}^k(\tilde{r})}\right)$$

Continue with probability:

$$\min\left(1, \frac{\mathcal{N}_{\sigma_m}(\tilde{m})}{M \cdot \mathcal{N}_{dr,\sigma_m}(\tilde{m})}\right)$$

Continue with probability:

$$\min\left(1, \frac{\mathcal{N}_{\sigma_C}(\tilde{u})}{M \cdot \mathcal{N}_{du,\sigma_C}(\tilde{u})}\right)$$

Continue with probability:

$\mathcal{M} := \{(e_1, \tilde{e_1}), (e_2, \tilde{e_2})\}$

$$\prod_{(a,b)\in\mathcal{M}} \min\left(1, \frac{\mathcal{N}_{\sigma_e}(b)}{M \cdot \mathcal{N}_{da,\sigma_e}(b)}\right)$$

$\tilde{r}, \tilde{m}, \tilde{u}, \tilde{e_1}, \tilde{e_2} \longrightarrow$

Accept iff

(1): condition

$$\mathcal{M} \cdot \begin{pmatrix} \tilde{r} \\ \tilde{m} \\ \tilde{u} \\ \tilde{e_1} \\ \tilde{e_2} \end{pmatrix} \stackrel{?}{=} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix} + d \cdot \begin{pmatrix} \mathrm{com}_1 \\ \mathrm{com}_2 \\ \mathrm{c}_1 \\ \mathrm{c}_2 \end{pmatrix}$$

(2): $\forall i \in [k] : ||\tilde{r}[i]||_2 \leq 2\sigma_C\sqrt{N}$

(3): $||\tilde{u}||_2 \leq 2\sigma_C\sqrt{N}, ||\tilde{e_1}||_2, ||\tilde{e_2}||_2 \leq 2\sigma_e\sqrt{N}$
$\wedge\ ||\tilde{m}||_2 \leq 2\sigma_m\sqrt{N}$

Figure 3.5.: Protocol for proof $\Pi_{\mathrm{ComEqCiph}}$

The proof $\Pi_{\mathrm{ComEqCiph}}$ shows that a commitment and a ciphertext contain the same message.

Completeness A prover with witness (r, m, u, e_1, e_2) can convince the verifier by running the proof as specified. Then, the verification conditions will be fulfilled as shown in the following:

$\begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix}$ in verification equation (1) has been calculated by the prover in the beginning of the protocol. Substituting these values yields

$$\mathcal{M} \cdot \begin{pmatrix} \tilde{r} \\ \tilde{m} \\ \tilde{u} \\ \tilde{e_1} \\ \tilde{e_2} \end{pmatrix} \stackrel{?}{=} \mathcal{M} \cdot \begin{pmatrix} \hat{r} \\ \hat{m} \\ \hat{u} \\ \hat{e_1} \\ \hat{e_2} \end{pmatrix} + d \cdot \begin{pmatrix} \mathrm{com}_1 \\ \mathrm{com}_2 \\ \mathrm{c}_1 \\ \mathrm{c}_2 \end{pmatrix}$$

where we can then use $\begin{pmatrix} \tilde{r} \\ \tilde{m} \\ \tilde{u} \\ \tilde{e_1} \\ \tilde{e_2} \end{pmatrix} = \begin{pmatrix} \hat{r} + dr \\ \hat{m} + dm \\ \hat{u} + du \\ \hat{e_1} + de_1 \\ \hat{e_2} + de_2 \end{pmatrix}$ from the calculations of the prover.

Its important to closely observe $\tilde{m} = \hat{m} + dm$. We argue that we can switch to R_t here. This is only possible if the ℓ_∞-norm of all components of this equation is smaller than $q/2$ to ensure that there was no reduction modulo q. Note that $||m||_\infty \leq t$ since m is the result of encodeVote$(v) \in R_t$. We get $||\hat{m}||_2 \leq 2\sigma_m\sqrt{N}$ from the tail-bound of Lemma 1 and hence $||\hat{m}||_\infty \leq 2\sigma_m\sqrt{N}$. Lastly, $||\tilde{m}||_\infty = ||\hat{m} + dm||_\infty = ||\hat{m}||_\infty + ||dm||_\infty = ||\hat{m}||_\infty + ||d||_1 \cdot ||m||_\infty = 2\sigma_m\sqrt{N} + \kappa t \ll q/2$ with $||d||_1 = \kappa$ since d has been generated accordingly as a challenge. The standard deviation σ_m is set in way to hide m and is sufficient to fulfil $2\sigma_m\sqrt{N} + \kappa t \ll q/2$. We describe the selection of σ_m in Section 4.2.1. As a result, we may interpret the given equation in R_t and write $\tilde{m} = \hat{m} + dm$. We also start interpreting com_2 as a polynomial over R_t from here on, which is possible since it has been calculated over R_t. With this result we get

$$\mathcal{M} \cdot \begin{pmatrix} \hat{r} + dr \\ \hat{m} + dm \\ \hat{u} + du \\ \hat{e_1} + de_1 \\ \hat{e_2} + de_2 \end{pmatrix} \stackrel{?}{=} \mathcal{M} \cdot \begin{pmatrix} \hat{r} \\ \hat{m} \\ \hat{u} \\ \hat{e_1} \\ \hat{e_2} \end{pmatrix} + d \cdot \begin{pmatrix} \mathrm{com}_1 \\ \mathrm{com}_2 \\ \mathrm{c}_1 \\ \mathrm{c}_2 \end{pmatrix}$$

which by matrix operations simplifies to

$$\mathcal{M} \cdot \begin{pmatrix} \hat{r} \\ \hat{m} \\ \hat{u} \\ \hat{e_1} \\ \hat{e_2} \end{pmatrix} + d \cdot \mathcal{M} \cdot \begin{pmatrix} r \\ m \\ u \\ e_1 \\ e_2 \end{pmatrix} \stackrel{?}{=} \mathcal{M} \cdot \begin{pmatrix} \hat{r} \\ \hat{m} \\ \hat{u} \\ \hat{e_1} \\ \hat{e_2} \end{pmatrix} + d \cdot \begin{pmatrix} \mathrm{com}_1 \\ \mathrm{com}_2 \\ \mathrm{c}_1 \\ \mathrm{c}_2 \end{pmatrix}$$

This can be written as

$$d \cdot \mathcal{M} \cdot \begin{pmatrix} r \\ m \\ u \\ e_1 \\ e_2 \end{pmatrix} \stackrel{?}{=} d \cdot \begin{pmatrix} \mathrm{com}_1 \\ \mathrm{com}_2 \\ \mathrm{c}_1 \\ \mathrm{c}_2 \end{pmatrix}$$

Now we substitute $\mathcal{M}$

$$d \cdot \begin{pmatrix} A_1 & & & & \\ A_2 & 1 & & & \\ & 1 & \mathrm{pk}_1 & t & \\ & & \mathrm{pk}_2 & & t \end{pmatrix} \cdot \begin{pmatrix} r \\ m \\ u \\ e_1 \\ e_2 \end{pmatrix} \stackrel{?}{=} d \cdot \begin{pmatrix} \mathrm{com}_1 \\ \mathrm{com}_2 \\ \mathrm{c}_1 \\ \mathrm{c}_2 \end{pmatrix}$$

Note that we can readily switch 1 and d from R_q to R_t since both are sufficiently small, i.e. $||1||_\infty = 1 \leq q/2$ and $||d||_\infty = 1 \leq q/2$. Hence, $d \cdot A_1 r = d\,\mathrm{com}_1$ follows directly from equation (3.1). The same is true for the second line, i.e. $d \cdot (A_2 \cdot r + 1 \cdot m) = d \cdot \mathrm{com}_2$. The third and fourth line also follow from equation (3.1) over R_q. Conditions(1) is therefore fulfilled.

