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

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

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

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

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 subprotocols. 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 [20], 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 [40] 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 ample 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 [36, 42, 44]. Using shuffles and

[*] This is the full version of [5] published at ACM CCS 2023.
[†] Work done in part while visiting Aarhus University.

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 [19] 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 [9] 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 $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_\tau$ then works as follows: We let the shuffler commit to BGV re-randomization ciphertexts $\hat{\boldsymbol{c}}_1, \ldots, \hat{\boldsymbol{c}}_\tau$ (encryptions of 0) using the linearly homomorphic BDLOP commitment scheme $\mathtt{Com}$. 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 $\hat{\boldsymbol{c}}_1, \ldots, \hat{\boldsymbol{c}}_\tau$ as $\mathtt{Com}(\hat{\boldsymbol{c}}_i)$ and shows that they are well-formed using zero-knowledge proofs.
2. The shuffler computes $\boldsymbol{d}_i = \boldsymbol{c}_i + \hat{\boldsymbol{c}}_i$ and sends shuffled elements $L = (\boldsymbol{d}_{\pi(i)})_{i \in [\tau]}$ to the receiver.
3. Finally, the prover shows that L is a list of openings of the commitments obtained from $\boldsymbol{c}_i + \mathtt{Com}(\hat{\boldsymbol{c}}_i)$ 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 τ. For this, we use a version of recent amortized proofs of shortness [15]. 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 [24]. 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 370τ KB and each decryption proof is of total size 157τ KB, where τ is the number of total ciphertexts. It takes only 0.74 ms to encrypt a ballot, while the mixing proof takes 260.5τ ms and the decryption proof takes 138.1τ ms. Verification is much faster, with only 17.9τ ms for the mixing and 30.2τ 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. [22] design a shuffle with a straight-forward approach similar to Neff [40] 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. Farzaliyev et al. [28] implements the mix-net by Costa et al. [22] using the amortization techniques by Attema et al. [7] for the commitment scheme by Baum et al. [9]. 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 10.4 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. [32] gave a new proof of correct shuffle based on Beneš 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, [32] 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. [17] 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. [17] 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. [25] 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. [18] that the protocol in [25] is not end-to-end verifiable unless all tallying authorities and all voters' voting devices are honest. This problem is solved by [18], but their construction still has the downside of only supporting homomorphic tallying. Strand [47] built a verifiable shuffle for the GSW cryptosystem, but this construction is too restrictive for practical use. Chillotti et al. [21] 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 κ be the computational and sec the statistical security parameter. We define the ring $R_q = \mathbb{Z}_q[X]/\langle X^N + 1\rangle$, its norms, the discrete Gaussian distribution $\mathcal{N}$, rejection sampling, and knapsack problems $\mathsf{SKS}^2_{n,k,\beta}$ and $\mathsf{DKS}^\infty_{n,k,\beta}$ in Appendix A. We use $S_B \subseteq R_q$ 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 decryption 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*

KGen *On input security parameter* 1^κ *and number of key-shares* ξ_2*, outputs public parameters* pp*, a public key* pk*, a secret key* sk*, and key-shares* $\{\mathsf{sk}_j\}$*,*

Enc *On input* pk *and messages* $\{m_i\}$*, outputs ciphertexts* $\{c_i\}$*,*

Dec *On input* sk *and ciphertexts* $\{c_i\}$*, outputs messages* $\{m_i\}$*,*

DDec *On input a secret key share* sk_{j^*} *and ciphertexts* $\{c_i\}$*, outputs decryption shares* $\{\mathsf{ds}_{i,j^*}\}$*,*

Comb *On input ciphertexts* $\{c_i\}$ *and decryption shares* $\{\mathsf{ds}_{i,j}\}$*, outputs either messages* $\{m_i\}$ *or* $\perp$*,*

and pp *are implicit inputs to* Enc, Dec, DDec *and* 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 $P_{\mathsf{sk}}(c)$ 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* $P_{\mathsf{sk}}(\cdot)$ *if an adversary* Adv *corrupting* $J \subseteq [\xi_2]$ *secret key shares* $\{\mathsf{sk}_j\}_{j\in J}$ *cannot convince* Comb *to accept maliciously created decryption shares* $\{\mathsf{ds}_{i,j}\}_{i\in[\tau],j\in J}$*. More concretely, the following probability is bounded by a negligible* $\epsilon(\kappa)$*:*

$$\Pr\left[\begin{array}{c|c} \mathsf{Dec}(\mathsf{sk}, \{c_i\}_{i\in[\tau]}) & (\mathsf{pp}, \mathsf{pk}, \mathsf{sk}, \{\mathsf{sk}_j\}_{j\in[\xi_2]}) \leftarrow \mathsf{KGen}(1^\kappa, \xi_2) \\ \neq & (\{c_1, \ldots, c_\tau\}) \leftarrow \mathsf{Adv}(\mathsf{pp}, \mathsf{pk}, \{\mathsf{sk}_j\}_{j\in J}) \\ \mathsf{Comb}(\{c_i\}_{i\in[\tau]}, \{\mathsf{ds}_{i,j}\}_{i\in[\tau]}^{j\in[\xi_2]}) : & \forall i \in [\tau] : P_{\mathsf{sk}}(c_i) = 1, \forall j \notin J : \\ \neq & \{\mathsf{ds}_{i,j}\}_{i\in[\tau]} \leftarrow \mathsf{DDec}(\mathsf{sk}_j, \{c_i\}_{i\in[\tau]}) \\ \perp & \{\mathsf{ds}_{i,j}\}_{i\in[\tau],j\in J} \leftarrow \mathsf{Adv}(\{\mathsf{ds}_{i,j}\}_{i\in[\tau],j\notin J}) \end{array}\right],$$

where the probability is taken over KGen *and* DDec.

We moreover define a simulation property, that shows that decryption 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* $P_{\mathsf{sk}}(\cdot)$ *if an adversary* Adv *corrupting* $J \subsetneq [\xi_2]$ *secret key shares* $\{\mathsf{sk}_j\}_{j\in J}$ *cannot distinguish the transcript of the decryption protocol from a simulation by a simulator* Sim *which only gets* $\{\mathsf{sk}_j\}_{j\in J}$ *as well as correct decryptions as input. More concretely, the following probability is bounded by a negligible* $\epsilon(\mathrm{sec})$*:*

$$|\Pr\left[b = b' \;:\; \begin{array}{c} (\mathsf{pp}, \mathsf{pk}, \mathsf{sk}, \{\mathsf{sk}\}_{j\in[\xi_2]}) \leftarrow \mathsf{KGen}(1^{\kappa}, \xi_2) \\ (\{c_1, \ldots, c_\tau\}) \leftarrow \mathsf{Adv}(\mathsf{pp}, \mathsf{pk}, \{\mathsf{sk}_j\}_{j\in J}) \\ \forall i \in [\tau] : P_{\mathsf{sk}}(c_i) = 1 \\ \{\mathsf{ds}^0_{i,j}\} \leftarrow \mathsf{DDec}(\{\mathsf{sk}_j\}_{j\in[\xi_2]}, \{c_i\}_{i\in[\tau]}) \\ \{\mathsf{ds}^1_{i,j}\} \leftarrow \mathsf{Sim}(\mathsf{pp}, \{\mathsf{sk}_j\}_{j\in J}, \{c_i, \mathsf{Dec}(\mathsf{sk}, c_i)\}_{i\in[\tau]}) \\ b \xleftarrow{\$} \{0,1\}, b' \leftarrow \mathsf{Adv}(\{\mathsf{ds}^b_{i,j}\}_{i\in[\tau], j\in[\xi_2]}) \end{array}\right] - \frac{1}{2}|,$$

where the probability is taken over KGen, DDec, Sim.

2.1.1 Our instantiation. Let $p \ll q$ be primes, define R_q and R_p for a fixed N, let $B_{\mathsf{Key}}, B_{\mathsf{Err}} \in \mathbb{N}$ be bounds. We use the BGV [19] encryption scheme, which consists of three algorithms: key generation (KGen), encryption (Enc) and decryption (Dec), where:

- KGen Samples a uniform element $a \xleftarrow{\$} R_q$, a short $s \xleftarrow{\$} S_{B_{\mathsf{Key}}}$ and noise $e \xleftarrow{\$} S_{B_{\mathsf{Err}}}$. The algorithm outputs the public key $\mathsf{pk} = (a, b) = (a, as + pe)$ and secret key $\mathsf{sk} = s$.
- Enc On input the public key $\mathsf{pk} = (a, b)$ and a message $m \in R_p$, samples a uniform $r \xleftarrow{\$} S_{B_{\mathsf{Key}}}$, noise $e', e'' \xleftarrow{\$} S_{B_{\mathsf{Err}}}$ and outputs the ciphertext $c = (u, v) = (ar + pe', br + pe'' + m)$.
- Dec On input secret key $\mathsf{sk} = s$ and ciphertext $c = (u, v)$, outputs message $m = (v - su \mod q) \mod p$.

The following theorem follows from [19] and [39].

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

We use this theorem to define the predicate $P_{\mathsf{sk}}(u, v)$ to be 1 iff $\|v - su\|_\infty < B_{\mathsf{Dec}}$ and otherwise 0. Since each ciphertext consists of 2 elements from R_q, it can be represented using $2N \log_2 q$ bits.

2.1.2 Threshold decryption. We quickly recap the passively secure distributed decryption protocol by Damgård *et al.* [11, 23, 24]. Here, the KGen algorithm on input $\xi_2 \in \mathbb{N}$ additionally outputs uniformly random shares $\mathsf{sk}_j = s_j$ of the secret key $\mathsf{sk} = s$ such that $s = s_1 + \cdots + s_{\xi_2}$ in R_q. This defines a passively secure threshold decryption protocol by using the linearity of the decryption function:

- DDec On input a secret key-share $\mathsf{sk}_j = s_j$ and a ciphertext $c = (u, v)$, does the following:
  1. Compute $m_j = s_j u$ and sample a uniformly random $E_j \xleftarrow{\$} R_q$ such that $\|E_j\|_\infty \leq 2^{\mathrm{sec}}(B_{\mathsf{Dec}}/p\xi_2)$ for statistical security parameter sec and noise-bound B_{Dec},
  2. Output $\mathsf{ds}_j = t_j = m_j + pE_j$.
- Comb On input ciphertext $c = (u, v)$ and set of decryption shares $\{\mathsf{ds}_j = t_j\}_{j\in[\xi_2]}$, outputs message $m = (v - t \mod q) \mod p$, where $t = t_1 + \cdots + t_{\xi_2}$.

The following theorem follows from [23, 24].

