Verifiable Mix-Nets and Distributed Decryption
for Voting from Lattice-Based Assumptions*

Diego F. Aranha
dfaranha@cs.au.dk
Aarhus University
Aarhus, Denmark

Kristian Gjøsteen
kristian.gjosteen@ntnu.no
Norwegian University of Science and Technology
Trondheim, Norway

Carsten Baum
cabau@dtu.dk
DTU Compute and
Aarhus University
Copenhagen, Denmark

Tjerand Silde†
tjerand.silde@ntnu.no
Norwegian University of Science and Technology
Trondheim, Norway

ABSTRACT
Cryptographic voting protocols have recently seen much interest from practitioners due to their (planned) use in countries such as Estonia, Switzerland, France, and Australia. Practical protocols usually rely on tested designs such as the mixing-and-decryption paradigm. There, multiple servers verifiably shuffle encrypted ballots, which are then decrypted in a distributed manner. While several efficient protocols implementing this paradigm exist from discrete log-type assumptions, the situation is less clear for post-quantum alternatives such as lattices. This is because the design ideas of the discrete log-based voting protocols do not carry over easily to the lattice setting, due to specific problems such as noise growth and approximate relations.

This work proposes a new verifiable secret shuffle for BGV ciphertexts and a compatible verifiable distributed decryption protocol. The shuffle is based on an extension of a shuffle of commitments to known values which is combined with an amortized proof of correct re-randomization. The verifiable distributed decryption protocol uses noise drowning, proving the correctness of decryption steps in zero-knowledge. Both primitives are then used to instantiate the mixing-and-decryption electronic voting paradigm from lattice-based assumptions.

We give concrete parameters for our system, estimate the size of each component and provide implementations of all important sub-protocols. Our experiments show that the shuffle and decryption protocol is suitable for use in real-world e-voting schemes.

KEYWORDS
Lattice-Based Cryptography, Electronic Voting, Implementation

1 INTRODUCTION
Mix-nets were originally proposed for anonymous communication [18], but have since been used extensively for cryptographic voting systems. A mix-net is a multi-party protocol that gets as input a collection of ciphertexts and outputs another collection of ciphertexts whose decryption is the same set, up to order. It guarantees that the permutation between input and output ciphertexts is hidden if at least one party is honest, while none of the servers involved learns the plaintexts.

Mix-nets are commonly used in cryptographic voting. Here, encrypted ballots are submitted to a bulletin board or ballot box with identifying information attached. These ciphertexts are then sent through a mix-net before decryption, to break the identity-ballot correlation. In addition to hiding the permutation, the correctness of the mix-net output must be verifiable. In applications such as voting it is important that the mix-net provides a proof of correctness that can be verified by anyone at any later point in time, to ensure universal verifiability.

A shuffle of a set of ciphertexts is another set of ciphertexts whose decryption is the same as the original set, up to order. Compared to a mix-net, it is performed by one server only (which does not learn the plaintexts). As for mix-nets, a shuffle is secret if it is hard for any external party to correlate input and output ciphertexts, and verifiable if there is a proof that decryptions are the same.

If we have a verifiable secret shuffle for some cryptosystem, then this can be used to construct a mix-net: for this, the nodes of the mix-net receive a set of ciphertexts as input, shuffle them sequentially and each provides a proof of correctness. The mix-net proof then consists of the intermediate ciphertexts along with the shuffle proofs. If at least one node in this chain is honest, it is hard to correlate the inputs and outputs.

For applications in cryptographic voting, it must also be guaranteed that the correct result can be obtained from the mix-net output, while nobody has the secret decryption key. One strategy is to use verifiable threshold decryption, where the key is secret-shared among a committee of decryption parties. Each of them contributes to the decryption and proves that they did so honestly.

Verifiable shuffling and verifiable distributed decryption protocols are well-known for cryptosystems based on discrete logarithm-type assumptions. For example, Neff [39] proposed the first efficient verifiable secret shuffle for ElGamal-like cryptosystems. Verifiable decryption for ElGamal-like cryptosystems can be constructed using standard Verifiable Secret-Sharing and Σ-protocols.

While amale voting schemes have been constructed based on the aforementioned outline, they essentially all rely on assumptions that are not secure against quantum computers. Given the need for the long-term privacy of elections, it is important to construct verifiable shuffles and distributed decryption from quantum-safe...
computational problems such as lattice assumptions. NIST recently standardized post-quantum key-encapsulation mechanisms and digital signatures based on lattices [35, 40, 42]. Using shuffles and distributed decryption schemes based on the same assumptions, it seems well-motivated to build a plausibly post-quantum voting scheme following the aforementioned approach. The main obstacle to simply adopting the protocols for discrete logarithms to lattices is the (presumed) lack of suitable efficient techniques for verification, as well as the problem of noise growth.

1.1 Our contributions

In this work, we make progress in the direction of plausibly quantum-secure voting. We design a verifiable secret shuffle for BGV ciphertexts [17] that is suitable for cryptographic voting systems dealing with arbitrary vote structures. In addition, we construct a verifiable distributed decryption protocol by compiling previous passively-secure constructions with zero-knowledge proofs and show how to integrate these and other building blocks into a voting scheme. Finally, we implemented the main parts of the verifiable shuffle and distributed decryption protocols to demonstrate the viability and efficacy of our overall design.

Lattice-based shuffle. To construct a mix-net for BGV ciphertexts we extend the shuffle of commitments to known values by Aranha et al. [4]. Their construction only works for BDLOP [7] commitments of message length 1, while we generalize their construction to an arbitrary length. Given such a generalized verifiable shuffle of commitment openings, our verifiable shuffle for input ciphertexts \( e_1, \ldots, e_r \) then works as follows: We let the shuffler commit to BGV re-randomization ciphertexts \( \hat{c}_1, \ldots, \hat{c}_r \) (encryptions of 0) using the linearly homomorphic BDLOP commitment scheme \( \text{Com} \). Together with efficient proofs of well-formedness for the committed re-randomization ciphertexts, this gives us a verifiable shuffle:

1. The shuffler commits to the re-randomization ciphertexts \( \hat{c}_1, \ldots, \hat{c}_r \) as \( \text{Com}(\hat{c}_i) \) and shows that they are well-formed using zero-knowledge proofs.
2. The shuffler computes \( d_i = e_i \pm \hat{c}_i \) and sends shuffled elements \( L = (d_{e(1)}, \ldots, d_{e(r)}) \) to the receiver.
3. Finally, the prover shows that \( L \) is a list of openings of the commitments obtained from \( c_i + \text{Com}(\hat{c}_i) \) using the extended shuffle of commitments to known values.

To prove the well-formedness of the ciphertexts, we utilize proofs of shortness where the proof size is sublinear in the number of ciphertexts \( r \). For this, we use a version of recent amortized proofs of shortness [13]. Unfortunately their construction as-is is suboptimal for our setting, so we adapt and re-prove their protocol.

Verifiable distributed decryption. As explained, a verifiable secret shuffle on its own is usually not sufficient to build a cryptographic voting system. The ciphertexts must also be decrypted, without introducing correctness and privacy problems. Our solution is to distribute the decryption operation in a verifiable way. We hand out key shares of the secret decryption key to each decryption server, and all of them perform a partial decryption of each ciphertext. In addition, we publish commitments to the key shares. The decryption servers then add noise to the partial decryption to hide information about their shares, in a process called noise drowning.

Finally, decryption servers publish the partial decryptions together with a proof of correctness of the decryption, and the plaintexts are computed in public by combining all the partial decryptions.

We use a decryption protocol for BGV ciphertexts that is similar to existing works such as [22]. Their construction is only passively secure. We, therefore, modify the protocol to be resistant to active attacks even if all decryption servers are malicious, and prove it secure. For this, we again utilize an (amortized) zero-knowledge proof of shortness that allows each decryption server to show that it behaved honestly during decryption.

Putting things together. Lattice-based cryptography is very delicate, and one has to be cautious when combining multiple sub-protocols into a larger (voting) construction. This is mainly due to noise in ciphertexts, which can lead to faulty decryptions, overly large parameters, or both.

In our construction, each shuffle adds noise to the ciphertexts, which means that to ensure the correctness of decryption we need to choose parameters based on the number of shuffles and the amount of noise added in each shuffle. Each partial decryption also adds noise to the ciphertexts to hide the secret key. Because of the noise drowning technique, the norm must be quite large, influencing the choice of parameters for the overall construction as well as the choice of zero-knowledge proof techniques involved.

In particular, it is important when measuring performance to use parameters suitable for the complete system, not parameters optimized for individual components only. In order to provide proper context for our contributions, we give a sketch of a full cryptographic voting protocol and provide example parameters. A simplified version could be used as a quantum-safe Helios [1] variant.

Implementation results. Our example parameters assume 4 mix-nodes and 4 decryption nodes. We have estimated the size of each component with respect to the parameters for the full protocol in addition to implementing all sub-protocols, showing that it can be used for large-scale real-world elections where ballots typically are counted and verified in batches of several thousand.

To summarize our implementation results, a ciphertext ballot is of size 80 KB (encoding a vote of size 4096 bits), each mixing proof is of total size 370 KB and each decryption proof is of total size 157 KB, where \( r \) is the number of total ciphertexts. It takes only 0.74 ms to encrypt a ballot, while the mixing proof takes 133.67 ms and the decryption proof takes 101.81 ms. Verification is much faster, with only 12.97 ms for the mixing and 28.59 ms for the decryption. These results improve on the state of the art considerably, see Section 7.

Quantum security. While our work constructs and implements a voting scheme from post-quantum assumptions, we do not claim that it is post-quantum secure. We discuss this in Appendix G.

1.2 Related work

Aranha et al. [4] provide a verifiable shuffle of known commitment openings together with concrete parameters and implementation of a complete voting protocol. However, their trust model has the limitation that the ballot box and the shuffle server must not collude to ensure the privacy of the ballots, which is too restrictive for most real-world settings. This is inherent for the protocol which can not
easily be extended to several shuffles unless layered encryption is used, and this would heavily impact the performance.

Costa et al. [20] design a shuffle with a straight-forward approach similar to Neff [39] based on roots of polynomials. Their protocol requires committing to two evaluations of a polynomial and then proving the correctness of the evaluation using a sequence of multiplication proofs which are quite costly in practice. Farzaliev et al. [27] implements the mix-net by Costa et al. [20] using the amortization techniques by Attema et al. [5] for the commitment scheme by Baum et al. [7]. Here, the proof size is approximately 14 MB per voter, a factor 40 larger than our shuffle proof, even for a smaller parameter set that does not take into account distributed decryption afterward. We expect our shuffle proof to be an additional factor 10 smaller than what we presented above with optimal parameters for the shuffle only \( q \approx 2^{32} \) and \( N = 1024 \) instead of \( q \approx 2^{78} \) and \( N = 4096 \). Furthermore, their proof generation and verification respectively take 1.54 and 1.51 second per vote, which is approximately 18.5 times slower than it takes to produce and verify our shuffle proof in sequence (when normalizing for clock frequency), with parameters that do not take decryption into account.

Recently, Herranz et al. [31] gave a new proof of correct shuffle based on Benes networks and sub-linear lattice-based proofs for arithmetic circuit satisfiability. However, the scheme is not implemented and the example parameters do not take the soundness slack of the amortized zero-knowledge proofs into account. Moreover, [31] does not consider the decryption of ballots, which would heavily impact the parameters of their protocol in practice.

A completely different approach to mix-nets is the so-called decryption mix-nets. The idea is that the input ciphertexts are actually nested encryptions. Each node in the mix-net is then responsible for decrypting one layer of each ciphertext. These can be made fully generic, relying only on public key encryption. Boyen et al. [15] carefully adapt these ideas to lattice-based encryption, resulting in a very fast scheme. Decryption mix-nets are well-suited to applications in anonymous communication. However, for voting applications, they are often less well-suited due to their trust requirements. An important goal for cryptographic voting is universal verifiability: after the election is done, anyone should be able to verify that the ballot decryption was done correctly without needing to trust anyone. This trust issue generalizes to any situation where it is necessary to convince someone that a shuffle has been performed correctly, but no auditor is available. Fast and generic decryption mix-nets such as Boyen et al. [15] need an auditor (potentially distributed) to verify the mix-net, but then it must be trusted during the operation. This conflicts with universal verifiability.

del Pino et al. [24] give a practical voting protocol based on homomorphic counting. They only support yes/no elections, and the total size depends directly on the number of candidates for larger elections. It was shown by Boyen et al. [16] that the protocol in [24] is not end-to-end verifiable unless all tallying authorities and all voters’ voting devices are honest. This problem is solved by [16], but their construction still has the downside of only supporting homomorphic tallying. Strand [45] built a verifiable shuffle for the GSW cryptosystem, but this construction is too restrictive for practical use. Chilloitti et al. [19] uses fully homomorphic encryption, which for the foreseeable future is most likely not efficient enough to be considered for practical deployment.

## 2 BUILDING BLOCKS

In this section, we define the building blocks that we use in our construction of the voting scheme. Then, in Section 3 we show how these can be put together.

Let \( \kappa \) be the computational and sec the statistical security parameter. We define the ring \( R_q = \mathbb{Z}_q[X]/(X^N + 1) \), its norms, the discrete Gaussian distribution \( N \), rejection sampling, and knapsack problems \( SKS_{n,k,j}^2 \) and \( DKS_{n,k,\beta}^2 \) in Appendix A. We use \( S_R \subseteq R_q \) to denote the subset of \( R_q \) where each coefficient is less or equal \( B \).

### 2.1 PKE with Distributed Decryption

We first present a definition of a secure public key encryption (PKE) scheme with a distributed decryption protocol. Such a scheme works like a regular PKE scheme but allows the secret key to be shared among decrypting servers. Then, for a given ciphertext, the decryption servers can compute decryption shares using their key shares which, when combined, reveal the plaintext. The goal here is that the decryption shares do not reveal information about the secret key shares.

**Definition 1 (PKE with Distributed Decryption).** A PKE scheme with distributed decryption consists of five algorithms: key generation (KGen), encryption (Enc), decryption (Dec), distributed decryption (DDec), and combine (Comb), where

- \( \text{KGen} \) on input security parameter \( \kappa^* \) and number of key shares \( \xi_2 \), outputs public parameters \( pp \), a public key \( pk \), a secret key \( sk \), and key shares \( \{sk_i\}_i \).
- \( \text{Enc} \) on input pk and messages \( \{m_i\}_i \), outputs ciphertexts \( \{c_i\}_i \).
- \( \text{Dec} \) on input \( sk \) and ciphertexts \( \{c_i\}_i \), outputs messages \( \{m_i\}_i \).
- \( \text{DDec} \) on a secret key share \( sk_j \) and ciphertexts \( \{c_i\}_i \), outputs decryption shares \( \{ds_{i,j}\}_i \).
- \( \text{Comb} \) on input ciphertexts \( \{c_i\}_i \) and decryption shares \( \{ds_{i,j}\}_i \), outputs either messages \( \{m_i\}_i \) or \( \perp \).

and \( pp \) are implicit inputs to \( \text{Enc}, \text{Dec}, \text{DDec} \) and \( \text{Comb} \).

For such a scheme, we require multiple security properties. (Threshold) correctness and IND-CPA security are standard and we only provide their definitions for completeness in Appendix A.4. Threshold verifiability and decryption simulatability are of more interest, which we define below.

Let \( P_{sk}(c) \) be an efficiently computable predicate that on input secret key \( sk \) and a ciphertext \( c \) outputs 1 or 0. Such a predicate signals that the ciphertext is reliably decryptable - which we need to consider as ciphertexts contain noise. We first define threshold verifiability, which models that distributed decryption is secure against active attacks.

**Definition 2 (Threshold Verifiability).** A PKE scheme with distributed decryption is threshold verifiable with respect to \( P_{sk}(c) \) if an adversary \( \text{Adv} \) corrupting \( J \subseteq \{\xi_2\} \) secret key shares \( \{sk_i\}_i \) cannot convince \( \text{Comb} \) to accept maliciously created decryption shares \( \{ds_{i,j}\}_{i \in J, j \in J} \). More concretely, the following probability is bounded by a negligible \( \epsilon(\kappa) \):

\[
\Pr \left[ \begin{array}{c}
\text{Dec(}sk, \{c_i\}_i) \notin \{\varepsilon\} \\
\text{Comb(}\{c_i\}_i, \{ds_{i,j}\}_{i \in J, j \in J}) \notin \{\varepsilon\} \\
\forall i \in J \quad P_{sk}(c_i) = 1 \\
\forall j \notin J \\
\forall i \in J \\
\forall j \in J \\
\forall \varepsilon \end{array} \right] \leq \epsilon(\kappa),
\]

where the probability is taken over KGen and DDec.
We moreover define a simulation property, that shows that description shares do not leak any information about the secret key. This models security against passive attackers.

Definition 3 (Distributed Decryption Simulatability). A PKE scheme with distributed decryption is simulatable with respect to $P_{\text{Adv}}$ if an adversary $\text{Adv}$ corrupting $T \subseteq \{\zeta\}$ secret key shares $(sk_j)_{j \in T}$ cannot distinguish the transcript of the decryption protocol from a simulation by a simulator $\text{Sim}$ which only gets $(sk_j)_{j \not\in T}$ as well as correct decryptions as input. More concretely, the following probability is bounded by a negligible $\epsilon(\sec)$:

$$\Pr[b = b'] = \frac{1}{2}$$

where the probability is taken over $KGen$, $\text{Dec}$, and $\text{Sim}$.

2.1.1 Our instantiation. Let $p < q$ be primes, define $R_q$ and $R_p$ for a fixed $N$, let $KGen$, $\text{Rec}$ be $\mathbb{Z}/p\mathbb{Z}$ be bounded. We use the BGV [17] encryption scheme, which consists of three algorithms: key generation ($KGen$), encryption ($\text{Enc}$) and decryption ($\text{Dec}$), where:

$KGen$ Samples a uniform element $a \xleftarrow{\$} R_q$, a short $s \xleftarrow{\$} S_{\text{Rec}}$ and noise $e \xleftarrow{\$} S_{\text{Rec}}$. The algorithm outputs the public key $pk = (a, b) = (a, s + pe)$ and secret key $sk = s$.