Conditions (2) and (3) are fulfilled as well since by Lemma 2 it is statistically indistinguishable whether all polynomials in $\tilde{r}$ and the polynomials $\tilde{m}, \tilde{u}, \tilde{e_1}, \tilde{e_2}$ stem from a normal distribution around another center than 0 or from a normal distribution with center 0. Additionally, the probability that the prover can output these polynomials is at least $\left(\frac{1-2^{-100}}{M}\right)^5$. Finally, the bound given by Lemma 1 for the ℓ_2-norm of these polynomials ensures that they are smaller or equal to $2\sigma_i\sqrt{N}$ except with negligible probability for their respective standard deviation σ_i, therefore fulfilling conditions (2) and (3).

In total, a prover with access to the witness can answer correctly for any challenge d, making the protocol *complete*. □

Soundness Given a commitment $\mathrm{com} = \begin{pmatrix} \mathrm{com}_1 \\ \mathrm{com}_2 \end{pmatrix}$, a ciphertext $\mathrm{c} = \begin{pmatrix} \mathrm{c}_1 \\ \mathrm{c}_2 \end{pmatrix}$ and a pair of transcripts for $\Pi_{\mathrm{ComEqCiph}}$ $(z_1, z_2, z_3, z_4, d, \tilde{r}, \tilde{m}, \tilde{u}, \tilde{e_1}, \tilde{e_2})$ and $(z_1, z_2, z_3, z_4, d', \tilde{r}', \tilde{m}', \tilde{u}', \tilde{e_1}', \tilde{e_2}')$ with $d \neq d'$ we can extract the witness (r, m, u, e_1, e_2) with a relaxation factor $f = d - d' \in \overline{C}_{R_q,\kappa}$. Note that f can be interpreted over R_t since $||f||_\infty \leq 2 \ll q/2$ and additionally is invertible over R_t. It holds that

$$\begin{pmatrix} A_1 & & & & \\ A_2 & 1 & & & \\ & 1 & \mathrm{pk}_1 & t & \\ & & \mathrm{pk}_2 & & t \end{pmatrix} \cdot \begin{pmatrix} \tilde{r} - \tilde{r}' \\ \tilde{m} - \tilde{m}' \\ \tilde{u} - \tilde{u}' \\ \tilde{e_1} - \tilde{e_1}' \\ \tilde{e_2} - \tilde{e_2}' \end{pmatrix} = f \cdot \begin{pmatrix} \mathrm{com}_1 \\ \mathrm{com}_2 \\ \mathrm{c}_1 \\ \mathrm{c}_2 \end{pmatrix}$$

We can calculate $r = \tilde{r} - \tilde{r}'$ $(r \in R_q^k)$ and since f is invertible over R_t we can extract the message m from the commitment by calculating $m \coloneqq \mathrm{com}_2 - f^{-1} \cdot A_2 r$ over R_t. This is a

valid opening (m, r, f) to the commitment since $\forall i \in [k] : ||r_i||_2 \leq ||\tilde{r}_i||_2 + ||\tilde{r}'_i||_2 \leq 4\sigma_C\sqrt{N}$ and $\begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \cdot r + f \cdot \begin{bmatrix} 0^n \\ m \end{bmatrix} = f \cdot \begin{bmatrix} \text{com}_1 \\ \text{com}_2 \end{bmatrix}$.

We extract the encryption randomness in a similar fashion and gain

$$\begin{pmatrix} r \\ m \\ u \\ e_1 \\ e_2 \end{pmatrix} := \begin{pmatrix} \tilde{r} - \tilde{r}' \\ \text{com}_2 - f^{-1} \cdot A_2 r \\ \tilde{u} - \tilde{u}' \\ \tilde{e_1} - \tilde{e_1}' \\ \tilde{e_2} - \tilde{e_2}' \end{pmatrix}$$

This is a valid encryption since

$$f \cdot \begin{bmatrix} \text{c}_1 \\ \text{c}_2 \end{bmatrix} = \begin{bmatrix} \text{pk}_1 \\ \text{pk}_2 \end{bmatrix} \cdot f \cdot u + t \begin{bmatrix} f \cdot e_1 \\ f \cdot e_2 \end{bmatrix} + \begin{bmatrix} f \cdot m \\ 0 \end{bmatrix}$$

and $||f||_\infty \leq 2$, $||u||_1 \leq 4\sigma_C N$, $||e_1||_1 \leq 4\sigma_e N$, $||e_2||_1 \leq 4\sigma_e N$ which gives a bound on the noise of $||f \cdot u||_\infty \leq 8\sigma_C N$, $||f \cdot e_1||_\infty \leq 8\sigma_e N$, $||f \cdot e_2||_\infty \leq 8\sigma_e N$. The paper on BGV gives an upper bound on the noise for ciphertexts to still be decryptable, namely $\frac{q}{2}$ (see [12], Lemma 6). Due to this it follows that the encryption is still valid here for the message $f \cdot m$ if $8\sigma_C N \leq \frac{q}{2}$ and $8\sigma_e N \leq \frac{q}{2}$. By choosing fitting parameters (especially $\sigma_C \cdot N \ll q, \sigma_e \cdot N \ll q$) we can ensure that this inequality holds and the encryption is valid given the encryption equation above.

Given two transcripts, we can extract the witness from the prover up to a relaxation factor f, making the protocol fulfil *special soundness.* □

Statistically Honest Verifier Zero Knowledge To simulate an accepting transcript, draw $d \xleftarrow{\$} C_{R_q,\kappa}$ $\tilde{r} \leftarrow \mathcal{N}_{\sigma_C}^k, \tilde{m} \leftarrow \mathcal{N}_{\sigma_m}, \tilde{u} \leftarrow \mathcal{N}_{\sigma_C}$ and $\tilde{e_1}, \tilde{e_2} \leftarrow \mathcal{N}_{\sigma_e}$. Then, calculate

$$\begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix} := \begin{pmatrix} A_1 & & & & \\ A_2 & 1 & & & \\ & 1 & \text{pk}_1 & t & \\ & & \text{pk}_2 & & t \end{pmatrix} \cdot \begin{pmatrix} \tilde{r} \\ \tilde{m} \\ \tilde{u} \\ \tilde{e_1} \\ \tilde{e_2} \end{pmatrix} - d \cdot \begin{pmatrix} \text{com}_1 \\ \text{com}_2 \\ \text{c}_1 \\ \text{c}_2 \end{pmatrix}$$