Theorem 2. *Let* sec *be the statistical security parameter. The distributed BGV encryption scheme is* correct *for input ciphertexts with* $\|v - us\|_\infty \leq (1 + 2^{\mathrm{sec}})B_{\mathsf{Dec}} < \lfloor q/2 \rfloor$*, 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 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 [13], 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* 1^{κ}*, outputs public params* pp,
- Com *On input message* m*, outputs commitment* c *and opening* r,
- Open *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 [9] commitment scheme. Let R_q be defined as above and let $\mathcal{N}_{\sigma_{\mathsf{C}}}$ be a Gaussian distribution with standard deviation σ_{C} and B_{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 $R_q^{l_c}$ using length-k randomness from $S^k_{B_{\mathsf{Com}}}$ outputting length-$(n + l_c)$ vectors. For this, we define

$$\boldsymbol{A}_{\mathsf{C},1} = \begin{bmatrix} \boldsymbol{I}_n & \widehat{\boldsymbol{A}}_{\mathsf{C},1} \end{bmatrix} \qquad \text{where } \widehat{\boldsymbol{A}}_{\mathsf{C},1} \xleftarrow{\$} R_q^{n\times(k-n)}$$

$$\boldsymbol{A}_{\mathsf{C},2} = \begin{bmatrix} \boldsymbol{0}^{l_c\times n} & \boldsymbol{I}_{l_c} & \widehat{\boldsymbol{A}}_{\mathsf{C},2} \end{bmatrix} \qquad \text{where } \widehat{\boldsymbol{A}}_{\mathsf{C},2} \xleftarrow{\$} R_q^{l\times(k-n-l_c)}.$$

  Let $\mathsf{pk} = \boldsymbol{A}_{\mathsf{C}} = \begin{bmatrix} \boldsymbol{A}_{\mathsf{C},1} \\ \boldsymbol{A}_{\mathsf{C},2} \end{bmatrix}$. $\boldsymbol{A}_{\mathsf{C}}$ has height $n + l_c$ and width k.
- Com On input $\boldsymbol{m} \in R_q^{l_c}$ samples $\boldsymbol{r_m} \xleftarrow{\$} S^k_{B_{\mathsf{Com}}}$ and computes

$$\mathsf{Com}_{\mathsf{pk}}(\boldsymbol{m}; \boldsymbol{r_m}) = \boldsymbol{A}_{\mathsf{C}} \cdot \boldsymbol{r_m} + \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{m} \end{bmatrix} = \begin{bmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \end{bmatrix} = [\![\boldsymbol{m}]\!].$$

  Com outputs $[\![\boldsymbol{m}]\!]$ and the opening $\boldsymbol{d} = (\boldsymbol{m}, \boldsymbol{r_m}, 1)$.
- Open Verifies whether an opening $(\boldsymbol{m}, \boldsymbol{r_m}, f)$, with $f \in \bar{\mathcal{C}}$, is a valid opening of $[\![\boldsymbol{m}]\!]$ by checking that $\|\boldsymbol{r_m}[i]\| \leq 4\sigma_{\mathsf{C}}\sqrt{N}$, for $i \in [k]$, and if

$$f \cdot \begin{bmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \end{bmatrix} \stackrel{?}{=} \boldsymbol{A}_{\mathsf{C}} \cdot \boldsymbol{r_m} + f \cdot \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{m} \end{bmatrix}.$$

  It outputs 1 if all conditions hold, and 0 otherwise.

We define the set $\bar{C}$ 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. [9].

Theorem 3. *The aforementioned commitment scheme is* computationally hiding *if the* $\mathsf{DKS}^{\infty}_{n+l_c,k,B_{\mathrm{Com}}}$ *problem is hard, and the scheme is* computationally binding *if the* $\mathsf{SKS}^{2}_{n,k,16\sigma_{\mathrm{C}}\sqrt{vN}}$ *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. [31]. 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 not revealing any secret randomness that they used in the process.

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 $\mathcal{P}$, $\mathcal{P}^*$, $\mathcal{V}$ and $\mathcal{V}^*$ be polynomial time algorithms.

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

We call an Interactive Proof a Zero-Knowledge proof[1] if it has the following three properties:

- **Completeness:** If $\mathcal{P}$ has a valid witness w such that $(x, w) \in$ R, then $\mathcal{V}$ accepts.
- **Knowledge Soundness:** If $\mathcal{P}^*$ 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 $\mathcal{P}^*$, 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 [29].

2.3.1 Linear relations among commitments. Assume that there are $\hat{n}$ BDLOP commitments

$$[\![\boldsymbol{m}_i]\!] = \begin{bmatrix} \boldsymbol{c}_{i,1} \\ \boldsymbol{c}_{i,2} \end{bmatrix}, \text{ for } 1 \le i \le \hat{n} \text{ where } \boldsymbol{c}_{i,2} \in R_q^{l_c}.$$

For the public scalar vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{\hat{n}-1}) \in R_q^{\hat{n}-1}$ the prover wants to prove that the following relation holds:

$$\mathcal{R}_{\mathrm{LIN}} = \left\{ (x, w) \;\middle|\; \begin{array}{c} x = (\mathrm{pk}, \{[\![\boldsymbol{m}_i]\!]\}_{i \in [\hat{n}]}, \boldsymbol{\alpha}) \land \\ w = (f, \{\boldsymbol{m}_i, \boldsymbol{r}_i\}_{i \in [\hat{n}]}) \land \\ \forall i \in [\hat{n}] : \mathrm{Open}_{\mathrm{pk}}([\![\boldsymbol{m}_i]\!], \boldsymbol{m}_i, \boldsymbol{r}_i, f) = 1 \\ \land\, \boldsymbol{m}_{\hat{n}} = \sum_{i=1}^{\hat{n}-1} \alpha_i \boldsymbol{m}_i \end{array} \right\}.$$

We will require proof of this relation at multiple points in our constructions. In Appendix B.1 we provide a ZK proof Π_{LIN} for this relation, which is a directly extended version of the linearity proof in [9]. It works like a standard Σ protocol when adapted to lattices.

The relation $\mathcal{R}_{\mathrm{LIN}}$ 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 $B = 2\sigma_{\mathrm{C}}\sqrt{N}$ and the protocol produces a proof transcript $\pi_{\mathrm{LIN}} = ((\{\boldsymbol{t}_i\}_{i \in [\hat{n}]}, u), \beta, (\{\boldsymbol{z}_i\}_{i \in [\hat{n}]}))$. Using the standard Fiat-Shamir transform, we make Π_{LIN} non-interactive.

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 nega-cyclic matrix in $\mathbb{Z}_q^{N \times N}$. Let $\boldsymbol{A}$ be a publicly known $r \times v$ matrix over R_q, that is, a $rN \times vN$ matrix over $\mathbb{Z}_q$. We will now consider how to prove generically in zero-knowledge that $\boldsymbol{t}_i = \boldsymbol{A}\boldsymbol{s}_i$ for bounded $\boldsymbol{s}_i$ and known $\boldsymbol{t}_i$ 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 τ 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 $\boldsymbol{A}$ be a publicly known $r \times v$-matrix over R_q, let $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_\tau$ be bounded elements in R_q^v and let $\boldsymbol{A}\boldsymbol{s}_i = \boldsymbol{t}_i$ for $i \in [\tau]$. Letting $\boldsymbol{S}$ be the matrix whose columns are $\boldsymbol{s}_i$ and $\boldsymbol{T}$ be the same matrix for $\boldsymbol{t}_i$, but defined over $\mathbb{Z}_q^N$ instead of R_q, then [8] give an efficient amortized zero-knowledge proof of knowledge for the relation

$$\mathcal{R}_{\mathrm{BND}} = \left\{ (x, w) \;\middle|\; \begin{array}{c} x = (\boldsymbol{A}, \boldsymbol{T}) \land w = \boldsymbol{S} \quad \land \forall i \in [\tau] : \\ \boldsymbol{t}_i = \boldsymbol{A}\boldsymbol{s}_i \land ||s_{i,j}||_2 \le 2 \cdot B_{\mathrm{BND}} \end{array} \right\}.$$

Let

$$\pi_{\mathrm{BND}} \leftarrow \Pi_{\mathrm{BND}}(\boldsymbol{S}; (\boldsymbol{A}, \boldsymbol{T}, \sigma_{\mathrm{BND}})), 0 \lor 1 \leftarrow \Pi_{\mathrm{BNDV}}((\boldsymbol{A}, \boldsymbol{T}, B_{\mathrm{BND}}); \pi_{\mathrm{BND}}),$$

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

2.3.3 Exact Amortized Proofs of Shortness. As can be seen from $\mathcal{R}_{\mathrm{BND}}$ the non-exact amortized proof has the disadvantage of introducing a "slack" factor $B_{\mathrm{BND}} = \sqrt{2N}\sigma_{\mathrm{BND}}$, 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 Π_{BND}, as one always has to assume that dishonestly provided encryptions or commitments only fulfills the larger bound.

[1] More concretely, an Honest-Verifier Zero Knowledge Proof of Knowledge

We will therefore also use a tighter ZK amortized proof of shortness which shows $\mathcal{R}_{\text{BND}}$ for the ℓ_∞-norm and with B_{BND} being 1. The disadvantage of this proof, over Π_{BND}, is that it does not scale as well with the number of statements that are proven as Π_{BND}.

For our exact amortized proof, we use a version of the protocol from Bootle et al. [15]. They give an efficient amortized sublinear zero-knowledge protocol for proving the knowledge of short vectors $\boldsymbol{s}_i$ and $\boldsymbol{e}_i$ over $\mathbb{Z}_q$ satisfying $\boldsymbol{A}\boldsymbol{s}_i + \boldsymbol{e}_i = \boldsymbol{t}_i$. We adapt their techniques for the case where $\boldsymbol{e}_i$ is zero and always prove that $\|\boldsymbol{s}_i\|_\infty \leq 1$. 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 $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_\tau$ as follows:

(1) The server creates encryptions $\boldsymbol{c}'_1, \ldots, \boldsymbol{c}'_\tau$ of 0 and commits to each $\boldsymbol{c}'_i$ as $\text{Com}(\boldsymbol{c}'_i)$. Then, by homomorphically adding $\boldsymbol{c}_i$ to $\text{Com}(\boldsymbol{c}'_i)$ we obtain commitments $\text{Com}(\hat{\boldsymbol{c}}_i)$ to the same plaintexts as in $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_\tau$, with "fresh" randomness.
(2) The shuffle server reveals the openings $\hat{\boldsymbol{c}}_i$, 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 $\text{Com}(\hat{\boldsymbol{c}}_i)$ 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_q, while BGV ciphertexts consist of two elements from R_q. We, therefore, extend the shuffle protocol by Aranha et al. to verifiably shuffle vectors in $R_q^{l_c}$. 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 [43].

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 [46], 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.