$\text{Enc}$ On input the public key $pk = (a, b)$ and a message $m \in R_p$, samples a uniform $r \xleftarrow{\$} S_{\text{Rec}}$, noise $e', e'' \xleftarrow{\$} S_{\text{Rec}}$ and outputs the ciphertext $c = (u, v) = (ar + pe', br + pe'' + m)$.

$\text{Dec}$ On input secret key $sk = s$ and ciphertext $c = (u, v)$, outputs message $m = (v - su) \mod q$.

The following theorem follows from [17] and [38].

Theorem 1. The BGV encryption scheme is correct if $\|v - su\|_{\infty} \leq B_{\text{Dec}} < \lfloor q/2 \rfloor$, and IND-CPA secure if the DKS $N_{\delta_{2}, \beta}$ problem is hard for some $\beta = \beta(N, q, S_{\text{Rec}}, B_{\text{Err}}, p)$.

We use this theorem to define the predicate $P_{\text{Adv}}(u, v)$ to be 1 iff $\|v - su\|_{\infty} < B_{\text{Dec}}$ and otherwise 0. Since each ciphertext consists of 2 elements from $R_q$, it can be represented using $2N \log_q q$ bits.

2.1.2 Threshold decryption. We quickly recap the passively secure distributed decryption protocol by Damgård et al. [9, 21, 22]. Here, the $KGen$ algorithm on input $\epsilon \xleftarrow{\$} \mathbb{N}$ additionally outputs uniformly random shares $sk_j = s_j$ of the secret key $sk = s$ such that $s = s_1 + \cdots + s_\epsilon$ in $R_q$. This defines a passively secure threshold decryption protocol by using the linearity of the decryption function:

$\text{DDec}$ On input a secret key-share $sk_j = s_j$ and a ciphertext $c = (u, v)$, does the following:

1. Compute $m_j = s_j$ and sample a uniformly random $E_j \xleftarrow{\$} R_q$ such that $\|E_j\|_{\infty} \leq 2\epsilon E_{\text{Dec}}(\|\epsilon\|_{\infty})$ for statistical security parameter $\sec$ and noise-bound $B_{\text{Dec}}$.
2. Output $ds_j = t_j = m_j + pe_j$.

Comb On input ciphertext $c = (u, v)$ and set of decryption shares $(ds_j)_{j \in \epsilon}$, outputs message $m = (v - t \mod q)$.

The following theorem follows from [21, 22].

Theorem 2. Let $\sec$ be the statistical security parameter. The distributed BGV encryption scheme is correct for input ciphertexts with $\|v - su\|_{\infty} \leq (1 + 2\epsilon E_{\text{Dec}})B_{\text{Dec}} < \lceil q/2 \rceil$, and is decryption simulatable against passive adversaries (i.e. fulfills Definition 3).

Each partial decryption consists of one element from $R_q$, namely the output of $\text{DDec}$, which means that the output from the passively secure protocol is of size $N \log_2 q$ bits per party.

This scheme is not secure against active adversaries, i.e. it does not have threshold verifiability. We, therefore, modify it in Section 6 to withstand active attacks.

2.2 Commitments

Commitment schemes were first introduced by Blum [11], and we use these at multiple points in this work to achieve verifiability.

Definition 4 (Commitment Scheme). A commitment scheme consists of three algorithms: key generation (Setup), commitment (Com) and opening (Open), where:

Setup On input security parameter $\epsilon$, outputs public params $pp$.

Com On input message $m$, outputs commitment $c$ and opening $r$. On input $m$, $c$ and $r$, outputs either 0 or 1, and the public parameters $pp$ are implicit inputs to Com and Open.

For the commitment scheme, we require that it is correct, binding, and hiding. Correctness means that an honestly generated commitment is always accepted by the opening algorithm. Binding requires that no PPT adversary can provide two different valid openings of a given commitment for different messages. Hiding means that the commitment itself does not reveal any information about the committed value. We provide these definitions for completeness in Appendix A.5.

2.2.1 Our instantiation. Our work uses the BDLOP [7] commitment scheme. Let $R_q$ be defined as above and let $N_{\delta_{2}}$ be a Gaussian distribution with standard deviation $\sigma_{c}$ and $B_{\text{Com}}$ be a noise bound. The algorithms are defined as follows:

Setup Outputs a $pk$ which allows to commit to length-$l_c$ messages from $R_q^{l}$ using length-$k$ randomness from $S_{\text{Com}}^{l_c}$ outputting length-$(n + l_c)$ vectors. For this, we define:

$$\text{AC}_{1} = I_n \quad \text{AC}_{2} = I_k$$

$$\text{AC}_{c} = \begin{cases} R_{n}(k-n) & \text{if } c < \frac{n}{2} \\ R_{n}(k-n) & \text{if } c > \frac{n}{2} \end{cases}$$

Let $pk = \text{AC} = \{\text{AC}_{1}, \text{AC}_{2}\}$. $\text{AC}$ has height $n + l_c$ and width $k$.

Com On input messages $m \in R_q^{k}$ samples $r \xleftarrow{\$} S_{\text{Com}}^{l_c}$ and computes

$$\text{Com}(m, r) = \text{AC} \cdot m + I_k$$

Com outputs $m$ and the opening $d = (m, r, m)$.

Open Verifies whether an opening $(m, r, m)$ is correct by checking that $\|m\|_{\infty} \leq 4\sigma_{c} \sqrt{N}$, for $i \in [k]$, and if

$$f : \{c_1, c_2\} \xrightarrow{\$} \text{AC} \cdot m + f : \{0, 1\}$$

It outputs 1 if all conditions hold, and 0 otherwise.
We define the set $\mathcal{C}$ in Appendix A. The openings generated by Com form a subset of those accepted by Open, which is necessary for efficient zero-knowledge proofs on BDLOP commitments. Observe that Open always accepts honestly generated openings (except with negligible probability) by setting $f = 1$. The following theorem follows from Baum et al. [7].

**Theorem 3.** The aforementioned commitment scheme is computationally hiding if the DKS$_{n,k,R}$ problem is hard, and the scheme is computationally binding if the SKS$_{n,k,16:\sqrt{q},V}$ problem is hard.

Each commitment consists of $n + l_c$ elements from $R_q$ and can hence be represented using $(n + l_c)N\log_2 q$ bits.

### 2.3 Zero-Knowledge Proofs

Zero-Knowledge (ZK) proofs were first introduced by Goldwasser et al. [30]. They are cryptographic protocols to show that a certain statement is true, without revealing the witness. We use ZK proofs in our constructions to achieve verifiability: protocol participants show that they indeed followed the protocol steps correctly, while in our constructions to achieve verifiability: protocol participants

#### 2.3.1 Linear relations among commitments.

Assume that there are non-interactive using the Fiat-Shamir transform [28].

Let $L$ be a language, and let $R$ be an NP-relation on $L$. Then, $x$ is an element in $L$ if there exists a witness $w$ such that $(x, w) \in R$. We let $P$, $P^*$, $V$, and $V^*$ be polynomial time algorithms.

**Definition 5 (Interactive Proofs).** An interactive proof protocol $\Pi$ consists of two parties: a prover $P$ and a verifier $V$, and a setup algorithm (Setup), where Setup, on input the security parameter $\lambda$, outputs public setup parameters $sp$. The protocol consists of a transcript $T$ of the communication between $P$ and $V$, with respect to $sp$, and the conversation terminates with $V$ outputting either 1 or 0. Let $(P(sp, x, w), V(sp, x))$ denote the output of $V$ on input $x$ after its interaction with $P$, who holds a witness $w$.

We call an Interactive Proof a Zero-Knowledge proof if it has the following three properties:

**Completeness:** If $P$ has a valid witness $w$ such that $(x, w) \in R$, then $V$ accepts.

**Knowledge Soundness:** If $P^*$ can make an honest verifier accept with large enough probability for statement $x$, then there exists a polynomial-time algorithm $E$ that can, through black-box access to $P^*$, extract $w$ such that $(x, w) \in R$.

**Honest Verifier Zero Knowledge:** There exists a PPT algorithm $S$, called simulator, that given only $x$ can create transcripts whose distribution is indistinguishable from those of an honest prover and verifier.

We give the formal definitions in Appendix A.6. Note that an interactive honest-verifier zero-knowledge proof protocol can be made non-interactive using the Fiat-Shamir transform [28].

#### 2.3.2 Amortized Proofs of Boundedness.

It is well-known that polynomials in $R_q$ can be represented as vectors in $\mathbb{Z}_q^N$ and multiplication by a polynomial $\hat{a}$ in $R_q$ can be expressed as a matrix-vector product with a negacyclic matrix in $\mathbb{Z}_q^{N \times N}$. Let $A$ be a publicly known $r \times o$ matrix over $R_q$, that is, a $R_q \times o$ matrix over $\mathbb{Z}_q$. We will now consider how to prove generically in zero-knowledge that $t_i = A_{si}$ for bounded $s_i$ and known $t_i$ over $\mathbb{Z}_q$. This is the same as proving correct multiplication over the ring $R_q$ of the respective elements. We use proofs that are amortized, meaning that the proof size is sublinear in the number $\tau$ of individual statements that we prove. Both the BGV encryption and BDLOP commitment can be expressed in this form and require bounds on inputs for correctness, so this ZK proof can be used to show that encryptions or commitments were honestly made.

Let $A$ be a publicly known $r \times o$-matrix over $R_q$, let $s_1, s_2, \ldots, s_r$ be bounded elements in $R_q$ and let $A_{si} = t_i$ for $i \in [r]$. Letting $S$ be the matrix whose columns are $s_i$ and $T$ be the same matrix for $t_i$, but defined over $\mathbb{Z}_q$ instead of $R_q$, then [6] give an efficient amortized zero-knowledge proof of knowledge for the relation

$$R_{BND} = \left\{(x, w) \mid x = (A(T) \wedge w = S \wedge \forall i \in [r]: t_i = A_{si} \wedge ||s_i|| \leq 2 \cdot B_{BND}^2 \right\}.$$ 

Let

$$\pi_{BND} \leftarrow \Pi_{BND}(S; (A(T), \pi_{BND})), 0 \vee 1 \leftarrow \Pi_{BND}\left((A(T), B_{BND}); \pi_{BND}\right),$$

denote the run of the proof and verification protocols, respectively, where the $\Pi_{BND}$-protocol, using Fiat-Shamir, produces a non-interactive proof of the form $\pi_{BND} = (C, Z)$, where $C$ is the output of a hash function, and the $\Pi_{BND}$-protocol verifies the NIZK $N_{\pi_{BND}}$ is a Gaussian distribution over $\mathbb{Z}$ with standard deviation $\sigma_{BND}$ and $B_{BND} = \sqrt{2\sqrt{q}}$. See Appendix B for more details.

#### 2.3.3 Exact Amortized Proofs of Shortness.

As can be seen from $R_{BND}$ the non-exact amortized proof has the disadvantage of introducing a “slack” factor $B_{BND} = \sqrt{2\sqrt{q}}$, meaning that the proven bound is substantially larger than what an honest party would generate. This ultimately leads to larger parameters for any application that uses $B_{BND}$, as one always has to assume that dishonestly provided encryptions or commitments only fulfills the larger bound.
We will therefore also use a tighter ZK amortized proof of shortness which shows $R_{\text{Bnd}}$ for the $\ell_2$-norm and with $B_{\text{Bnd}}$ being 1. The disadvantage of this proof, over $\Pi_{\text{Bnd}}$, is that it does not scale as well with the number of statements that are proven as $\Pi_{\text{Bnd}}$.

For our exact amortized proof, we use a version of the protocol from Bootle et al. [13]. They give an efficient amortized sublinear zero-knowledge proof for proving the knowledge of short vectors $s_i$ and $e_i$ over $\mathbb{Z}_q$ satisfying $\text{AS}_s + e_i = t_i$. We adapt their techniques for the case where $e_i$ is zero and always prove that $\|s_i\|_{\infty} \leq 1$. Our amortized protocol will be denoted throughout this work as $(\Pi_{\text{SMALL}}, \Pi_{\text{SMALLV}})$. These modifications are non-trivial and require us to re-prove that the construction is a ZK proof. We present more details in Section 4.

### 2.4 Verifiably Shuffling Ciphertexts

We construct a shuffle of BGV ciphertexts $c_1, \ldots, c_r$ as follows:

1. The shuffle server creates encryptions $c'_1, \ldots, c'_r$ of 0 and commits to each $c'_i$ as $\text{Com}(c'_i)$. Then, by homomorphically adding $c_i$ to $\text{Com}(c'_i)$ we obtain commitments $\text{Com}((\hat{c}_i))$ to the same plaintexts as in $c_1, \ldots, c_r$, with “fresh” randomness.

2. The shuffle server reveals the openings $\hat{c}_i$, but in random order. It then runs the verifiable shuffle protocol of [4] to prove that these openings are indeed the correct openings of the commitments.

To make the full construction verifiable, we use additional zero-knowledge proofs: the shuffle server will have to show that the $\text{Com}((\hat{c}_i))$ are valid BDLOP commitments with bounded noise and contain well-formed encryptions of 0 (i.e. have small noise as well). For this, we use the ZK proofs introduced in the previous subsection. But this is insufficient because the protocol of [4] only supports BDLOP commitments of single elements from $R_{\eta}$, while BGV ciphertexts consist of two elements from $R_{\eta}$. We, therefore, extend the shuffle protocol by Aranha et al. to verifiably shuffle vectors in $R_{\eta}^2$. The full construction is described in Section 5.

### 2.5 Verifiable Decryption

We verifiably decrypt the BGV ciphertexts containing the votes in the voting scheme. In order to avoid a single party that has the secret decryption key (and could decrypt the inputs into the mix-net) we secret-share the key among multiple decryption servers.

The decryption algorithm introduced in Section 2.1 is only passively secure, but we assume that attackers may act maliciously in the voting scheme. We, therefore, modify the passively secure decryption protocol as follows:

- During key generation, a BDLOP commitment to each share is generated and published. The opening information is given to the shareholder.

- Each decryption share will additionally contain a proof that the decryption share is well-formed; the decryption algorithm proves that the decryption share is generated using the committed key share and that the randomness used is bounded. We will use the ZK proofs in Section 2.3.

We fully describe these transformations and prove them secure in Section 6. We do, however, not implement the (verifiable) key generation for our construction, which can e.g. be obtained by modifying a threshold key generation protocol such as [41].

### 3 THE VOTING SCHEME

The high-level architecture for the counting phase of our protocol is shown in Figure 1. As it follows a standard design [44], we do not describe its security properties further here, but refer the reader to Appendix H for a more formal treatment. We also have left out some aspects, such as voter authentication, to focus on the core building blocks of our construction.

![Figure 1: The high-level counting phase of our voting protocol.](image)

The voting protocol requires a trusted set of players to run the setup, a set of voters $\text{Voter}_i$ and their computers $\text{Comp}_i$, a ballot box $\text{Ballot}$, a collection of shuffle servers $S_k$, a collection of decryption servers $D_j$ and one or more auditors $\text{Audit}$. We will assume that there are $\xi_1$ shuffle servers and $\xi_2$ decryption servers in total. The voting protocol consists of a setup phase, a registration phase, a casting phase, a counting phase as well as a verification algorithm to check casting and counting.

**Setup Phase.** A trusted set of players runs the key generation algorithm $KGen$ of the PKE scheme with Distributed Decryption. The key generation can either be done in a trusted fashion or distributed using the protocol by Rotaru et al. [41]. The derived public parameters $pk$ are given to every participant, while the decryption key shares $sk_j$ are given to the decryption servers $D_j$.

A key pair $(sk_{\mu}, vk_{\mu})$ for a EUF-CMA-secure signature scheme is also given and generated to the ballot box. The verification key $vk_{\mu}$ is given to every participant.

**Casting Phase.** Each voter $\text{Voter}_i$ instructs its computer $\text{Comp}_i$ which ballot to cast. The computer encrypts the ballot under the public key $pk$ and creates a ballot proof, sending the encrypted ballot and proof to the box $\text{Ballot}$. The ballot proof is tied to the voter’s identity and is supposed to stop copy-and-paste attacks against privacy. In the security proof, the ballot proof must allow us to...
extract ballots from adversarially generated encryptions. Either we can use an argument of knowledge, but to simplify the security proof we often encrypt the ballot under two distinct keys and use an argument of equality. The ballot box will check the proof and signs the encrypted ballot and the proof using \( sk_B \). This signature \( \sigma_1 \) is sent to the voter’s computer. The computer verifies the signature \( \sigma_1 \) from Ballot using \( vk_B \) and only accepts if it is valid. It then shows the voter the encrypted ballot, proof, and signature, which constitutes the voter’s receipt. The voter \( \text{Voter}_i \) accepts the ballot as cast if and only if the computer accepts it with a receipt.

**Counting phase.** The ballot box \( \text{Ballot} \) sends the encrypted ballots and ballot proofs that it has seen to the auditor \( \text{Audit} \) as well as every decryption server \( D_j \). Ballot then sorts the list of encrypted ballots \( \{c_i^{(0)}\} \) and sends this to the first shuffle server \( S_1 \) and every decryption server. If some voter has cast more than one ballot, only the encrypted ballot seen last is included in this list.

The \( \ell_1 \) shuffle servers \( S_1, S_2, \ldots, S_{\ell_1} \) consecutively use the shuffle algorithm on the input encrypted ballots \( \{c_i^{(k-1)}\} \), passing the shuffled and re-encrypted ballots \( \{c_i^{(k)}\} \) to the next shuffle server. They also pass the shuffled re-encrypted ballots \( \{c_i^{(k)}\} \) and the shuffle proof \( \pi_S \) to \( \text{Audit} \) and every decryption server.

Each decryption server verifies that the data from Ballot as well as each shuffle server is consistent (input-output wise), and that every shuffle proof \( \pi_S \) verifies for the respective ciphertexts. Only then will they run the distributed decryption algorithm \( \text{DDec} \) with their decryption key share \( sk_j \) and send their partial decryption shares \( t_i,j \) of each ballot \( c_i^{(\ell_1)} \) to the auditor as well as each recipient of the output. To obtain the result, each recipient can then run \( \text{Comb} \) on the partial decryption shares \( t_i,j \).

