

From ElGamal to Lattice-Based: Enhancing Helios Voting with Quantum-Safe Cryptography

João Pedro Cardoso Barbosa^[0009–0005–9055–6073], Gustavo Zambonin^[0000–0002–6962–9347], Thaís Bardini Idalino^[0000–0002–8836–1104], and Ricardo Felipe Custódio^[0000–0001–9611–5694]

Departamento de Informática e Estatística, Universidade Federal de Santa Catarina
Florianópolis, 88040-900, Brazil
{joao.c.barbosa,gustavo.zambronin}@posgrad.ufsc.br
{thais.bardini,ricardo.custodio}@ufsc.br

Abstract. While digitalizing voting processes is convenient for both the voter and the entities responsible for counting the votes, it introduces a high degree of complexity in balancing extremely high privacy requirements and the need for verifiability. Cryptography is a fundamental instrument for achieving these properties. Nevertheless, the advent of quantum computing compromises the security of classical algorithms, requiring new voting schemes based on secure problems against this type of adversary. The literature offers proposals for quantum-safe voting schemes that are largely theoretical, creating a critical gap between those schemes and real-world applications that hinders their practical adoption. In this work, we bridge this gap by implementing the quantum-safe voting protocol due to Aranha et al. (CT-RSA '21). We also instantiate and sketch a proof of the sum of lattice-based commitments to achieve our goal. To prove the viability of our implementation, we integrate the protocol into the widely used Helios Voting, enhancing it with complex ballots, return codes, and quantum security. This constitutes a step towards a voter-convenient system that achieves integrity and preserves privacy of voters in the face of emerging quantum threats.

Keywords: Electronic voting · Implementation · Commitment schemes · Post-quantum cryptography · Helios Voting.

1 Introduction

Historically, elections have been a mechanism for endorsing the power of choice of a certain group. Considering the digital landscape, modernizing the classical voting process may seem like a logical step toward bringing convenience and scalability to a procedure that demands both time and effort. However, voting is a distinct task, where the main challenge is to find a balance between verifiability and privacy when these properties may antagonize each other. The underlying challenge of these protocols is the computation of the tally without breaking privacy of voters while being able to prove that all votes were correctly counted.

To solve this problem, cryptography is essential for threading this fine balance, providing three main frameworks: (i) *homomorphic tallying*, where all the encrypted ballots are added and then deciphered so no single vote is revealed; (ii) *blind signatures*, where the ballots are anonymized before being sent; and (iii) *mix-nets*, where the ciphers are permuted in a way that there is no correlation between an encrypted ballot and the vote contained within. Primitives such as proofs of knowledge, commitment schemes, and homomorphism are the building blocks for the secure multi-party communication that characterizes e-voting.

Practical electronic voting systems, such as Helios [1], and those historically used in Norway [10] and Estonia [14], depend on such underlying primitives. Particularly, their security relies on the hardness of number-theoretic problems, such as the discrete logarithm or large integer factorization. However, considering a sufficiently powerful quantum computer, the security of the aforementioned systems is compromised by Shor’s algorithm [18], endangering the privacy of voters and system verifiability.

Fortunately, there has been progress regarding quantum-safe e-voting [3, 5, 7, 16], which mostly rely on lattice-based primitives. Still, we argue that little effort has been made to implement them in their entirety and integrate them into a setting or system that approaches real-world use. Next, we discuss the current literature on quantum-safe e-voting and then detail our contributions to address the aforementioned issues regarding the practicality of such systems.

1.1 Related Work

Classical voting systems like the construction by Cramer et al. [8] established important foundations for secure electronic voting. This multi-authority scheme uses proofs of knowledge for equality of discrete logarithms combined with Shamir’s (t, n) -threshold cryptosystem [17] over ElGamal encryption [9]. The protocol achieves universal verifiability, which allows any observer to verify the election outcome independently, through a public *bulletin board* that displays encrypted ballots, while preserving voter privacy via homomorphic tallying. The widely-used Helios system [1] implements this approach for vote computation, but like other classical schemes, it remains vulnerable to quantum attacks.

Concerning quantum-safe voting schemes, Chillotti et al. [7] propose a voting scheme based on fully homomorphic encryption to count the ballots, which does not require knowledge proofs. The contribution allows for 1-out-of- n elections, and even though no implementation or choice of parameters was provided, the chosen primitives can serve as a basis to infer that the construction is impractical, as pointed out by several authors [3, 5, 16].

An alternative method is proposed by del Pino et al. [16]. They present a voting scheme for simple yes/no or 1-out-of- n elections using homomorphic tallying. However, the scheme does not provide sufficient verifiability for practical use. Building on this work, Boyen et al. [5] introduces a protocol that achieves universal verifiability with comparable performance and similar ballot structure. However, the authors of both proposals do not disclose the level of detail about complete implementations, focusing on prototypes of key components.

A scheme that uses a different approach is the construction by Aranha et al. [3]. This mix-net-based system relies on a verifiable shuffle of known values over Baum et al. [4] commitments to permute and verify the ballots. The authors propose a verifiable variation of the encryption scheme due to Lyubashevsky and Neven [12], and return codes [10] for cast-as-intended ballot verification. To the best of our knowledge, this is the only quantum-safe voting system capable of supporting complex ballot structures necessary for diverse voting methods such as ranked-choice, approval, and single transferable vote systems.

In this scheme, votes are individually deciphered, allowing the server that shuffles the votes to break privacy of voters if the encrypted ballots are published by the *ballot box*. The authors argue that achieving both privacy and universal verifiability is challenging, and settle for a limited form of verifiability. Aranha et al. [3] implement some of the necessary primitives, such as the commitment scheme, the verifiable encryption scheme, and the proof of shuffle. However, they do not provide the implementation of the entire protocol and other required mechanisms, only describing it on paper.

In a follow-up work, Aranha et al. [2] give a new shuffle for ciphertexts without decryption to solve the privacy issues that could arise from the *shuffle server* being aware of the encrypted ballots. However, the first scheme is still faster and produces significantly smaller proofs, besides allowing complex ballots. Again, the implementation covers the cryptographic primitives, but not other practical aspects of the protocol.

1.2 Our Contribution

We observe that the proposal of Aranha et al. [3] is the closest to a practical instantiation of a quantum-safe e-voting scheme, while still allowing features such as complex ballots. Hence, we choose to extend this line of work by providing a fully featured e-voting system on top of the cryptographic primitives provided by the authors. To the best of our knowledge, our work is the first publicly available implementation of a quantum-safe e-voting protocol, available as open-source code at <https://github.com/jp-db/voting-protocol-acns> (commit e81000b).

To achieve our goal, we give an instantiation of the *proof for sum* Π_{Sum} over Baum et al. [4] commitments. The protocol is briefly mentioned by the authors, but no formal definition or proof is provided. Therefore, we formalize Π_{Sum} and implement a non-interactive version of the protocol on top of the primitives provided by Aranha et al. [3]. We expose all cryptographic building blocks, phases and algorithms of the protocol through Python bindings. We also provide serializers to convert polynomials to and from familiar data types, facilitating storage of internal election data. Our library allows integrating the protocol with pre-existent voting applications and other use cases.

To bridge the gap between theory and practice, we integrate our implementation into the established Helios voting system [1]. We leverage the primary Helios server as the *ballot box* while deploying the remaining protocol participants as independent servers. Additionally, we conduct a comparative performance and behavior analysis of our protocol and the original Helios system. We delve into

the properties gained and lost by transitioning from one system to another and show that quantum-safe e-voting is currently attainable.

2 Preliminaries

We first establish foundational concepts and notations necessary for understanding the voting protocol. We also briefly discuss the fundamentals and properties of the polynomial rings employed, and the parameters used for the cryptographic primitives of the schemes in our work.

For a distribution $\mathcal{D}$ and a finite set A , we write $d \xleftarrow{\$} \mathcal{D}$ when d is drawn according to $\mathcal{D}$ and $a \xleftarrow{\$} A$ when a is uniformly sampled from A . For any two vectors $\mathbf{v}, \mathbf{u} \in \mathbb{Z}^N$, where $N \in \mathbb{Z}$, we denote the inner product of $\langle \mathbf{v}, \mathbf{u} \rangle$ in $\mathbb{Z}^N$ as $\langle \mathbf{v}, \mathbf{u} \rangle = \sum_{i=0}^{N-1} v_i \cdot u_i$, and the ℓ_p -norm follows the definition below.