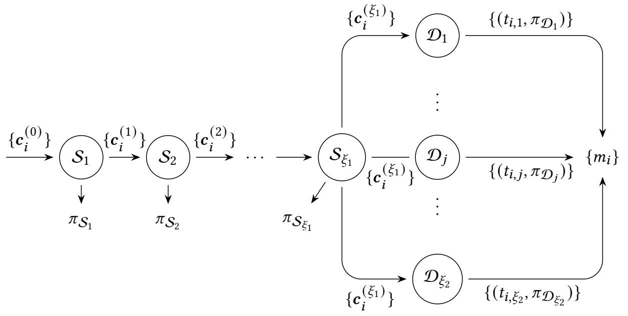
$\{\boldsymbol{c}_i^{(\xi_1)}\}$
$\mathcal{D}_1$
$\{(t_{i,1}, \pi_{\mathcal{D}_1})\}$
$\{\boldsymbol{c}_i^{(0)}\}$
$\mathcal{S}_1$
$\{\boldsymbol{c}_i^{(1)}\}$
$\mathcal{S}_2$
$\{\boldsymbol{c}_i^{(2)}\}$
$\cdots$
$\mathcal{S}_{\xi_1}$
$\{\boldsymbol{c}_i^{(\xi_1)}\}$
$\mathcal{D}_j$
$\{(t_{i,j}, \pi_{\mathcal{D}_j})\}$
$\{m_i\}$
$\pi_{\mathcal{S}_1}$
$\pi_{\mathcal{S}_2}$
$\pi_{\mathcal{S}_{\xi_1}}$
$\{\boldsymbol{c}_i^{(\xi_1)}\}$
$\mathcal{D}_{\xi_2}$
$\{(t_{i,\xi_2}, \pi_{\mathcal{D}_{\xi_2}})\}$

Figure 1: The high-level counting phase of our voting protocol. Each shuffle server $\mathcal{S}_k$ receives a set of ciphertexts $\{\boldsymbol{c}_i^{(k-1)}\}$, shuffles them, and outputs a new set of ciphertexts $\{\boldsymbol{c}_i^{(k)}\}$ and a proof $\pi_{\mathcal{S}_k}$. When all shuffle proofs are verified, each decryption server $\mathcal{D}_j$ partially decrypts every ciphertext and outputs the partial decryptions $\{t_{i,j}\}$ and a proof of correctness $\pi_{\mathcal{D}_j}$. Votes $\{m_i\}$ are reconstructed from the partial decryptions.

The voting protocol requires a *trusted set of players* to run the setup, a set of *voters* Voter_i and their *computers* Comp_i, a *ballot box* Ballot, a collection of *shuffle servers* $\mathcal{S}_k$, a collection of *decryption servers* $\mathcal{D}_j$ and one or more *auditors* Audit. We will assume that there are ξ_1 shuffle servers and ξ_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. [43]. The derived public parameters pk are given to every participant, while the decryption key shares sk_j are given to the decryption servers $\mathcal{D}_j$.

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

Casting phase. Each voter Voter_i instructs its computer 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 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 σ_i is sent to the voter's computer. The computer verifies the signature σ_i 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 Voter_i accepts the ballot as cast if and only if the computer accepts it with a receipt.

Counting phase. The ballot box Ballot sends the encrypted ballots and ballot proofs that it has seen to the auditor Audit as well as every decryption server $\mathcal{D}_j$. Ballot then sorts the list of encrypted ballots $\{\boldsymbol{c}_i^{(0)}\}$ and sends this to the first shuffle server $\mathcal{S}_1$ 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 ξ_1 shuffle servers $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{\xi_1}$ consecutively use the shuffle algorithm on the input encrypted ballots $\{\boldsymbol{c}_i^{(k-1)}\}$, passing the shuffled and re-encrypted ballots $\{\boldsymbol{c}_i^{(k)}\}$ to the next shuffle server. They also pass the shuffled re-encrypted ballots $\{\boldsymbol{c}_i^{(k)}\}$ and the shuffle proof $\pi_{\mathcal{S}_k}$ to 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 $\pi_{\mathcal{S}_k}$ verifies for the respective ciphertexts. Only then will they run the distributed decryption algorithm DDec with their decryption key share sk_j and send their partial decryption shares $t_{i,j}$ of each ballot $\boldsymbol{c}_i^{(\xi_1)}$ to the auditor as well as each recipient of the output. To obtain the result, each recipient can then run $\mathtt{Comb}$ 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 are consistent with the data from Ballot, that every shuffle proof verifies, and then runs the combining algorithm $\mathtt{Comb}$ on the received partial decryption shares $t_{i,j}$ from each $\mathcal{D}_j$. If all checks pass then the auditor accepts, otherwise, it rejects. Finally, 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 σ_i using the ballot box' 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 Comp_i 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

Bootle et al. [15] give an efficient amortized sublinear zero-knowledge protocol for proving the knowledge of short vectors $\boldsymbol{s}_i$ and $\boldsymbol{e}_i$ over $\mathbb{Z}_q$ satisfying $\boldsymbol{A}\boldsymbol{s}_i+\boldsymbol{e}_i = \boldsymbol{t}_i$. For our setting, we adapt their techniques for the case where $\boldsymbol{e}_i$ is zero, and prove that $\|\boldsymbol{s}_i\|_\infty \leq 1$[2].

We explain the main idea of [15] for proving knowledge of a preimage $\boldsymbol{s}$ of $\boldsymbol{t} = \boldsymbol{A}\boldsymbol{s}$ and then generalize to an amortized proof for τ elements with sublinear communication.

The approach follows an ideal linear commitments-technique with vector commitments $\mathsf{Com}_{\mathsf{L}}(\cdot)$ over $\mathbb{Z}_q$. The prover initially commits to the vector $\boldsymbol{s}$ as well as an auxiliary vector $\boldsymbol{s}_0$ of equal length. Implicitly, this defines a vector of polynomials $\boldsymbol{f}(X) = \boldsymbol{s}_0(X) + \boldsymbol{s}$ for the prover. Now consider the vector of polynomials $\boldsymbol{f}(X) \circ (\boldsymbol{f}(X) - \boldsymbol{1}) \circ (\boldsymbol{f}(X) + \boldsymbol{1})$, where $\circ$ denote the coordinate-wise product, then the coefficients of X^0 are exactly $\boldsymbol{s} \circ (\boldsymbol{s} - \boldsymbol{1}) \circ (\boldsymbol{s} + \boldsymbol{1})$ and therefore $\boldsymbol{0}$ if and only if the aforementioned bound on $\boldsymbol{s}$ holds. In that case, each aforementioned polynomial in $\boldsymbol{f}(X) \circ (\boldsymbol{f}(X) - \boldsymbol{1}) \circ (\boldsymbol{f}(X) + \boldsymbol{1})$ is divisible by X. Therefore, the prover computes the coefficient vectors

$$1/X \cdot \boldsymbol{f}(X) \circ (\boldsymbol{f}(X) - \boldsymbol{1}) \circ (\boldsymbol{f}(X) + \boldsymbol{1}) = \boldsymbol{v}_2 X^2 + \boldsymbol{v}_1 X + \boldsymbol{v}_0$$

and commits to these. Additionally, define the value $\boldsymbol{d} = \boldsymbol{t} - \boldsymbol{A}\boldsymbol{f} = -\boldsymbol{A}\boldsymbol{s}_0$, which the prover also commits to.

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

(1) $\mathsf{Com}_{\mathsf{L}}(\boldsymbol{s}_0) \cdot x + \mathsf{Com}_{\mathsf{L}}(\boldsymbol{s})$ opens to $\overline{\boldsymbol{f}}$.
(2) $\mathsf{Com}_{\mathsf{L}}(\boldsymbol{v}_2) \cdot x^2 + \mathsf{Com}_{\mathsf{L}}(\boldsymbol{v}_1) \cdot x + \mathsf{Com}_{\mathsf{L}}(\boldsymbol{v}_0)$ opens to the value $\frac{1}{x} \cdot \overline{\boldsymbol{f}} \circ (\overline{\boldsymbol{f}} + 1) \circ (\overline{\boldsymbol{f}} - 1)$.

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

The ideal linear commitments in [15] get realized using an Encode-then-Hash scheme. In this commitment scheme, the prover commits to vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{Z}_q^{l_{\mathsf{msg}}}$:

(1) Sample n random vectors $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n \in \mathbb{Z}_q^{\eta}$
(2) Let Encode be the encoding function of an $[l, l_{\mathsf{msg}} + \eta, d]$ Reed-Solomon Code with code-length l, message length $l_{\mathsf{msg}}+\eta$ and minimal distance d. Compute $\boldsymbol{e}_i \leftarrow \mathsf{Encode}(\boldsymbol{x}_i \| \boldsymbol{r}_i)$ for each $i \in [n]$.
(3) Construct matrix $\boldsymbol{E} = \mathsf{RowsToMatrix}(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n)$ where $\boldsymbol{e}_i$ is row i.
(4) Commit to each *column* of $\boldsymbol{E}$ using a hash, then compress all commitments to Merkle root M.
(5) Send M to the verifier.

[2] The authors of [15] mention that this optimization is possible, but neither present the modified protocol nor a proof.

For the prover to show to the verifier that $\boldsymbol{x}$ is an opening of the linear combination $\sum_{i=1}^{n} \gamma_i \boldsymbol{x}_i$:

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

This is a proof of the respective statement due to the random choice of the set I. Intuitively, if each row of $\boldsymbol{E}$ is in the code[3], but they do not sum up to $\boldsymbol{x}$, then the linear combination of the codewords in $\boldsymbol{E}$ must differ from $\mathsf{Encode}(\boldsymbol{x} \| \boldsymbol{r})$ in at least d positions, which is the minimum distance of the code. By the random choice of I and by setting η appropriately, the verifier would notice such a disagreeing entry with high probability. At the same time, because only η columns of $\boldsymbol{E}$ are opened, this leaks no information about the vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ if the evaluation points of the output of 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 $\boldsymbol{t}_i = \boldsymbol{A}\boldsymbol{s}_i$ for τ values $\boldsymbol{t}_i$ known to the verifier, subject to $\boldsymbol{s}_i$ again being ternary vectors. The goal is to establish the latter for all $\boldsymbol{t}_i$ simultaneously while verifying only one equation and sending only one vector $\overline{\boldsymbol{f}}$. Then the prover commits to $\boldsymbol{s}_i$ as well as an additional blinding value $\boldsymbol{s}_0$. Let $a_1, \ldots, a_\tau \in \mathbb{Z}_q$ be distinct interpolation points and define the ith Lagrange polynomial

$$\ell_i(X) = \prod_{i \neq j} \frac{X - a_j}{a_i - a_j}.$$

Additionally, let $\ell_0(X) = \prod_{i=1}^{\tau}(X - a_i)$. Then every $f \in \mathbb{Z}_q[X]/\ell_0(X)$ can be written uniquely as $f(X) = \sum_{i=1}^{\tau} \lambda_i \ell_i(X)$ and any $g \in \mathbb{Z}_q[X]/\ell_0(X)^b$ as a linear combination of $\{\ell_i(X)\ell_0(X)^j\}_{j=0}^{b-1}$. Define the polynomial

$$\boldsymbol{f}(X) = \sum_{i=0}^{\tau} \boldsymbol{s}_i \ell_i(X),$$

and observe that $\boldsymbol{f}(X) \circ (\boldsymbol{f}(X) - \boldsymbol{1}) \circ (\boldsymbol{f}(X) + \boldsymbol{1})$ is divisible by $\ell_0(X)$ if and only if all $\ell_i(X)$-coefficients of $\boldsymbol{f}(X)$ for $i \in [\tau]$ are 0. Additionally, since $\ell_i(X) \cdot \ell_j(X) = 0 \bmod \ell_0(X)$ if $i, j \in [n], i \neq j$ this then also implies that the $\boldsymbol{s}_i$ are ternary. Moreover, we only have to commit to additional $3 \cdot \tau$ coefficients of $\{\ell_i(X)\ell_0(X)^j\}_{j=0}^{b-1}$ to prove well-formedness of any evaluation of $\boldsymbol{f}(X)$ sent by the prover.