**Verification.** The auditor verifies the data from Ballot (it checks that the ballot proofs of knowledge verify), that the encrypted ballots received by the first shuffle server are consistent with the data from Ballot, that every shuffle proof verifies, and then runs the combining algorithm \( \text{Comb} \) on the received partial decryption shares \( t_i,j \) from each \( D_j \). If all checks pass then the auditor accepts, otherwise, it rejects. Finally, \( \text{Audit} \) outputs the list of messages, including public key material, as its transcript.

One can easily design a verification algorithm that takes as input a transcript, a result, and optionally a receipt, and either accepts or rejects. The verification algorithm simply runs the auditor with the public key material and the messages listed in the transcript and checks if the auditor’s result matches the input. If a receipt is present, it also verifies the signature \( \sigma_1 \) using the ballot box’s verification key \( vk_B \). Checks that the encrypted ballot and ballot proof are present in the ballot box data set, and that the encrypted ballots are present in the first shuffle server’s input.

Note that there are many variations of this protocol. It can be used with so-called return codes, which allow human verification of the vote cast and detect a cheating computer \( \text{Comp}_p \) of the voter.

Many comparable schemes are phrased in terms of an ideal bulletin board, where every player posts their messages. Implementing a bulletin board is tricky in practice, so instead, we have described the scheme as a conventional cryptographic protocol passing messages via a network.

It is worth noting that for our concrete scheme, anyone can redo the auditor’s work (since no secret key material is involved) by running the verification algorithm (and parts of the code algorithm) on the public data, making the voting protocol (universally) verifiable.

### 4 Exact Amortized ZK Proofs

Boote et al. [13] give an efficient amortized sublinear zero-knowledge protocol for proving the knowledge of short vectors \( s \) and \( e_i \) over \( \mathbb{Z}_q \) satisfying \( As + e_i = t_i \). For our setting, we adapt their techniques for the case where \( e_i \) is zero, and prove that \( \|s_i\|_\infty \leq 1^3 \).

We explain the main idea of [13] for proving knowledge of a preimage \( s \) of \( t = As \) and then generalize to an amortized proof for \( \tau \) elements with sublinear communication.

The approach follows an ideal linear commitments-technique with vector commitments \( \text{Com}_L(\cdot) \) over \( \mathbb{Z}_q \). The prover initially commits to the vector \( s \) as well as an auxiliary vector \( s_0 \) of equal length. Implicitly, this defines a vector of polynomials \( f(x) = s_0(x) + s \) for the prover. Now consider the vector of polynomials \( f(x) \circ (f(x) - 1) \circ (f(x) + 1) \), where \( \circ \) denotes the coordinate-wise product, then the coefficients of \( X^0 \) are exactly \( s \circ (s - 1) \circ (s + 1) \) and therefore \( 0 \) if and only if the aforementioned bound on \( s \) holds.

In that case, each aforementioned polynomial in \( f(x) \circ (f(x) - 1) \circ (f(x) + 1) \) is divisible by \( X \). Therefore, the prover computes the coefficient vectors

\[
1/X \cdot f(X) \circ (f(X) - 1) \circ (f(X) + 1) = v_2 X^2 + v_1 X + v_0
\]

and commits to these. Additionally, define the value \( d = t - Af = -As_0 \), which the prover also commits to.

The verifier now sends a challenge \( \vec{x} \), for which the prover responds with \( \vec{f} = f(x) \). The prover also uses the linear property of the commitment scheme to show that:

1. \( \text{Com}_L(s_0) \cdot x + \text{Com}_L(s) \) opens to \( \vec{f} \).
2. \( \text{Com}_L(v_2) \cdot x^2 + \text{Com}_L(v_1) \cdot x + \text{Com}_L(v_0) \) opens to the value \( \frac{1}{x} \cdot \vec{f} \circ (f(x) + 1) \circ (f(x) - 1) \).

The prover additionally opens the commitment to \( d \) and the verifier checks that it opens to \( \frac{1}{x} \cdot (t - Af) \). Here, the first two commitment openings allow us to deduce that the correct \( \vec{f} \) is sent by the prover and that the values that committed as \( s \) are indeed commitments to \( \{-1, 0, 1\} \). Then, from opening \( d \) we get that the committed \( s \) is the preimage of \( t \) under \( A \).

The ideal linear commitments in [13] get realized using an Encode-then-Hash scheme. In this commitment scheme, the prover commits to vectors \( x_1, \ldots, x_n \in \mathbb{Z}_q^{l_{\text{msg}}} \):

1. Sample \( n \) random vectors \( r_1, \ldots, r_n \in \mathbb{Z}_q^n \).
2. Let \( \text{Encode} \) be the encoding function of an \( [l, l_{\text{msg}} + \eta, d] \) Reed-Solomon Code with code-length \( l \), message length \( l_{\text{msg}} + \eta \) and minimal distance \( d \). Compute \( e_i \leftarrow \text{Encode}(x_i) || r_i \) for each \( i \in [n] \).
3. Construct matrix \( E = \text{RowsToMatrix} (e_1, \ldots, e_n) \) where \( e_i \) is row \( i \).
4. Commit to each column of \( E \) using a hash, then compress all commitments to Merkle root \( M \).
5. Send \( M \) to the verifier.

\footnote{The authors of [13] mention that this optimization is possible, but neither present the modified protocol nor a proof.}
For the prover to show to the verifier that \( x \) is an opening of the linear combination \( \sum_{i=1}^{n} \gamma_i x_i \):

1. It computes \( r = \sum_{i=1}^{\ell} \eta_i f_i \) and sends \( r \) to the verifier.
2. The verifier chooses a subset \( I \) of size \( \eta \) from \([1]\).
3. The prover opens the commitment for each column \( i \in I \) of \( E \) and proves that it lies in the Merkle tree \( M \) by revealing the path.
4. The verifier checks that \( \text{Encode}(x|r) \) coincides at position \( i \) with the respective linear combination of all \( n \) opened values in column \( i \) of \( E \).

This is a proof of the respective statement due to the random choice of the set \( I \). Intuitively, if each row of \( E \) is in the code\(^3\), but they do not sum up to \( x \), then the linear combination of the codewords in \( E \) must differ from \( \text{Encode}(x|r) \) in at least \( \delta \) positions, which is the minimum distance of the code. By the random choice of \( I \) and by setting \( \eta \) appropriately, the verifier would notice such a disagreement entry with high probability. At the same time, because only \( \eta \) columns of \( E \) are opened, this leaks no information about the vectors \( x_1, \ldots, x_\ell \) if the evaluation points of the output of \( \text{Encode} \) are different from those of the input, i.e. if the code is not systematic.

For the case of more than one secret, the prover wants to show that \( t_1 = A s_i \) for \( \tau \) values \( t_1 \) known to the verifier, subject to \( s_i \) again being ternary vectors. The goal is to establish the latter for all \( t_1 \) simultaneously while verifying only one equation and sending only one vector \( f \). Then the prover commits to \( s_j \) as well as an additional blinding value \( s_0 \). Let \( a_1, \ldots, a_\ell \in \mathbb{Z}_2 \) be distinct interpolation points and define the \( i \)th Lagrange polynomial

\[
\ell_i(x) = \prod_{j \neq i} \frac{x - a_j}{a_i - a_j}.
\]

Additionally, let \( q_0(x) = \prod_{j=1}^{\ell} (X-a_i) \). Then every \( f \in \mathbb{Z}_2[X]/q_0(X) \) can be written uniquely as \( f(X) = \sum_{i=0}^{\ell} \lambda_i \ell_i(X) \) and any \( g \in \mathbb{Z}_2[X]/q_0(X) \) as a linear combination of \( \{ \ell_i(X) \ell_j(X) \}^{i,j=0}_{\ell-1} \). Define the polynomial

\[
f(X) = \sum_{i=0}^{\tau} s_i \ell_i(X),
\]

and observe that \( f(X) \circ (f(X) = 1) \circ (f(X) + 1) \) is divisible by \( q_0(X) \) if and only if all \( \ell_i(X) \)-coefficients of \( f(X) \) for \( i \in [\tau] \) are 0.

Additionally, since \( \ell_i(X) \cdot \ell_j(X) = 0 \mod q_0(X) \) if \( i, j \in [\ell] \) and \( i \neq j \) then also implies that the \( s_i \) are ternary. Moreover, we only have to commit to additional \( 3 \cdot r \) coefficients of \( \{ \ell_i(X) \ell_j(X) \}^{i,j=0}_{\ell-1} \) to prove well-formedness of any evaluation of \( f(X) \) sent by the prover.

The protocol is described in detail in Figure 5 in Appendix C. As our construction substantially deviates from that of [13] we show that the protocol indeed is a ZKPoK. In Appendix C we show that the following holds:

**Theorem 4.** The amortized zero-knowledge proof of exact openings is complete when the secrets \( s_j \) has ternary coefficients, it is special sound if the SKS\(_{\ell=0,1} \) problem is hard and the hash-function is collision-resistant, and it is statistically honest-verifier zero-knowledge.

Towards defining the size of the proof, we see that the proof size is dominated by the sending of the homomorphic commitments (step 9 in Figure 5) and the opening of the column-wise commitments of \( E \) via Merkle tree paths (step 11).

- In step 9, prover sends polynomials which are openings to the homomorphic commitments of total size \( 3\sigma N \log_2 q \) and additional randomness of total size \( 3\eta \log_2 q \).
- In step 11, the preimages of the hash column commitments \( (E_i) \) have length \( (3r + 2\eta)\log_2 q \) while the Merkle tree paths add another \( 2k \eta(1 + \log_2 \ell) \) bits.

This leads to a proof of size

\[
(3\sigma N + (3r + 2\eta)\log_2 q + 2k \eta(1 + \log_2 \ell) \text{ bits}) \quad (1)
\]

in total. The second part is essentially independent of \( r \), which decides how good the proof amortizes after fixing the lattice components. We get the optimal result by setting \( 3\sigma N \approx (3r + 2\eta) \).

## 5 VERIFIABLE SHUFFLE OF CIPHERTEXTS

The recent work by Aranha et al. [4] presents an efficient protocol \( \Pi_{\text{SHUFF}} \) for a shuffle of openings of the lattice-based commitments from Section 2.2 using proofs of linear relations. The protocol of [4] only supports committed secrets coming from \( R_q \). We now extend their protocol to verifiably shuffle vectors in \( R_q^\ell \).

### 5.1 The Extended Shuffle for Commitments

To prove a shuffle, both the prover and verifier are given a list of commitments \( [m_1], \ldots, [m_r] \) as well as potential messages \( (\hat{m}_1, \ldots, \hat{m}_r) \) from \( R_q \). The prover additionally obtains openings \( m_i, r_i, \hat{f}_i \) and wants to prove that the set of plaintext elements is the same set as the underlying elements of the commitments for some secret permutation \( \pi \) of the indices in the lists. More formally, our goal is to prove the following relation

\[
\mathcal{R}_{\text{SHUFF}} = \{(x, w) \mid \begin{array}{c}
x = ([m_1], \ldots, [m_r], \hat{m}_1, \ldots, \hat{m}_r), \\
\forall i \in [r] : f_i \cdot (m_{\pi^{-1}(i)} - \hat{m}_{\pi^{-1}(i)}) = r_i \cdot f_i \cdot c_{\pi^{-1}(i)} \cup c_{\pi^{-1}(i)} \cup c_{\pi^{-1}(i)} \\
= \mathcal{A}c_{r_1} + f_i \cdot 0 \hat{m_i} \land |r_i| \geq 6\sigma c_{\sqrt{\mathcal{N}}}
\end{array}\}.
\]

Towards proving this relation, we observe that it is sufficient to let the verifier choose a random element \( h \leftarrow R_q \). Then instead of proving a shuffle on \( m_1, \ldots, m_r \), the prover instead performs the proof on \( (m_1, \rho), \ldots, (m_r, \rho) \) where \( \rho = (1, h, \ldots, h^{r-1})^T \). The problem with this approach is that we must also be able to apply \( \rho \) to the commitments \( [m_1], \ldots, [m_r] \), without re-committing to the inner messages and proving correctness in zero-knowledge.

Since each commitment \( [m] \) can be written as

\[
\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \mathcal{A}c_r + 0 \begin{bmatrix} m \end{bmatrix}
\]

we can write \( c_1 = \mathcal{A}c_r \) and \( c_2 = \mathcal{A}c_d + m \). From this we can create a new commitment \( [\rho, m] \) under the new commitment key \( pk' = (\mathcal{A}c_1, \rho \mathcal{A}c_2) \) where \( c'_1 = c_1 \) remains the same, while we set \( c'_2 = (\rho, c_2) \). This does not increase the bound of the randomness.
of the commitment. Since 
\[
A_{C,2} = \begin{bmatrix} 0^{k \times n} & I_k & \lambda_3 \end{bmatrix} \text{ where } \lambda_3 \in \mathbb{R}^{(k-n-l_k)},
\]
it holds that 
\[
a'_2 = \rho A_{C,2} = \begin{bmatrix} 0^n & \rho^T & \rho \lambda_3 \end{bmatrix}.
\]
It is easy to see that breaking the binding property for \(pk'\) is no easier than breaking the binding property for \(pk\).

**Proposition 1.** If there exists an efficient attacker \(Adv\) that breaks the binding property on commitments under the key \(pk'\) with probability \(\epsilon\), then there exists an efficient algorithm \(Adv'\) that breaks the binding property on \(pk\) with the same probability.

We can now construct the protocol \(\Pi_{SHUFF}^k\):

1. Initially, prove \(P\) and verifier \(V\) hold \(\langle [m_i], \hat{m}_i \rangle \in [r]\) for a public key \(pk = (A_{C,1}, A_{C,2})\) while the prover additionally hold secrets \([m_i, r_i] \in [r], \pi \in S_r\).
2. \(V\) chooses \(h \stackrel{\$}{\leftarrow} R_q\) and sends it to \(P\). Both parties compute 
\[
p = (1, h, \ldots, h^{k-1})^T.
\]
3. \(P\) and \(V\) for each \([m_i] = (c_{1,i}, c_{2,i})\) compute \([p, m_i] = (c_{1,i}, p, c_{2,i})\) and messages \([p, \hat{m}_i]\). \(P\) uses same permutation \(\pi\), randomness \(r_i\) as before. The commitment key \(pk' = (A_{C,1}', A_{C,2}')\) is used by both.
4. If \(\Pi_{SHUFF}^k\) accepts then \(V\) accepts in \(\Pi_{SHUFF}^k\), otherwise rejects.

We show the following in Appendix D:

**Lemma 1.** Assume that \(\Pi_{SHUFF}^k\) is an HVZK Proof of Knowledge (PoK) for the relation \(R_{SHUFF}\) with soundness error \(\epsilon'\). Then \(\Pi_{SHUFF}^k\) is a HVZK PoK for the relation \(R_{SHUFF,k}\) with soundness error \(\epsilon = 2\epsilon' + 3\left(\frac{k-1}{q}\right)^N\).

### 5.2 Verifiable Shuffle of BGV Ciphertexts

We now implement the verifiable shuffle for ciphertexts that we outlined in Section 2.4. To recap quickly, the idea behind the shuffle of BGV ciphertexts \(c_1, \ldots, c_k\) is as follows:

1. The shuffle server creates encryptions \(c'_1, \ldots, c'_k\) of 0 and commits to each \(c'_i\) as \(\text{Com}_c\). Then, by homomorphically adding \(c_i\) to \(\text{Com}_c\), we obtain commitments \(\text{Com}_c\) to the same plaintexts as in \(c_1, \ldots, c_k\), with “fresh” randomness.
2. The shuffle server reveals the openings \(\hat{c}_i\), but in random order. It then runs the verifiable shuffle protocol from the previous subsection to prove that these openings are indeed the commitments’ correct (permutated) openings.

In the following, we describe the resulting approach in more detail.

**Public parameters.** Let \(p \ll q\) be primes, let \(R_q\) and \(R_p\) be defined as above for a fixed \(N\), and let \(B_{\text{Dec}}, B_{\text{Err}} \in \mathbb{N}\) be bounds for an instance of our chosen PKE scheme. We assume properly generated keys and ciphertexts according to the KeyGen and Enc algorithms in Section 2.1.

The shuffle server \(S\) takes as input a set of \(r\) publicly known BGV ciphertexts \(\{c_i\}_{i=1}^r\), where the total noise in each ciphertext is bounded by \(B_{\text{Dec}}, \i.e. each ciphertext fulfills \(P_{sk}()\).

**Randomizing.** First, \(S\) randomizes all the received ciphertexts. Towards this it creates a new set of ciphertexts \(\{c'_i\}_{i=1}^r\):
\[
c'_i = (u'_i, v'_i) = (ar'_i + pe'_{i,1}, br'_i + pe'_{i,2}),
\]
where \(r'_i \leftarrow S_{\text{BErr}}, \text{and } c'_i, e'_{i,2} \leftarrow S_{\text{BErr}}\), as in fresh ciphertexts. This corresponds to creating fresh, independent encryptions of 0. \(S\) will not publish these \(c'_i\).

**Committing.** \(S\) now commits to the \(c'_i\). Towards this, we re-write the commitment matrix from Section 2.2 for \(k = 2\) and add the public key of the encryption scheme to get a \((n+2) \times (k+3)\) matrix \(\mathbf{A}_M\), where \(0^n\) are row-vectors of length \(n\), \(a_{1,1}, a_{1,2}\) are column vectors of length \(n\), and \(a_{2,3}, a_{3,3}\) are row vectors of length \(k-n-2\) and \(A_{1,3}\) is of size \(n \times (k-n-2)\). Then,
\[
\text{Com}(u'_i, v'_i) = \left[\begin{array}{c} (ar'_i + pe'_{i,1}, br'_i + pe'_{i,2}) \\ a_1^{0} & a^{0} & 0 & 0 & 0 \\ a^{0} & a^{0} & 0 & a^{0} & 0 \end{array} \right] \in \mathbb{R}^{2r \times 5},
\]
where \(r_i \in \mathbb{R}^k\) is the randomness used in the commitment. Further, let \((u_i, v_i) \leftarrow S_{\text{BErr}}\) be the trivial commitment to \((u_i, v_i)\) with no randomness. Then, given the commitment \(\text{Com}(u'_i, v'_i)\) and \((u_i, v_i)\) we can compute a commitment
\[
\left[\begin{array}{c} \hat{u}_i, \hat{v}_i \end{array} \right] = \left[\begin{array}{c} (u_i, v_i) \end{array} \right] o + \left[\begin{array}{c} (u'_i, v'_i) \end{array} \right].
\]
Thus, the commitments \(\text{Com}(u'_i, v'_i)\) contain re-randomized encryptions of the original ciphertexts. \(S\) can therefore open a permutation of the \((\hat{u}_i, \hat{v}_i)\) and prove correctness of the shuffled opening using algorithm \(\Pi_{SHUFF}^k\). To ensure correctness, we have to additionally show that each \(u'_i, v'_i\) was created so that decryption is correct, i.e., it has small enough noise.