Definition 1 (ℓ_p -norm of the elements in $\mathbb{Z}^N$). *For any $p \geq 1 \in \mathbb{Z}$ and any vector $\mathbf{v} \in \mathbb{Z}^N$, the ℓ_p -norm of $\mathbf{v}$, and the special case ℓ_∞ -norm, are respectively given by the equations*

$$\|\mathbf{v}\|_p = \left(\sum_{i=0}^{N-1} |v_i|^p \right)^{1/p} \quad \text{and} \quad \|\mathbf{v}\|_\infty = \max_{i \in \{0, \dots, N-1\}} |v_i|.$$

2.1 Rings

Let $r \in \mathbb{N}^+$ and $N = 2^r$. We consider the ring $R = \mathbb{Z}[X]/\langle X^N + 1 \rangle$ and $\mathbf{I}_k \in R^{k \times k}$ denotes an identity matrix of dimension $k \times k$ over R . For any $f = \sum_i \alpha_i X^i$ and $g = \sum_i \beta_i X^i$ in R , the function $\phi: R \rightarrow \mathbb{Z}^N$ defined by $\phi(f) = (\alpha_0, \dots, \alpha_{N-1})$ converts a polynomial in R to a coefficient vector in $\mathbb{Z}^N$. Then, the ℓ_p -norm of elements in R is denoted by $\|f\|_p = \|\phi(f)\|_p$, and the inner product is denoted by $\langle f, g \rangle = \langle \phi(f), \phi(g) \rangle$.

Let $\delta \in \mathbb{N}^+$ and $p \in \mathbb{N}^+$ be a prime. Then, $R_p = R/\langle p \rangle$ is the polynomial ring modulo $X^N + 1$ with coefficients in $\{0, \dots, p-1\}$. For any $f \in R_p$, we select the coefficients in the range $[-\frac{p-1}{2}, \frac{p-1}{2}]$ and compute norms and inner products as if f was in R . We now restate a lemma that is relevant to establish properties of the cryptographic primitives used in our work.

Lemma 1 ([13], Corollary 1.2). *Let $N \geq \delta > 1$ be powers of 2 and $p \equiv 2\delta + 1 \pmod{4\delta}$ be a prime. Then, $X^N + 1$ factors into δ irreducible polynomials $X^{N/\delta} - r_i$, so that*

$$X^N + 1 \equiv \prod_{i=1}^{\delta} (X^{N/\delta} - r_i) \pmod{p}$$

for distinct $r_i \in R_p$. Additionally, any $f \in R_p \setminus \{0\}$ is invertible in R_p if it satisfies

$$\|f\|_\infty < \frac{p^{1/\delta}}{\sqrt{\delta}} \quad \text{or} \quad \|f\|_2 < p^{1/\delta}.$$

We hereafter expect that the parameters p , N and δ follow the constraints in Lemma 1. Therefore, when $p \equiv 2\delta + 1 \pmod{4\delta}$, $X^N + 1$ factors into δ irreducible polynomials and any non-zero element in R_p with sufficiently small norms ℓ_2 or ℓ_∞ is invertible. The invertibility of polynomials is essential for two reasons [4]. It allows the establishing of statistical *binding* and statistical *hiding* properties for the commitment scheme. Additionally, it enables the instantiation of a challenge space that is easily sampled from, and contains small elements where every difference of distinct elements is invertible in R_p . We recall that, considering the restrictions above, for any $f \in R_p$, if $\|f\|_\infty \leq 2$, f is invertible in R_p .

Let $\kappa \in \mathbb{Z}^+$. We define $\mathcal{C} = \{c \in R_p \mid \|c\|_\infty = 1, \|c\|_1 = \kappa\}$ as the challenge space, where $\mathcal{C} \subset R_p$ and $|\mathcal{C}| = 2^\kappa \cdot \binom{N}{\kappa}$. It consists of all elements in R_p with κ coefficients of absolute value equal to 1 and $N - \kappa$ coefficients equal to 0. Consequently, $|\mathcal{C}|$ derives from the $\binom{N}{\kappa}$ ways to choose the κ non-zero positions, and each of these positions can be either 1 or -1 . We also denote the set of differences as $\overline{\mathcal{C}} = \{c - c' \mid c, c' \in \mathcal{C}, c \neq c'\}$. Additionally, we define $\beta_\infty < p^{1/\delta}/\sqrt{\delta}$ as the ℓ_∞ -norm bound for the elements in the finite set $D_{\beta_\infty} \subset R_p$, such that $D_{\beta_\infty} = \{f \in R_p \mid \|f\|_\infty \leq \beta_\infty\}$. For a sufficiently small β_∞ , all the non-zero elements in the sets $\mathcal{C}$, $\overline{\mathcal{C}}$ and D_{β_∞} are invertible in R_p .

2.2 Discrete Gaussian

The primitives of the protocol require values sampled uniformly at random from the discrete Gaussian distribution. First, the Gaussian function over $\mathbb{R}^k$ centered on $\mathbf{v} \in \mathbb{R}^k$ with standard deviation σ is given by

$$\rho(\mathbf{x})_{\mathbf{v},\sigma}^N = \exp\left(\frac{-\|\mathbf{x} - \mathbf{v}\|_2^2}{2\sigma^2}\right).$$

The discrete Gaussian distribution is obtained by normalizing the function in R^k . For $\mathbf{x} \in R^k$ and $\rho_\sigma^{kN}(R^k) = \sum_{\mathbf{x} \in R^k} \rho_\sigma^{kN}(\mathbf{x})$, the distribution is denoted by

$$\mathcal{N}_{\mathbf{v},\sigma}^k(\mathbf{x}) = \frac{\rho_{\mathbf{v},\sigma}^{kN}(\mathbf{x})}{\rho_\sigma^{kN}(R^k)}.$$

When $\sigma = 1$ or $\mathbf{v} = 0$, the values are omitted from the notation [3].

3 Primitives

In this section, we introduce the two main cryptographic primitives of the protocol by Aranha et al. [3]. We briefly describe each primitive and its purpose. The first is the commitment scheme by Baum et al. [4] that allows for a message to be committed, only to be later revealed. Additionally, properties regarding the committed message can be verified without revealing it via zero-knowledge proofs. The second is the verifiable encryption scheme that, besides the usual encryption and decryption algorithms, allows certain properties from the plaintext to be verified with the ciphertext.

The protocol leverages the interplay between the commitment and verifiable encryption schemes. A vote, represented as a polynomial in R_p , is committed to, generating a commitment and an opening. The verifiable encryption scheme encrypts this opening and verifies its *binding* property with the commitment. Thus, both schemes operate within the same ring R_p and share parameters. Additionally, we abstract the *mix-net* as the algorithm **SHUFFLE**, which receives a collection of votes $[v_1, \dots, v_n]$, chooses a random permutation π on $\{1, \dots, n\}$ and outputs $[v_{\pi(1)}, \dots, v_{\pi(n)}]$.

3.1 Commitment Scheme

A commitment scheme is a fundamental primitive to conceal and bind the value of the vote or other polynomials. Here, we define the Baum et al. [4] commitment scheme **C** as a tuple of algorithms (**KEYGEN_C**, **COM**, **OPEN**). Let k be the width and n be the height of the commitment matrix. For the knowledge proofs, we denote the standard deviation for the commitment scheme's discrete Gaussian to be $\sigma_C = 11 \cdot \kappa \cdot \beta_\infty \cdot \sqrt{kN}$. Hereafter, we present the algorithms for the R_p message space and discuss some fundamental properties of the voting scheme.