The distribution of the simulated values are statistically indistinguishable from a real, non-aborting transcript since the simulator acts similar to $\mathcal{S}$ from Lemma 2. Therefore the protocol is *statistically honest verifier zero knowledge.* □

3.4.2. Binary Proof

One of the properties we need to show for votes to be correct is that they are binary, i.e. all entries of the vote are either zero or one. Formally, for a vote $\mathbf{v} \in \mathbb{Z}_t^\omega$ we need to show that $\forall i \in [1, \ldots, \omega] : \mathbf{v}_i \in \{0, 1\}$. Note that this hold iff

$$\begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \ldots \\ \mathbf{v}_\omega \end{pmatrix} \cdot \begin{pmatrix} \mathbf{v}_1 - 1 \\ \mathbf{v}_2 - 1 \\ \ldots \\ \mathbf{v}_\omega - 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \ldots \\ 0 \end{pmatrix}$$

Since we embed votes into polynomials from R_t using Theorem 2 and the functions `encodeVote` and `decodeVote` from section 3.1 we set $v \coloneqq \text{encodeVote}(\mathbf{v}) \in R_t$ and with $1_p = \text{encodeVote}\left((1, \ldots, 1)^\intercal \in \mathbb{Z}_t^\omega\right)$ the above equation corresponds to

$$v \cdot \left(v - 1_p\right) = 0_p$$

We use a proof of product by Attema, Lyubashevsky and Seiler [5] to show the binary property. See Figure 3.6 for the protocol.

We instantiate the product proof with $B_0 \coloneqq A_1, b_1, \ldots, b_4 \coloneqq A_2$ and the messages as $m_1 \coloneqq v, m_2 \coloneqq v - 1_p, m_3 \coloneqq 0_p$ to prove that $m_1 \cdot m_2 = m_3$ which then corresponds to $v \cdot \left(v - 1_p\right) = 0_p$. We call the resulting proof Π_{Binary}. Since the protocol of the product proof uses a little different notation, note that $\chi \coloneqq S_{R_t,\beta}, \mathfrak{s} \coloneqq \sigma_C$ and $D_{\mathfrak{s}} \coloneqq \mathcal{N}_{\sigma_C}$ in our notation.

The product proof fulfils the properties *completeness, soundness* and *honest-verifier zero knowledge* [5]. Since our binary proof is an instatiation of the product proof, it also fulfils these properties.

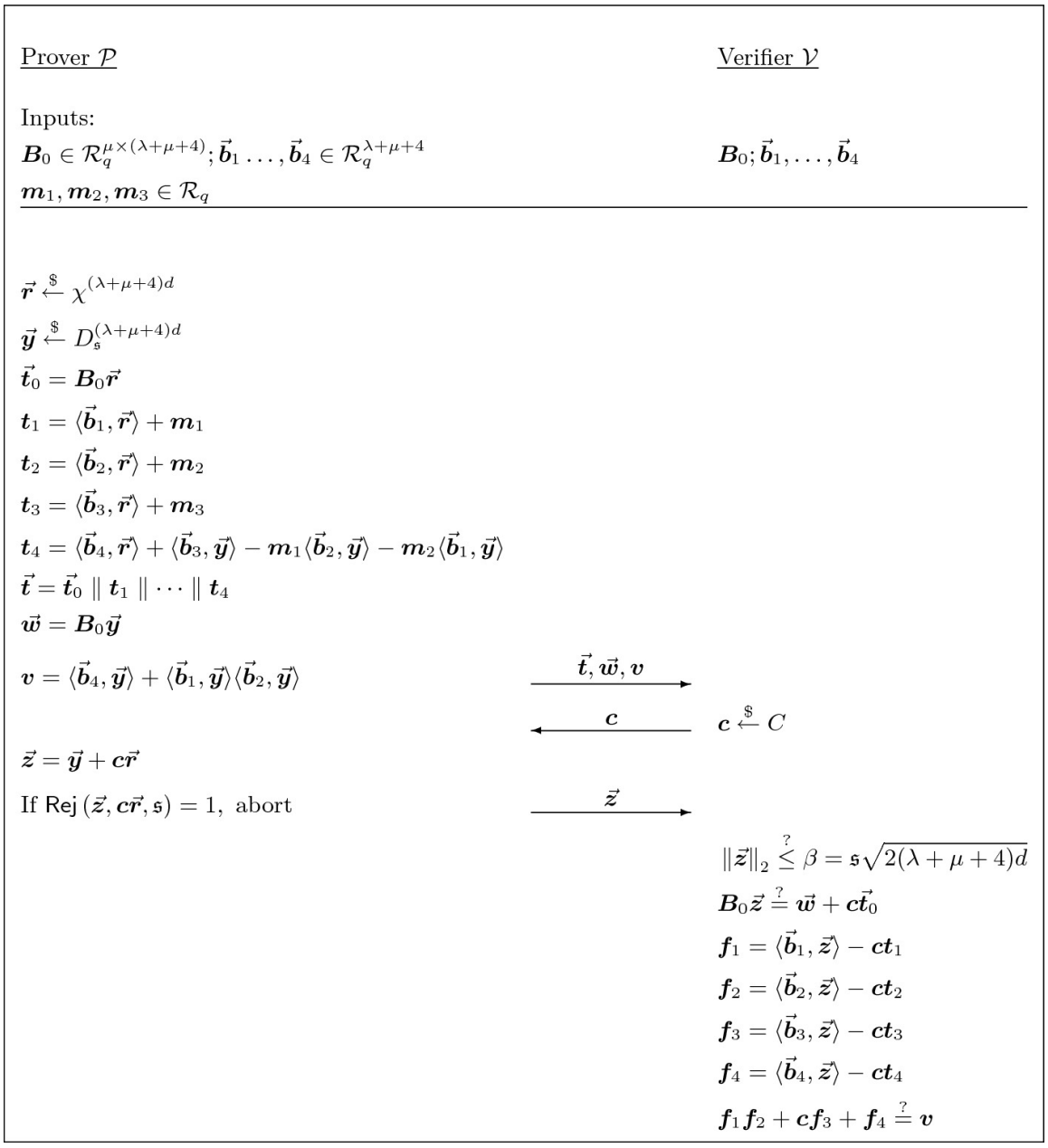

Figure 3.6.: Protocol for proof Π_{Product}, used for proof Π_{Binary}

The proof Π_{Product} [5] shows that $m_1 \cdot m_2 = m_3$ on respective BDLOP commitments which we use to construct the proof Π_{Binary} that proves that $v \cdot (v - 1_p) = 0_p$ to show that the vote v only contains the values 0 and 1 in each of its slots. Note that we use our commitments in a ring R_t instead of R_q like shown in the protocol.