**Proving correctness of commitments.** Let \(\mathbf{A}_M\) be the \((n+2) \times (k+3)\) matrix defined above. Then \(S\) needs to prove that, for all \(i\), it knows secret short vectors \(r'_i\) of length \(k+3\) that are solutions to the following equations:
\[
t_i = \mathbf{A}_M r'_i = \left[\begin{array}{c} (ar'_i + pe'_{i,1}, br'_i + pe'_{i,2}) \end{array} \right], \|r'_i\|_\infty \leq B_{\text{Dec}}.
\]
To show this, \(S\) runs the \(\Pi_{\text{SMALL}}\)-protocol on \(\mathbf{A}_M, \{r'_i\}_{i=1}^r, \{t_i\}_{i=1}^r\). \(S\) uses Fiat-Shamir to ensure the non-interactivity of the proof.

**The full protocol.** We summarize this protocol as \(\Pi_{\text{Mix}}\):

1. \(S\) obtains ciphertexts \(\{c_i\}_{i=1}^r = \{(u_i, v_i)\}_{i=1}^r\).
2. For each \(i \in [r]\), samples \(r'_i, e'_{1,2} \in \mathbb{F}_q^k\) as above. It then creates commitments \(\text{Com}(u'_i, v'_i) = [ar'_i + pe'_{i,1}, br'_i + pe'_{i,2}]\) using randomness \(r_i\) for each such commitment.
3. Let \(t_i = \text{Com}(u'_i, v'_i)\) and \(r'_i = [r'_i, r'_i, e'_{1,2}]^T\). Then \(S\) computes \(\pi_{\text{SMALL}} \leftarrow \Pi_{\text{SMALL}}^k\) for matrix \(\mathbf{A}_M\), input vectors \(\{r'_i\}\), target vectors \(\{t_i\}\) and bound \(B_{\text{Dec}}\).
4. Let \(\hat{c}_i = (u_i + u'_i, v_i + v'_i)\) and \(L\) be a random permutation of \(\{\hat{c}_i\}_{i=1}^r\). Then \(S\) computes \(\pi_{\text{SHUFF}} \leftarrow \Pi_{\text{SHUFF}}^k\) with input commitments \(\text{Com}(\hat{u}_i, \hat{v}_i)\) \(\{i\}_{i=1}^r\), commitment messages \(\hat{c}_i\) \(\{i\}_{i=1}^r\), randomness \(\{r_i\}_{i=1}^r\) and ciphertexts \(L\).
5. \(S\) outputs \((\{t_i\}_{i=1}^r, \pi_{\text{SMALL}}, L, \pi_{\text{SHUFF}})\).
Given such a string \((\{t_i\}, \sigma)\) from \(S\) as well as ciphertext vector \(\{c_i\}_i\) any third party \(V\) can now run the following algorithm \(\Pi_{\text{Mix}}\) to verify the mix:

1. Run the verification algorithm of \(\Pi_{\text{SMALL}}\) for \(\sigma\) on inputs \(A_{\text{Mix}}, \{t_i\}_i\) and \(B_{\text{Mix}}\). If verification fails: output 0.
2. For \(v_i \in [r]\) set \(c_{i,v_i} = c_i + t_i\).
3. Run the verification algorithm of \(\Pi_{\text{SHUF}}\) for \(\sigma_{\text{SHUF}}\) on input \(\{(c_{i,v_i})\}_i\). If the verification fails, then output 0. Otherwise, output 1.

In the following, define noise bound \(B_{\text{Mix}}\) to be the maximum level of noise in ciphertexts \(c_i\), i.e., the maximal noise of the randomness \(r'_i, e'_i, c'_i\) used to create the ciphertexts.

We want that the outputs of the mixing protocol fulfill the following relation \(R_{\text{Mix}}\):

\[
\begin{align*}
(x, w) & = (c_1, \ldots, c_r, e_1, \ldots, e_r, c'_1, \ldots, c'_r, c'_1', \ldots, c'_r') \\
(w, c'_i) & = A_{\text{Mix}}(r'_i, e'_i, c'_i) \\
\|c'_i\| & \leq B_{\text{Mix}}(\pi_i) = c_i + c'_i
\end{align*}
\]

If the noise-levels in all \(c_i\) and \(c'_i\) are bounded by \(B_{\text{Dec}}\) and \(B_{\text{Mix}}\) respectively, and \((B_{\text{Dec}} + B_{\text{Mix}}) \leq q^2\), then all \(c_i\) and \(c'_i\) will, for some permutation \(\pi\), decrypt to the same message \(m_i\) under sk.

In Appendix E we analyze the guarantees of \(\Pi_{\text{SMALL}}\) in more detail.

5.3 Communication of a BGV Shuffle

The mixing phase transcript contains \(\tau\) new ciphertexts generated by the server, which are of size \(2rN\log_2 q\) bits.

The server must provide a proof of shuffle and an amortized proof of shortness for \(r'_1, e'_1, c'_1, r'_2, e'_2, c'_2\). Both proofs prove a relation about commitments to the randomization factors \(u'_1, v'_1\) added to the old ciphertexts to get the new ciphertexts. Each commitment to \(u'_i, v'_i\) is of size \((n + 2)N\log_2 q\) bits. We denote the proof by \(\sigma_{\text{SMALL}}\).

The shuffle proof consists of \(\tau\) commitments of size \((n + 1)N\log_2 q\) bits, \(\tau R_{\text{Mix}}\)-elements of size \(N\log_2 q\) bits and a proof of linearity per ciphertext. This adds up to an overall size of \((N + 2)N\log_2 q + 2(k - n)N\log_2 (6\sigma_C)\) \(\tau\) bits for the proof of shuffle, and in total

\[
((2n + 6)N\log_2 q + 2(k - n)N\log_2 (6\sigma_C))\tau + |\sigma_{\text{SMALL}}| \text{ bits.}
\]

6 VERIFIABLE DISTRIBUTED DECRIPTION

In this section, we provide a construction for a PKE scheme with distributed decryption which is secure against active attacks. We combine the distributed decryption protocol from Section 2.1 with zero-knowledge proofs to achieve this. In a nutshell, the DDec algorithm that we introduced in Section 2.1 first requires that each decryption server chooses a uniformly random \(E_j\) from a bounded distribution. Next, it outputs a linear combination \(t_j\) involving a ciphertext element \(u\), the decryption key share \(s_j\), and \(E_j\). To make this actively secure, we will let the key generation algorithm output a commitment to \(s_j\). Then, to show that it computed \(t_j\) correctly from \(u\), the decryption server will reveal a commitment to \(E_j\) as well as two zero-knowledge proofs: i) it will show that \(E_j\) is bounded as required, and ii) it will show that \(t_j\) is indeed computed using a linear combination.

6.1 The Actively Secure Protocol

Let the ring \(R_q\), the statistical security parameter \(\sigma\), and bounds \(B_{\text{Mix}}, B_{\text{Dec}}, B_{\text{COM}}\) be public information, together with the plaintext modulus \(p\) for the PKE scheme. Let \(AC\) be the public commitment matrix of a BDLO instance for message size \(l_e = 1\).

- \(KGen_A(\tau, \sigma_2)\):
  1. Get \((pk, sk, s_1, \ldots, s_{\sigma_2})\) \(\Leftarrow KGen(\tau, \sigma_2)\) as in the passive distributed encryption protocol.
  2. \(\forall j \in [\sigma_2] \text{ compute } (s_j, \pi_j) \Leftarrow \text{Com}(s_j).
  3. Output \(pk_A = (pk, s_1, \ldots, s_{\sigma_2})\) and finally \(sk_A = sk\) and \(sk_A' = (s_j, \pi_j)\) for all \(j \in [\sigma_2]\).
- \(\text{Enc}_A\) and \(\text{Dec}_A\) work just like the original \(\text{Enc}\) and \(\text{Dec}\) in the passively secure threshold encryption scheme, ignoring additional information in \(pk_A\).
- \(DDec(\{c_i\}_{i \in [\tau]}\) where \(c_i = (u_i, v_i)\):
  1. For each \(i \in [\tau]\) compute \(m_i = s_j u_i\), sample uniform noise \(E_{i,j} \Leftarrow R_q\) such that \(\|\|E_{i,j}\|\| \leq 2\sigma_{\text{EC}}(B\text{Dec}/p\|\|E\|\|)\) and compute the decryption share \(t_{i,j} = m_i + pE_{i,j}\).
  2. For each \(i \in [\tau]\) compute \((s_j, \pi_j) \Leftarrow \text{Com}(s_j)\) and use the \(\Pi_{\text{LIN}}\)-protocol to compute a proof for the linear relation \(t_{i,j} = s_j u_i + pE_{i,j}\) from \(\pi_{L_{i,j}} = \Pi_{\text{LIN}}((s_j, r_{i,j}), (E_{i,j}, r''_{i,j})); ((s_j, E_{i,j}, t_{i,j}, (u_i, p))\).

3. Each commitment \(\|E_{i,j}\|\) of the form

\[
\begin{bmatrix}
I_r & a_{1,1} & A_{1,2} & 0 \\
0^2 & 1 & a_{2,2} & t_{i,j}' + 0 \\
\end{bmatrix}
\]

where \(\|r_{i,j}\| \leq B_{\text{Dec}}\) is the randomness used in the commitments. Run the zero-knowledge protocol \(\Pi_{\text{KN}}(\{(c_{i,j}))_{i \in [\tau]}\); \(\text{Ap}(\|E_{i,j}\|_{i \in [\tau]}))\) to obtain the amortized zero-knowledge PoK \(\sigma_{\text{KN}}\).

4. Output \(ds_{i,j} = (t_{i,j})_{i \in [\tau]}, \sigma_{\text{D}}\) with the decryption proof \(\Pi_{\text{D}}(\{E_{i,j}\}_{i \in [\tau]}; \{\pi_{L_{i,j}}\}_{i \in [\tau]}; \sigma_{\text{KN}})\).

* Combine \((\{c_i\}_{i \in [\tau]}; \{ds_{i,j}\}_{i \in [\tau]}; \sigma_{\text{D}})\):
  1. Parse \(ds_{i,j}\) as \((t_{i,j})_{i \in [\tau]}, \sigma_{\text{D}}\).
  2. Verify the proofs \(\pi_{L_{i,j}}\).
  3. Verify the proofs \(\sigma_{\text{KN}}\).

If any verification protocol returned 0 then output \(\bot\). Otherwise, compute

\[
m_i = (u_i - t_i \mod q) \mod p,\text{ where } t_i = t_{i,1} + \cdots + t_{i,\tau}\text{ for } i = 1, \ldots, \tau.
\]

and output the set of messages \(m_1, \ldots, m_{\tau}\).

The randomness \(r_{i,j}\) has much smaller \(\sigma_{\text{D}}\) norm than \(E_{i,j}\), and hence, we will run the \(\Pi_{\text{BDK}}\) protocol with small standard deviation \(\sigma_{\text{BDK}}\) for rows 1 to \(k\), while row \(k + 1\) will have large \(\sigma_{\text{BDK}}\). This trivially works for \(\Pi_{\text{BDK}}\) as all operations, also in the extractor for the soundness-proof, are coordinate-wise.
The following theorems refer to definitions of threshold correctness, threshold verifiability, and distributed decryption simulatability given in Section 2.1. In the following theorems, let the noise bounds $B_{\text{Dec}}$ and $B_{\text{Bnd}}$ satisfy $(1 + B_{\text{Dec}}) \cdot 2^{\text{sec}} < 2B_{\text{Bnd}} < [q/2]$.

**Theorem 5.** Let ciphertext-noise be bounded by $B_{\text{Dec}}$, and let the noise added in DDec be bounded by $2^{\text{sec}}B_{\text{Dec}}$. Suppose the passively secure protocol is threshold correct and the protocols $\Pi_{\text{LIN}}$ and $\Pi_{\text{BND}}$ are complete. Then the actively secure protocol is threshold correct.

Informally, since $B_{\text{Dec}} + 2^{\text{sec}}B_{\text{Dec}} < q/2$, it follows that decryption is correct. Furthermore, since $(1 + B_{\text{Dec}}) \cdot 2^{\text{sec}} < 2B_{\text{Bnd}} < q/2$ and $\Pi_{\text{LIN}}$ and $\Pi_{\text{BND}}$ are complete, the arguments will be accepted, which means that the decryption proof will be accepted.

**Theorem 6.** Let $\text{Adv}_0$ be an adversary against threshold verifiability for the actively secure protocol with advantage $e_0$. Then there exists adversaries $\text{Adv}_1$ and $\text{Adv}_2$ against soundness for $\Pi_{\text{LIN}}$ and $\Pi_{\text{BND}}$, respectively, with advantages $e_1$ and $e_2$, such that $e_0 \leq e_1 + e_2$. The runtime of $\text{Adv}_1$ and $\text{Adv}_2$ are essentially the same as of $\text{Adv}_0$.

We sketch the argument. We only consider ciphertexts with noise bounded by $B_{\text{Dec}}$, so we may assume that the noise in any particular ciphertext is bounded by $B_{\text{Dec}}$.

If the decryption is incorrect for a particular ciphertext, then for some $j$ no relation $\ell_{i,j} = s_ju_i + pE_{i,j}$ holds for an $E_{i,j}$ of the norm at most $2B_{\text{Bnd}}$. This can happen in two ways: Either the argument for the linear combination of the commitments to $E_{i,j}$ and $s_j$ is incorrect or the bound on $E_{i,j}$ is incorrect. In the former case, we trivially get an adversary $\text{Adv}_1$ against soundness for $\Pi_{\text{LIN}}$. Similar for the case of $\Pi_{\text{BND}}$.

**Theorem 7.** Suppose the passively secure protocol is simulatable and $\Pi_{\text{LIN}}$ and $\Pi_{\text{BND}}$ are honest-verifier zero-knowledge. Then there exists a simulator for the actively secure protocol such that for any distinguisher $\text{Adv}_0$ for this simulator with advantage $e_0$, there exists an adversary $\text{Adv}_1$ against hiding for the commitment scheme, with advantage $e_1$, and distinguishers $\text{Adv}_2$, $\text{Adv}_3$ and $\text{Adv}_4$ for the simulators for the passively secure protocol, $\Pi_{\text{LIN}}$, and $\Pi_{\text{BND}}$, respectively, with advantages $e_2$, $e_3$, $e_4$, such that $e_0 \leq e_1 + e_2 + e_3 + e_4$. The runtime of $\text{Adv}_1$, $\text{Adv}_2$, $\text{Adv}_3$ and $\text{Adv}_4$ are essentially the same as of $\text{Adv}_0$.

We sketch the argument. Using appropriate simulators, the simulator simulates the arguments and the passively secure distributed decryption algorithm. It replaces the commitment to the noise $E_{i,j}$ by commitments to zero.

The claim about the simulator follows from a straightforward hybrid argument. We begin with distributed decryption.

First, we replace the $\Pi_{\text{LIN}}$ arguments with simulated arguments, which gives us a distinguisher $\text{Adv}_2$ for the $\Pi_{\text{LIN}}$ honest verifier simulator. Second, we replace the $\Pi_{\text{BND}}$ arguments with simulated arguments, which gives us a distinguisher $\text{Adv}_3$ for the $\Pi_{\text{BND}}$ honest verifier simulator. Third, we replace the commitments to the noise $E_{i,j}$ with random commitments, giving us an adversary $\text{Adv}_4$ against hiding for the commitment scheme. Fourth, we replace the passively secure distributed decryption algorithm with its simulator, which gives us a distinguisher $\text{Adv}_1$ for the simulator.

After four changes, we are left with the claimed simulator for the actively secure protocol, and the claim follows.

### 6.2 Communication Complexity of DistDec

Each partial decryption consists of one element from $R_n$, namely the output of DDec, which means that the output from the passively secure protocol is of size $\xi_2 n \log_2 q$ bits.

Each decryption server outputs a commitment $\Pi_2 \Pi_1$ to the added noise and proof of linearity per ciphertext, and an amortized proof of shortness for all the added noise values. Each server has a public commitment of their decryption key-share to be used in the proof of linearity, but we neglect this as it is constant.

Each commitment $\Pi_2 \Pi_1$ is of size $(n + 1)N \log_2 q$ bits, and each proof of linearity is of size $(k - n)N(\log_2(6\tau_0^2) + \log_2(6\tau_0^2))$ bits because the partial decryption is given in the clear and one commitment is re-used in all equations. Finally, each of the amortized proofs is of size $k\eta N \log_2(6\tau_0^2) + \eta \log_2(6\tau_0^2)$ bits because of the different norms of the secret values as noted earlier. As the bounds in the amortized proof depend on the number of commitments in the statement, each amortized proof is for a batch of $N$ equations at once to control the growth of parameters.