Algorithm 1 **KEYGEN_C**

Input: security parameter $\lambda \in \mathbb{N}^+$
Output: public key $B = \begin{bmatrix} B_1 \\ b_2 \end{bmatrix}$

- 1 $B_1 = [I_n \ B'_1]$ where $B'_1 \xleftarrow{\$} R_p^{n \times (k-n)}$
- 2 $b_2 = [0^n \ 1 \ b'_2]$ where $(b'_2)^\top \xleftarrow{\$} R_p^{(k-n-1)}$
- 3 **return** $B = \begin{bmatrix} B_1 \\ b_2 \end{bmatrix}$

Algorithm 2 **COM**

Input: public key $B = \begin{bmatrix} B_1 \\ b_2 \end{bmatrix}$, message $m \in R_p$, randomness $r_m \xleftarrow{\$} D_{\beta_\infty}^k$
Output: commitment and opening pair $(\llbracket m \rrbracket, d)$

- 1 $\llbracket m \rrbracket = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = B \cdot r_m + \begin{bmatrix} 0 \\ m \end{bmatrix}$
- 2 $d = (m; r_m, 1)$
- 3 **return** $(\llbracket m \rrbracket, d)$

Algorithm 3 **OPEN**

Input: public key $B = \begin{bmatrix} B_1 \\ b_2 \end{bmatrix}$, commitment $\llbracket m \rrbracket = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$, opening tuple $(m; r_m, f)$ with $f \in \overline{\mathcal{C}}$
Output: whether $(m; r_m, f)$ is a valid opening for $\llbracket m \rrbracket$

- 1 **if not** $f \cdot \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \stackrel{?}{=} B \cdot r_m + f \cdot \begin{bmatrix} 0 \\ m \end{bmatrix}$ **then return** 0
- 2 **for** $i \leftarrow 1$ **to** k **do**
- 3 **if not** $\|r_m[i]\| \leq 4\sigma_C \sqrt{N}$ **then return** 0
- 4 **return** 1

Algorithm 1 generates a public commitment key $\mathbf{B}$ of dimension $k \times (n+1)$. To commit to a message m , Algorithm 2 uses this public key along with a randomness $\mathbf{r}_m$, producing both a commitment $\llbracket m \rrbracket$ and its corresponding opening $(m; \mathbf{r}_m, f)$. The opening information is a cryptographic proof that binds the commitment to its original message. Therefore, Algorithm 3 can be used to assert the validity of a commitment $\llbracket m \rrbracket$ for a message m by checking if $\mathbf{r}_m$ and f correctly open $\llbracket m \rrbracket$. We refer to Algorithm 2 as **COM**; to simplify, we may invoke it as

$$(\llbracket m \rrbracket, \mathbf{r}_m) \leftarrow \mathbf{COM}(\mathbf{B}, m).$$

We remark that when $f = 1$, only $\mathbf{r}_m$ is necessary to open $\llbracket m \rrbracket$.

The commitment scheme adheres to the properties of *hiding* and *binding* over the message m and its randomness $\mathbf{r}_m$. Examining the structure of a commitment

$$\llbracket m \rrbracket = \begin{bmatrix} \mathbf{c}_1 \\ c_2 \end{bmatrix} = \mathbf{B} \cdot \mathbf{r}_m + \begin{bmatrix} \mathbf{0} \\ m \end{bmatrix},$$

c_2 hides the message, and $\mathbf{c}_1$ binds the commit to a single $\mathbf{r}_m$. This allows for one of the main properties of the voting scheme: recovering a message m from its commitment using the key $\mathbf{B}$ and $\mathbf{r}_m$. We formalize this recovery process with the algorithm **REC**, which takes as input $\llbracket m \rrbracket$, $\mathbf{B}$, and $\mathbf{r}_m$ and outputs the corresponding message m following Proposition 1.

Proposition 1. *Let $\llbracket m \rrbracket$ be a commitment for $m \in R_p$ generated with a randomness $\mathbf{r}_m$ and key $\mathbf{B}$. The message m can be recovered with $\mathbf{r}_m$, $\mathbf{B}$ and $\llbracket m \rrbracket$.*

Proof. The proof is a straightforward equation. By replacing the commit value to its preliminary form $\llbracket m \rrbracket = \mathbf{B} \cdot \mathbf{r}_m + \begin{bmatrix} \mathbf{0} \\ m \end{bmatrix}$, and subtracting $\mathbf{B} \cdot \mathbf{r}_m$ from $\llbracket m \rrbracket$, we directly obtain the value $\begin{bmatrix} \mathbf{0} \\ m \end{bmatrix}$, thereby recovering m . $\square$

3.2 Proofs of Knowledge

We now discuss zero-knowledge proofs integral to our voting protocol: the *proof of shuffle of known values* Π_{Shuffle} and the *proof for sum* Π_{Sum} . Zero-knowledge proofs are protocols that allow a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ of a statement's truth without revealing any information beyond its validity. In essence, $\mathcal{P}$ possesses a tuple of secret information w and shares a tuple q of public knowledge with $\mathcal{V}$ and, via interactive communication, convinces $\mathcal{V}$ regarding the validity of a statement. However, the proofs are executed in their non-interactive versions with the Fiat-Shamir with aborts transform [11]. For a proof of knowledge Π , we define the algorithm **GEN** $_{\Pi}$ for generating the non-interactive version of Π from q and w and the algorithm **VER** $_{\Pi}$ for verifying Π with q .

Proof of Shuffle of Known Values. A verifiable shuffle is a mechanism to assert that a shuffled set of messages generated a specific set of commitments without revealing the original message-commitment pairings. The main contribution by Aranha et al. [3] is the proof Π_{Shuffle} over commitments in R_p , operating as

the vote anonymization mechanism for the *mix-net* of the protocol. Let $\mathbf{B}$ be a public key, $\llbracket \mathbf{X} \rrbracket = \{\llbracket x_1 \rrbracket, \dots, \llbracket x_n \rrbracket\}$ be a sequence of commitments generated from the messages $\mathbf{X} = \{x_1, \dots, x_n\}$ and randomness $\mathbf{R} = \{\mathbf{r}_{x_1}, \dots, \mathbf{r}_{x_n}\}$, and $\hat{\mathbf{X}} = \{\hat{x}_1, \dots, \hat{x}_n\}$ be a sequence of shuffled messages. We respectively define the private w and public q information as $w = (\mathbf{X}, \mathbf{R})$ and $q = (\llbracket \mathbf{X} \rrbracket, \hat{\mathbf{X}}, \mathbf{B})$.

Proof for Sum. This proof demonstrates that a committed value is a linear combination of two other committed values, without disclosing the underlying values. It is essential during the cast-as-intended verification mechanism, where return codes allow voters to verify that their ballots were correctly cast. Using this proof, it can be verified that a return code was generated using the actual vote value. Formally, we define Π_{Sum} as the proof for the relation $m'' = \alpha m + \beta m'$, where $\alpha, \beta \in R_p$ are two public polynomials and $\llbracket m \rrbracket, \llbracket m' \rrbracket, \llbracket m'' \rrbracket$ are three commitments with their corresponding messages m, m', m'' and randomness $\mathbf{r}, \mathbf{r}', \mathbf{r}''$. We respectively define the private w and public q information as

$$w = (m, \mathbf{r}, m', \mathbf{r}', m'', \mathbf{r}'') \quad \text{and} \quad q = (\alpha, \beta, \llbracket m \rrbracket, \llbracket m' \rrbracket, \llbracket m'' \rrbracket, \mathbf{B}).$$

This protocol was briefly referred to by [4], being introduced as similar to Π_{Lin} that proves the relation $m' = \alpha m$. However, since no explicit proof was provided, we instantiate the protocol and sketch a proof in Section 4.

3.3 Verifiable Encryption

A verifiable encryption scheme $\mathbf{VE}$ is a tuple of algorithms $(\mathbf{KEYGEN}_{\mathbf{VE}}, \mathbf{ENC}_{\mathbf{VE}}, \mathbf{VER}_{\mathbf{VE}}, \mathbf{DEC}_{\mathbf{VE}})$ where a property of a message can be verified with their respective ciphertext. For the voting protocol, Aranha et al. [3] proposed a $\mathbf{VE}$ that merges the Lyubashevsky and Neven [12] encryption scheme with a proof of knowledge for the relation between a commitment and its opening.

To enable this behavior, the parameters in $\mathbf{VE}$ must align with the dimensions of the k -height and n -width commitment matrix. Therefore, the encryption message space is R_p^k , and n is the dimension of the public information to verify the ciphertext. The evaluated relation by the algorithm is $\mathbf{T}\boldsymbol{\mu} = \mathbf{u} \pmod{p}$, for a plain text $\boldsymbol{\mu} \in D_{\beta_\infty}^k$ and a fixed $\mathbf{u} \in R_p^n$ and $\mathbf{T} \in R_p^{n \times k}$.

At its core, the scheme works by encrypting the opening randomness $\mathbf{r}_m$ used to generate a commitment to the vote, rather than encrypting the vote itself. When combined with the message recovery technique described in Proposition 1, this approach allows for a ballot to be constructed containing two essential components (i) a commitment to the vote; and (ii) an encrypted version of the randomness used to create that commitment. The vote can be revealed later by decrypting the randomness and using it to open the commitment. Thus, the relation $\mathbf{T}\boldsymbol{\mu} = \mathbf{u} \pmod{p}$ verified by the scheme ensures that the randomness $\mathbf{r}_m$ was used to generate the commitment $\llbracket m \rrbracket = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$ with the key $\mathbf{B} = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \end{bmatrix}$. This is achieved via the *binding* property, so the relation is $\mathbf{B}_1 \mathbf{r}_m = \mathbf{c}_1 \pmod{p}$.