3.4.3. Vote Sum

In addition to the property that votes are *binary*, i.e. all candidates either get zero or one votes, we also need to show that the total number of votes cast does not exceed an election specific limit τ. For example, in an election with $\tau = 3$, a voter can cast votes for three candidates (or less), but not for all seven candidates. In case a voter wants to cast less votes than τ, they encode all votes they don't want to cast in the placeholder votes as described in section 3.1. Since votes are encoded as vectors with ω entries of values from $\mathbb{Z}_t$ which are either zero or one we can simply show that the total number of votes including placeholder votes is equal to τ by summing up all entries of the vote. We call this the *vote sum*. For $\vec{v} \in \mathbb{Z}_t^\omega$ we define the vote sum as

$$\text{voteSum}(\vec{v}) \in \mathbb{Z}_t[x]_{\leq N/\omega} \coloneqq \sum_{i=1}^{\omega} \vec{v}_i$$

Given the isomorphism of R_t and $\mathbb{Z}_t^\omega$ as described by Theroem 2 we can calculate voteSum for votes represented as polynomials $v \in R_t$ by simply calculating $\text{voteSum}(\text{decodeVote}(v))$ with decodeVote as described in Section 3.1:

$$\text{voteSum}(v \in R_t) \coloneqq \text{voteSum}(\text{decodeVote}(v) \in \mathbb{Z}_t^\omega))$$

For a vote $v \in R_t$, the corresponding commitment $c_v = \begin{bmatrix} \text{com}_1 \\ \text{com}_2 \end{bmatrix} \coloneqq \text{BDLOP.Commit}(v; r_v)$ and a vote limit τ we prove the relation

$$\mathcal{R}_{\text{voteSum}} = \left\{ (u, w) \;\middle|\; \begin{array}{l} u = (c_v, A_1, A_2, \tau), w = (v, r_v, f) : \\ \text{voteSum}(v) = \tau \wedge \text{BDLOP.Open}(c_v, v, r_v, f) = 1 \end{array} \right\}$$

by using the proof Π_{voteSum} as shown in Figure 3.7.

Depending on the election system one wants to achieve, a rangeproof for the interval $[0, \ldots, \tau]$ can be used here instead. This would allow voters to cast less votes than the maximum possible and would remove the need for placeholder options that are used with the current construction. We leave a rangeproof of this sort to future work.

Completeness A prover with witness $w = (v, r_v, f)$ can convince the verifier by running the proof as specified. Then, the verification conditions will be fulfilled as shown in the following.

Conditions (1) - (5) ensure that the masked commitment c_z has been calculated correctly. Note that the commitment $c_{\hat{v}}$ can be calculated by the verifier themselves since $c_{\hat{v}} \coloneqq c_v - \begin{bmatrix} 0 \\ \hat{\tau} \end{bmatrix}$ and the commitments c_s and c_z are sent as part of the protocol. Using Π_{Sum} the verifier can now check that the messages in these three commitments fulfil the relation $z = 1 \cdot s + \hat{d} \cdot \hat{v}$. The completeness of the conditions (1) - (5) therefore follows from the completeness of Π_{Sum} and that Π_{Sum} accepts.

Condition (6) is also fulfilled since c_z has been calculated honestly by the prover. The opening $o = (z, r_z, 1)$ is valid since it contains the correct commitment randomness r_z and message z.

Prover $\mathcal{P}$		Verifier $\mathcal{V}$
$\hat{\tau} \coloneqq \text{encodeVote}((\tau, 0, \ldots, 0)^\intercal \in \mathbb{Z}_t^\omega)$		
$\hat{v} \coloneqq v - \hat{\tau} \in R_t, c_{\hat{v}} \coloneqq c_v - \begin{bmatrix} 0 \\ \hat{\tau} \end{bmatrix}$		
$x \xleftarrow{\$} R_t$		
$\hat{x} \coloneqq \text{encodeVote}((\text{voteSum}(x), 0, \ldots, 0)^\intercal)$		
$s \coloneqq x - \hat{x}$		
$r_s \xleftarrow{\$} S^k_{R_t,\beta}$		
$c_s \coloneqq \text{BDLOP.Commit}(s, A_1, A_2; r_s)$	$\xrightarrow{c_s}$	
	$\xleftarrow{\hat{d}}$	$\hat{d} \xleftarrow{\$} \mathbb{Z}_t$
$z \coloneqq s + \hat{d}\hat{v}$		
$r_z \xleftarrow{\$} S^k_{R_t,\beta}$		
$c_z \coloneqq \text{BDLOP.Commit}(z, A_1, A_2; r_z)$		
$o \coloneqq (z, r_z, f \coloneqq 1)$	$\xrightarrow{c_z, o = (z, r_z, f)}$	
Instantiate Π_{Sum} with		Instantiate Π_{Sum} with
$\alpha_1 \coloneqq 1, \alpha_2 \coloneqq \hat{d}, c \coloneqq c_s, c' \coloneqq c_{\hat{v}}, c'' \coloneqq c_z$		$\alpha_1 \coloneqq 1, \alpha_2 \coloneqq \hat{d}, c \coloneqq c_s,$
$w = (r \coloneqq r_s, r' \coloneqq r_v, r'' \coloneqq r_z, x \coloneqq s,$		$c' \coloneqq c_{\hat{v}}, c'' \coloneqq c_z$
$x' \coloneqq \hat{v}, f \coloneqq 1, f' \coloneqq 1, f'' \coloneqq 1)$		
$\ldots$	$\xrightarrow{\Pi_{\text{Sum}}}$	
	$\xleftarrow{\quad\quad}$	$\ldots$
$\ldots$	$\xrightarrow{\quad\quad}$	
		Accept iff
		(1) - (5) from Π_{Sum}
		(6): $\text{BDLOP.Open}(c_z, o) \stackrel{?}{=} 1$
		(7): $\text{voteSum}(z) \stackrel{?}{=} 0$

Figure 3.7.: Protocol for proof Π_{voteSum}

The proof Π_{voteSum} shows that $\text{voteSum}(v) = \tau$ for a given vote limit τ. The general idea is to mask the commitment to v into another commitment z, which can then be opened. The linear relation between z and v is shown by setting $\hat{\tau} = \text{encodeVote}((\tau, 0, \ldots, 0)^\intercal \in \mathbb{Z}_t^\omega)$, $\hat{v} \coloneqq v - \hat{\tau}$ and using the proof Π_{Sum} to show that the commitments $c_z, c_s, c_{\hat{v}}$ fulfil the relation $z = 1 \cdot s + \hat{d} \cdot \hat{v}$

Lastly, note that iff $\text{voteSum}(v) = \tau$ it holds that for $\hat{v} = v - \hat{\tau}$, $\text{voteSum}(\hat{v}) = 0$. Additionally, s has been chosen with $\text{voteSum}(s) = 0$. It results that $\text{voteSum}(z) = \text{voteSum}(s + \hat{d}\hat{v}) = \text{voteSum}(s) + \text{voteSum}(\hat{d} \cdot \hat{v}) = \text{voteSum}(s) + \hat{d} \cdot \text{voteSum}(\hat{v}) = 0 + \hat{d} \cdot 0 = 0$. Therefore conditions (7) is fulfilled.