The total size of the distributed decryption is

$$\xi_2 (n + 2) N \log_2 q + (k - n)N(\log_2(6\tau_0^2) + \log_2(6\tau_0^2)) + k\eta N \log_2(6\tau_0^2) + \eta \log_2(6\tau_0^2)$$

### 7 PERFORMANCE

We provide an overview of parameters and descriptions in Table 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\eta$</td>
<td>Computational security parameter</td>
<td>At least $128$ bits</td>
</tr>
<tr>
<td>$N$</td>
<td>Number of servers</td>
<td>In bits</td>
</tr>
<tr>
<td>$p$</td>
<td>Degree of polynomial $N + n + b_p$</td>
<td>In bits</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Suitability modulus</td>
<td>$p \times$ prime</td>
</tr>
<tr>
<td>$\xi$</td>
<td>Number of ciphertexts and commitments needed</td>
<td>From $p \cdot 2^{\text{sec}}$ and $2^{\text{sec}} + n \times 2^{\text{sec}} + 1$</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>Value of linearizing commitment noise deduced to_leader</td>
<td>In bits</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>Length of commitment noise</td>
<td>In bits</td>
</tr>
<tr>
<td>$\phi$</td>
<td>Challenge space for Linear (k proof of commitments)</td>
<td>$k \times 2^{\text{sec}}$ + $k \times (2^{\text{sec}} - 1)$</td>
</tr>
<tr>
<td>$\mu$</td>
<td>Minimum</td>
<td>In bits</td>
</tr>
<tr>
<td>$\Phi$</td>
<td>Set of adversaries at nonce of round $s$</td>
<td>$\Phi_0 \cup \Phi_1 \cup \Phi_2 \cup \Phi_3$</td>
</tr>
<tr>
<td>$\Lambda$</td>
<td>Bound for the commitment noise</td>
<td>$\gamma$</td>
</tr>
<tr>
<td>$\Phi_0$</td>
<td>Bound for two key-verifier interaction scheme</td>
<td>$\Lambda$</td>
</tr>
<tr>
<td>$\Phi_1$</td>
<td>Bound for noise in ciphertext</td>
<td>$\Lambda$</td>
</tr>
<tr>
<td>$\Phi_2$</td>
<td>Standard deviation in linear (k proof of commitments)</td>
<td>$\Lambda$</td>
</tr>
<tr>
<td>$\Phi_3$</td>
<td>Standard deviation in Linear (k proof of commitments)</td>
<td>$\Lambda$</td>
</tr>
<tr>
<td>$\Phi_4$</td>
<td>Standard deviation for one-line amortized proof of noise</td>
<td>$\Lambda$</td>
</tr>
<tr>
<td>$\Phi_5$</td>
<td>Standard deviation for one-line amortized proof of noise</td>
<td>$\Lambda$</td>
</tr>
<tr>
<td>$\Phi_6$</td>
<td>Number of proofs in $\Pi_{\text{LIN}}$</td>
<td>$2 \times N^2 + 2N$</td>
</tr>
<tr>
<td>$\Phi_7$</td>
<td>Number of proofs in $\Pi_{\text{BND}}$</td>
<td>$2 \times N^2 + 2N$</td>
</tr>
<tr>
<td>$\Phi_8$</td>
<td>Total number of messages/million of rounds</td>
<td>$2 \times N^2 + 2N$</td>
</tr>
<tr>
<td>$\Phi_9$</td>
<td>Encoding length in $\Pi_{\text{LIN}}$</td>
<td>$2 \times N^2 + 2N$</td>
</tr>
<tr>
<td>$\Phi_{10}$</td>
<td>Encoding length in $\Pi_{\text{BND}}$</td>
<td>$2 \times N^2 + 2N$</td>
</tr>
<tr>
<td>$\Phi_{11}$</td>
<td>Dimension of Reed-Solomon Code in $\Pi_{\text{LIN}}$</td>
<td>$2 \times N^2 + 2N$</td>
</tr>
<tr>
<td>$\Phi_{12}$</td>
<td>Dimension of Reed-Solomon Code in $\Pi_{\text{BND}}$</td>
<td>$2 \times N^2 + 2N$</td>
</tr>
</tbody>
</table>

**Table 1: System parameters and constraints.**

### 7.1 Concrete Parameters and Total Size

We begin by fixing the rejection-sampling parameter as $M = 3$, leading to a general abort probability of $2/3$ for each rejection sampling proof. This allows us to define the standard deviations in all proof instances.

We pick the noise in the BGV ciphertexts as well as in commitments to ternary distributions, as this gives tight control on the noise growth during the protocols.
To be able to choose concrete parameters for the mix-net, we need to estimate how much noise is added to the ciphertexts through the two stages of the protocol: 1) the shuffle phase, and 2) the decryption phase. This follows from a standard analysis that incorporates the slacks of the ZK proofs involved in the protocols and will be one lower bound on choosing q as the noise should not wrap around computations mod q.

For our example, we let the number of shuffle and decryption servers be $\xi_1 = \xi_2 = 4$. We fix the plaintext modulus to be $p = 2$, statistical security parameter sec = 40 (a common choice in the MPC literature), and need $N = 4096$ when $q$ is chosen as outlined above in order for the underlying lattice problems to be hard, see details in Table 5. This allows for votes of size 4096 bits, which is a feasible size for real-world elections representing a wide range of voter options.

Finally, we must decide on parameters for the exact proof of shortness from Section 4. The soundness of the protocol depends on the ratio between the number of equations and the size of the modulus. We choose to compute the proof in batches of size $N$ instead of computing the proof for all $r$ commitments at once and will have to run each proof twice to achieve negligible soundness error. After choosing appropriate parameters for code length and the number of tested rows $\eta$, the total size of $\pi_{\text{SMALL}}$ by instantiating equation 1, is $\approx 20r$ KB.

We summarize the concrete sizes of each part of the protocol in Table 2. Each voter submits a ciphertext size of approximately 80 KB. The size of the mix-net, including ciphertexts, commitments, shuffle proof, and proof of shortness, is approximately 370 KB per mixing node $S_i$. The size of the decryption phase, including partial decryptions, commitments, proofs of linearity, and proofs of boundedness, is approximately 157 KB per decryption node $D_j$.

See Appendix F for more details on the choice of parameters.

### 7.2 Implementation

We developed a proof-of-concept implementation to compare with previous results in the literature. Our performance figures were collected on an Intel Kabylake Core i7-7700 CPU machine with 64GB of RAM running single-threaded at 3.6GHz, with Turbo Boost disabled to reduce measurement variability. The results can be found in Tables 3 and 4. Our research prototype can be found at https://github.com/dfaranha/lattice-verifiable-mixnet.

First, we compare the performance of the main building blocks with an implementation of the shuffle-proof protocol proposed in [4]. That work used the FLINT library to implement arithmetic involving polynomials of degree $N = 1024$ with 32-bit coefficients, fitting a single machine word. Their parameters were not compatible with the fast Number Theoretic Transform (NTT), so a CRT decomposition to two half-degree polynomials was used instead. The code was made available, so a direct comparison is possible.

In this work, the degree is much larger ($N = 4096$) and coefficients are multi-word ($q \approx 2^{88}$), but the parameters are compatible with the NTT. We implemented polynomial arithmetic with the efficient FLINT [2] library using the RNS representation for coefficients accelerated with AVX2 instructions. We observed that our polynomial multiplication is around 19 times faster than [4] (61, 314 cycles instead of 1, 165, 997), despite parameters being considerably larger. We also employed the FLINT library for arithmetic routines not supported in NFLlib, such as polynomial inversion, but that incurred some non-trivial costs to convert representations between two libraries. We adapted [47] and [46] for Gaussian sampling and adjusting the standard deviation $\sigma$ accordingly.

Computing a commitment takes 0.45 ms on the target machine, which is 2x faster than [4]. Opening a commitment is slower due to conversions between libraries for performing the norm-test. Our implementation of BGV encryption at 0.74 ms is much faster than the 69 ms reported for verifiable encryption in [4], while decryption is improved by a factor of 10. Distributed decryption with passive security costs an additional 1.56 ms per party, but the zero-knowledge proofs for active security increase the cost. The shuffle proof performance is 20.1 ms per vote, thus slightly faster than the 27 ms reported in [4].

For the other sub-protocols, we benchmarked executions with $\tau = 1000$ and report the execution time amortized per vote for both prover and verifier in Table 4. In the case of $\Pi_{\text{SMALL}}$, we implement the performance-critical polynomial arithmetic and encoding scheme, since this is already representative of the overall performance. From the table, we can compute the cost of distributed decryption $\Pi_{\text{Dec}}$ with active security as (1.56 + 0.45 + 17.3 + 82.5) = 101.81 ms per vote, the cost of verification $\Pi_{\text{DecV}}$ as (6.3 + 22.2) = 28.5 ms per vote, the cost of $\Pi_{\text{Mix}}$ as (0.74 + 0.45 + 112.3 + 20.1) = 133.6 ms and the cost of $\Pi_{\text{MixV}}$ as (5.0 + 7.9) = 12.9 ms per vote. This result compares quite favorably with the costs of 1.54 s and 1.51 s per vote to respectively generate/verify a proof in the lattice-based shuffle proof of [27] in a Kaby Lake processor running at a similar frequency. Our total numbers are 18.5 times faster after adjusting for clock frequency, while storage overhead is much lower.

---

<table>
<thead>
<tr>
<th>$e^{(\ell)}$</th>
<th>$\ell^o$</th>
<th>$\pi_{\text{SHUFFE}}$</th>
<th>$\pi_{\text{L1,L2}}$</th>
<th>$\pi_{\text{SMALL}}$</th>
<th>$\pi_{\text{BIND}}$</th>
<th>$\pi_{\text{Sig}}$</th>
<th>$\pi_{\text{D}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>80 KB</td>
<td>40($L + 1$) KB</td>
<td>150 KB</td>
<td>35 KB</td>
<td>20 KB</td>
<td>2r KB</td>
<td>370 KB</td>
<td>157+ KB</td>
</tr>
</tbody>
</table>

Table 2: Size of the ciphertexts, commitments, and proofs.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>$\Pi_{\text{LIN}} + \Pi_{\text{LINV}}$</th>
<th>$\Pi_{\text{SHUFFE}} + \Pi_{\text{SHUFFE}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>(17.3 + 6.3) $\tau$ ms</td>
<td>(20.1 + 7.9) $\tau$ ms</td>
</tr>
</tbody>
</table>

Table 3: Timings for cryptographic operations. Numbers were obtained by computing the average of $10^4$ executions measured using the cycle counter available on the platform.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>$\Pi_{\text{BIND}} + \Pi_{\text{BINDV}}$</th>
<th>$\Pi_{\text{SMALL}} + \Pi_{\text{SMALLV}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>(82.5 + 22.2) $\tau$ ms</td>
<td>(112.3 + 5.0) $\tau$ ms</td>
</tr>
</tbody>
</table>

Table 4: Timings for cryptographic protocols, obtained by computing the average of 100 executions with $\tau = 1000$. 
We have proposed a verifiable secret shuffle of BGV ciphertexts and a verifiable distributed decryption protocol. Together, these two novel constructions are practical and solve a long-standing problem in the design of quantum-safe cryptographic voting systems.

Verifiable secret shuffles for discrete logarithm-based cryptography has seen a long sequence of incremental designs follow Neff’s breakthrough construction. While individual published improvements were often fairly small, the overall improvement in performance over time was significant. We expect that our designs can be improved in a similar fashion. In particular, we expect that the size of the proofs can be significantly reduced. While it is certainly straightforward to download a few hundred gigabytes today (compare with high-quality video streaming), many voters will be discouraged and this limits the universality of verification in practice. It, therefore, seems reasonable to focus further effort on reducing the size of the proofs.

The distributed decryption protocol does not have an adjustable threshold. In practice, this is not much of a problem, since the keys will be shared among many key holders. Only when counting starts is the key material given to the decryption servers. Key reconstruction can then be combined with a key distribution protocol.

Shuffles followed by distributed decryption is one paradigm for the design of cryptographic voting systems. Another possible paradigm is to use key shifting in the shuffles. This would then allow us to use a single party for decryption (though it must still be verifiable, e.g., using the protocols [29, 43]). Key shifting can be done with many of the same techniques that we use for distributed decryption, but there seems to be difficulties in amortizing the proofs. This means that key shifting with just the techniques we use will be significantly slower and of increased size, as we would need additional proofs of linearity for each ciphertext in each shuffle.

Follow-up work by Hagåsen and Slåde [32] shows how our voting protocol can be combined with a return-code mechanism to achieve individual voter verifiability against a cheating ballot box.

Finally, we note that our scheme and concrete instantiation using the NTT is optimized for speed, and that it is possible to slightly decrease the parameters by instantiating the encryption scheme based on the SKS and DKS problems in higher dimensions $k$ using a smaller, but still a power of 2, ring-dimension $N$. We leave this as future work. We also remark that lattice-based cryptography, and especially lattice-based zero-knowledge proofs such as the recent work by Lyubashevsky et al. [36], continuously improves the state-of-the-art, and we expect future works to improve the concrete efficiency of our protocol.

REFERENCES


A PRELIMINARIES

Let \( N \) be a power of 2 and \( q \) a prime such that \( q \equiv 1 \mod 2N \). We define the rings \( R = \mathbb{Z}[X]/(X^N + 1) \) and \( R_q = R/qR_q \), that is, \( R_q \) is the ring of polynomials modulo \( X^N + 1 \) with integer coefficients modulo \( q \). This way, \( X^N + 1 \) splits completely into \( N \) irreducible factors modulo \( q \), which allows for very efficient computation in \( R_q \) due to the number theoretic transform (NTT) \([34]\). We define the norms of elements \( f(X) = \sum \alpha_i X^i \in R \) to be the norms of the coefficient vector as a vector in \( \mathbb{Z}^N \):

\[
\|f\|_1 = \sum |\alpha_i|, \quad \|f\|_2 = \sum |\alpha_i|^2, \quad \|f\|_\infty = \max |\alpha_i|.
\]

For an element \( f \in R_q \) we choose coefficients as representatives in

\[
\left[ -q^{-\frac{1}{2}}, q^{-\frac{1}{2}} \right],
\]

and then compute the norms as if \( f \) is an element in \( R \). For vectors \( \mathbf{a} = (a_1, \ldots, a_k) \in \mathbb{R}^k \) we define the \( \ell_2 \) norm to be

\[
\|\mathbf{a}\|_2 = \sqrt{\sum |a_i|^2}
\]

and analogously for the \( \ell_1 \) and \( \ell_\infty \) norm. It is easy to see the following relations between the norms of elements in \( R_q \):

\[
\|f\|_\infty \leq \alpha, \quad \|f\|_1 \leq \beta, \quad \text{then} \quad \|f\|_2 \leq \alpha \beta,
\]

\[
\|f\|_2 \leq |\mathbf{a}|, \quad \|g\|_2 \leq \beta, \quad \text{then} \quad \|f \circ g\|_2 \leq |\mathbf{a}| \beta.
\]

We also define the sets \( S_\beta = \{ x \in R_q \ | \ |x|_\infty \leq \beta \} \) as well as

\[
C = \{ c \in R_q \mid |c|_1 = 1, \quad |c|_\infty = \gamma, \quad \bar{C} = \{ c - \epsilon c \ | \ \epsilon \neq c \in C \}.
\]

A.1 The Discrete Gaussian Distribution

The continuous normal distribution over \( \mathbb{R}^k \) centered at \( \mathbf{v} \in \mathbb{R}^k \) with standard deviation \( \sigma \) is given by

\[
n_{C, \mathbf{v}}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left( -\frac{1}{2\sigma^2} |x - \mathbf{v}|^2 \right).
\]

When sampling randomness for our lattice-based commitment and encryption schemes, we will need samples from the discrete Gaussian distribution. This distribution is achieved by normalizing the
We first define the Search Knapsack problem in the $\ell_2$ norm (DKS$^{\infty}$). The DKS$^{\infty}$ problem is equivalent to the Module-LWE problem when the number of samples is limited.

Definition 7. The DKS$_{n,k,\beta}$ problem is to distinguish the distribution $\{ B_{\gamma_1} A' : y \}$ for a short $y$ from a bounded distribution $S_\varepsilon$ when given $A'$. An algorithm Adv has advantage $\varepsilon$ in solving the DKS$_{n,k,\beta}$ problem if

$$\Pr[b = 1 | y \leftarrow S_\varepsilon, b \leftarrow \text{Adv}(A', [ I_n A'] : y)]$$

$$\leq \Pr[b = 1 | u \leftarrow R_q, b \leftarrow \text{Adv}(A', u)] - \varepsilon,$$

for a uniformly sampled $A' \leftarrow R_{q}$.

See [33] for more details about module lattice problems.

A.4 Security of Distributed Decryption

Definition 8 (Chosen Plaintext Security). We say that the public key encryption scheme is secure against chosen plaintext attacks if an adversary $A$, after choosing two messages $m_0$ and $m_1$ and receiving an encryption of either $m_0$ or $m_1$ (chosen at random), cannot distinguish which message $c$ is an encryption of. Hence, we want that

$$|\Pr[b = 1 | (pp, pk, sk) \leftarrow \text{KGen}(\lambda) \quad (m_0, m_1, st) \leftarrow \mathcal{A}(pp, pk) \quad b \leftarrow \mathcal{A}(c, st)] - \frac{1}{2}| \leq \varepsilon(\lambda),$$

where the probability is taken over KGen and Enc.

Definition 9 (Threshold Correctness). We say that the public key distributed encryption scheme is threshold correct with respect to $P_{sk}(-)$ if the following probability equals 1:

$$\Pr[\begin{pmatrix} \text{Comb}((r_i)_{i \in [t]}, (d_{ij})_{i \in [t], j \in [z_i]}) \quad \text{Dec}(sk, c_{i}) \end{pmatrix} = \end{pmatrix} \begin{pmatrix} \text{KGen}(\lambda, \varepsilon_2) \quad \mathcal{A}(pp, pk) \quad \mathcal{A}(c, st) \end{pmatrix} | \begin{pmatrix} \text{PP}_{sk} = 1, \forall i \in [t] : P_{sk}(c_i) = 1, \forall j \in [z_i] : (d_{ij})_{i \in [t], j \in [z_i]} \end{pmatrix} \end{pmatrix} \end{pmatrix} \leq \varepsilon(\lambda),$$

where the probability is taken over KGen and DDec.

A.5 Security Definitions for Commitments

Definition 10 (Hiding). We say that a commitment scheme is hiding if an adversary $A$, after choosing two messages $m_0$ and $m_1$ and receiving a commitment $c$ to either $m_0$ or $m_1$ (chosen at random), cannot distinguish which message $c$ is a commitment to. Hence, we want that

$$|\Pr[b = 1 | (pp, pk) \leftarrow \text{Setup}(\lambda) \quad (m_0, m_1, st) \leftarrow \mathcal{A}(pp) \quad b \leftarrow \mathcal{A}(c, st)] - \frac{1}{2}| \leq \varepsilon(\lambda),$$

where the probability is taken over Setup and Com.