The protocol is described in detail in Figure 5 in Appendix C. As our construction substantially deviates from that of [15] 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 $\boldsymbol{s}_i$ has ternary coefficients, it is* special sound *if the* $\mathsf{SKS}^2_{r,v,1}$ *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 openings of the homomorphic commitments (step 9 in Figure 5) and the opening of the column-wise commitments of $\boldsymbol{E}$ via Merkle tree paths (step 11) More concretely:

- In step 9), prover sends polynomials which are openings to the homomorphic commitments of total size $3vN \log_2 q$ and additional randomness of total size $3\eta \log_2 q$.
- In step 11), the preimages of the hash column commitments $(\boldsymbol{E}|_I)$ have length $(3\tau + 2)\eta \log_2 q$ while the Merkle tree paths add another $2\kappa\eta(1 + \log_2 l)$ bits.

This leads to a proof of size

$$(3vN + (3\tau + 2)\eta) \log_2 q + 2\kappa\eta(1 + \log_2 l) \text{ bits} \tag{1}$$

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

5 VERIFIABLE SHUFFLE OF CIPHERTEXTS

The recent work by Aranha et al. [4] presents an efficient protocol Π_{SHUF} 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 $R_q^{l_c}$.

5.1 The Extended Shuffle for Commitments

To prove a shuffle, both the prover and verifier are given a list of commitments $[\![\boldsymbol{m}_1]\!], \ldots, [\![\boldsymbol{m}_\tau]\!]$ as well as potential messages $(\hat{\boldsymbol{m}}_1, \ldots, \hat{\boldsymbol{m}}_\tau)$ from $R_q^{l_c}$. The prover additionally obtains openings $\boldsymbol{m}_i, \boldsymbol{r}_i, f_i$ 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 π of the indices in the lists. More formally, our goal is to prove the following relation

$$\mathcal{R}_{\text{SHUF}^{l_c}} = \left\{ (x, w) \;\middle|\; \begin{array}{l} x = ([\![\boldsymbol{m}_1]\!], \ldots, [\![\boldsymbol{m}_\tau]\!], \hat{\boldsymbol{m}}_1, \ldots, \hat{\boldsymbol{m}}_\tau), \\ w = (\pi, f_1, \ldots, f_\tau, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_\tau), \pi \in S_\tau, \\ \forall i \in [\tau] :\; f_i \cdot [\![\boldsymbol{m}_{\pi^{-1}(i)}]\!] = f_i \cdot \begin{bmatrix} \boldsymbol{c}_{1,\pi^{-1}(i)} \\ \boldsymbol{c}_{2,\pi^{-1}(i)} \end{bmatrix} \\ = \boldsymbol{A}_{\text{C}} \boldsymbol{r}_i + f_i \cdot \begin{bmatrix} \boldsymbol{0} \\ \hat{\boldsymbol{m}}_i \end{bmatrix} \wedge \;\; \|\boldsymbol{r}_i[j]\| \leq 4\sigma_C \sqrt{N} \end{array} \right\}.$$

Towards proving this relation, we observe that it is sufficient to let the verifier choose a random element $h \xleftarrow{\$} R_q$. Then instead of proving a shuffle on $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_\tau$, the prover instead performs the same proof on $\langle \boldsymbol{m}_1, \rho \rangle, \ldots, \langle \boldsymbol{m}_\tau, \rho \rangle$ where $\rho = (1, h, \ldots, h^{l_c - 1})^\top$. The problem with this approach is that we must also be able to apply ρ to the commitments $[\![\boldsymbol{m}_1]\!], \ldots, [\![\boldsymbol{m}_\tau]\!]$, without re-committing to the inner product and proving correctness in zero-knowledge.

Since each commitment $[\![\boldsymbol{m}]\!]$ can be written as

$$\begin{bmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \end{bmatrix} = \boldsymbol{A}_{\text{C}} \boldsymbol{r} + \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{m} \end{bmatrix}$$

we can write $\boldsymbol{c}_1 = \boldsymbol{A}_{\text{C},1} \boldsymbol{r}$ and $\boldsymbol{c}_2 = \boldsymbol{A}_{\text{C},2} \boldsymbol{r} + \boldsymbol{m}$. From this we can create a new commitment $[\![\langle \boldsymbol{\rho}, \boldsymbol{m} \rangle]\!]$ under the new commitment key $\mathsf{pk}' = (\boldsymbol{A}_{\text{C},1}, \rho \boldsymbol{A}_{\text{C},2})$ where $\boldsymbol{c}_1' = \boldsymbol{c}_1$ remains the same, while we set $c_2' = \langle \rho, \boldsymbol{c}_2 \rangle$. This does not increase the bound of the randomness

[3] For the proof to work, the verifier additionally has to verify this claim or rather, that all rows are close to actual codewords. One mechanism to achieve this is to commit to an additional auxiliary row and also open a random linear combination of all rows, including the auxiliary row.

of the commitment. Since

$$\boldsymbol{A}_{\mathrm{C},2} = \begin{bmatrix} \boldsymbol{0}^{l_c \times n} & \boldsymbol{I}_{l_c} & \widehat{\boldsymbol{A}}_2 \end{bmatrix} \text{ where } \widehat{\boldsymbol{A}}_2 \in R_q^{l \times (k-n-l_c)},$$

it holds that

$$\boldsymbol{a}_2' = \boldsymbol{\rho} \boldsymbol{A}_{\mathrm{C},2} = \begin{bmatrix} \boldsymbol{0}^n & \boldsymbol{\rho}^\top & \boldsymbol{\rho} \widehat{\boldsymbol{A}}_2 \end{bmatrix}.$$

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* ϵ*, then there exists an efficient algorithm* Adv′ *that breaks the binding property on* pk *with the same probability.*

We can now construct the protocol $\Pi_{\mathrm{SHUF}}^{l_c}$:

(1) Initially, prover $\mathcal{P}$ and verifier $\mathcal{V}$ hold $\{\llbracket \boldsymbol{m}_i \rrbracket, \hat{\boldsymbol{m}}_i\}_{i \in [\tau]}$ for a public key $\mathsf{pk} = (\boldsymbol{A}_{\mathrm{C},1}, \boldsymbol{A}_{\mathrm{C},2})$ while the prover additionally hold secrets $\{\boldsymbol{m}_i, \boldsymbol{r}_i\}_{i \in [\tau]}, \pi \in S_\tau$.

(2) $\mathcal{V}$ chooses $h \xleftarrow{\$} R_q$ and sends it to $\mathcal{P}$. Both parties compute $\boldsymbol{\rho} \leftarrow (1, h, \ldots, h^{l_c - 1})^\top$.

(3) $\mathcal{P}$ and $\mathcal{V}$ for each $\llbracket \boldsymbol{m}_i \rrbracket = (\boldsymbol{c}_{1,i}, \boldsymbol{c}_{2,i})$ compute $\llbracket \langle \boldsymbol{\rho}, \boldsymbol{m}_i \rangle \rrbracket = (\boldsymbol{c}_{1,i}, \langle \boldsymbol{\rho}, \boldsymbol{c}_{2,i} \rangle) = (\boldsymbol{c}_{1,i}, \boldsymbol{c}_{2,i}')$.

(4) $\mathcal{P}$ and $\mathcal{V}$ run Π_{SHUF} on input commitments $\{\llbracket \langle \boldsymbol{\rho}, \boldsymbol{m}_i \rangle \rrbracket\}_{i \in [\tau]}$ and messages $\langle \boldsymbol{\rho}, \hat{\boldsymbol{m}}_i \rangle$. $\mathcal{P}$ uses same permutation π, randomness $\boldsymbol{r}_i$ as before. The commitment key $\mathsf{pk}' = (\boldsymbol{A}_{\mathrm{C},1}', \boldsymbol{A}_{\mathrm{C},2}')$ is used by both.

(5) If Π_{SHUF} accepts then $\mathcal{V}$ accepts in $\Pi_{\mathrm{SHUF}}^{l_c}$, otherwise rejects.

We show the following in Appendix E:

Lemma 1. *Assuming that the commitment scheme is binding and that* $\tau \leq \frac{1}{2 - 2\exp(-N/q)}$. *Then the protocol* $\Pi_{SHUF}^{l_c}$ *is a HVZK PoK for the relation* $\mathcal{R}_{SHUF^{l_c}}$ *with soundness error* $\kappa_\Gamma = 2/q^N + 2\tau/|C|^\tau + 4(\tau + 1) \cdot (\tau \cdot (l_c - 1) + 1)/q^{2N}$*, where* C *is the challenge space of the proof of linearity employed in* Π_{SHUF}.

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 $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_\tau$ is as follows:

(1) The shuffle server creates encryptions $\boldsymbol{c}_1', \ldots, \boldsymbol{c}_\tau'$ of 0 and commits to each $\boldsymbol{c}_i'$ as $\mathrm{Com}\boldsymbol{c}_i'$. Then, by homomorphically adding $\boldsymbol{c}_i$ to $\mathrm{Com}\boldsymbol{c}_i'$ we obtain commitments $\mathrm{Com}\hat{\boldsymbol{c}}_i$ to the same plaintexts as in $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_\tau$, with "fresh" randomness.

(2) The shuffle server reveals the openings $\hat{\boldsymbol{c}}_i$, 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 (permuted) 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 $B_{\mathsf{Key}}, B_{\mathsf{Err}} \in \mathbb{N}$ 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 $\mathcal{S}$ takes as input a set of τ publicly known BGV ciphertexts $\{\boldsymbol{c}_i\}_{i=1}^\tau$, where the total noise in each ciphertext is bounded by B_{Dec}, i.e. each ciphertexts fulfills $P_{\mathsf{sk}}(\cdot)$.