Since a prover with witness $w = (v, r_v, f)$ can convince the verifier by fulfilling all conditions, the protocol is *complete*. □

Soundness We can give a direct argument for the soundness of the protocol. The prover has to commit to a value s in a commitment c_s before receiving the challenge $\hat{d}$. Since the BDLOP commitment is binding, given the commitments $c_{\hat{v}}$ and c_s we can assume that the values $\hat{v}$ and s are fixed. Then it holds that if $\text{voteSum}(\hat{v}) \neq 0$ or $\text{voteSum}(s) \neq 0$ there exists only one $\hat{d}$ such that $\text{voteSum}(z) = 0$. When the soundness of Π_{Sum} is not broken, then $z = s + \hat{d}\hat{v}$. In this case, an attack can only convince the verifier with probability $1/t$.

Honest Verifier Zero Knowledge To simulate an accepting transcript for a given commitment c, draw $\hat{d} \xleftarrow{\$} \mathbb{Z}_t$, $r_z \xleftarrow{\$} S^k_{R_t,\beta}$ and $\vec{z} \xleftarrow{\$} \left\{x \in \mathbb{Z}_t^\omega \mid \text{voteSum}(x) = 0\right\}$ and set $z \coloneqq \text{encodeVote}(\vec{z})$ Then calculate $c_z = \text{BDLOP.Commit}(z, A_1, A_2; r_z)$ and set $o = (z, r_z, 1)$. Note that these values fulfil conditions (6),(7). Additionally, draw $s \xleftarrow{\$} R_t, r_s \xleftarrow{\$} S^k_{R_t,\beta}$ and calculate $c_s = \text{BDLOP.Commit}(s, A_1, A_2; r_s)$.

Next, simulate Π_{Sum} such that conditions (1)-(5) are fulfilled for the commitments $c_z, c_s, c_{\hat{v}}$ with $c_{\hat{v}} \coloneqq c - \begin{bmatrix} 0 \\ \hat{\tau} \end{bmatrix}$ and $\alpha_1 = 1, \alpha_2 = \hat{d}$. For the simulability of Π_{Sum} see [7, Section 4.4]

Since its possible to simulate a transcript for Π_{CoSum}, the proof is *honest verifier zero knowledge*. □

4. Construction

Using the proofs in the preceding chapters, we can construct a post-quantum secure variant of ElectionGuard. We first go over how to construct a voting system from the aforementioned building blocks and then describe the parameter selection for exemplary instantiations. Finally, we give a summary of the security of the resulting election scheme.

4.1. Post-Quantum Secure ElectionGuard

To prepare an election with our version of ElectionGuard, the election authority together with the trustees generates the public parameters as well as the secret key for the BGV encryption scheme. This can either be done in a key generation ceremony or, potentially, in a distributed manner with appropriate protocols, which we however leave for future work.

After an election has been set up, voters can cast their votes. This is where we focused our work since this is the centerpiece of the election. See Figure 4.1 for a visualisation of the basic principle our construction implements.

To cast a vote v, voters generate a pair of a ciphertext c and a commitment com. These messages need to fulfil three requirements:

1. the ciphertext and the commitment need to contain the same vote
2. the commitment needs to contain a vote that only consists of the values 0 or 1
3. the commitment needs to contain a vote that cast votes for a maximum of τ candidates

The first requirement is proven by using $\Pi_{\text{ComEqCiph}}$ showing that the ciphertext and the commitment contain the same vote and are well-formed. The second requirement is ensured by including a proof Π_{Binary} to show that all candidates receive a maximum of 1 and a minimum of 0 votes. Lastly, the third requirement is proven by using Π_{voteSum}, showing that the total number of votes including dummy votes is τ.

Both Π_{Binary} and Π_{voteSum} show their respective property for the message contained in the commitment, together ensuring that the vote contained is well-formed. Then the proof $\Pi_{\text{ComEqCiph}}$ ensures that the vote contained in the ciphertext is well-formed, which will later be used for calculation of the election tally. Therefore, these three proofs together are sufficient to guarantee that the vote is well-formed. All three proofs are transformed into non-interactive zero knowledge proofs using the Fiat-Shamir heuristic for the quantum random oracle model (See Section 2.1). This corresponds to the current, non-quantum version of ElectionGuard where the voter proofs that their vote is well-formed in a similar fashion with proofs for the exponential ElGamal encryption [10, 3.3. Ballot Encryption].

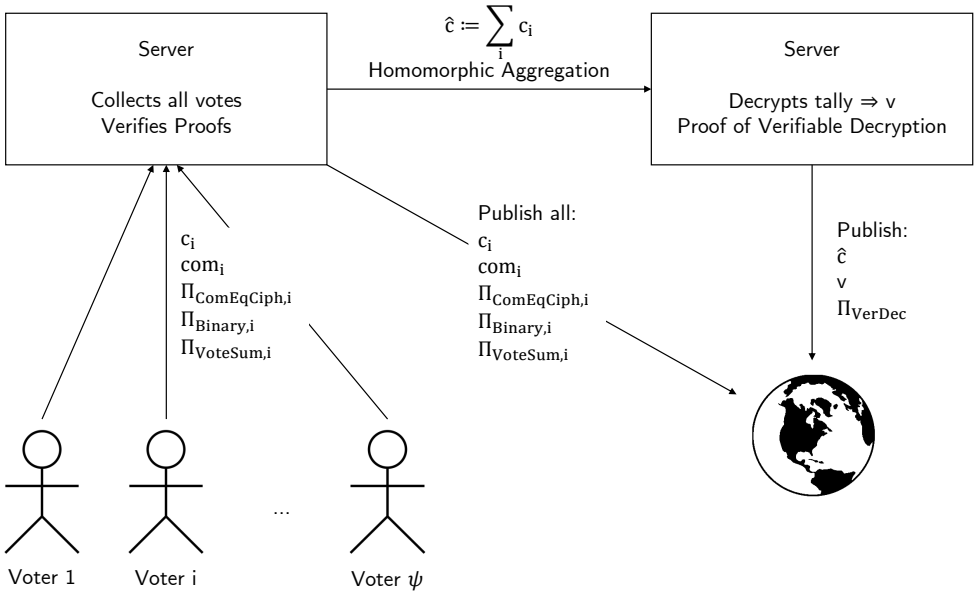

Figure 4.1.: Construction Overview

The basic principle of our construction, which is equal to the principle of ElectionGuard.

All the ciphertexts and commitments along their proofs of correctness are then published for later verification. Additionally, the aspects of ElectionGuard relating to end-to-end verifiability can be add here, i.e. confirmation codes as well as ballot casting or spoiling [10].

After the voters have cast their votes, the election authority can add all votes together using the additive homomorphic property of BGV. This does not require another zero knowledge proof since this calculation can be performed by anyone with access to the ciphertexts of all votes. The result of the election is now encrypted in the sum of all ciphertexts. This is again very similar to ElectionGuard in its current, non-quantum version [10, 3.4 Ballot Aggregation].