Next, we briefly recall the inputs and outputs of the algorithms in a verifiable encryption scheme, as they are used to instantiate the voting protocol. For more details, we refer the reader to [3].

- **KEYGEN_{VE}**: receives a security parameter $\lambda \in \mathbb{N}^+$ and returns a key pair (sk, pk) ;
- **ENC**: receives a public key pk , a pair (u, T) and a message $\mu \in D_{\beta_\infty}^k$ and outputs a ciphertext with the structure $e = (v, w, c, z)$;
- **VER**: receives a ciphertext $e = (v, w, c, z)$, a pair (u, T) and a public key pk and returns the proof of the relation $T\mu = u \pmod{p}$;
- **DEC**: receives a ciphertext $e = (v, w, c, z)$, a pair (u, T) , a public key pk and the secret key pk and outputs the message μ .

3.4 Return Codes

Return codes are a mechanism that allows a voter to detect if their voting device is malicious. In this framework, the voter receives a code from another device after casting their ballot. Each code is mapped to a vote option. Therefore, the voter can use it to verify that the ballot was correctly cast. This operation is achieved with a table exclusive for each voter that maps a vote option to the output of a pseudorandom function PRF. For an arbitrary $\alpha \in \mathbb{N}$, let $k \in \{0, 1\}^\alpha$ be any binary string. Then, $\text{PRF}_k : \{0, 1\}^* \times R_p \rightarrow \{0, 1\}^\alpha$ is a pseudorandom function keyed by k . It maps a pair consisting of an arbitrary-length binary string and an element from the ring R_p to a fixed-length binary string of length α .

We define ϵ as the cardinality of the set of all possible vote combinations, being dependent on the ballot structure. For instance, for an m -out-of- n election, where n is the number of candidates and voters can select any subset of size $1 \leq m \leq n$, the total number of possible combinations is given by $\epsilon = 2^n - 1$, also denoting the size of the return code table.

To hide the vote contents, each voter V possesses a secret blinding key $a \in R_p$ and a commitment to this key $c_a = \llbracket a \rrbracket$ that is made public. When V casts a vote $v \in R_p$, a pre-code $\hat{r} = a + v \pmod{p}$ is computed. The encrypted ballot sent to the server includes commitments for the vote c_v and pre-code $c_{\hat{r}}$. Thus, the relation $\hat{r} = a + v$ can be proven via Π_{Sum} with the commitments $c_{\hat{r}}, c_a, c_v$, bonding v to $\hat{r}$. Finally, the return code $r \leftarrow \text{PRF}_k(V, \hat{r})$ is computed from the pre-code and sent to the voter, who can verify if the ballot was correctly cast.

4 Proof for Sum

We now explore the instantiation of Π_{Sum} and give a proof for a generalized version of the relation. Let $\alpha, \beta \in R_p$ be two public polynomials, and

$$\llbracket m \rrbracket = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}, \quad \llbracket m' \rrbracket = \begin{bmatrix} c'_1 \\ c'_2 \end{bmatrix}, \quad \llbracket m'' \rrbracket = \begin{bmatrix} c''_1 \\ c''_2 \end{bmatrix},$$

commitments with respective openings

$$d = (m; r, f), \quad d' = (m'; r', f), \quad d'' = (m''; r'', f).$$

We revisit the public q and private w information previously defined in Section 3.2 and instantiate Π_{Sum} in Figure 1 for the relation $m'' = \alpha m + \beta m'$. The proof is

a three-move protocol between the Prover $\mathcal{P}$ and Verifier $\mathcal{V}$ that employs the Fiat-Shamir with aborts [11] framework. Figure 1 details the calculations and communication between $\mathcal{P}$ and $\mathcal{V}$.

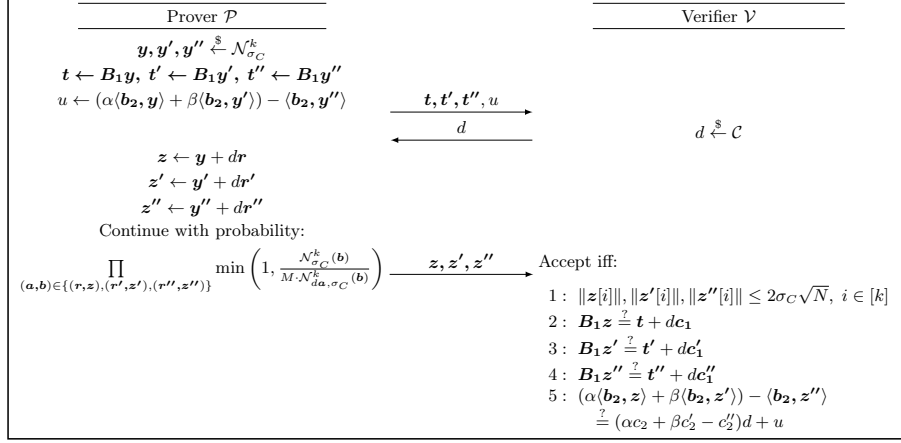


Fig. 1. The generalized three-move protocol Π_{Sum} .

We observe that when $f \neq 1$, the proof can be readily generalized to accommodate a broader class of relations $\alpha \tilde{m} + \beta \tilde{m}'$, where the messages m, m', m'' and their corresponding randomnesses $\mathbf{r}, \mathbf{r}', \mathbf{r}''$ are scaled by the factor f . Building upon the work of Aranha et al. [3] for the relation $\tilde{m}' = \alpha \tilde{m} + \beta$, we demonstrate that our Π_{Sum} proof can be similarly generalized in conjunction with Lemma 2.

Lemma 2. *Let $\alpha, \beta \in R_p$, $\mathbf{B}$ the commitment key, and $\llbracket m \rrbracket, \llbracket m' \rrbracket, \llbracket m'' \rrbracket$ three commitments with respective openings $(m; \mathbf{r}, f), (m'; \mathbf{r}', f), (m''; \mathbf{r}'', f)$. Then, Π_{Sum} is a Honest-Verifier Zero Knowledge Proof of the relation*

$$R_{\text{Sum}} = \left\{ (q, w) \left| \begin{array}{l} q = (\alpha, \beta, \llbracket m \rrbracket, \llbracket m' \rrbracket, \llbracket m'' \rrbracket, \mathbf{B}), \\ w = (\tilde{m}, \tilde{\mathbf{r}}, \tilde{m}', \tilde{\mathbf{r}}', \tilde{\mathbf{r}}'', f), \\ \mathbf{OPEN}(\mathbf{B}, \llbracket m \rrbracket, \tilde{m}, \tilde{\mathbf{r}}, f) \\ \quad = \mathbf{OPEN}(\mathbf{B}, \llbracket m' \rrbracket, \tilde{m}', \tilde{\mathbf{r}}', f) \\ \quad = \mathbf{OPEN}(\mathbf{B}, \llbracket m'' \rrbracket, \alpha \cdot \tilde{m} + \beta \cdot \tilde{m}', \tilde{\mathbf{r}}'', f) \\ \quad = 1 \end{array} \right. \right\}.$$

Proof. We assume that the polynomial-time prover can be rewound to obtain two accepting proofs with the same first message $(\mathbf{t}, \mathbf{t}', \mathbf{t}'', u)$ but differing challenges $d, \bar{d}$ and responses $\mathbf{z}, \mathbf{z}', \mathbf{z}'', \bar{\mathbf{z}}, \bar{\mathbf{z}}', \bar{\mathbf{z}}''$. Then, valid openings $(\tilde{m}; \tilde{\mathbf{r}}, f), (\tilde{m}'; \tilde{\mathbf{r}}', f)$, and $(\alpha \tilde{m} + \beta \tilde{m}'; \tilde{\mathbf{r}}'', f)$ for the commitments $\llbracket m \rrbracket, \llbracket m' \rrbracket, \llbracket m'' \rrbracket$ respectively can be extracted by the following process: from two accepting proofs, the values of $\tilde{\mathbf{r}} = \mathbf{z} - \bar{\mathbf{z}}, \tilde{\mathbf{r}}' = \mathbf{z}' - \bar{\mathbf{z}}'$ and $\tilde{\mathbf{r}}'' = \mathbf{z}'' - \bar{\mathbf{z}}''$ can be recovered, and we can set $f = d - \bar{d}$, where it must hold that

$$(\alpha \langle \mathbf{b}_2, \tilde{\mathbf{r}} \rangle + \beta \langle \mathbf{b}_2, \tilde{\mathbf{r}}' \rangle) - \langle \mathbf{b}_2, \tilde{\mathbf{r}}'' \rangle \stackrel{?}{=} f(\alpha c_2 + \beta c'_2 - c''_2).$$