Randomizing. First, $\mathcal{S}$ randomizes all the received ciphertexts. Towards this it creates a new set of ciphertexts $\{\boldsymbol{c}_i'\}_{i=1}^\tau$:

$$\boldsymbol{c}_i' = (u_i', v_i') = (ar_i' + pe_{i,1}', br_i' + pe_{i,2}'),$$

where $r_i' \xleftarrow{\$} S_{B_{\mathsf{Key}}}$ and $e_{i,1}', e_{i,2}' \xleftarrow{\$} S_{B_{\mathsf{Err}}}$ as in fresh ciphertexts. This corresponds to creating fresh, independent encryptions of 0. $\mathcal{S}$ will *not* publish these $\boldsymbol{c}_i'$.

Committing. $\mathcal{S}$ now commits to the $\boldsymbol{c}_i'$. Towards this, we re-write the commitment matrix from Section 2.2 for $l_c = 2$ and add the public key of the encryption scheme to get a $(n+2) \times (k+3)$ matrix $\boldsymbol{A}_{\mathsf{M}}$, where $\boldsymbol{0}^n$ are row-vectors of length n, $\boldsymbol{a}_{1,1}, \boldsymbol{a}_{1,2}$ are column vectors of length n, $\boldsymbol{a}_{2,3}, \boldsymbol{a}_{3,3}$ are row vectors of length $k - n - 2$ and $\boldsymbol{A}_{1,3}$ is of size $n \times (k - n - 2)$. Then,

$$\mathsf{Com}(u_i', v_i') = \llbracket (ar_i' + pe_{i,1}', br_i' + pe_{i,2}') \rrbracket = \boldsymbol{A}_{\mathsf{M}} \boldsymbol{r}_i'$$
$$= \begin{bmatrix} \boldsymbol{I}_n & \boldsymbol{a}_{1,1} & \boldsymbol{a}_{1,2} & \boldsymbol{A}_{1,3} & 0 & 0 & 0 \\ \boldsymbol{0}^n & 1 & 0 & \boldsymbol{a}_{2,3} & a & p & 0 \\ \boldsymbol{0}^n & 0 & 1 & \boldsymbol{a}_{3,3} & b & 0 & p \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_i \\ r_i' \\ e_{i,1}' \\ e_{i,2}' \end{bmatrix},$$

where $\boldsymbol{r}_i \in R_q^k$ is the randomness used in the commitment. Further, let $\llbracket (u_i, v_i) \rrbracket_{\boldsymbol{0}}$ be the trivial commitment to (u_i, v_i) with no randomness. Then, given the commitment $\llbracket (u_i', v_i') \rrbracket$ and $\llbracket (u_i, v_i) \rrbracket_{\boldsymbol{0}}$ we can be compute a commitment

$$\llbracket (\hat{u}_i, \hat{v}_i) \rrbracket = \llbracket (u_i, v_i) \rrbracket_{\boldsymbol{0}} + \llbracket (u_i', v_i') \rrbracket.$$

Thus, the commitments $\llbracket (\hat{u}_i, \hat{v}_i) \rrbracket$ contain re-randomized encryptions of the original ciphertexts. $\mathcal{S}$ can therefore open a permutation of the $(\hat{u}_i, \hat{v}_i)$ and prove correctness of the shuffled opening using algorithm $\Pi_{\mathrm{SHUF}}^{l_c}$. 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 $\boldsymbol{A}_{\mathsf{M}}$ be the $(n+2) \times (k+3)$ matrix defined above. Then $\mathcal{S}$ needs to prove that, for all i, it knows secret short vectors $\boldsymbol{r}_i'$ of length $k + 3$ that are solutions to the following equations:

$$\boldsymbol{t}_i = \boldsymbol{A}_{\mathsf{M}} \boldsymbol{r}_i' = \llbracket (ar_i' + pe_{i,1}', br_i' + pe_{i,2}') \rrbracket, \left\| \boldsymbol{r}_i' \right\|_\infty \leq B_\infty.$$

To show this, $\mathcal{S}$ runs the Π_{SMALL}-protocol on $\boldsymbol{A}_{\mathsf{M}}$, $\{\boldsymbol{r}_i'\}_{i=1}^\tau$, $\{\boldsymbol{t}_i\}_{i=1}^\tau$. $\mathcal{S}$ uses Fiat-Shamir to ensure the non-interactivity of the proof.

The full protocol. We summarize this protocol as Π_{MIX}:

(1) $\mathcal{S}$ obtains ciphertexts $\{\boldsymbol{c}_i\}_{i \in [\tau]} = \{(u_i, v_i)\}_{i \in [\tau]}$.

(2) $\mathcal{S}$ for each $i \in [\tau]$ samples $r_i', e_{i,1}', e_{i,2}'$ as above. It then creates commitments $\{\llbracket u_i', v_i' \rrbracket = \llbracket ar_i' + pe_{i,1}', br_i' + pe_{i,2}' \rrbracket\}_{i \in [\tau]}$ using randomness $\boldsymbol{r}_i$ for each such commitment.

(3) Let $\boldsymbol{t}_i = \llbracket (u_i', v_i') \rrbracket$ and $\boldsymbol{r}_i' = [\boldsymbol{r}_i^\top, r_i', e_{i,1}, e_{i,2}]^\top$. Then $\mathcal{S}$ computes $\pi_{\mathrm{SMALL}} \leftarrow \Pi_{\mathrm{SMALL}}$ for matrix $\boldsymbol{A}_{\mathsf{M}}$, input vectors $\{\boldsymbol{r}_i'\}$, target vectors $\{\boldsymbol{t}_i\}$ and bound B_∞.

(4) Let $\hat{\boldsymbol{c}}_i = (u_i + u_i', v_i + v_i')$ and L be a random permutation of $\{\hat{\boldsymbol{c}}_i\}_{i \in [\tau]}$. Then $\mathcal{S}$ computes $\pi_{\mathrm{SHUF}} \leftarrow \Pi_{\mathrm{SHUF}}^{l_c}$ with input commitments $\{\llbracket (\hat{u}_i, \hat{v}_i) \rrbracket\}_{i \in [\tau]}$, commitment messages $\{\hat{\boldsymbol{c}}_i\}_{i \in [\tau]}$, randomness $\{\boldsymbol{r}_i\}_{i \in [\tau]}$ and ciphertexts L.

(5) $\mathcal{S}$ outputs $(\{\boldsymbol{t}_i\}_{i \in [\tau]}, \pi_{\mathrm{SMALL}}, L, \pi_{\mathrm{SHUF}})$.

Given such a string $(\{\boldsymbol{t}_i\}_{i\in[\tau]}, \pi_{\text{SMALL}}, L, \pi_{\text{SHUF}})$ from $\mathcal{S}$ as well as ciphertext vector $\{\boldsymbol{c}_i\}_{i\in[\tau]}$ any third party $\mathcal{V}$ can now run the following algorithm Π_{MIXV} to verify the mix:

(1) Run the verification algorithm of Π_{SMALLV} for π_{SMALL} on inputs $\boldsymbol{A}_{\text{M}}, \{\boldsymbol{t}_i\}_{i\in[\tau]}$ and B_∞. If verification fails: output 0.
(2) For $\forall i \in [\tau]$ set $\llbracket \hat{\boldsymbol{c}}_i \rrbracket = \llbracket \boldsymbol{c}_i \rrbracket_0 + \boldsymbol{t}_i$.
(3) Run the verification algorithm of $\Pi^{l_c}_{\text{SHUFV}}$ for π_{SHUF} on input $\{\llbracket \hat{\boldsymbol{c}}_i \rrbracket\}_{i\in[\tau]}, L$. If the verification fails, then output 0. Otherwise, output 1.

In the following, define noise bound B_{Mix} to be the maximum level of noise in ciphertexts $\boldsymbol{c}'_i$, i.e. the maximal noise of the randomness $r'_i, e'_{i,1}, e'_{i,2}$ used to create the ciphertexts.

We want that the outputs of the mixing protocol fulfill the following relation $\mathcal{R}_{\text{MIX}}$:

$$\left\{ (x, w) \;\middle|\; \begin{array}{l} x = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_\tau, \hat{\boldsymbol{c}}_1, \ldots, \hat{\boldsymbol{c}}_\tau, \llbracket \boldsymbol{c}'_1 \rrbracket, \ldots, \llbracket \boldsymbol{c}'_\tau \rrbracket), \\ w = (\pi, \boldsymbol{r}'_1, \ldots, \boldsymbol{r}'_\tau,), \pi \in S_\tau, \forall i \in [\tau] : \\ \llbracket \boldsymbol{c}'_i \rrbracket = \boldsymbol{A}_{\text{M}} \boldsymbol{r}'_i, \left\| \boldsymbol{r}'_i \right\|_\infty \leq B_{\text{Mix}}, \hat{\boldsymbol{c}}_{\pi(i)} = \boldsymbol{c}_i + \boldsymbol{c}'_i \end{array} \right\}.$$

If the noise-levels in all $\boldsymbol{c}_i$ and $\boldsymbol{c}'_i$ are bounded by B_{Dec} and B_{Mix} respectively, and $(B_{\text{Dec}} + B_{\text{Mix}}) < \lfloor q/2 \rfloor$, then all $\boldsymbol{c}_i$ and $\hat{\boldsymbol{c}}_{\pi(i)}$ will, for some permutation π, decrypt to the same message m_i under sk. In Appendix D we analyze the guarantees of Π_{MIX} in more detail.

5.3 Communication of a BGV Shuffle

The mixing phase transcript contains τ new ciphertexts generated by the server, which are of size $2\tau N \log_2 q$ bits.

The server must provide a proof of shuffle and an amortized proof of shortness for $r'_i, e'_{i,1}, e'_{i,2}$. Both proofs prove a relation about commitments to the randomization factors u'_i, v'_i 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 π_{SMALL}.

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

$$((2n+6)N \log_2 q + 2(k-n)N \log_2(6\sigma_{\text{C}}))\tau + |\pi_{\text{SMALL}}| \text{ bits}.$$

6 VERIFIABLE DISTRIBUTED DECRYPTION

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 sec, and bounds $B_{\text{Err}}, B_{\text{Com}}, B_{\text{Dec}}$ be public information, together with the plaintext modulus p for the PKE scheme. Let $\boldsymbol{A}_{\text{C}}$ be the public commitment matrix of a BDLOP instance for message size $l_c = 1$.