Definition 11 (Binding). We say that a commitment scheme is binding if an adversary $A$, after creating a commitment $c$, cannot find two valid openings to $c$ for different messages $m$ and $\hat{m}$. Hence, we want that

$$\Pr[\begin{pmatrix} \text{Open}(m, c, r) = 1 \quad \text{Com}(m, \hat{m}) \leftarrow \mathcal{A}(pp) \end{pmatrix} | \begin{pmatrix} m \neq \hat{m} \end{pmatrix} \leq \varepsilon(\lambda).$$

where the probability is taken over the random coins of Setup.
### A.6 Security Definitions for ZK Proofs

**Definition 12 (Completeness).** We say that a proof protocol $Π$ is complete if $V$ outputs 1 when $P$ knows a witness $w$ and both parties follow the protocol. Hence, for any efficient sampling algorithm $P_0$ we want that

$$\Pr \left[ \left( P_0(s, x, w) \right) \cdot (V(s, x, w)) = 1 \right] = 1,$$

where the probability is taken over $Set_{\Sigma}$, $P$ and $V$.

**Definition 13 (Knowledge Soundness).** We say that a proof protocol $Π$ is knowledge sound if, when a cheating prover $P^*$ that does not know a witness $w$ is able to convince an honest verifier $V$, there exists a polynomial time algorithm extractor $E$ which, given black-box access to $P^*$, can output a witness $w$ such that $(x, w) \in R$. Hence, we want that

$$\Pr \left[ \left( Sp\rightarrow Set_{\Sigma} \left( x, w \right) \right) \cdot (V(s, x, w)) = 1 \right] \geq 1 - \epsilon(x),$$

where the probability is taken over $Set_{\Sigma}$, $P^*$ and $E$.

**Definition 14 (Honest-Verifier Zero-Knowledge).** We say that a proof protocol $Π$ is honest-verifier zero-knowledge if an honest verifier $V$ that follows the protocol cannot learn anything beyond the fact that $x \in L$. Hence, we want for real accepting transcripts $T(\langle P(s, x, w) \rangle, \langle V(s, x, w) \rangle)$ between a prover $P$ and a verifier $V$, and a accepting transcript $S(\langle P(s, x, w) \rangle, \langle V(s, x, w) \rangle)$ generated by simulator $S$ that only knows $x$, that

$$\Pr \left[ \left( Sp\rightarrow Set_{\Sigma} \left( x, w \right) \right) \cdot (V(s, x, w)) = 1 \right] \geq 1 - \epsilon(x),$$

where the probability is taken over $Set_{\Sigma}$ and $V^*$.

### B PROOFS OF LINEAR RELATIONS AND AMORTIZED NON-EXACT ZKPOK

#### B.1 Zero-Knowledge Proof of Linear Relations

![Figure 3: $\Pi_{Lin}$ is a Sigma-protool to prove the relation $R_{Lin}$.](image)

For the public scalar vector $\mathbf{a} = (a_1, \ldots, a_{q-1}) \in \mathbb{R}_{q}^{m}$ the prover wants to prove that the following relation holds:

$$R_{Lin} = \left\{ (x, w) \mid x = (pk, \langle [m_1] \rangle_{i \in [\hat{n}]}, \mathbf{a}) \wedge w = (f_1, \langle [m_1], [r_1], [r_i] \rangle_{i \in [\hat{n}]}) \wedge m_{a_i} = \hat{w}_{i \in [\hat{n}]} \frac{\mathbf{a}}{a_i} B_{Lin} \right\}.$$

$\Pi_{Lin}$ in Figure 3 is a zero-knowledge proof of knowledge (ZKPoK) of this relation (it is a directly extended version of the linearity proof in [7]). It works like a standard $Σ$-protocol when adapted to lattices, and we, therefore, do not explain its inner workings further and instead refer the reader to [7].

The relation $R_{Lin}$ is relaxed because of the additional factor $f$ in the opening, which appears in the soundness proof. It does not show up in protocol $Π_{Lin}$, because an honest prover uses $f = 1$. The bound is $B = 2\sqrt{N} \sqrt{L}$ and the protocol produces a proof transcript of the form $\pi_{Lin} = (\langle [\hat{t}_i] \rangle_{i \in \hat{n}}, \hat{b}, (\langle [\hat{z}_i] \rangle_{i \in \hat{n}})).$

The following is a direct adaption of Baum et al. [7, Lemma 8].

**Theorem 8.** The zero-knowledge proof of linear relations is complete if the randomness $r_1$ is bounded by $B_{Lin}$ in the $\ell_\infty$ norm, it is 2-special sound if the $SKS^2$ problem is hard, and it is statistical honest-verifier zero-knowledge. The probability of abort is $1 - (1/M)^{\hat{a}}$ (where $M$ is a parameter showing up in Rejection Sampling as defined in Equation 2).

Even though the success probability is $(1/M)^{\hat{a}}$, we will only use this protocol for small values $\hat{a}$ and choose parameters so that the rejection probability is small ($\approx 1/3$), which ensures that the protocol is efficient in practice.

When applying the Fiat-Shamir transform [28] to make the proof non-interactive, we let $\mathbf{b}$ be the output of a hash function applied to the first message and $x$. Then, the proof transcript is reduced to $\pi_{Lin} = (\beta, \langle [\hat{z}_i] \rangle_{\hat{n}})$. To estimate the size of the transcript, we assume that due to the Gaussian distribution of $\hat{y}_i$, the size of $\hat{z}_i$ is within 6 standard deviations of the distribution. Using this, one obtains that each $\hat{z}_i$ is of size $kN \log_2(6\sigma_\mathbf{b})$ bits. To improve the efficiency of the proof one can compress $\hat{z}_i$ to $\tilde{h}(k - n)N \log_2(6\sigma_\mathbf{b})$ bits by checking an approximate equality instead, which was e.g. described in Aranha et al. [4, Section 3.2] in more detail.

#### B.2 Bounded Amortized Zero-Knowledge Proof

We now introduce a Zero-Knowledge Proof for a statement similar to the one considered in 4, except that the bounds shown on the secret do not have to be as accurate. This is sufficient in certain situations, and such proofs can be much more efficient (in terms of computation and communication) than those from Section 4.

Let $A$ be a publicly known $r \times \nu$-matrix over $\mathbb{R}_q$, let $s_1, s_2, \ldots, s_r$ be bounded elements in $\mathbb{R}_q^{\nu}$ and let $A_i = t_i$ for $i \in [\nu]$. Letting $S$ be the matrix whose columns are $s_i$ and $T$ be the same matrix for $t_i$, but defined over $Z_N^{\nu}$ instead of $\mathbb{R}_q$ as in the previous subsection, then Baum et al. [6] give an efficient amortized zero-knowledge proof of knowledge for the relation

$$R_{Bnd} = \left\{ (x, w) \mid x = (A, T) \wedge w = S \wedge \forall i \in [\nu] : t_i = A_i \wedge ||s_i||_2 \leq 2 \cdot B_{Bnd} \right\}.$$

The protocol $\Pi_{Bnd}$ is depicted in Figure 4. We can use a challenge matrix $C$ with entries sampled from the set $C_{Bnd} = \{0, 1\}$, then this
Verifiable Mix-Nets and Distributed Decryption

C.1 Soundness

Lemma 2. Let Encode be the encoding of a Reed-Solomon code of dimension $g' = nN + \eta$ and length $l$. Furthermore, let $g' \leq g \leq l < q$. Suppose that there is an efficient deterministic prover $P^*$ convincing

\[ \epsilon > 2 \max \left\{ \left( \frac{g}{l - \eta} \right)^\eta, \frac{1}{q - \eta} + \left( \frac{g - g'}{6l} \right)^\eta, \right. \]

\[ \left. 2 \left( \frac{1 - 2(g - g')}{3l} \right)^\eta \right\}., \]

Then there exists an efficient probabilistic extractor Ext which, given access to $P^*$ either produces vectors $s_i \in \{-1, 0, 1\}^{3Nq}$ such that $t_i = \sum_{j \in \hat{X}} A_{s_i}$ for all $i \in \mathbb{Z}$, or breaks the binding property of the commitment scheme (ComAux, OpenAux), or finds a hash collision in expected time at most $64T$ where

\[ T := \frac{3}{\epsilon} + \epsilon (2 - (q/(1 - \eta)))^\eta \]

and running $P^*$ takes unit time.
Proof. First, we construct an extractor \( \text{Ext} \) as in [13] which does the following 8 times:

1. First, run \( \mathcal{P}^* \) on random challenges from \( \mathcal{V} \) until an accepting transcript is found. Abort if none is found after \( 8/e \) steps.
2. Let \( I_j \) be the challenge set where \( \mathcal{P}^* \) responded and let \( E[I_j, \text{MerklePathSet}_i] \) be the opened columns and Merkle tree paths. Fixing the other challenges, \( \text{Ext} \) adaptively re-runs the proof for different challenges \( I_2, I_3, \ldots \) that contain so far unopened columns and collects these. If any collision in the Merkle tree is found then \( \text{Ext} \) outputs the hash collision and terminates, otherwise it continues until it has collected a set \( J \) of at least \( g \) columns, or until \( 8e/6 \) time passed.
3. Finally, \( \text{Ext} \) re-runs \( \mathcal{P}^* \) 16 times with completely fresh challenges, obtaining new \( E[I_j, \text{MerklePathSet}_i] \). If for some of these instances, the accepting transcript contains a hash collision in the Merkle tree (colliding with \( J \)) or \( c_d \) is opened with a different opening than in the first step, then output the hash collision or the respective two different openings of \( c_d \).

Using a standard heavy-row argument, [13] shows that \( \text{Ext} \)'s total runtime is bounded by the term mentioned in the Lemma. Moreover, define \( S \) as the event that the \( \mathcal{P}^* \) outputs a valid proof in the last step and \( C \) be the event that the values \( \mathcal{P}^* \) outputs to \( \text{Ext} \) in the last step of the extractor are consistent (i.e. no hash collisions and the commitment was not opened differently than before). Then [13] show that it must hold that \( \Pr[S \land C] > e/2 \). We now show that if we cannot use \( J \) to decide to a valid witness, then the success probability of \( \mathcal{P}^* \) must be lower than the given bound.

Define \( C^* \) as the RS code obtained from \( C \) (generated by \( \text{Encode} \)) when restricted to the indices of \( J \). As \( |J| = g \), \( C^* \) has length \( g \) and minimum distance \( d^* = g - g' + 1 \). For any \( x \in \mathbb{Z}_q^g \) we define the minimum distance of \( x \) to \( C^* \) as \( d'(C^*, x) = \min_{c \in C^*} d(c, x) \).

Let \( E^* := E[J] \) be the matrix that was extracted by the extractor and let \( d^* \) be the opening message of the commitment. Assume that there exists \( x \in \mathbb{Z}_q^{2g} \) such that \( d'(C^*, xE^*) \geq d'/3 \). Then by Bootle et al. [12, Appendix B], any random linear combination of \( E^* \) (in particular, we compute and output such a combination in the proof as \( \bar{h} \)) has distance \( \geq d'/6 \) from \( C^* \) except with probability \( 1/(q - \tau) \). Similar to in [13] we can use this to deduce that in such a case, it must hold that

\[
\frac{e}{2} < \frac{1}{q - \tau} + \left( 1 - \frac{g - g'}{6d} \right) \eta
\]

which contradicts the bound on \( \eta \) in this lemma. Therefore, each row of \( E^* \) must be within \( d'/3 \) of \( C^* \), meaning that it is efficiently decodable. Let \( h^*, s_i^*, r_{ij}^*, o_{ij}^* \) be the respective decoded values and \( r_{ij}^*, r_{ij}^* \) be the randomness. We consider the composition of the aforementioned row values as \( V \) and the randomness used in the encoding as \( R \). By applying another result from Bootle et al. [12, Appendix B] we have that for any vector \( y \in \mathbb{Z}_q^{2g} \) it holds that \( d'(\text{Encode}_j(yV, yR), yE^*) < d'/3 \). In other words, any linear transformation \( y \) when applied to the possibly noisy codewords \( E^* \) is within distance \( d'/3 \) of the codeword obtained from encoding \( V, R \) after applying the same transformation \( y \). That means that \( \bar{f} \) is constructed from \( s_{i0}^*, r_{ij}^* \) as we would expect.

Similar as in Bootle et al. [13, Corollary 3.7] one can show that if there are \( \leq (\epsilon/4)(q - \tau) \) choices of \( x \) such that

\[
\bar{f} = \ell_0(x)s^*_0 + \sum_{i=1}^{2} \sum_{j=0}^{r} \ell_i(x)\ell_0(x)^j s_{i,j}^* \quad \tag{3}
\]

\[
\frac{1}{\ell_0(x)} \cdot f \circ (\bar{f} - 1) \circ (\bar{f} + 1) \quad \tag{4}
\]

\[
= \ell_0(x)s^*_0 + \sum_{i=1}^{2} \sum_{j=0}^{r} \ell_i(x)\ell_0(x)^j s_{i,j}^* \quad \tag{5}
\]

then \( \epsilon < 4(1 - 2(q - 3)/q, \tau) \), contradicting the bound on \( \epsilon \) in the lemma. By multiplying 4 with \( \ell_0(X) \) we obtain the equation

\[
\bar{f} \circ (\bar{f} - 1) \circ (\bar{f} + 1) = \ell_0(X)^2x^0 - \sum_{i=1}^{2} \sum_{j=0}^{r} \ell_i(x)\ell_0(X)^j s_{i,j}^* \quad \tag{6}
\]

which equals 0. Replacing \( \bar{f} \) according to Equation 3 means that the above expression is of degree at most \( 9 \cdot \tau \). But it is 0 for more choices of \( x \) because \( \epsilon > 36\tau/\tau(q - \tau) \), meaning that the expression itself must be the zero-polynomial. Reducing Equation 6 modulo \( \ell_0(X) \) and knowing that all \( \ell_i(X) \) are independent, it follows that for all \( i \in [\tau] \) the value \( s_{i0}^* \) is in \((-1, 0, 1)^N\).

Additionally, we have that

\[
d^* = \frac{1}{\ell_0(X)} \cdot \left( \sum_{i=1}^{\tau} t_i\ell_i(X) - A\bar{f} \right)
\]

and replacing again \( \bar{f} \) with Equation 3 yields \( d^* \) to equal

\[
-As^*_0 + \frac{1}{\ell_0(X)} \cdot \left( \sum_{i=1}^{\tau} t_i\ell_i(X) - A\left( \sum_{i=1}^{\tau} \sum_{j=0}^{r} \ell_i(x)\ell_0(x)^j s_{i,j}^* \right) \right)
\]

Since \( d^* \) has been committed to before \( x \) is chosen, it must be that \( d^* \) is the constant of the polynomial on the right, so we have that \( d^* = -As^*_0 \) and therefore

\[
\sum_{i=1}^{\tau} \sum_{j=0}^{r} \ell_i(x)\ell_0(x)^j s_{i,j}^* = \sum_{i=1}^{\tau} t_i\ell_i(X).
\]

Again reducing modulo \( \ell_0(X) \) reveals that

\[
\sum_{i=1}^{\tau} \ell_i(x)As_{i0}^* = \sum_{i=1}^{\tau} t_i\ell_i(X)
\]

and by the independence of the \( \ell_i(X) \) modulo \( \ell_0(X) \) we have that \( t_i = As_{i0}^* \) for all \( i \in [\tau] \), which proves the claim. \( \square \)

C.2 Zero-Knowledge

Lemma 3. There exists an efficient simulator \( \text{Sim} \) which, given \( x, \beta_0, \beta_{ij}, \ldots, \beta_{1,2}, I \) outputs a protocol transcript of the protocol in Figure 5 whose distribution is indistinguishable from a real transcript between an honest prover and honest verifier.

Proof. Towards constructing a simulation, observe that
We compute the abort probability of Step 3. First, assume that Ext' outputs the same opening with a probability of at least 1/2 in 2/3rds of the heavy rows. It can easily be shown that we can otherwise construct an algorithm that breaks the binding property of the commitment scheme with an expected constant number of calls to Ext' and by using Proposition 1. Moreover, by a counting argument, there must be > \(\frac{3}{2}\left(\frac{l-1}{q}\right)^N\) heavy rows: Assume to the contrary that there are at most \(\frac{3}{2}\left(\frac{l-1}{q}\right)^N\) heavy rows. Let each of the heavy rows have only one outputs (verifier always accepts), and each other row be filled with \(\epsilon/2\) ones. This is the maximal case of having only \(\frac{3}{2}\left(\frac{l-1}{q}\right)^N\) heavy rows. But then the acceptance probability can be at most

\[
\frac{3}{2} \left(\frac{l-1}{q}\right)^N + \left[1 - 3\left(\frac{l-1}{q}\right)^N\right] \epsilon/2 < \frac{3}{2} \left(\frac{l-1}{q}\right)^N + \epsilon/2
\]

which contradicts the assumption that \(\mathcal{P}'\) has success > \(\epsilon\).

Assume that Ext' extracts a valid witness \(\pi, \{f_i, r_i\}_{i \in [r]}\) for input commitments \(\{(p_i, m_i)\}_{i \in [r]}\) and messages \(\{p_i, m_i\}\) while the extracted \(\hat{m}_i = (f_i c_{2,i} - A_{C,2} r_i)/f_i\) do not form a permutation on the \(\hat{m}_i\). Then there exists an \(i \in [r]\) such that

\[
\hat{f}_i \cdot \langle p, c_{2, \pi^{-1}(i)} \rangle = \langle p, A_{C,2} r_i \rangle + f_i (p, \hat{m}_i)
\]

but

\[
\hat{f}_i \cdot c_{2, \pi^{-1}(i)} = A_{C,2} r_i + f_i (\hat{m}_i + \delta)
\]

where \(\hat{m}_i = \hat{m}_i + \delta\) for a non-zero vector \(\delta\). Combining both equations, we get that 0 = \(\langle p, \delta \rangle\). This implies that the polynomial \(\sum_{i=0}^{l-1} \delta[i]X^i\) that has coefficients from \(\delta\) must be zero at point \(h\) whose powers generate the vector \(p\). Since this polynomial is of degree \(l - 1\), by Arana et al. [4, Lemma 2] it can be 0 in at most \((l-1)^N\) positions without being the 0-polynomial itself. But since the transcript is extractable and thus accepting for strictly more than \((l-1)^N\) choices of \(h\) as we have a constant fraction more rows that are heavy, we must have that \(\delta = 0\) to begin with.