Finally, the result of the election can be determined by decrypting the sum calculated in the step before. This is done using verifiable decryption with proof Π_{VerDec} showing that the decryption is in fact correct. This operation can again be done in a form of decryption party in the presence of the trustees, or with a distributed proof of verifiable decryption that could be the result of further work. The proof is run in a non-interactive fashion as before and published so that anyone can verify it. Again ElectionGuard works in a comparable fashion in this step in its current, non-quantum specification [10, 3.4 Verifiable Decryption]

Optionally before decryption, the ciphertext that is the result of the homomorphic sum of all votes could be multiplied by a constant that is one in all candidate slots and zero in all placeholder slots. This would hide how many votes where cast into the placeholder slots and only reveal the votes cast for actual candidates. This approach would therefore prevent some leakage. However, we omit this step here to keep a tight bound on the ℓ_∞-norm

of the ciphertext and to keep the BGV scheme simple by not using any multiplication operations. Future work could however implement this feature.

To conclude the election, the tally alongside all votes and all proofs can be published, constituting the full *election record* of our post-quantum secure ElectionGuard variant.

4.2. Parameter Selection

We give two sets of parameters that are selected to work for two exemplary, real elections. Additionally, we state important constraints that the parameters have to follow and that are used to derive the choice of parameters. The exemplary elections have been picked to represent different, potential use-cases for the results of our work.

The first exemplary election is the German Sozialwahl 2023[1] where a total of 52 million voters could cast their vote for one of 13 electoral lists. Every voter could only cast a single vote.

The second exemplary elections are the "Elections of the Student Body 2023"[2] for the student parliament at Karlsruhe Institute of Technology (KIT) where about 21 thousand students were called to cast five votes each between a total of 54 candidates. Additionally, a second vote had to be cast for one out of eight electoral lists, which we will however omit for simplicity in our further inspection of this election.

4.2.1. Parameter constraints

One of the most important constraints in our construction is ensuring that $x^N + 1$ splits into ω factors (i.e. candidate slots) as described in Theorem 1 and 2. This requires that $\frac{1}{\sqrt{\omega}} \cdot t^{1/\omega} > 2$ and that $t = 2\omega + 1 \bmod 4\omega$ with $N \geq \omega > 1$ being powers of 2. We use this for embedding votes into the ring R_t in the commitments.

Since $q \gg t$ this also means that $\frac{1}{\sqrt{\omega}} \cdot t^{1/\omega} > 2 \Rightarrow \frac{1}{\sqrt{\omega}} \cdot q^{1/\omega} > 2$ and subsequently the challenges d are invertible in R_t and R_q. We set q and t to be co-prime to each other, i.e. $\gcd(q, t) = 1$.

When choosing κ i.e. the ℓ_1-norm of elements in the challenge space $C_{R_t,\kappa}$ we aim to have a challenge space of sufficient size to get appropriate proof soundness. We can achieve this by ensuring $\binom{N}{\kappa} \cdot 2^\kappa > 2^\lambda$ for security parameter λ as described in section 2.2.

For the different normal distributions we need to choose a total of four standard deviations: σ_χ for the error distribution of BGV, σ_C for masking the commitment randomness of the BDLOP commitments, σ_e for masking the BGV errors and finally $\sigma_{\hat{e}}$ for masking the aggregated noise of BGV ciphertexts.

We set σ_χ according to the homomorphic encryption standard [2, Section 2.1.5] to the value $\sigma_\chi = \frac{8}{\sqrt{2\pi}} \approx 3.2$. Accordingly, we set σ_C like suggested in the paper on BDLOP commitments [7, Table 1] to $\sigma_C = 11 \cdot \kappa \cdot \beta \cdot \sqrt{kN}$.

To find a value for σ_e, we need to give a bound on the ℓ_2-norm of errors sampled from χ, which is $\forall e \leftarrow \chi : ||e||_2 \leq 2\sigma_\chi\sqrt{N}$ according to the tail-bound from Lemma 1.

[1] See `https://www.sozialwahl.de/english`, Date accessed: 28.11.2023

[2] See `https://wahl.asta.kit.edu/en.html`, Date accessed: 28.11.2023

To use the indistinguishably between normal distributions from Lemma 2 we need the standard deviation of the normal distribution we want to hide our value in to be $\sigma_e \in \omega(T\sqrt{\log(N)})$. We achieve this by setting $T = 2\sigma_\chi\sqrt{N}$ such that $||e||_2 \leq T$ and it follows that $\sigma_e \in \omega(2\sigma_\chi\sqrt{N\log(N)})$. With $M \approx 3$, $M = \exp(12/\alpha + 1/(2\alpha))$ and $\sigma_e = \alpha T$ we get $\alpha \approx 11$ from Lemma 2 and can subsequently set $\sigma_e \coloneqq 11 \cdot 2\sigma_\chi\sqrt{N}$.

A similar approach can be taken to set a value for σ_m. By Lemma 2 we need to set the standard deviation $\sigma_m \in \omega\left(\sqrt{N}t\sqrt{\log(N)}\right)$ for $||m||_2 \leq \sqrt{N}t \coloneqq T$. We achieve this by setting $\sigma_m = \alpha T$ and with a value of 11 for α we get $\sigma_m \coloneqq 11 \cdot \sqrt{N}t$

To guarantee the completeness of the proof Π_{VerDec}, we need to set our parameters such that $2\psi\sigma_e(2N + \sqrt{N}) \leq q/2$ and $2B_{\hat{e}} \leq 4\psi\sigma_e N^2(2\sqrt{N}+1) < (q/2t)$ with $B_{\hat{e}} = \sqrt{2N}\sigma_{\hat{e}}$ (see Section 3.3.2), where ψ is the number of ciphertexts that have been summed up homomorphically, meaning we can set $\psi \coloneqq$ #Voters to the total number of eligible voters.

We check that the $\text{SKS}^2_{n,k,\beta}$-problem is hard using Lemma 3 with $\beta < t^{1/\omega}$, and $\beta < \sqrt{\frac{N}{2\pi e}} \cdot t^{n/k} \cdot 2^{-128/(k\cdot N)} - \sqrt{N}/2$, which makes the BDLOP commitments statistically binding and computationally hiding in our case. Alternatively, one could set the parameters to achieve a "optimal" setting as suggested in the paper on BDLOP [7, 4.3 Instantiations] and aim for an equal level of computational hardness for the hiding and binding property.

4.2.2. Parameter Selection

We provide an exemplary parameter selection in Table 4.1 for the German Sozialwahl 2023 and the Elections of the Student Body 2023 at KIT. The parameters have been calculated to respect the constrains described in Section 4.2.1.

Note that especially the Student Body election has lots of candidates (54) and hence needs a lot of candidate slots (64). This results in a big plaintext modulus t and even bigger ciphertext modulus q, however these values are still feasible since the resource heavy calculation of each voters ballot and the accommodating proofs of correctness is distributed to each voters device, while the vote server only needs to form the homomorphic sum and the proof of verifiable decryption. Thus, even parameters that might be unusual in other settings can be realistic to use in our setting.

We claim more than 128 bits security against a quantum adversary for the parameters given using the homomorphic encryption standard [2, Table 2] and its results using the LWE estimator by Albrecht et al [3] with the BKZ.qsieve cost model.