- $\mathsf{KGen}_A(1^\kappa, \xi_2)$:
  (1) Get $(\mathsf{pk}, \mathsf{sk}, s_1, \ldots, s_{\xi_2}) \leftarrow \mathsf{KGen}(1^\kappa, \xi_2)$ as in the passive distributed encryption protocol.
  (2) $\forall j \in [\xi_2]$ compute $(\llbracket s_j \rrbracket, \boldsymbol{d}_j) \leftarrow \mathsf{Com}(s_j)$.
  (3) Output $\mathsf{pk}_A = (\mathsf{pk}, \llbracket s_1 \rrbracket, \ldots, \llbracket s_{\xi_2} \rrbracket)$ and finally $\mathsf{sk}_A = \mathsf{sk}$ and $\mathsf{sk}_{A,j} = (s_j, \boldsymbol{d}_j)$ for all $j \in [\xi_2]$.
- Enc_A and Dec_A work just like the original Enc and Dec in the passively secure threshold encryption scheme, ignoring additional information in pk_A.
- $\mathsf{DDec}(\mathsf{sk}_{A,j}, \{c_i\}_{i\in[\tau]})$ where $c_i = (u_i, v_i)$:
  (1) For each $i \in [\tau]$ compute $m_{i,j} = s_j u_i$, sample uniform noise $E_{i,j} \leftarrow R_q$ such that $\left\| E_{i,j} \right\|_\infty \leq 2^{\text{sec}}(B_{\text{Dec}}/p\xi_2)$ and compute the decryption share $t_{i,j} = m_{i,j} + pE_{i,j}$.
  (2) For each $i \in [\tau]$ compute $(\llbracket E_{i,j} \rrbracket, \boldsymbol{r}''_{i,j}) \leftarrow \mathsf{Com}(E_{i,j})$ and use the Π_{LIN}-protocol to compute a proof for the linear relation $t_{i,j} = s_j u_i + pE_{i,j}$ from

$$\begin{aligned} \pi_{L_{i,j}} \leftarrow \Pi_{\text{LIN}}(((s_j, \boldsymbol{r}_j), (E_{i,j}, \boldsymbol{r}''_{i,j})); \\ (\llbracket s_j \rrbracket, \llbracket E_{i,j} \rrbracket, t_{i,j}), (u_i, p)). \end{aligned}$$

  (3) Each commitment $\llbracket E_{i,j} \rrbracket$ is of the form

$$\begin{aligned} &\begin{bmatrix} \boldsymbol{I}_n & \boldsymbol{a}_{1,1} & \boldsymbol{A}_{1,2} \\ \boldsymbol{0}^n & 1 & \boldsymbol{a}_{2,2} \end{bmatrix} \cdot \boldsymbol{r}''_{i,j} + \begin{bmatrix} 0 \\ E_{i,j} \end{bmatrix} \\ = &\underbrace{\begin{bmatrix} \boldsymbol{I}_n & \boldsymbol{a}_{1,1} & \boldsymbol{A}_{1,2} & 0 \\ \boldsymbol{0}^n & 1 & \boldsymbol{a}_{2,2} & 1 \end{bmatrix}}_{\boldsymbol{A}_{\text{D}}} \begin{bmatrix} \boldsymbol{r}''_{i,j} \\ E_{i,j} \end{bmatrix}, \end{aligned}$$

  where $\left\| \boldsymbol{r}''_{i,j} \right\|_\infty \leq B_{\text{Com}}$ is the randomness used in the commitments. Run the zero-knowledge protocol Π_{BND} on $(\{(E_{i,j}, \boldsymbol{r}''_{i,j}\}_{i\in[\tau]}); (\boldsymbol{A}_{\text{D}}, \{\llbracket E_{i,j} \rrbracket\}_{i\in[\tau]}))$ to obtain the amortized zero-knowledge PoK π_{BND_j}.
  (4) Output $\mathsf{ds}_j = (\{t_{i,j}\}_{i=1}^\tau, \pi_{\mathcal{D}_j})$ with the decryption proof $\pi_{\mathcal{D}_j} = (\{\llbracket E_{i,j} \rrbracket\}_{i=1}^\tau, \{\pi_{L_{i,j}}\}_{i=1}^\tau, \pi_{\text{BND}_j})$.
- $\mathsf{Comb}_A(\{c_i\}_{i=1}^\tau, \{\mathsf{ds}_j\}_{j\in[\xi_2]})$:
  (1) Parse ds_j as $(\{t_{i,j}\}_{i=1}^\tau, \pi_{\mathcal{D}_j})$.
  (2) Verify the proofs $\pi_{L_{i,j}}$.
  (3) Verify the proofs π_{BND_j}.
  (4) If any verification protocol returned 0 then output $\perp$. Otherwise, compute

$$\begin{aligned} m_i = (v_i - t_i \mod q) \mod p, \text{ where} \\ t_i = t_{i,1} + \cdots + t_{i,\xi_2} \text{ for } i = 1, \ldots, \tau, \end{aligned}$$

  and output the set of messages $m_1, \ldots, m_\tau$.

The randomness $\boldsymbol{r}''_{i,j}$ has much smaller ℓ_∞ norm than $E_{i,j}$, and hence, we will run the Π_{BND} protocol with small standard deviation σ_{BND} for rows 1 to k, while row $k+1$ will have large $\hat{\sigma}_{\text{BND}}$.This trivially works for Π_{BND} 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_{Dec} and $\hat{B}_{\text{BND}}$ satisfy $(1 + B_{\text{Dec}}) \cdot 2^{\text{sec}} < 2\hat{B}_{\text{BND}} < \lfloor q/2 \rfloor$.

Theorem 5. *Let ciphertext-noise be bounded by* B_{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* Π_{LIN} *and* Π_{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 $(1 + B_{\text{Dec}}) \cdot 2^{\text{sec}} < 2\hat{B}_{\text{BND}} < q/2$ and Π_{LIN} and Π_{BND} are complete, the arguments will be accepted, which means that the decryption proof will be accepted.

Theorem 6. *Let* Adv_0 *be an adversary against threshold verifiability for the actively secure protocol with advantage* ϵ_0*. Then there exists adversaries* Adv_1 *and* Adv_2 *against soundness for* Π_{LIN} *and* Π_{BND}*, respectively, with advantages* ϵ_1 *and* ϵ_2*, such that* $\epsilon_0 \leq \epsilon_1 + \epsilon_2$*. The runtime of* Adv_1 *and* Adv_2 *are essentially the same as of* Adv_0*.*

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

If the decryption is incorrect for a particular ciphertext, then for some j no relation $t_{i,j} = s_j u_i + pE_{i,j}$ holds for an $E_{i,j}$ of the norm at most $2\hat{B}_{\text{BND}}$. 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 Adv_1 against soundness for Π_{LIN}. Similar for the case of Π_{BND}.

Theorem 7. *Suppose the passively secure protocol is simulatable and* Π_{LIN} *and* Π_{BND} *are honest-verifier zero-knowledge. Then there exists a simulator for the actively secure protocol such that for any distinguisher* Adv_0 *for this simulator with advantage* ϵ_0*, there exists an adversary* Adv_4 *against hiding for the commitment scheme*[4]*, with advantage* ϵ_4*, and distinguishers* Adv_1*,* Adv_2 *and* Adv_3 *for the simulators for the passively secure protocol,* Π_{LIN} *and* Π_{BND}*, respectively, with advantages* $\epsilon_1, \epsilon_2, \epsilon_3$*, such that* $\epsilon_0 \leq \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4$*. The runtime of* Adv_1*,* Adv_2*,* Adv_3 *and* Adv_4 *are essentially the same as of* 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 Π_{LIN} arguments with simulated arguments, which gives us a distinguisher Adv_2 for the Π_{LIN} honest verifier simulator. Second, we replace the Π_{BND} arguments with simulated arguments, which gives us a distinguisher Adv_3 for the Π_{BND} honest verifier simulator. Third, we replace the commitments to the noise $E_{i,j}$ with random commitments, giving us an adversary 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 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_q, namely the output of DDec, which means that the output from the passively secure protocol is of size $\xi_2 \tau N \log_2 q$ bits.

Each decryption server outputs a commitment $[\![E_{i,j}]\!]$ 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 $[\![\cdot]\!]$ is of size $(n + 1)N \log_2 q$ bits, and each proof of linearity is of size $(k - n)N(\log_2(6\sigma_{\text{C}}) + \log_2(6\hat{\sigma}_{\text{C}})$ 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 $k\hat{n}N \log_2(6\sigma_{\text{BND}}) + \hat{n} \log_2(6\hat{\sigma}_{\text{BND}})$ 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

$$\begin{aligned}\xi_2((n + 2)N \log_2 q + (k - n)N(\log_2(6\sigma_{\text{C}}) + \log_2(6\hat{\sigma}_{\text{C}}))\\ + k\hat{n} \log_2(6\sigma_{\text{BND}}) + \hat{n} \log_2(6\hat{\sigma}_{\text{BND}}))\tau \text{ bits.}\end{aligned}$$

7 PERFORMANCE

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

Parameter	Explanation	Constraints
κ	Computational security parameter	At least 128 bits
sec	Statistical security parameter	40 bits
N	Degree of polynomial $X^N + 1$ in R_p, R_q	N a power of two
p	Plaintext modulus	p a small prime
q	Ciphertext and commitment modulus	Prime $q = 1 \mod 2N$ s.t. $\max\|v - su\| \ll q/2$
k	Portion of homomorphic commitment vector dedicated to binding	
n	Length of commitment vector	
C	Challenge space for Linear ZK proofs of commitments	$C = \left\{c \in R_p \mid \|c\|_\infty = 1, \|c\|_1 = \nu\right\}$
ν	Maximum ℓ_1-norm of elements in C	
S_B	Set of elements of ∞-norm at most B	$S_B = \{x \in R_p \mid \|x\|_\infty \leq B\}$
B_{Com}	Bound on the commitment noise	–
B_{Key}	Bound for secret key in encryption scheme	Chosen as 1
B_{Err}	Bound for noise in ciphertexts	Chosen as 1
σ_{C}	Standard deviation in linear ZK proofs for one-time commitments	Chosen to be $\sigma_{\text{C}} = 0.954 \cdot \nu \cdot B_{\text{Com}} \cdot \sqrt{kN}$
$\hat{\sigma}_{\text{C}}$	Standard deviation in linear ZK proofs for reusable commitments	Chosen to be $\hat{\sigma}_{\text{C}} = 22 \cdot \nu \cdot B_{\text{Com}} \cdot \sqrt{kN}$
σ_{BND}	Standard deviation for the one-time amortized proof in mixing	Chosen to hide the commitment randomness $\boldsymbol{r}''_{i,j}$
$\hat{\sigma}_{\text{BND}}$	Standard deviation for the one-time amortized proof in mixing	Chosen to hide the decryption noise $E_{i,j}$
$\hat{n}$	Dimension of proof in Π_{BND}	$\hat{n} \geq \kappa + 2$
ξ_1, ξ_2	Number of shuffle- and decryption-servers	
τ	Total number of messages/number of voters	For soundness we need $(\tau^\delta + 1)/\|R_q\| < 2^{-\kappa}$
l	Encoding length in Π_{SMALL}	–
l_c	Length of the committed message in Π_{SMALL}	–
η	Randomness of encodings in Π_{SMALL}	–
g	Dimension of Reed-Solomon Code in Π_{SMALL}	–

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 come from ternary distributions, as this gives tight control on the noise growth during the protocols.

[4] A more careful argument could allow us to dispense with this adversary. We have opted for a simpler argument since the commitment scheme is also used elsewhere.