When we set $\tilde{m} = c_2 - f^{-1}(\mathbf{b}_2, \tilde{\mathbf{r}})$, $\tilde{m}' = c'_2 - f^{-1}(\mathbf{b}_2, \tilde{\mathbf{r}}')$ and $\tilde{m}'' = c''_2 - f^{-1}(\mathbf{b}_2, \tilde{\mathbf{r}}'')$, we have that $\tilde{m}'' = \alpha\tilde{m} + \beta\tilde{m}'$ by the previous equation. We briefly demonstrate the relation with the following steps:

$$\begin{aligned}
(\alpha\langle\mathbf{b}_2, \tilde{\mathbf{r}}\rangle + \beta\langle\mathbf{b}_2, \tilde{\mathbf{r}}'\rangle) - \langle\mathbf{b}_2, \tilde{\mathbf{r}}''\rangle &\stackrel{?}{=} f(\alpha c_2 + \beta c'_2 - c''_2) \\
f^{-1}((\alpha\langle\mathbf{b}_2, \tilde{\mathbf{r}}\rangle + \beta\langle\mathbf{b}_2, \tilde{\mathbf{r}}'\rangle) - \langle\mathbf{b}_2, \tilde{\mathbf{r}}''\rangle) &\stackrel{?}{=} \alpha c_2 + \beta c'_2 - c''_2 \\
c''_2 - f^{-1}(\langle\mathbf{b}_2, \tilde{\mathbf{r}}''\rangle) &\stackrel{?}{=} \alpha c_2 + \beta c'_2 - f^{-1}(\alpha\langle\mathbf{b}_2, \tilde{\mathbf{r}}\rangle + \beta\langle\mathbf{b}_2, \tilde{\mathbf{r}}'\rangle) \\
\tilde{m}'' &\stackrel{?}{=} \alpha\tilde{m} + \beta\tilde{m}'.
\end{aligned}$$

The remainder of the proof regarding the opening validity and bounds follows the arguments in [3, 4]. $\square$

5 Voting Protocol

The proposed protocol by Aranha et al. [3] has four phases (*setup*, *registration*, *casting* and *counting*). Moreover, it relies on the commitment scheme $\mathbf{C}$ and the verifiable encryption $\mathbf{VE}$ algorithms. The participant players are (i) a ballot box B ; (ii) a return code generator R ; (iii) a shuffler S ; (iv) a set of auditors $\mathbf{A}$; and (v) a set of voters $\mathbf{V}$ of cardinality τ . Each voter holds their respective voting device D and another device F , which should not be the same type as D , to receive the return code. We give a formalization of all scheme algorithms presented in [3]. Each referenced algorithm is accompanied by a detailed description of its inputs and outputs. We also provide a detailed description of all phases of the protocol.

Algorithm 4 SETUP	Algorithm 5 REGISTER
Input: $\lambda \in \mathbb{N}^+$	Input: $pk = (pk_C, pk_V, pk_R)$
Output: pk, dk, ck	Output: vvk, vck
1 $pk_C \leftarrow \mathbf{KEYGEN}_C(\lambda)$	1 $a \xleftarrow{\$} R_p$
2 $(pk_V, dk_V) \leftarrow \mathbf{KEYGEN}_{VE}(\lambda)$	2 $(c_a, d_a) \leftarrow \mathbf{COM}(pk_C, a)$
3 $(pk_R, dk_R) \leftarrow \mathbf{KEYGEN}_{VE}(\lambda)$	3 $vvk = c_a$
4 $pk = (pk_C, pk_V, pk_R)$	4 $vck = (a, c_a, d_a)$
5 $dk = (pk_C, dk_V)$	5 return (vvk, vck)
6 $ck = (pk_C, pk_V, dk_R)$	
7 return (pk, dk, ck)	

During the *setup phase*, a trusted set of players run **SETUP** (Algorithm 4), receiving as input the security parameter λ and returning the system-wide public key pk , the decryption key dk and the code key ck . All participants receive pk , while dk is sent to the shuffler S and ck is sent to the return code generator R .

During the *registration phase*, a trusted set of players for each voter $V \in \mathbf{V}$ run **REGISTER** (Algorithm 5). The algorithm receives the system-wide public key pk and outputs a vote verification key vvk and a vote casting key vck . The vck is given to the device D of the voter, and vvk is made public. After registering every voter, the return code server R selects a key k for the PRF and publishes it. Then,

for each voter $V \in \mathbf{V}$ holding a casting key $vck = (a, c_a, d_a)$, and considering all voting possibilities $\mathcal{V} = \{v_1, \dots, v_\epsilon\}$, a trusted set of players compute the return code table $\{(v_i, \text{PRF}_k(V, v_i + a)) : v_i \in \mathcal{V}\}$.

Algorithm 6 CAST	Algorithm 7 CODE
Input: $pk = (pk_C, pk_V, pk_R)$, $vck = (a, c_a, d_a)$, v Output: ev, Π^v 1 $\hat{r} \leftarrow v + a$ 2 $(c, d) \leftarrow \text{COM}(pk_C, v)$ 3 $(c_{\hat{r}}, d_{\hat{r}}) \leftarrow \text{COM}(pk_C, \hat{r})$ 4 $w = (v, d, a, d_a, d_{\hat{r}})$ 5 $q = (1, 1, c, c_a, c_{\hat{r}}, pk_C)$ 6 $\Pi_{\text{Sum}}^{\hat{r}} \leftarrow \text{GEN}_{\Pi_{\text{Sum}}}(q, w)$ 7 $(v, w, c, z) \leftarrow \text{ENC}_{\text{VE}}(pk_V, d)$ 8 $e_{\hat{r}} \leftarrow \text{ENC}_{\text{VE}}(pk_R, d_{\hat{r}})$ 9 $ev = (c, d, v, w, c)$ 10 $\Pi^v = (z, c_{\hat{r}}, e_{\hat{r}}, \Pi_{\text{Sum}}^{\hat{r}})$ 11 return (ev, Π^v)	Input: $ck = (pk_C, pk_V, dk_R)$, $vvk = c_a$, $ev = (c, d, v, w, c)$, $\Pi^v = (z, c_{\hat{r}}, e_{\hat{r}}, \Pi_{\text{Sum}}^{\hat{r}})$ Output: $\hat{r}$ 1 $e = (v, w, c, z)$ 2 $q = (1, 1, c, c_a, c_{\hat{r}}, pk_C)$ 3 if $\text{VER}_{\Pi_{\text{Sum}}}(\Pi_{\text{Sum}}^{\hat{r}}, q)$ and $\text{VER}_{\text{VE}}(pk_V, e)$ and $\text{VER}_{\text{VE}}(pk_R, e_{\hat{r}})$ then 4 $d_{\hat{r}} \leftarrow \text{DEC}_{\text{VE}}(dk_R, e_{\hat{r}})$ 5 $\hat{r} \leftarrow \text{REC}(c_{\hat{r}}, pk_C, d_{\hat{r}})$ 6 return $\hat{r}$ 7 return $\perp$

During the *casting phase*, V instructs their device D to run **CAST** (Algorithm 6). The algorithm inputs are the system public key pk , the vote casting key vck , and the desired vote $v \in R_p$. It outputs two tuples: the encrypted ballot ev and the ballot proof Π^v . Essentially, ev holds the information to later reveal the vote, and Π^v contains the data to validate the vote and return code.

Afterwards, D sends ev and Π^v to the ballot box B , which refers them to R . Subsequently, R runs **CODE** (Algorithm 7) that receives as input the code key ck , the verification key vvk from V , ev and Π^v and returns the pre-code $\hat{r}$. Then, the return code can be computed with $r \leftarrow \text{PRF}_k(\hat{r})$ and sent to F that forwards it to V . Finally, V compares r with the code in their table and accepts the ballot only if they are equal. Additionally, D signs both the ev and Π^v in the name of V . Then, B and R verify the signature, and if it is valid, R generates a countersignature of the entire content, sending it to D for verification.

During the *counting phase*, the ballot box B and return code server R send their encrypted ballots and ballot proofs to the auditors $\mathbf{A}$. If they are consistent, the auditors approve the ballots. Then, B orders the encrypted ballots and sends them to the shuffler S that runs **COUNT** (Algorithm 8). The algorithm receives the decryption key dk and the encrypted ballots $\{ev_1, \dots, ev_\tau\}$ and returns a shuffled list of votes $\{v_1, \dots, v_\tau\}$ and the proof of shuffle Π^c . Both outputs are sent to the auditors.