	Parameter explanation	**Sozialwahl**	**Student Body Election**
#Voters	Upper limit on number of voters	$\sim$ 52 Mil.	$\sim$ 21k
ψ	Limit of number of voters we used to calculate bound for accumulated noise $\hat{e}$	$\sim$ 52 Mil.	$\sim$ 21k
#Cand.	Number of candidates	13	54
τ	Number of votes a single voter can cast	1	5
ω	Number of factors of $x^N + 1$, i.e. number of candidate slots	16	64
N	Ring dimension of R_q and R_t	8192	16384
t	Plaintext modulus defining R_t with $t = 2\,\omega + 1 \mod 4\,\omega$	$\approx 2^{48}$	$\approx 2^{256}$
q	Ciphertext modulus defining R_q with $q = 2\,\omega + 1 \mod 4\,\omega$	$\approx 2^{123}$	$\approx 2^{323}$
κ	ℓ_1-norm of elements in the challenge space $C_{R_t,\kappa}$	24	22
β	ℓ_∞-norm of "small" elements in $S_{R_t,\beta}, S_{R_q,\beta}$	1	1
k	Width (over R_t) of the commitment matrices	3	3
n	Height (over R_t) of the commitment matrix A_1	1	1
l	Dimension of messages embedded into BDLOP commitments	1	1
σ_χ	Standard deviation used to generate noise during BGV encryption	$\frac{8}{\sqrt{2\pi}} \approx 3.2$	$\frac{8}{\sqrt{2\pi}} \approx 3.2$
σ_C	Standard deviation used to mask commitment randomness	≈ 41400	≈ 53700
σ_m	Standard deviation used to mask messages	$\approx 2^{58}$	$\approx 2^{266.5}$
σ_e	Standard deviation used to mask noise	≈ 6360	≈ 9000
$\sigma_{\hat{e}}$	Standard deviation used to mask accumulated noise $\hat{e}$	$\approx 2^{66}$	$\approx 2^{57.5}$
$B_{\hat{e}}$	Verification bound on accumulated noise $\hat{e}$	$\approx 2^{73}$	$\approx 2^{65}$

Table 4.1.: Parameter settings for two exemplary elections

The exemplary elections are chosen to represent different use-cases of ElectionGuard and use the amount of voters and candidates from the respective election 2023. The values of all parameters have been calculated in accordance to the constrains described in section 4.2.1.

4.3. Compatibility with ElectionGuard

Our construction has been built in a way to offer the largest possible compatibility with ElectionGuard. In this work we have described the centerpiece of election guard, i.e. the ballot creation and aggregation, the homomorphic tally and the verifiable decryption. In its current form we rely on a key generation ceremony for setting up an election.

To create a full, post-quantum version of ElectionGuard some further work is necessary. First, the key generation could be implemented in a distributed manner using a threshold scheme. Additionally, the verifiable decryption algorithm could be transformed into a distributed variant where all guardians (trustees) calculate their partial decryption and proof and then combine their results. Refer to the work by Bernabé Rodríguez as well as Ghanem and Moursy [11, 14] for pointers how this could be implemented.

Furthermore, the end-to-end verifiability components of ElectionGuard need to be added back to the construction. This especially concerns the generation of confirmation codes, which we have omitted here for simplicity. Additionally, a voting system implementing post-quantum ElectionGuard needs to implement ballot casting or spoiling.

4.4. Security Properties

E-Voting schemes can be described with a number of properties to evaluate their functionality and security. We now want to mention some relevant aspects and analyse whether they apply to our construction and to what degree. The definitions of the functional and security requirements written in italics are taken word for word from the review of Cryptographic Electronic Voting by Kho, Heng and Chin [17]. Since ElectionGuard is not a full e-voting scheme on its own but instead aims to be a toolkit to enhance other election schemes with, some of the properties depend heavily on the concrete context they are used in. We want to give an impression which properties can be achieved in which settings and therefore describe some of the important properties.

Robustness *"Any dishonest party cannot disrupt elections."*

The degree to which this property is fulfilled depends on how the system is implemented. For example, attacks on availability like DDOS-attacks could disrupt elections. The zero knowledge proofs we use at least guarantee that votes cast are valid and therefore prevent manipulation of the tally.

Fairness *"No partial tally is revealed."*

This property is fulfilled by our scheme since only the final tally is decrypted and revealed.

Verifiability *"The election results cannot be falsified. There are two types of verifiability: 1) Individual verifiability: The voter can verify whether their vote is included in the final tally. 2) Universal verifiability: All valid votes are included in the final tally and this is publicly verifiable."*

The verifiability of our scheme depends on the additional implementation of confirmation codes and the ballot casting or spoiling principle. Using these two additions however, our scheme fulfils both types of verifiability.

Transparency *"Maximise transparency in the vote casting, vote storing and vote counting process while preserving the secrecy of the ballots."*

We achieve transparency by publishing all votes alongside their proofs of correctness and a proof of verifiable decryption after homomorphic aggregation of all cast votes. This way anyone can reproduce the verification of the election result.

Practicality *"The implementation of requirements and assumptions should be able to adapt to large-scale elections."*

As the two exemplary sets of parameters have shown we have provided a practical voting scheme that can even be used for large-scale elections like the German Sozialwahl.

Privacy and vote secrecy *"The cast votes are anonymous to any party."*

This property is fulfilled by our scheme since the votes are stored in ciphertexts and hiding commitments and only the final tally is decrypted, hence the individual votes are anonymous. This property especially benefits from the addition of post-quantum cryptography to even keep the voters choices secret when the traditional hardness assumption like DLog or prime factorisation do not hold anymore.

Coercion-resistance *"Coercers cannot insist that voters vote in a certain way and the voter cannot prove his vote to the information buyer."*

This is an important property, however the degree to which this is fulfilled depends on the concrete instantiation of our scheme. To make the scheme coercion-resistant, one could for example include the concept of revoting into post-quantum ElectionGuard.

Authentication *"Only eligible voters were allowed to vote."*

The question on how to ensure that only eligible voters are allowed to cast votes and unqualified voters are not is not at the core of our work and we therefore leave it unanswered here. This is something a concrete implementation ouf our work needs to concern itself with.

5. Conclusion

This research aimed to create a post-quantum counterpart to the logic behind ElectionGuard, a toolkit for end-to-end verifiable voting systems. We have identified this as an issue since the current version of ElectionGuard relies on exponential ElGamal and its underlying security assumption of the discrete logarithm problem being hard does not hold with sufficient developments in quantum computers. This is a severe problem since even elections held today could be influenced by the threat of later vote decryption. Hence, our goal was to transfer the main parts of ElectionGuard, i.e. the vote encryption, proofs of correctness, vote aggregation and verifiable decryption, to a post-quantum setting.