$\boldsymbol{c}_i^{(k)}$	$[\![R_q^{l_c}]\!]$	$\pi_{\textsc{Shuf}}$	$\pi_{L_{i,j}}$	$\pi_{\textsc{Small}}$	$\pi_{\textsc{Bnd}}$	$\pi_{\mathcal{S}_i}$	$\pi_{\mathcal{D}_j}$
80 KB	$40(l_c+1)$ KB	150τ KB	35 KB	20τ KB	2τ KB	370τ KB	157τ KB

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

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 τ 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 η, the total size of $\pi_{\textsc{Small}}$, by instantiating equation 1, is $\approx 20\tau$ 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 370τ KB per mixing node $\mathcal{S}_k$. The size of the decryption phase, including partial decryptions, commitments, proofs of linearity, and proofs of boundedness, is approximately 157τ KB per decryption node $\mathcal{D}_j$.

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 Kaby Lake 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 is publicly available 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) for polynomial multiplication, so a CRT decomposition to two half-degree polynomials was used instead. The code was made available, so a direct comparison is possible.

Primitive	Commit	Open	Encrypt	Decrypt	DistDec
Time	0.45 ms	2.76 ms	0.74 ms	0.64 ms	1.56 ms

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.

In this work, the degree is much larger ($N = 4096$) and coefficients are multi-word ($q \approx 2^{78}$), but the parameters are compatible with the NTT. We implemented polynomial arithmetic with the efficient NFLlib [2] library using the RNS representation for coefficients and 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 [49] and [48] for Gaussian sampling and adjusting the standard deviation σ accordingly. We employ BLAKE3 [41] for fast hashing inside the various proofs.

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 44.9 ms per vote, thus slower than the 27 ms reported in [4] due to slower rejection sampling.

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 $\Pi_{\textsc{Small}}$, we implement only 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 $\Pi_{\textsc{Dec}}$ with active security as $(1.56 + 0.45 + 43.4 + 92.7) = 138.1$ ms per vote, the cost of verification $\Pi_{\textsc{DecV}}$ as $(6.4{+}23.9) = 30.2$ ms per vote, the cost of $\Pi_{\textsc{Mix}}$ as $(0.74 + 0.45 + 214.4 + 44.9) = 260.5$ ms and the cost of $\Pi_{\textsc{MixV}}$ as $(10.0 + 7.9) = 17.9$ ms per vote[5]. 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 [28] in a Kaby Lake processor running at a similar frequency. Our total numbers are around 10.4 times faster after adjusting for clock frequency, while storage overhead is much lower.

8 CONCLUDING REMARKS

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.

[5] Note that in the conference version of this paper we only included the timings for running $\Pi_{\textsc{Small}}$ once (107.2 ms), while it is necessary to run it twice to get the appropriate soundness levels required. The comparison is updated accordingly.

Protocol	$\Pi_{\mathrm{Lin}} + \Pi_{\mathrm{LinV}}$	$\Pi^{l_c}_{\mathrm{Shuf}} + \Pi^{l_c}_{\mathrm{ShufV}}$
Time	$(43.4 + 6.4)\tau$ ms	$(44.9 + 7.9)\tau$ ms

Protocol	$\Pi_{\mathrm{Bnd}} + \Pi_{\mathrm{BndV}}$	$\Pi_{\mathrm{Small}} + \Pi_{\mathrm{SmallV}}$
Time	$(92.7 + 23.9)\tau$ ms	$(214.4 + 10.0)\tau$ ms

Table 4: Timings for cryptographic protocols, obtained by computing the average of 100 executions with $\tau = 1000$.

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 straight-forward 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 [30, 45]). 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 Høgåsen and Silde [33] 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^2 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. [37], continuously improves the state-of-the-art, and we expect future works to improve the concrete efficiency of our protocol.

REFERENCES

[1] Ben Adida. 2008. Helios: Web-based Open-Audit Voting. In *USENIX Security 2008: 17th USENIX Security Symposium*, Paul C. van Oorschot (Ed.). USENIX Association, San Jose, CA, USA, 335–348.

[2] Carlos Aguilar Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian, and Tancrède Lepoint. 2016. NFLlib: NTT-Based Fast Lattice Library. In *Topics in Cryptology – CT-RSA 2016 (Lecture Notes in Computer Science, Vol. 9610)*, Kazue Sako (Ed.). Springer, Heidelberg, Germany, San Francisco, CA, USA, 341–356. https://doi.org/10.1007/978-3-319-29485-8_20

[3] Andris Ambainis, Ansis Rosmanis, and Dominique Unruh. 2014. Quantum Attacks on Classical Proof Systems: The Hardness of Quantum Rewinding. In *55th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Philadelphia, PA, USA, 474–483. https://doi.org/10.1109/FOCS.2014.57

[4] Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, Tjerand Silde, and Thor Tunge. 2021. Lattice-Based Proof of Shuffle and Applications to Electronic Voting. In *Topics in Cryptology – CT-RSA 2021 (Lecture Notes in Computer Science, Vol. 12704)*, Kenneth G. Paterson (Ed.). Springer, Heidelberg, Germany, Virtual Event, 227–251. https://doi.org/10.1007/978-3-030-75539-3_10

[5] Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, and Tjerand Silde. 2023. Verifiable Mix-Nets and Distributed Decryption for Voting from Lattice-Based Assumptions. ACM CCS. https://eprint.iacr.org/2022/1725

[6] Thomas Attema, Serge Fehr, and Nicolas Resch. 2023. A Generalized Special-Soundness Notion and its Knowledge Extractors. *Cryptology ePrint Archive* (2023).

[7] Thomas Attema, Vadim Lyubashevsky, and Gregor Seiler. 2020. Practical Product Proofs for Lattice Commitments. In *Advances in Cryptology – CRYPTO 2020, Part II (Lecture Notes in Computer Science, Vol. 12171)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 470–499. https://doi.org/10.1007/978-3-030-56880-1_17

[8] Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël del Pino, Jens Groth, and Vadim Lyubashevsky. 2018. Sub-linear Lattice-Based Zero-Knowledge Arguments for Arithmetic Circuits. In *Advances in Cryptology – CRYPTO 2018, Part II (Lecture Notes in Computer Science, Vol. 10992)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 669–699. https://doi.org/10.1007/978-3-319-96881-0_23

[9] Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner, and Chris Peikert. 2018. More Efficient Commitments from Structured Lattice Assumptions. In *SCN 18: 11th International Conference on Security in Communication Networks (Lecture Notes in Computer Science, Vol. 11035)*, Dario Catalano and Roberto De Prisco (Eds.). Springer, Heidelberg, Germany, Amalfi, Italy, 368–385. https://doi.org/10.1007/978-3-319-98113-0_20

[10] Carsten Baum, Daniel Escudero, Alberto Pedrouzo-Ulloa, Peter Scholl, and Juan Ramón Troncoso-Pastoriza. 2020. Efficient Protocols for Oblivious Linear Function Evaluation from Ring-LWE. In *SCN 20: 12th International Conference on Security in Communication Networks (Lecture Notes in Computer Science, Vol. 12238)*, Clemente Galdi and Vladimir Kolesnikov (Eds.). Springer, Heidelberg, Germany, Amalfi, Italy, 130–149. https://doi.org/10.1007/978-3-030-57990-6_7

[11] Rikke Bendlin and Ivan Damgård. 2010. Threshold Decryption and Zero-Knowledge Proofs for Lattice-Based Cryptosystems. In *TCC 2010: 7th Theory of Cryptography Conference (Lecture Notes in Computer Science, Vol. 5978)*, Daniele Micciancio (Ed.). Springer, Heidelberg, Germany, Zurich, Switzerland, 201–218. https://doi.org/10.1007/978-3-642-11799-2_13

[12] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. 2015. SoK: A Comprehensive Analysis of Game-Based Ballot Privacy Definitions. In *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Jose, CA, USA, 499–516. https://doi.org/10.1109/SP.2015.37

[13] Manuel Blum. 1984. How to Exchange (Secret) Keys. *ACM Transactions on Computer Systems* 1 (1984), 175–193.

[14] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. 2017. Linear-Time Zero-Knowledge Proofs for Arithmetic Circuit Satisfiability. In *Advances in Cryptology – ASIACRYPT 2017, Part III (Lecture Notes in Computer Science, Vol. 10626)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer, Heidelberg, Germany, Hong Kong, China, 336–365. https://doi.org/10.1007/978-3-319-70700-6_12

[15] Jonathan Bootle, Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. 2021. More Efficient Amortization of Exact Zero-Knowledge Proofs for LWE. In *ESORICS 2021: 26th European Symposium on Research in Computer Security, Part II (Lecture Notes in Computer Science, Vol. 12973)*, Elisa Bertino, Haya Shulman, and Michael Waidner (Eds.). Springer, Heidelberg, Germany, Darmstadt, Germany, 608–627. https://doi.org/10.1007/978-3-030-88428-4_30

[16] Jonathan Bootle, Vadim Lyubashevsky, and Gregor Seiler. 2019. Algebraic Techniques for Short(er) Exact Lattice-Based Zero-Knowledge Proofs. In *Advances in Cryptology – CRYPTO 2019, Part I (Lecture Notes in Computer Science, Vol. 11692)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 176–202. https://doi.org/10.1007/978-3-030-26948-7_7

[17] Xavier Boyen, Thomas Haines, and Johannes Müller. 2020. A Verifiable and Practical Lattice-Based Decryption Mix Net with External Auditing. In *ESORICS 2020: 25th European Symposium on Research in Computer Security, Part II (Lecture Notes in Computer Science, Vol. 12309)*, Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider (Eds.). Springer, Heidelberg, Germany, Guildford, UK, 336–356. https://doi.org/10.1007/978-3-030-59013-0_17

[18] Xavier Boyen, Thomas Haines, and Johannes Müller. 2021. Epoque: Practical End-to-End Verifiable Post-Quantum-Secure E-Voting. In *IEEE European Symposium on Security and Privacy, EuroS&P 2021, Vienna, Austria, September 6-10, 2021*. IEEE, 272–291. https://doi.org/10.1109/EuroSP51992.2021.00027

[19] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, Shafi Goldwasser (Ed.). Association for Computing Machinery, Cambridge, MA, USA, 309–325. https://doi.org/10.1145/2090236.2090262

[20] David Chaum. 1981. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM* 24, 2 (1981), 84–88. https://doi.org/10.1145/358549.358563

[21] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2016. A Homomorphic LWE Based E-voting Scheme. In *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*, Tsuyoshi Takagi (Ed.). Springer, Heidelberg, Germany, Fukuoka, Japan, 245–265. https://doi.org/10.1007/978-3-319-29360-8_16