Finally, each auditor runs **VERIFY** (Algorithm 9), inputting the system public key pk , the encrypted ballots $\{ev_1, \dots, ev_\tau\}$ from B , the shuffled votes $\{v_1, \dots, v_\tau\}$ and the shuffle proof Π^c . The algorithm outputs the proof verification, and the auditor publishes the results.

Algorithm 8 COUNT	Algorithm 9 VERIFY
Input: $dk = (pk_C, dk_V), \{ev_1, \dots, ev_\tau\}$ Output: $(\{v_1, \dots, v_\tau\}, \Pi^c)$ 1 foreach $ev_i \in \{ev_1, \dots, ev_\tau\}$ do 2 $(c_i, d_i, v_i, w_i, c_i) \leftarrow ev_i$ 3 $d_i \leftarrow \text{DEC}_{VE}(dk_V, v_i, w_i, c_i)$ 4 $v'_i \leftarrow \text{REC}(c_i, pk_C, d_i)$ 5 $\mathcal{S} \leftarrow \{v'_1, \dots, v'_\tau\}$ 6 $\{v_1, \dots, v_\tau\} \leftarrow \text{SHUFFLE}(\mathcal{S})$ 7 $w = (\mathcal{S}, \{d_1, \dots, d_\tau\})$ 8 $q = (\{c_1, \dots, c_\tau\}, \{v_1, \dots, v_\tau\}, pk_C)$ 9 $\Pi^c \leftarrow \text{GEN}_{\Pi_{\text{Shuffle}}}(q, w)$ 10 return $(\{v_1, \dots, v_\tau\}, \Pi^c)$	Input: $pk = (pk_C, pk_V, pk_R), \{ev_1, \dots, ev_\tau\}, \{v_1, \dots, v_\tau\}, \Pi^c$ Output: Π_{Shuffle} verification 1 foreach $ev_i \in \{ev_1, \dots, ev_\tau\}$ do 2 $(c_i, d_i, v_i, w_i, c_i) \leftarrow ev_i$ 3 $q = (\{c_1, \dots, c_\tau\}, \{v_1, \dots, v_\tau\}, pk_C)$ 4 return $\text{VER}_{\Pi_{\text{Shuffle}}}(\Pi_c, q) = 1$

6 Implementation

Building upon the primitives implemented by Aranha et al. [3] as our foundation, we extend their contributions by developing the complete protocol. Besides the phases and algorithms from Section 5, we also create the message recovery mechanism derived from Proposition 1, the Π_{Sum} generation and verification algorithms, and selected the **SHUFFLE** and PRF algorithms that will be further detailed in Section 6.2.

As illustrated in Figure 2, our implementation is structured as a Python library (gray-shaded region) encompassing the core protocol. This core library is assisted by supporting mechanisms represented by the outer sections. These mechanisms were either implemented directly within the Python code or integrated through C for enhanced polynomial manipulation.

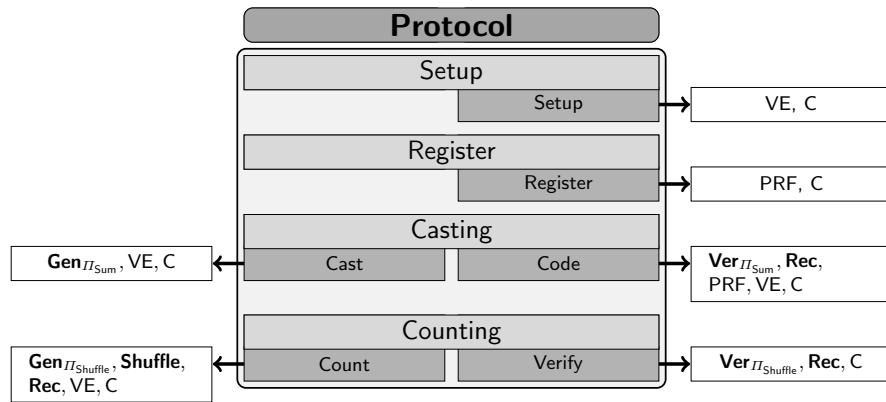


Fig. 2. Implemented protocol structure with the phases, algorithms, and primitives.

6.1 C Primitives

Our code depends on core primitives from [3], relying on the library FLINT [19] for efficient polynomial arithmetic within rings. To allow integration with the Python library, we perform adaptations to encapsulate and expose functions and the commitment and verifiable encryption schemes through a shared object. Furthermore, to allow for the intertwined behavior of the primitives, we employ the same parameter values as in [3].

Regarding the developed functions, in this component, we include **REC**, **GEN** $_{H_{\text{Sum}}}$ and **VER** $_{H_{\text{Sum}}}$. To enable interaction between the commitment scheme and the verifiable encryption scheme, we implement mechanisms to allow commitment openings to be ciphered and deciphered. This is necessary because the implementation utilizes distinct polynomial representations for each scheme.

6.2 Python Library

Primarily, the library loads the shared object and maps the primitives to Python classes and functions via the `ctypes` Python library. Then, for each primitive, an interface was developed to simplify the function callings. The algorithms and phases of the protocol were created from these building blocks following the specifications in Section 5. Figure 3 illustrates the structure of the developed Python library and the inner-module dependencies.

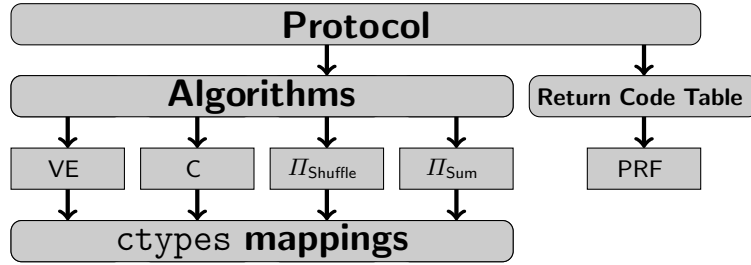


Fig. 3. Structure of the Python library showing dependencies between cryptographic primitives and other structures.

The **SHUFFLE** algorithm employs a Fisher-Yates shuffle for its simplicity. However, a more robust approach would involve utilizing a proper *mix-net* as described by Chaum [6]. We selected HMAC_SHA512 as the PRF for the return code table, employing a key and function image size of 512 bits. This choice was made considering that the SHA512 algorithm still provides acceptable security against quantum algorithms [15]. Finally, a serializer was developed to convert `ctypes`-mapped structures to and from JSON, transforming the polynomials into an easier-to-store format. The serializer represents each polynomial as a string, each matrix dimension as a list, and each structure or tuple as a JSON object.

6.3 Integration with Helios

Structurally, the Helios system follows the same phase organization as the protocol by Aranha et al. [3], facilitating the integration. The main distinction between the two protocols is in the frameworks used for vote anonymization. Helios employs *homomorphic tallying*, where encrypted ballots are summed during the counting phase. Auditors then decrypt and publish the final result. Conversely, Aranha et al. [3] utilize a *mix-net*, which deciphers and shuffles the encrypted ballots. Auditors subsequently verify the proof of shuffle.

For our implementation, we employ the main Helios server as both the ballot box B and the trusted player that runs **SETUP** and **REGISTER** and computes the return code tables. The remaining players—the return code server R , the shuffler S , and the auditor A —are implemented as Python APIs that share a common codebase. Each API exposes only the endpoints relevant to its specific role within the system, allowing the reuse of common structures. Keys, ballots, and proofs are serialized into JSON and stored within a relational database.

Following the description and notations in Section 5, Figure 4 describes our complete election protocol within Helios, presenting the data flow between all players. Each election is initialized with the addresses of all participating servers. Upon freezing, where the election becomes immutable, the Helios server runs **SETUP**, aggregating and publishing all public information to the other players. This information includes the public key and the addresses of each server.

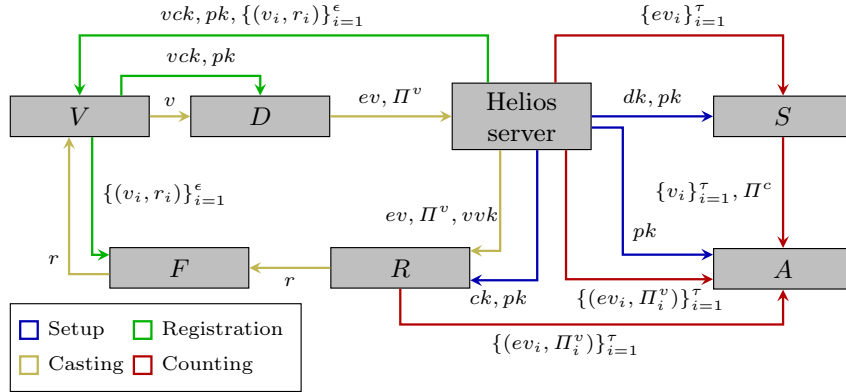


Fig. 4. Data flow of the entire protocol within Helios. We highlight the circular casting process where voter send their vote and receive a return code.