We have first detailed our mathematical setting and described the BGV encryption scheme and the BDLOP commitment scheme. To construct a post-quantum version of ElectionGuard, we follow a dual approach using both a ciphertext and a commitment simultaneously, first showing the equality of their content using our proof $\Pi_{\text{ComEqCiph}}$. Then we apply the proofs Π_{Binary} and Π_{VoteSum} to show that cast votes in the commitments are well-formed, i.e. only consists of zeros and ones and are not exceeding the election limit. Then we perform the vote aggregation using the homomorphic property of the ciphertexts. Finally, we run a verifiable decryption using the proof Π_{VerDec} and publish the tally, concluding the election.

With our work, we have laid the foundation to create a post-quantum version of ElectionGuard. We have presented the key components to run elections. Further work needs to additionally investigate and include the end-to-end verifiability components as well as a distributed key generation and multi-party variant of our verifiable decryption using a threshold approach. Our work shows that a post-quantum approach to ElectionGuard is feasible and worth pursuing further.

Bibliography

[1] Ben Adida. "Helios: Web-based Open-Audit Voting." In: *USENIX security symposium*. Vol. 17. 2008, pp. 335–348.

[2] Martin Albrecht et al. *Homomorphic Encryption Standard*. Cryptology ePrint Archive, Paper 2019/939. 2019. URL: https://eprint.iacr.org/2019/939.

[3] Martin R Albrecht, Rachel Player, and Sam Scott. "On the concrete hardness of learning with errors". In: *Journal of Mathematical Cryptology* 9.3 (2015), pp. 169–203.

[4] Diego F. Aranha et al. "Lattice-Based Proof of Shuffle and Applications to Electronic Voting". In: *Topics in Cryptology – CT-RSA 2021*. Ed. by Kenneth G. Paterson. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 227–251. ISBN: 978-3-030-75539-3. DOI: 10.1007/978-3-030-75539-3_10.

[5] Thomas Attema, Vadim Lyubashevsky, and Gregor Seiler. "Practical Product Proofs for Lattice Commitments". In: *Advances in Cryptology – CRYPTO 2020*. Ed. by Daniele Micciancio and Thomas Ristenpart. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 470–499. ISBN: 978-3-030-56880-1. DOI: 10.1007/978-3-030-56880-1_17.

[6] Wojciech Banaszczyk. "New Bounds in Some Transference Theorems in the Geometry of Numbers". In: *Mathematische Annalen* 296 (1993), pp. 625–635.

[7] Carsten Baum et al. "More Efficient Commitments from Structured Lattice Assumptions". In: *Security and Cryptography for Networks*. Ed. by Dario Catalano and Roberto De Prisco. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 368–385. ISBN: 978-3-319-98113-0. DOI: 10.1007/978-3-319-98113-0_20.

[8] Carsten Baum et al. "Sub-linear Lattice-Based Zero-Knowledge Arguments for Arithmetic Circuits". In: *Advances in Cryptology – CRYPTO 2018*. Ed. by Hovav Shacham and Alexandra Boldyreva. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 669–699. ISBN: 978-3-319-96881-0. DOI: 10.1007/978-3-319-96881-0_23.

[9] Josh Benaloh. "Verifiable Secret-Ballot Elections". PhD thesis. Sept. 1987. URL: https://www.microsoft.com/en-us/research/publication/verifiable-secret-ballot-elections/.

[10] Josh Benaloh and Michael Naehrig. *ElectionGuard Specification v1.1*. 2022. URL: https://github.com/microsoft/electionguard/releases/download/v1.1/EG_spec_v1_1.pdf (visited on 12/08/2023).

[11] Julen Bernabé Rodríguez. *Non-linear operations and verifiable decryption in BGV*. 2023. URL: http://hdl.handle.net/10609/148192.

[12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) Fully Homomorphic Encryption without Bootstrapping". In: *ACM Transactions on Computation Theory* 6.3 (July 2014), pp. 1–36. ISSN: 1942-3454, 1942-3462. DOI: 10.1145/2633600. URL: https://dl.acm.org/doi/10.1145/2633600 (visited on 08/13/2023).

[13] Jelle Don et al. *Security of the Fiat-Shamir Transformation in the Quantum Random-Oracle Model*. Cryptology ePrint Archive, Paper 2019/190. 2019. URL: https://eprint.iacr.org/2019/190.

[14] Sahar M. Ghanem and Islam A. Moursy. "Secure Multiparty Computation via Homomorphic Encryption Library". In: *2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS)*. 2019, pp. 227–232. DOI: 10.1109/ICICIS46948.2019.9014698.

[15] Oded Goldreich and Yair Oren. "Definitions and properties of zero-knowledge proof systems". In: *Journal of Cryptology* 7.1 (1994), pp. 1–32.

[16] *Introduction to the BGV Encryption Scheme*. URL: https://www.inferati.com/blog/fhe-schemes-bgv (visited on 08/13/2023).

[17] Yun-Xing Kho, Swee-Huay Heng, and Ji-Jian Chin. "A review of cryptographic electronic voting". In: *Symmetry* 14.5 (2022), p. 858.

[18] Qipeng Liu and Mark Zhandry. *Revisiting Post-Quantum Fiat-Shamir*. Cryptology ePrint Archive, Paper 2019/262. 2019. URL: https://eprint.iacr.org/2019/262.

[19] Vadim Lyubashevsky. "Lattice Signatures without Trapdoors". In: *Advances in Cryptology – EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 738–755. ISBN: 978-3-642-29011-4. DOI: 10.1007/978-3-642-29011-4_43.

[20] Vadim Lyubashevsky and Gregor Seiler. *Short, Invertible Elements in Partially Splitting Cyclotomic Rings and Applications to Lattice-Based Zero-Knowledge Proofs*. Cryptology ePrint Archive, Paper 2017/523. 2017. URL: https://eprint.iacr.org/2017/523.

[21] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (1997), pp. 1484–1509. DOI: 10.1137/S0097539795293172.

[22] Tjerand Silde. "Short Paper: Verifiable Decryption for BGV". In: *Financial Cryptography and Data Security. FC 2022 International Workshops*. Ed. by Shin'ichiro Matsuo et al. Cham: Springer International Publishing, 2023, pp. 381–390. ISBN: 978-3-031-32415-4.

A. Appendix

A.1. SageMath Code for Vote Encoding

```
F = GF(97)
P = PolynomialRing(F, 'x');
x = P.gen()
mod = x^1024 + 1
factors = mod.factor()[:]
factors = list(zip(*factors))[0]
k = len(factors)
print(str(mod) + " factors into " + str(k) + " factors")
print("Factors: " + str(list(factors)))

# calculate base e
e = [1 for i in range(k)]
for i in range(k):
    A = list(factors)
    B = list()
    A.remove(factors[i])
    B.append(factors[i])
    while len(A) >= 1:
        c = A.pop()
        e[i] = e[i] - e[i] * inverse_mod(prod(B) , c) * prod(B)
        B.append(c)
print("Base: " + str(e))

# example values calculation
a = [5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2]
b = [3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4]

# encode
a = sum([a*b for a,b in zip(a,e)])
b = sum([a*b for a,b in zip(b,e)])

# example calculation
c = a * b % mod

# decode
res = [c % factors[i] for i in range(k)]
print(res)
```