**E. BGV MIXING SECURITY**

First, we define completeness, soundness and simulatability for a mixing protocol \(\Pi_{\text{Mix}}\) executed by a prover \(\mathcal{P}\), with respect to a generic encryption scheme \(\mathcal{E} = (K\text{Gen}, \text{Enc}, \text{Dec})\).

**Definition 15 (Mixing Completeness).** We say that the mixing protocol \(\Pi_{\text{Mix}}\) is complete if for honest PPT parties \(\mathcal{P}\) and \(\mathcal{V}\) that follows the protocol then \(\mathcal{P}\) on input a set of honestly generated ciphertexts will output a new set of ciphertexts together with a proof such that \(\mathcal{V}\) accepts the proof and the output ciphertexts decrypt to the same set of messages as the input ciphertexts. Hence, we want that the following is less than \(1 - \epsilon(x):\)

\[
P_{\mathcal{V}}(\mathcal{V}(\text{pp, pk, sk}) \leftarrow K\text{Gen}(\mathcal{V}^*)) \left[ \mathcal{V}(\text{pp, pk, sk}) \leftarrow K\text{Gen}(\mathcal{V}^*) \right]
\]

where the probability is taken over \(K\text{Gen}\) and \(\mathcal{P}\).

**Definition 16 (Mixing Soundness).** We say that the mixing protocol \(\Pi_{\text{Mix}}\) is sound if a dishonest PPT adversary \(\mathcal{A}\) that can behave arbitrarily on input a set of honestly generated ciphertexts will not be able to output a new set of ciphertexts together with a proof such that an honest \(\mathcal{V}\) accepts the proof but the output ciphertexts...
decrypt to a different set of messages than the input ciphertexts. Hence, we want that the following is less than \( \epsilon(x) \):

\[
Pr \left[ \begin{array}{c}
\{m_i \}_{i \in [r]} \neq \text{Dec}(sk, \{e_i \}_{i \in [R]}) \\
\{e_i \}_{i \in [R]} \leftarrow \text{Enc}(pk, \{m_i \}_{i \in [r]}) \\
\{e_i \}_{i \in [R]} \leftarrow \text{Enc}(pk, \{m_i \}_{i \in [r]}) \\
\{e_i \}_{i \in [R]} \leftarrow \text{Enc}(pk, \{m_i \}_{i \in [r]})
\end{array} \right| \leq \sum_{b \in \{0, 1\}} Pr(b) \left[ \begin{array}{c}
\{m_i \}_{i \in [r]} \leftarrow \text{Dec}(sk, \{e_i \}_{i \in [R]}) \\
\{e_i \}_{i \in [R]} \leftarrow \text{Enc}(pk, \{m_i \}_{i \in [r]}) \\
\{e_i \}_{i \in [R]} \leftarrow \text{Enc}(pk, \{m_i \}_{i \in [r]}) \\
\{e_i \}_{i \in [R]} \leftarrow \text{Enc}(pk, \{m_i \}_{i \in [r]})
\end{array} \right]
\]

where the probability is taken over \( \text{KGen} \), \( \text{Enc} \) and \( \mathcal{A} \).

**Definition 17 (Mixing Simulatability).** We say that the mixing protocol \( \Pi_{\text{Mix}} \) is simulatable if a PPT adversary \( \mathcal{A} \) that on input a set of honestly generated ciphertexts can not distinguish between a real execution of the mixing protocol with accepting output and a protocol execution from a PPT simulator \( \mathcal{S} \) (given a set honestly mixed output ciphertexts) producing a simulated mixing proof. Hence, we want that

\[
Pr \left[ \begin{array}{c}
\{m_i \}_{i \in [r]} \leftarrow \text{Dec}(sk, \{e_i \}_{i \in [R]}) \\
\{e_i \}_{i \in [R]} \leftarrow \text{Enc}(pk, \{m_i \}_{i \in [r]}) \\
\{e_i \}_{i \in [R]} \leftarrow \text{Enc}(pk, \{m_i \}_{i \in [r]}) \\
\{e_i \}_{i \in [R]} \leftarrow \text{Enc}(pk, \{m_i \}_{i \in [r]})
\end{array} \right| \leq \sum_{b \in \{0, 1\}} Pr(b) \left[ \begin{array}{c}
\{m_i \}_{i \in [r]} \leftarrow \text{Dec}(sk, \{e_i \}_{i \in [R]}) \\
\{e_i \}_{i \in [R]} \leftarrow \text{Enc}(pk, \{m_i \}_{i \in [r]}) \\
\{e_i \}_{i \in [R]} \leftarrow \text{Enc}(pk, \{m_i \}_{i \in [r]}) \\
\{e_i \}_{i \in [R]} \leftarrow \text{Enc}(pk, \{m_i \}_{i \in [r]})
\end{array} \right]
\]

where the probability is taken over \( \text{KGen} \), \( \text{Enc} \), \( \mathcal{P} \), \( S \) and \( \mathcal{A} \).

Next, we state the security of \( \Pi_{\text{Mix}} \), where the encryption scheme is instantiated with the BGV scheme in Section 2 and the mixing protocol is instantiated with the verifiable shuffle in Section 5.

**Theorem 10 (Completeness).** If the protocols \( \Pi_{\text{Small}} \) and \( \Pi_{\text{Shuf}} \) are complete, then \( \Pi_{\text{Mix}} \) always terminates. Moreover, if the input ciphertexts \( e_i \) have noise bounded by \( \text{BDec} \), and the total noise added in \( \Pi_{\text{Mix}} \) is bounded by \( \text{BMix} \) such that \( \text{BDec} + \text{BMix} < \lfloor q/2 \rfloor \), then the output ciphertexts \( \hat{e}_i \) decrypt to the same set of messages as \( e_i \).

We sketch the argument. The simulator simulates the zero-knowledge proofs of \( \Pi_{\text{Small}} \) and \( \Pi_{\text{Shuf}} \), respectively, with advantages \( \epsilon_1, \epsilon_2 \), such that \( \epsilon_0 \leq \epsilon_1 + \epsilon_2 < \epsilon_4. \) The runtime of \( \Pi_{\text{Small}} \) and \( \Pi_{\text{Shuf}} \) is simulatable.

We begin with the verifiable shuffle protocol.

First, we replace the \( \Pi_{\text{Shuf}} \) arguments with simulated arguments, which gives us a distinguisher \( \text{Adv}_1 \) for the \( \Pi_{\text{Shuf}} \) honest verifier simulator. Second, we replace the \( \Pi_{\text{Small}} \) arguments by simulated arguments, which gives us a distinguisher \( \text{Adv}_2 \) for the \( \Pi_{\text{Small}} \). Third, we replace the commitments to ciphertexts with commitments to zero, which gives us an adversary \( \text{Adv}_3 \) against hiding for the commitment scheme. Fourth, we replace the output with given ciphertexts to the same messages, which gives us an adversary \( \text{Adv}_4 \) against CPA security.

After the changes, we are left with the claimed simulator for the actively secure protocol and the claim follows.

**F CHOOSING PARAMETERS CONCRETELY**

We let the success probability of each of the zero-knowledge protocols be \( 1/M \approx 1/3 \). We will use the following parameters, where we note that the commitments used in the shuffle and in the amortized proofs are only used once, while the proof of linearity in the decryption protocol depends on a commitment to the secret key share each time. However, that is the only part that is reused, and we can use a smaller standard deviation for the other commitment.

The proofs of linearity have two terms, and each of them must have a success probability of \( 1/\sqrt{3} \). This gives \( \sigma_c = 0.954 \sigma_{\text{Com}} \text{VKN} \).

For the re-usable commitments we get \( \sigma_c = 22 \sigma_{\text{Com}} \text{VKN} \). The amortized proof also has two checks, and we get a standard deviation of \( 0.954 \|S^*C^*\|_2 \), where \( \sigma_{\text{Simul}} \) and \( \sigma_{\text{Comp}} \) are depending on the norm of the elements in the rows of \( S^* \).

We let the noise bounds \( \text{BKey} = \text{BErr} = 1 \) for the encryption.

To be able to choose concrete parameters for the mix-net, we need to estimate how much noise is added to the ciphertexts through the two stages of the protocol: 1) the shuffle phase, and 2) the decryption phase. Each part of the system contributes the following amount of noise to the ciphertexts:

- Fresh ciphertext: \( B_{\text{Start}} = \rho \|e_{r,2} - e_{r,1}\|_\infty + \|m\|_\infty \).
We get the following upper bounds, which for

\[
\begin{align*}
B_{\text{Dec}} & = (\xi_1 + 1) p (2N + 1) + \left(\frac{p - 1}{2}\right), \\
B_{\text{SHUF}} & = p (2N + 1),
\end{align*}
\]

which for \(\xi_1\) shuffles gives us
\[
B_{\text{Dec}} = (\xi_1 + 1) p (2N + 1) + \left(\frac{p - 1}{2}\right).
\]

Finally, we need to make sure that
\[
B_{\text{Dec}} + B_{\text{DDec}} < q/2,
\]

where
\[
B_{\text{DDec}} = 2 p \xi_2 B_{\text{BND}}
\]
because of the soundness slack of the amortized proof of bounded values. A honestly generated value \(E_{i,j}\) is bounded by \(2^{3\sec} B_{\text{Dec}}/p \xi_2\), but the proof can only guarantee that the values are shorter than some larger bound \(2 B_{\text{BND}}\) (following Baum et al. [6, Lemma 3]) that depends on the number of equations in the statement. Define \(S_{i,k}'\) to be the first \(k\) rows of \(S'\) and define \(S_{i,k+1}'\) to be the last row of \(S'\). For batches of \(N\) equations, we then get that:
\[
B_{\text{BND}} \leq \sqrt{2N} \cdot \sigma_{\text{BND}} \leq \sqrt{2N} \cdot 0.954 \cdot \max_{2} \left\| S'_{i,k} C' \right\|_2
\]
\[
\leq 1.35 \cdot \sqrt{N} \cdot \max_{2} \left\| S_{i,k+1}' \right\|_2 \cdot \max_{2} \left\| C' \right\|_\infty
\]
\[
\leq 1.35 \cdot k \cdot \sqrt{N} \cdot N \cdot \sigma_{\text{BND}}
\]
and, similarly,
\[
B_{\text{BND}} \leq \sqrt{2N} \cdot \sigma_{\text{BND}} \leq \sqrt{2N} \cdot N \cdot \left\| E_{i,j} \right\|_\infty
\]

with \(B_{\text{BND}}\) for rows 1 to \(k\) of \(Z\) and \(B_{\text{BND}}^{\sigma}\) for the last.

We fix plaintext modulus \(p = 2\), statistical security parameter \(\text{sec} = 40\), and need \(N = 4096\) when \(q\) is large to provide proper security. This allows for votes of size \(4096\) bits, which should be a feasible size for real-world elections. We let the number of shuffle and decryption servers be \(\xi_1 = 4\). It follows that
\[
B_{\text{Dec}} < 2^{27}
\]
and
\[
B_{\text{DDec}} < 2^{76.5}
\]
We then set \(q = 2^{28}\), and verify that
\[
\max_{i \in [r]} \left\| u_i - su_i \right\| < 2 \cdot (2^{17} + 2^{76.5}) < q.
\]

Finally, we must decide on parameters for the exact proof of shortness. The soundness of the protocol depends on the ratio between the number of equations and the size of the modulus. We choose to compute the proof in batches of size \(N\) instead of computing the proof for all \(r\) commitments at once. Then we get
\[
18 N/(q - N) \approx 2^{-62}
\]
and hence, we must compute each proof twice in parallel to achieve negligible soundness. Furthermore, we choose
\[
g \approx 2^{20}, \ell \approx 2^{20}, n \approx 2^{35}, q = 325\text{ to keep the soundness} \approx 2^{-62}
\]

The total size of \(\pi_{\text{SMALL}}\), by instantiating \(1\), is \(\approx 20\text{ KB}\).

We give a complete set of parameters in Table 5, and the concrete sizes of each part of the protocol in Table 2. Each voter submits a ciphertext size of approximately 80 KB. The size of the mix-net, including ciphertexts, commitments, shuffle proof, and proof of shortness, is approximately 370 KB per mixing node \(S_t\). The size of the decryption phase, including partial decryptions, commitments, proofs of linearity, and proofs of boundedness, is approximately 157 KB per decryption node \(D_j\).

### G SECURITY IN THE QUANTUM RANDOM ORACLE MODEL

In this work, we have chosen parameters for all primitives such as to make our voting protocol secure against all known classical attacks. Since we only use assumptions that are assumed to be post-quantum secure, it is obvious to ask if our construction is also post-quantum secure. We cannot answer this within this work, due to the complexity of proving such a statement.

As a “second-best” approach, we can alternatively look at the post-quantum security of the individual building blocks. Here, of particular importance are the NIZKs that this work uses. We use two different types of proofs, namely those exploiting the homomorphism of an underlying OWF (such as \(\Pi_{\text{LIN}}, \Pi_{\text{BND}}\)) and those that rely only on commitments and a combinatorial argument (\(\Pi_{\text{SMALL}}\)). Both of these are made non-interactive in the ROM using the Fiat-Shamir transform, which becomes the QROM in the quantum setting. Here, the recent work of [26] could be used to show that \(\Pi_{\text{SMALL}}\) is online-extractable in the QROM and therefore still secure, for adjusted parameters.

Unfortunately, the situation is a bit more problematic for the homomorphism-based proofs. There, the most efficient QROM Fiat-Shamir approach that we are aware of is [25], which applies to \(\Sigma\)-protocols. Their work implies a large loss in parameters that they show to be inherent, and this loss grows with the number of rounds of the protocol. Even worse, new techniques would have to be developed to prove the security of \(\Pi_{\text{BND}}\) as it seems unlikely that [25] applies to it. To achieve provable security of all these NIZKs in the QROM, it would be better to replace the homomorphic OWF-based protocols with Commit-And-Open-based proofs following \(\Pi_{\text{SMALL}}\). We expect that this would come at a significant cost in proof size as well as prover runtime, impacting the practicality of our construction.

A more optimistic view, which we share, is that known counterexamples in the QROM on NIZKs such as [3] are contrived and that there are no known attacks (beyond Grover’s algorithm) for the NIZKs that we use. One could therefore argue that our construction is plausibly post-quantum. We leave a more detailed post-quantum security analysis, which also includes parameter choices to withstand attacks based on Grover’s algorithm, for future work.
H SECURITY OF THE VOTING PROTOCOL

Here we provide a more formal description of the voting protocol described in Section 3, give security notions, sketch a security proof, and discuss the security properties of the full voting protocol.

H.1 Verifiable Voting with Return Codes

A verifiable cryptographic voting scheme in our architecture is usually defined in terms of algorithms for the tasks of election setup, casting ballots, counting cast ballots, and verifying the count. We also need algorithms for voter registration and pre-code computation to support return codes. Finally, we need algorithms for shuffling and distributed decryption to accurately model the counting process.

The setup algorithm Setup outputs a public key pk, decryption key shares dk, and a code key ck.

The register algorithm Reg takes a public key pk as input and outputs a voter verification key vvk, a voter casting key vck and a function f from ballots to pre-codes.

The cast algorithm Cast takes a public key pk, a voter casting key vck and a ballot b, and outputs an encrypted ballot ev and a ballot proof πv.

The code algorithm Code takes a code key ck, an encrypted ballot ev and a proof πv as input and outputs a pre-code i or ⊥. (If the code key ck is ⊥, the algorithm outputs 0 or 1.)

The shuffle algorithm Shuffle takes a public key pk and a sequence of encrypted ballots ev, and outputs a sequence of encrypted ballots ev’ and a proof of shuffle πσ.

The verify algorithm Verify takes a public key pk, two sequences of encrypted ballots ev, and ev’ and a proof πσ, and outputs 0 or 1.

The distributed decryption algorithm DistDec takes a decryption key dk and a sequence of encrypted ballots ev, and outputs a sequence of ballot decryption shares sv and a decryption proof πd.

The combining algorithm Comb takes a public key pk, a sequence of encrypted ballots ev, ballot decryption share sequences sv1, sv2, . . . , svl, with proofs πd1, πd2, . . . , πdl, and outputs either ⊥ or a sequence of ballots v1, v2, . . . , vl.

A cryptographic voting scheme is lcorrect if for any (pk, (dkj)), (ck) output by Setup and any (vvk1, vck1, f1), . . . , (vvkl, vckl, fl) output by Reg(pk), any ballots v1, . . . , vl, any (ev1, . . . , evl) output by Cast(pk, vck1, a1), i = 1, . . . , l, any sequence of l sequences of encrypted ballots ev1(i) with proofs πs1(i), output by Shuffle(pk, ev1(j−1)), any ballot decryption shares sv1, . . . , svl with proofs πd1, . . . , πdl, output by DistDec(dk, ev1(i)), i = 1, l, any sequence of l′ and any (v′1, . . . , v′l′) possibly output by Comb(pk, ev1(i), sv1, . . . , svl′, πd1, . . . , πdl′), then:

- Code(c, vvk1, ev1, πs1) = f1(v1). Code(⊥, vvk1, ev1, πs1) = 1,
- Verify(pk, ev1(j−1), ev1(j), πs1) = 1 for j = 1, 2, . . . , l,
- Comb(pk, ev1(i), sv1, . . . , svl′, πd1, . . . , πdl′) did not output ⊥, and
- v1, . . . , vl equals v′1, . . . , v′l′, up to order.

We also require that the distribution of evi only depends on pk and vi, not vckj.

For any such scheme we define a decryption algorithm Dec that first applies a number of shuffles (possibly zero) to the single ciphertext, then applies DistDec and Comb in sequence. Note that this algorithm will not actually be used, but it simplifies the definition of security.

H.2 Our Scheme

Our voting scheme combines the BGV encryption together with our shuffle (Section 5) and distributed decryption (Section 6). We adopt the techniques from Aranha et al. [4] to get extractability and code voting, but omit the details.