Voter registration can be performed in two ways: (i) after the election setup for a predefined set of voters, or, if the election is made public, (ii) delegated to just before the voter casts their first ballot. This allows for more flexibility than rigidly following the order of the phases, and gives the election manager further customization options while maintaining the data flow illustrated in Figure 4.

Voter Devices. For ease of implementation, the voter phone F is abstracted as a separate server, registered with the voter and possessing the return code table. The original Helios implementation employs client-side ballot encryption within the browser used by the voter. However, our reliance on the shared object for algorithm execution complicates this approach in our current context. To achieve browser compatibility, the code must be compiled to WebAssembly (WASM) and integrated into the Helios booth.

Only the first step of this process was achieved by substituting the discrete Gaussian and AES with implementations that do not rely on certain SIMD operations, currently unsupported by WASM. Hence, we developed the voting device D as a desktop application communicating with the main Helios Server. This approach still allows for secure ballot encryption at the cost of usability.

Gains and Losses in Properties. Our implementation enhances the security of the Helios system against quantum threats. Additionally, it supports complex ballots, opening possibilities for intricate voting options and return codes for cast-as-intended ballot verification. However, this comes at the cost of losing universal verifiability and compromised browser compatibility. Although the cost of verifiability may limit the direct ability of each voter to independently validate the final election outcome, the system can still achieve a significant level of integrity and remain trustworthy, as long as at least one auditor is honest.

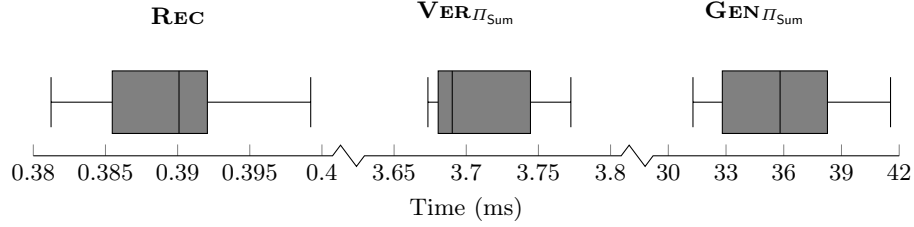
7 Efficiency Analysis

We benchmark our implementation on an AMD Ryzen 7 5700X @ 3.4GHz to obtain performance-related metrics for both the algorithms and the protocols. Each test was executed 10 times, and the mean of all results is presented. The main parameter to consider is the number of voters τ . However, we also discuss the structure of the vote, assuming an m -out-of- n election.

7.1 Auxiliary Algorithms

We illustrate in Figure 5 the performance of the auxiliary algorithms. Notably, **REC**, **VER** $_{\Pi_{\text{Sum}}}$ are relatively fast since they only perform simpler polynomial operations, while **GEN** $_{\Pi_{\text{Sum}}}$ exhibits a markedly higher execution time as a result of the Fiat-Shamir with aborts increasing the prover time by a constant factor. Additionally, the **SHUFFLE** algorithm has an average performance of 0.069τ ms, and given the entire runtime of the protocol, it is comparatively inexpensive, even for a large number of voters.

Considering that a voter can choose any number of candidates $1 \leq m \leq n$ for their vote, increasing n has a significant impact on the number of executions ϵ of the PRF. We remark that this behavior adds substantial overhead for the trusted party during the *registration* phase. Although all voting options can be computed once for all voters, the **HMAC_SHA512** function has to be individually executed for each option, with an average execution time of 0.113ϵ ms per voter.

**Fig. 5.** Performance for **REC** and Π_{Sum} .**Table 1.** Comparison between the phases of our implementation and Helios.

		Algorithm						
		Setup	Registration		Casting		Counting	
τ			REGISTER	PRF	CAST	CODE	COUNT	VERIFY
Helios	2^4	0.005 s	≈ 0.000 s	-	4.871 s	-	0.058 s	0.059 s
	2^6	0.005 s	≈ 0.000 s	-	19.529 s	-	0.065 s	0.058 s
	2^8	0.005 s	0.002 s	-	78.520 s	-	0.091 s	0.058 s
	2^{10}	0.005 s	0.006 s	-	310.896 s	-	0.200 s	0.058 s
	2^{12}	0.005 s	0.056 s	-	1243.953 s	-	0.636 s	0.058 s
	2^{14}	0.005 s	0.258 s	-	4995.920 s	-	2.355 s	0.058 s
Ours	2^4	0.004 s	0.025 s	0.078 s	1.782 s	0.787 s	0.464 s	0.091 s
	2^6	0.004 s	0.101 s	0.314 s	7.563 s	3.144 s	1.992 s	0.365 s
	2^8	0.004 s	0.397 s	1.250 s	30.040 s	12.538 s	8.210 s	1.462 s
	2^{10}	0.004 s	1.612 s	5.038 s	121.974 s	50.479 s	33.127 s	5.907 s
	2^{12}	0.004 s	6.340 s	19.806 s	487.045 s	202.983 s	133.222 s	23.793 s
	2^{14}	0.004 s	25.197 s	81.112 s	1896.569 s	786.010 s	519.343 s	90.285 s

Therefore, if the return code mechanism is used, elections with many candidates can be impractical, even considering that the trusted party and the other players have plenty of computational resources.

7.2 Protocol Performance and Comparison with Helios

In Table 1, the Helios protocol is compared with our implementation. For our set of tests, we detached both protocols from their web-based components to provide a fairer and replicable environment and considered an election with eight possibilities and one auditor. In our implementation, the variation in the number of options can add significant overhead during the return code table computation represented as PRF in *registration*. While in the Helios system, it affects the number of knowledge proofs created for the ballot during *casting*.

Both *setup* phases are inexpensive and independent of the number of voters. In contrast, the remaining phases grow linearly with the number of voters. Furthermore, our use of a shared library accelerates several protocol processes. Executing them in compiled code offers a significant performance advantage compared to the original Helios implementation, which relies solely on Python.

Since the original Helios implementation lacks the return code mechanism, it does not execute **PRF** and **CODE**. This primarily affects the differences during registration. While our **REGISTER** is relatively inexpensive since it only executes polynomial sampling and simple operations, the **PRF** execution time can scale significantly, especially with more voting possibilities.

The main time overhead in both protocols is the *casting* phase, with our implementation outperforming Helios even with the **CODE** algorithm. In Helios case, this overhead is due to the *casting* phase incorporating vote verification when a ballot is cast. In contrast, our implementation performs heavy operations in *casting* with the return code mechanism that executes Π_{Sum} and *counting* via the generation and verification of the shuffle proof. However, for both protocols, the time during the *casting* phase is distributed between all voters, being executable in parallel. Therefore, in a real situation, the burden of this phase is lessened.

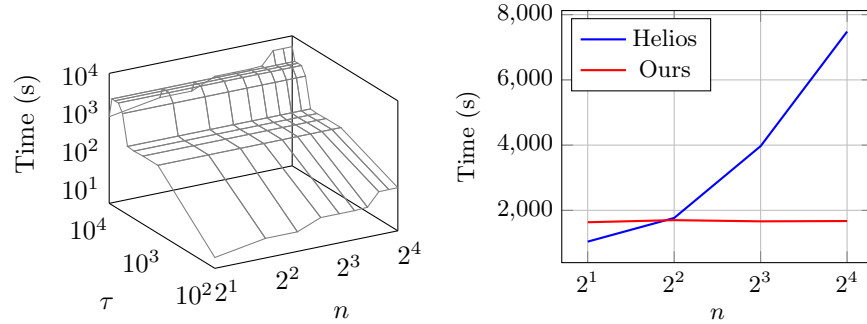
During *counting*, since Helios relies on homomorphic tallying, the number of voters has less impact during the phase, with the main time growth occurring during the aggregation of the ballots, while the verification time is constant. In contrast, the number of voters affects both **COUNT** and **VERIFY** in our implementation, with **COUNT** being significantly slower than **VERIFY**, since, similar to Π_{Sum} , the generation of the proof of shuffle is substantially more expensive than the verification.

Our implementation presents a tradeoff compared to Helios, exhibiting faster *casting* and significantly slower *counting*. While *counting* introduces a one-time, potentially costly election management overhead, it is particularly noteworthy that our *casting* outperforms Helios, especially considering our on reliance lattice-based primitives, since the reduced time greatly enhances the voter experience.

As stated earlier, increasing the number of candidates greatly affects the casting phase in Helios. Figure 6b compares the casting performance of our implementation with Helios for varying numbers of candidates n . Our implementation behaves more consistently, exhibiting comparable performance with four candidates and superior performance as that number increases. Figure 6a further illustrates this behavior by showing the combined impact of τ and n during casting in Helios.

8 Conclusion

Considering the threat of quantum computers and the NIST deadline for deprecation of classical algorithms [15], the development of sound and reliable quantum-safe voting systems is critical for privacy of voters, both for elections in the future and in the present, considering an adversary that stores the ballots for later decryption. The scientific community has already identified and acted on



(a) Helios casting phase for varied τ and (b) Casting with $\tau = 10,000$ and varied n .

Fig. 6. Comparison between the casting phase of our implementation and Helios.

this necessity by proposing multiple schemes [2, 3, 5, 7, 16]. However, there are still barriers that distance those schemes from practical use cases.

Given the urgency and importance of this situation, we proposed the first complete and public implementation of a quantum-safe e-voting protocol. We argue that this is an important step to close the distance between quantum-safe e-voting and the real world. This implementation is packaged as a generic Python library and integrated with the widely adopted Helios voting system.

Overall, we expected that a quantum-safe system would exhibit lower performance than one utilizing classical cryptography. Surprisingly, certain portions of the protocol performed similarly or better than Helios. This is a very positive result since many elections using Helios are conducted in smaller scenarios, with at most a couple thousand voters. The experiments show that our protocol can be a competitive candidate for a practical quantum-safe voting system.

8.1 Future Work

We suggest the following improvements to our proposal: (i) broader browser compatibility, likely integrating our system into the Helios booth through an agnostic framework such as WASM; (ii) generalization of the return code mechanism to send messages to voters via secure channels; (iii) evaluation of the schemes proposed by del Pino et al. [16] and Boyen et al. [5] after being adapted to work with Helios, as they employ the commitment scheme due to Baum et al. [4]. Particularly, Boyen et al. [5] construction is closer property-wise to Helios, due to both schemes offering end-to-end verifiability and homomorphic counting with multi-authority decryption via secret sharing.

Acknowledgments. We thank Gustavo de Castro Biage and the anonymous reviewers for their helpful comments and suggestions.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

Bibliography

- [1] Adida, B.: Helios: Web-based Open-Audit Voting. In: 17th USENIX Security Symposium, pp. 335–348 (Jul 2008), <https://doi.org/10.5555/1496711.1496734>, URL https://www.usenix.org/legacy/event/sec08/tech/full_papers/adida/adida.pdf
- [2] Aranha, D.F., Baum, C., Gjøsteen, K., Silde, T.: Verifiable Mix-Nets and Distributed Decryption for Voting from Lattice-Based Assumptions. In: CCS '23: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, pp. 1467–1481 (Nov 2023), <https://doi.org/10.1145/3576915.3616683>, URL <https://eprint.iacr.org/2022/422>
- [3] Aranha, D.F., Baum, C., Gjøsteen, K., Silde, T., Tunge, T.: Lattice-Based Proof of Shuffle and Applications to Electronic Voting. In: Paterson, K.G. (ed.) Topics in Cryptology – CT-RSA 2021, Lecture Notes in Computer Science, vol. 12704, pp. 227–251 (May 2021), https://doi.org/10.1007/978-3-030-75539-3_10, URL <https://eprint.iacr.org/2021/338>
- [4] Baum, C., Damgård, I., Lyubashevsky, V., Oechsner, S., Peikert, C.: More Efficient Commitments from Structured Lattice Assumptions. In: Catalano, D., de Prisco, R. (eds.) Security and Cryptography for Networks (SCN 2018), Lecture Notes in Computer Science, vol. 11035, pp. 368–385 (Sep 2018), https://doi.org/10.1007/978-3-319-98113-0_20, URL <https://eprint.iacr.org/2016/997>
- [5] Boyen, X., Haines, T., Müller, J.: Epoque: Practical End-to-End Verifiable Post-Quantum-Secure E-Voting. In: 2021 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 272–291 (Sep 2021), <https://doi.org/10.1109/EuroSP51992.2021.00027>, URL <https://eprint.iacr.org/2021/304>
- [6] Chaum, D.L.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM **24**(2), 84–90 (Feb 1981), <https://doi.org/10.1145/358549.358563>
- [7] Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: A Homomorphic LWE Based E-voting Scheme. In: Takagi, T. (ed.) Post-Quantum Cryptography (PQCrypto 2016), Lecture Notes in Computer Science, vol. 9606, pp. 245–265 (Feb 2016), https://doi.org/10.1007/978-3-319-29360-8_16, URL https://ilachill.github.io/papers/CGGI16a-An_homomorphic_LWE_based_E-voting_Scheme.pdf
- [8] Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. European Transactions on Telecommunications **8**(5), 481–490 (Sep 1997), <https://doi.org/10.1002/ett.4460080506>
- [9] Elgamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory **31**(4), 469–472 (Jul 1985), <https://doi.org/10.1109/TIT.1985.1057074>
- [10] Gjøsteen, K., Lund, A.S.: An experiment on the security of the Norwegian electronic voting protocol. Annals of Telecommunications **71**, 299–307 (Aug 2016), <https://doi.org/10.1007/s12243-016-0509-8>

- [11] Lyubashevsky, V.: Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures. In: Matsui, M. (ed.) *Advances in Cryptology – ASIACRYPT 2009*, Lecture Notes in Computer Science, vol. 5912, pp. 598–616 (Dec 2009), https://doi.org/10.1007/978-3-642-10366-7_35, URL <https://iacr.org/cryptodb/data/paper.php?pubkey=18550>
- [12] Lyubashevsky, V., Neven, G.: One-Shot Verifiable Encryption from Lattices. In: Coron, J.S., Nielsen, J.B. (eds.) *Advances in Cryptology – EUROCRYPT 2017*, Lecture Notes in Computer Science, vol. 10210, pp. 293–323 (Apr 2017), https://doi.org/10.1007/978-3-319-56620-7_11, URL <https://eprint.iacr.org/2017/122>
- [13] Lyubashevsky, V., Seiler, G.: Short, Invertible Elements in Partially Splitting Cyclotomic Rings and Applications to Lattice-Based Zero-Knowledge Proofs. In: Nielsen, J.B., Rijmen, V. (eds.) *Advances in Cryptology – EUROCRYPT 2018*, Lecture Notes in Computer Science, vol. 10820, pp. 204–224 (Apr 2018), https://doi.org/10.1007/978-3-319-78381-9_8, URL <https://eprint.iacr.org/2017/523>
- [14] Ülle Madise, Martens, T.: E-voting in Estonia 2005. The first practice of country-wide binding Internet voting in the world. In: Krimmer, R. (ed.) *Electronic Voting 2006 – 2nd International Workshop*, Co-organized by Council of Europe, ESF TED, IFIP WG 8.6 and E-Voting.CC, pp. 15–26 (Aug 2006), URL <https://dl.gi.de/items/b15050c3-a8ea-46f9-8976-1584b3cf499f>
- [15] Moody, D., Perlner, R., Regenscheid, A., Robinson, A., Cooper, D.: Transition to Post-Quantum Cryptography Standards. NIST Internal Report (Initial Public Draft) 8547, National Institute of Standards and Technology (Dec 2024), <https://doi.org/10.6028/NIST.IR.8547.ipd>
- [16] del Pino, R., Lyubashevsky, V., Neven, G., Seiler, G.: Practical Quantum-Safe Voting from Lattices. In: CCS ’17: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 1565–1581 (Oct 2017), <https://doi.org/10.1145/3133956.3134101>, URL <https://eprint.iacr.org/2017/1235>
- [17] Shamir, A.: How to share a secret. *Communications of the ACM* **22**(11), 612–613 (Nov 1979), <https://doi.org/10.1145/359168.359176>
- [18] Shor, P.W.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Review* **41**(2), 303–332 (Apr 1999), <https://doi.org/10.1137/S0036144598347011>
- [19] The FLINT team: FLINT: Fast Library for Number Theory (Mar 2025), version 3.2.1, <https://flintlib.org>