- Setup computes pkC ← KeyGenC, (pkV, dkV) ← KeyGenVE, (pkR, dkR) ← KeyGenVE as well as key shares dkij for every decryption server. The public key pk = (pkC, pkV, pkR), the decryption share is dk = (pkC, dkij) and the code key is ck = (pkC, pkV, dkR).
- Reg takes pk = (pkC, pkV, pkR) as input. It samples a $\in$ Rp and computes (c0, d0) ← CompCk, a. The voter verification key is vv = (c, a, d, a) and the function f is v = v + a.
- Cast takes pk = (pkC, pkV, pkR), vck = (a, c0, d0) and v as input. It computes v ← EncVE(pkV, v), v′ ← a + v and w ← EncVE(pkR, f), along with a proof πv0 that v and w are well-formed ciphers and that e0 is a commitment to the difference of the encryptions. The encrypted ballot is ev = a, while the ballot proof is πv = (w, πv0).
- Code takes ck = (pkC, pkV, dkR), a voter verification key vv, an encrypted ballot ev = v and a ballot proof πv = (w, πv0) as input. It verifies πv0 and outputs ⊥ if verification fails. Otherwise, it computes $\hat{v} ← DecVE(dkR, w)$ and outputs $\hat{v}$. (If ck = ⊥, it outputs 1 if and only if it accepts πv0.)
- The shuffle algorithm Shuffle and the verify algorithm Verify are as described in Section 5. The distributed decryption algorithm DistDec and the combining algorithm Comb are as described in Section 6.

It is straightforward to verify that the scheme is correct.

H.3 Security Notions

Our notion of confidentiality is similar to the usual ballot box privacy notions [10]. An adversary that sees both the contents of the ballot box, the intermediate shuffles and the decrypted ballot shares should not be able to determine who cast which ballot. This should hold even if the adversary can see pre-codes, learn the code key, some voter casting keys, and some decryption key shares, insert adversarially generated ciphertexts into the ballot box, introduce adversarially generated intermediate shuffles and publish adversarially chosen decrypted ballot shares.

Our notion of integrity is again fairly standard, adapted to return codes. An adversary should not be able to cause an incorrect pre-code or inconsistent decryption or non-unique decryption, even if the adversary knows all of the key material.
We define security notions for a verifiable cryptographic voting scheme using an experiment where an adversary $\mathcal{A}$ is allowed to reveal keys, make challenge queries, create ciphertexts, ask for ciphertexts to be shuffled, create shuffles, and ask for ballot shares. We use this experiment to define games both for confidentiality and integrity. The experiment works as follows:

- Sample $b \xleftarrow{} \{0, 1\}$ and let $L, L', L''$ be empty lists.
- $(pk, (dk_i), ck) \leftarrow$ Setup. For $i = 1, \ldots, L': (vck_i, vck_i, f_i) \leftarrow$ Reg$(pk)$. Send $(pk, vck_1, \ldots, vck_{L'})$ to $\mathcal{A}$.
- On a voter reveal query $i$, send $(vck_i, f_i)$ to $\mathcal{A}$. On a decrypt reveal query $i$, send $ck_i$ to $\mathcal{A}$. On a code reveal query, send $ck$ to $\mathcal{A}$.
- On a challenge query $(i, v_0, v_1)$, compute $(ev, \sigma) \leftarrow$ Cast$(pk, vck_i, q_b)$. $\hat{r} \leftarrow$ Code$(ck, vck_i, ev, \sigma)$, append $(i, v_0, v_1, ev, \sigma)$ to $L$. Send $(ev, \sigma)$ to $\mathcal{A}$.
- On a chosen ciphertext query $(i, ev, \sigma)$, compute $\hat{r} \leftarrow$ Code$(ck, vck_i, ev, \sigma)$. If $\hat{r} \neq \perp$, append $(i, \perp, ev, \sigma)$ to $L$. Send $\hat{r}$ to $\mathcal{A}$.
- On a shuffle query $(i, ev)$, compute $(ev', \sigma) \leftarrow$ Shuffle$(pk, ev)$, then record $(ev, ev', \sigma)$ in $L'$. Send $(ev, \sigma)$ to $\mathcal{A}$.
- On a chosen shuffle query $(ev, ev', \sigma)$, we record it in $L''$ if and only if Verify$(pk, ev, ev', \sigma) = 1$.
- On a ballot decryption share query $(i, ev)$, we then compute $(sv_i, \pi_{d,i}) \leftarrow$ DistDec$(dk_i, ev)$, record $(i, ev, sv_i, \pi_{d,i})$ in $L''$ and send $(sv_i, \pi_{d,i})$ to $\mathcal{A}$.
- On a test query $(ev, sv_1, \ldots, sv_{L'}, \pi_{d,1}, \ldots, \pi_{d,L'})$, then compute the result $\leftarrow$ Comb$(pk, ev, sv_1, \ldots, sv_{L'}, \pi_{d,1}, \ldots, \pi_{d,L'})$ and send result to $\mathcal{A}$.

Eventually, the adversary outputs a bit $b'$.

The confidentiality game follows the usual left-or-right game pattern, where an adversary makes challenge queries and must determine the value of the bit $b$. The test query is irrelevant to the confidentiality game.

The integrity game follows the usual pattern where the adversary aims to achieve certain inconsistencies, either during a code query or a test query. The inconsistencies are that a pre-code does not match the encrypted ballot, that an outcome verifies as correct but is inconsistent with the challenge ciphertexts chosen for counting, or that there is no unique decryption. The test query is not strictly needed. We could have had the adversary output its encrypted ballots and ballot decryption shares instead of making a test query. But the test query pattern is convenient in many similar settings, so we include it.) The bits $b, b'$ are not really used in the game for integrity, nor is the shuffle query. The challenge query is used to create honestly encrypted ballots.

Confidentiality fails trivially if the counting phase trivially reveals the challenge bit. This happens unless the left-hand and right-hand ballots are identical and up to order. (Recall that the adversary should figure out who cast which ballots, not what ballots were cast.) Confidentiality also fails trivially if the adversary makes more than one challenge query or chosen ciphertext query for any given voter. And confidentiality fails trivially if the adversary reveals too much key material. We should not count executions where confidentiality fails trivially towards the adversary’s advantage. Technically, we count this using a freshness event when evaluating the advantage.

In the execution of this experiment, we say that a sequence of encrypted ballots $ev$ is valid if tuples $(i_1, v_0, v_1, ev_1, \pi_{0,1}), \ldots, (i_L, v_0, v_1, ev_L, \pi_{0,L})$ in $L$ and $L'$ contains a sequence of tuples $(ev^{(j-1)}_1, ev^{(j)}_1, \pi_{0,1}), j = 1, \ldots, L$, such that $ev_0 = (ev_1, \ldots, ev_L)$ and $ev_L = ev$. In this case we also say that $ev$ derives from $(i_1, v_0, v_1, ev_1, \pi_{0,1}), \ldots, (i_L, v_0, v_1, ev_L, \pi_{0,L})$. A valid sequence $ev$ is honest if at least one of the tuples $(ev^{(j-1)}_1, ev^{(j)}_1, \pi_{0,1})$ originated with a shuffle query. A valid sequence $ev$ is balanced if the ballot sequence $(v_0, \ldots, v_1)$ equals $(v_0, \ldots, v_1)$, up to order.

We define events related to confidentiality and integrity. Let $E_g$ be the event that $b = b'$. Let $E_f$ denote the event that an execution is fresh, which is true if the following are satisfied: there is no decrypt reveal query for any valid $i$; for any $i$, there is either no challenge query, or at most one challenge query and no voter reveal query or chosen ciphertext query; and for any ballot decryption share query $(\cdot, ev)$, the sequence $ev$ is honest and balanced at the time of the ballot decryption share query.

Let $F_i$ (incorrect pre-code) be the event that for some chosen ciphertext query $(i, ev, \sigma)$ where Code$(ck, vck_i, ev, \sigma) = \hat{r} \neq \perp$, we have that either $\text{Dec}((dk_i, ev)) = \perp$ or $\text{Dec}((dk_i, ev)) = v$ and $f_i(v, \sigma) \neq \hat{r}$.

Let $F_c$ (failure to bound) be the event that a test query gets result $\perp$ when $ev$ is valid and $(ev, sv_1, \pi_{d,1})$ is in $L''$ for $i = 1, \ldots, L_d$.

Let $F_d$ (inconsistent decryption) be the event that a test query $(ev, sv_1, \ldots, sv_{L'}, \pi_{d,1}, \ldots, \pi_{d,L'})$ with result $= (v_1, \ldots, v_L)$, where $ev$ derives from $(i_1, v_0, v_1, ev_1, \pi_{0,1}), \ldots, (i_L, v_0, v_1, ev_L, \pi_{0,L})$, there is no permutation $\pi$ on $(1, 2, \ldots, L)$ such that $\sigma_{\pi(k)} = \perp$ or $\sigma_{\pi(k)} = \pi_{\sigma(k)}$ for $k = 1, 2, \ldots, L$.

Let $F_k$ (no unique decryption) be the event that two test queries $(ev, sv_1, \ldots, sv_{L'}, \pi_{d,1}, \ldots, \pi_{d,L'})$ and $(ev, sv_1', \ldots, sv_{L'}', \pi_{d,1}', \ldots, \pi_{d,L'}')$ for some valid $ev$ get results $\text{result}$ and $\text{result}'$ that are not equal up to order, and neither of which is equal to $\perp$. The advantage of the adversary is

$$\max\{2 \cdot |\Pr[E_g \land E_f] - \Pr[E_f]/2|, \Pr[F_i \lor F_c \lor F_d \lor F_k]\}$$

### H.4 Security Proof Sketch

We briefly sketch a proof for how to bound the advantage of an adversary against the cryptographic voting scheme in terms of adversaries against the shuffle, the distributed decryption scheme, the commitment scheme or the encryption scheme.

#### Confidentiality

We begin by analyzing the confidentiality event $\Pr[E_g \land E_f]$.

The proof would proceed as a sequence of games, where the first game is the interaction between the experiment and the adversary.

In the next game, we stop the adversary with a forced guess $b' = 0$ immediately upon any query that would make the execution non-fresh. Note that a query that makes the execution non-fresh can be recognized with no secret information, and at the time the query is made. A brief computation shows that this changes nothing, but in further analysis, we may assume that the execution remains fresh.

We next simulate all the zero-knowledge proofs involved, which is straight-forward in the random oracle model since all our proofs are HVZK.
Next, we change the challenge query so that instead of computing the precode as $\hat{r} = a + v$, it samples $\hat{r}$ uniformly at random. If this change is observable, we get an adversary against hiding for the commitment scheme.

Next, for any ballot decryption share query for a sequence $(ev_1, \ldots, ev_v)$, we decrypt $ev_i$ to $v_i$, then use the HVZK simulator from Section 6 to simulate the decryption share given the decryption $v_i$. This change is unobservable. (To get an adversary, we guess a decryption key share $i$ for which the adversary will never make a decrypt reveal query, and simulate the other decryption key shares as random shares. When the adversary makes a ballot decryption share query for $i$, we compute the ballot decryption shares for the other decryption key shares and compute the $i$th ballot decryption share to give the correct result when combined.)

Next, for chosen ciphertext queries, we decrypt the $w$ using $dk_i$ and subtract $a$ to recover $a$, and then record $(i, n, ev, \pi_v)$ instead of $(i, \perp, \perp, ev, \pi_v)$. The soundness of the ballot proof (details omitted), we now have that every tuple $(i, ev_i, \pi_v)$ in $L$ satisfies $Dec(dk_i, ev) = v_k$.

For any ballot decryption share query for an honest and balanced sequence $(ev_1, \ldots, ev_v)$ deriving from $(\perp, v_0, v_1, \perp, \ldots, \perp, v_{v_1}, v_{v_2}, \ldots)$, sample a permutation $\pi$ on \{1, 2, \ldots, $v$\} and use $v_i = v_{\pi(i)}$ instead of decrypting $ev_i$. If this change is observable, we either get an adversary against soundness for the shuffle (when the decryption of the output of a shuffle is not equal to the decryption of the input to the shuffle, up to order) or an adversary against the encryption scheme (when the adversary notices that the ballot decryption shares are inconsistent with the encrypted ballots). The latter adversary re-randomizes the shuffle with random values instead of encryptions of zero.

At this point, the decryption key shares $\{dk_i\}$ are no longer used. Also, the pre-code encrypted in the challenge query is independent of the challenge ballots.

Finally, we encrypt a random ballot instead of the left or right ballot for challenge queries. We get a real-or-random adversary against the encryption scheme if this change is observable.

At this point, the challenge bit $b$ is no longer used. It follows that the adversary has no advantage in this game. By the above arguments, the claim that the difference between $Pr[E_F \wedge E_f]$ and $Pr[|E_f|/2]$ is appropriately bounded follows.

**Integrity.** Next, we analyze the integrity events. In this case, the adversary may have revealed every secret key, and there is no need for the execution to be fresh.

If a chosen ciphertext query results in an incorrect pre-code, then we immediately get an adversary against the soundness of the ballot proof (details omitted). It follows that the probability of $F_i$ happening is appropriately bounded.

In the event that $F_i$ happens, note that every encrypted ballot either originates with a challenge query or a chosen ciphertext query, the shuffles applied to the encrypted ballots originate with shuffle queries or chosen shuffle queries, and the ballot decryption shares all originate with ballot decryption share queries. By the completeness and soundness of the various arguments, and the bound on the number of shuffles, we get the probability of $F_i$ happening is appropriately bounded.

In the event that $F_d$ happens, then either the output of some shuffle does not decrypt to the same as the input to the shuffle, in which case we get an adversary against the soundness of the shuffle or the distributed decryption does not decrypt correctly, in which case we get an adversary against the soundness of the distributed decryption. It follows that the probability of $F_d$ happening is appropriately bounded.

In the event that $F_d$ happens, then either the decryption of the encrypted ballots is not unique, in which case we get an adversary against the soundness of the proofs ensuring valid ciphertexts (in the ballot proofs and the shuffle proofs), or one or both results are incorrect, in which case we get an adversary against the soundness of the shuffle. It follows that the probability of $F_d$ happening is appropriately bounded.

The claim that $Pr[F_1 \lor F_2 \lor F_d \lor F_u]$ is appropriately bounded follows.

**H.5 Voting System Security Properties**

**H.5.1 Integrity.** Integrity for a voting system is modelled using a game between an adversary and a set of voters, some of which may be corrupt. The adversary tells the honest voters what ballots to cast. If the count phase eventually runs and ends with a result, the adversary wins if the result is inconsistent with the ballots accepted as cast by the honest voters. (Recall that only the voter’s last ballot cast is counted, so if the voter first accepts a ballot as cast and then tries to cast another ballot and this fails, the end result is that they have not accepted a ballot as cast.)

We can define a variant notion called $\epsilon$-integrity where we allow a small error, and say that the adversary wins if the result is inconsistent with any $(1 - \epsilon)$ fraction of the ballots accepted as cast by honest voters. (We need this since return codes for a single voter must be human-comparable, and can therefore collide with some non-negligible probability.)

**Analysis.** The voter will only accept the ballot as cast if the correct return code is received. If the correct return code is received, then the correct pre-code must have been computed at some point (except with some small probability due to collisions in the PRF). If the return code generator $R$ is honest, the integrity of the cryptographic voting scheme implies that this can only happen if the correct ballot has been encrypted. If the auditor $A$ is honest, the result will only be accepted if the encrypted ballot has been included among those sent to the first shuffler. By the integrity of the cryptographic voting system, all such ballots must then be included in the result.

If the voter’s computer $Comp$ and the ballot box $B$ and the auditor $A$ are honest, the encrypted ballot will be included among those sent to the first shuffler. By the integrity of the cryptographic voting system, all such ballots must then be included in the result.

If a voter receives a return code without casting a ballot, the voter will no longer accept their ballot as cast.

In summary, $\epsilon$-integrity holds if the auditor and either both the voters’ phones and the return code generator are honest, or both the voters’ computers and the ballot box are honest.

**H.5.2 Verifiability.** In a verifiable voting protocol, every voter gets a receipt after accepting a ballot as cast. Also, the auditor outputs
If every shuffle server is compromised, the adversary learns the correspondence between decrypted ballots and voters, and can trivially win the privacy game.

If every decryption server is compromised, the adversary learns the decryption key, and can trivially win the privacy game.

If a voter casts more than one ballot, a compromised return code generator or voter phone will always be able to decide if they are the same or not by observing the return code sent to the voter. If the ballots are distinct, the return code generator will learn information about which ballots were submitted, and typically learn both ballots. (We could prove privacy when the voter casts more than one ballot and the return code generator and the voter phone are both are honest, but this requires adding a restricted challenge query that does not reveal the precode of the cryptosystem experiment.)

Suppose the honest voters cast at most one ballot each, their computers remain honest, and at least one shuffle server and one decryption server are honest. Then privacy follows from the confidentiality of the cryptographic voting system since the protocol execution can be interpreted as an interaction with the cryptosystem experiment. The protocol and our assumptions ensure a fresh execution.

Note that cut-and-paste attacks against confidentiality, which commonly affect this type of voting protocol, do not work against this protocol because the ballot proof includes an encryption of the return code and proof that the return code is correct which is tied to the voter’s public key material. Cut-and-paste attacks would anyway constitute a valid attack on the cryptosystem.

In summary, privacy holds if the honest voters’ computers are honest, there is at least one honest shuffle server and one honest decryption server, and no honest voter casts more than one ballot.

Analysis. Verifiability for our protocol follows by integrity for the underlying cryptosystem, since the execution of our protocol can be thought of as interaction with the experiment for the underlying cryptosystem, where the honest computer’s actions correspond to challenge queries, and part of the verification algorithms’ work corresponds to a test query. The structure of the protocol then ensures that if the result output by a test query is inconsistent with corresponding ballots output by challenge queries, integrity fails for the underlying cryptosystem.

In summary, if the underlying cryptosystem has integrity, the voting protocol is verifiable.

H.5.3 Privacy. Privacy for a voting protocol is modelled as a left-or-right game with an adversary and a set of voters, some of which may be corrupt. The adversary gives pairs of ballots to honest voters, who will all cast the left or right ballot. The adversary must decide which they cast. (This essentially amounts to deciding who cast which ballot.)

The adversary can corrupt players and also control the network. We shall assume that players use secure channels to communicate. This means that only the fact that players are communicating and the length of their communications leak. Since message flows and message lengths are fixed and public knowledge, we can ignore the network in the subsequent analysis.

We want to avoid adversaries that deduce the honest voters’ ballots trivially from the result, we require that the adversary organizes the pairs of ballots given to honest voters in such a way that the ballots cast by honest voters are independent of whether the voters cast the left or the right ballot.

Analysis. If some honest voter’s computer Comp is compromised, the adversary can trivially win the privacy game.