[22] Núria Costa, Ramiro Martínez, and Paz Morillo. 2019. Lattice-Based Proof of a Shuffle. In *FC 2019 Workshops (Lecture Notes in Computer Science, Vol. 11599)*, Andrea Bracciali, Jeremy Clark, Federico Pintore, Peter B. Rønne, and Massimiliano Sala (Eds.). Springer, Heidelberg, Germany, Frigate Bay, St. Kitts and Nevis, 330–346. https://doi.org/10.1007/978-3-030-43725-1_23

[23] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. 2013. Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits. In *ESORICS 2013: 18th European Symposium on Research in Computer Security (Lecture Notes in Computer Science, Vol. 8134)*, Jason Crampton, Sushil Jajodia, and Keith Mayes (Eds.). Springer, Heidelberg, Germany, Egham, UK, 1–18. https://doi.org/10.1007/978-3-642-40203-6_1

[24] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. 2012. Multiparty Computation from Somewhat Homomorphic Encryption. In *Advances in Cryptology – CRYPTO 2012 (Lecture Notes in Computer Science, Vol. 7417)*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 643–662. https://doi.org/10.1007/978-3-642-32009-5_38

[25] Rafaël del Pino, Vadim Lyubashevsky, Gregory Neven, and Gregor Seiler. 2017. Practical Quantum-Safe Voting from Lattices. In *ACM CCS 2017: 24th Conference on Computer and Communications Security*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, Dallas, TX, USA, 1565–1581. https://doi.org/10.1145/3133956.3134101

[26] Jelle Don, Serge Fehr, and Christian Majenz. 2020. The Measure-and-Reprogram Technique 2.0: Multi-round Fiat-Shamir and More. In *Advances in Cryptology – CRYPTO 2020, Part III (Lecture Notes in Computer Science, Vol. 12172)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 602–631. https://doi.org/10.1007/978-3-030-56877-1_21

[27] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. 2022. Efficient NIZKs and Signatures from Commit-and-Open Protocols in the QROM. In *Advances in Cryptology – CRYPTO 2022, Part II (Lecture Notes in Computer Science, Vol. 13508)*, Yevgeniy Dodis and Thomas Shrimpton (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 729–757. https://doi.org/10.1007/978-3-031-15979-4_25

[28] Valeh Farzaliyev, Jan Willemson, and Jaan Kristjan Kaasik. 2023. Improved lattice-based mix-nets for electronic voting. *IET Inf. Secur.* 17, 1 (2023), 18–34.

[29] Amos Fiat and Adi Shamir. 1987. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology – CRYPTO'86 (Lecture Notes in Computer Science, Vol. 263)*, Andrew M. Odlyzko (Ed.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 186–194. https://doi.org/10.1007/3-540-47721-7_12

[30] Kristian Gjøsteen, Thomas Haines, Johannes Müller, Peter Rønne, and Tjerand Silde. 2022. Verifiable Decryption In The Head. In *Information Security and Privacy: 27th Australasian Conference, ACISP 2022, Wollongong, NSW, Australia, November 28–30, 2022, Proceedings* (Wollongong, NSW, Australia). Springer-Verlag, Berlin, Heidelberg, 355–374. https://doi.org/10.1007/978-3-031-22301-3_18

[31] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1985. The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract). In *17th Annual ACM Symposium on Theory of Computing*. ACM Press, Providence, RI, USA, 291–304. https://doi.org/10.1145/22145.22178

[32] Javier Herranz, Ramiro Martínez, and Manuel Sánchez. 2021. Shorter lattice-based zero-knowledge proofs for the correctness of a shuffle. In *Financial Cryptography and Data Security. FC 2021 International Workshops: CoDecFin, DeFi, VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers 25*. Springer, 315–329.

[33] Audhild Høgåsen and Tjerand Silde. 2022. Return Codes from Lattice Assumptions. *E-VOTE-ID* (2022). https://doi.org/10.15157/diss/025

[34] Adeline Langlois and Damien Stehlé. 2015. Worst-Case to Average-Case Reductions for Module Lattices. *Des. Codes Cryptography* 75, 3 (June 2015), 565–599. https://doi.org/10.1007/s10623-014-9938-4

[35] Patrick Longa and Michael Naehrig. 2016. Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography. In *CANS 16: 15th International Conference on Cryptology and Network Security (Lecture Notes in Computer Science, Vol. 10052)*, Sara Foresti and Giuseppe Persiano (Eds.). Springer, Heidelberg, Germany, Milan, Italy, 124–139. https://doi.org/10.1007/978-3-319-48965-0_8

[36] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. 2020. *CRYSTALS-DILITHIUM*. Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[37] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Maxime Plançon. 2022. Lattice-Based Zero-Knowledge Proofs and Applications: Shorter, Simpler, and More General. In *Advances in Cryptology – CRYPTO 2022, Part II (Lecture Notes in Computer Science, Vol. 13508)*, Yevgeniy Dodis and Thomas Shrimpton (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 71–101. https://doi.org/10.1007/978-3-031-15979-4_3

[38] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. 2021. Shorter Lattice-Based Zero-Knowledge Proofs via One-Time Commitments. In *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part I (Lecture Notes in Computer Science, Vol. 12710)*, Juan Garay (Ed.). Springer, Heidelberg, Germany, Virtual Event, 215–241. https://doi.org/10.1007/978-3-030-75245-3_9

[39] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2013. A Toolkit for Ring-LWE Cryptography. In *Advances in Cryptology – EUROCRYPT 2013 (Lecture Notes in Computer Science, Vol. 7881)*, Thomas Johansson and Phong Q. Nguyen (Eds.). Springer, Heidelberg, Germany, Athens, Greece, 35–54. https://doi.org/10.1007/978-3-642-38348-9_3

[40] C. Andrew Neff. 2001. A Verifiable Secret Shuffle and Its Application to e-Voting. In *ACM CCS 2001: 8th Conference on Computer and Communications Security*, Michael K. Reiter and Pierangela Samarati (Eds.). ACM Press, Philadelphia, PA, USA, 116–125. https://doi.org/10.1145/501983.502000

[41] Jack O'Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O'Hearn. 2020. BLAKE3: one function, fast everywhere. https://www.blake3.io. , 31 pages.

[42] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. 2020. *FALCON*. Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[43] Dragos Rotaru, Nigel P. Smart, Titouan Tanguy, Frederik Vercauteren, and Tim Wood. 2022. Actively Secure Setup for SPDZ. *J. Cryptol.* 35, 1 (jan 2022), 32 pages. https://doi.org/10.1007/s00145-021-09416-w

[44] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. 2020. *CRYSTALS-KYBER*. Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[45] Tjerand Silde. 2022. Short Paper: Verifiable Decryption for BGV. In *Financial Cryptography and Data Security. FC 2022 International Workshops - CoDecFin, DeFi, Voting, WTSC, Grenada, May 6, 2022, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 13412)*, Shin'ichiro Matsuo, Lewis Gudgeon, Ariah Klages-Mundt, Daniel Perez Hernandez, Sam Werner, Thomas Haines, Aleksander Essex, Andrea Bracciali, and Massimiliano Sala (Eds.). Springer, 381–390. https://doi.org/10.1007/978-3-031-32415-4_26

[46] Kristian Gjøsteen. 2022. *Practical Mathematical Cryptography*. CRC Press.

[47] Martin Strand. 2019. A Verifiable Shuffle for the GSW Cryptosystem. In *FC 2018 Workshops (Lecture Notes in Computer Science, Vol. 10958)*, Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala (Eds.). Springer, Heidelberg, Germany, Nieuwpoort, Curaçao, 165–180. https://doi.org/10.1007/978-3-662-58820-8_12

[48] Shuo Sun, Yongbin Zhou, Yunfeng Ji, Rui Zhang, and Yang Tao. 2022. Generic, efficient and isochronous Gaussian sampling over the integers. *Cybersecur.* 5, 1 (2022), 10.

[49] Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. 2020. COSAC: COmpact and Scalable Arbitrary-Centered Discrete Gaussian Sampling over Integers. In *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, Jintai Ding and Jean-Pierre Tillich (Eds.). Springer, Heidelberg, Germany, Paris, France, 284–303. https://doi.org/10.1007/978-3-030-44223-1_16

```
Rej($\boldsymbol{z}, \boldsymbol{v}, b, M, \sigma$)
  if $b = 1$ and $\langle \boldsymbol{z}, \boldsymbol{v} \rangle < 0$: return 1
  $\mu \xleftarrow{\$} [0, 1)$
  if $\mu > \frac{1}{M} \cdot \exp\left[\frac{-2\langle \boldsymbol{z}, \boldsymbol{v} \rangle + \|\boldsymbol{v}\|_2^2}{2\sigma^2}\right]$ : return 1
  else : return 0
```

Figure 2: Rejection sampling

A PRELIMINARIES

Let N be a power of 2 and q a prime such that $q \equiv 1 \mod 2N$. We define the rings $R = \mathbb{Z}[X]/\langle X^N + 1\rangle$ and $R_q = R/qR$, 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) [35]. 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 $\mathbb{Z}^N$:

$$||f||_1 = \sum |\alpha_i|, ||f||_2 = \left(\sum \alpha_i^2\right)^{1/2}, ||f||_\infty = \max\{|\alpha_i|\}.$$

For an element $\bar{f} \in R_q$ we choose coefficients as the representatives in $\left[-\frac{q-1}{2}, \frac{q-1}{2}\right]$, and then compute the norms as if $\bar{f}$ is an element in R. For vectors $\boldsymbol{a} = (a_1, \ldots, a_k) \in R^k$ we define the ℓ_2 norm to be $\|\boldsymbol{a}\|_2 = \sqrt{\sum \|a_i\|_2^2}$, and analogously for the ℓ_1 and ℓ_∞ norm. It is easy to see the following relations between the norms of elements in R_q:

$$\|f\|_\infty \leq \alpha, \|g\|_1 \leq \beta, \text{ then } \|fg\|_\infty \leq \alpha\beta,$$
$$\|f\|_2 \leq \alpha, \|g\|_2 \leq \beta, \text{ then } \|fg\|_\infty \leq \alpha\beta.$$

We also define the sets $S_B = \{x \in R_q \mid \|x\|_\infty \leq B\}$ as well as

$$C = \left\{c \in R_q \mid \|c\|_\infty = 1, \|c\|_1 = \nu\right\}, \bar{C} = \left\{c - c' \mid c \neq c' \in C\right\}.$$

A.1 The Discrete Gaussian Distribution

The continuous normal distribution over $\mathbb{R}^k$ centered at $\boldsymbol{v} \in \mathbb{R}^k$ with standard deviation σ is given by

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

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 continuous distribution over R^k by letting

$$\mathcal{N}^k_{\boldsymbol{v},\sigma}(\boldsymbol{x}) = \frac{\rho^{kN}_{\boldsymbol{v},\sigma}(\boldsymbol{x})}{\rho^{kN}_\sigma(R^k)} \text{ for } \boldsymbol{x} \in R^k, \rho^{kN}_\sigma(R^k) = \sum_{\boldsymbol{x} \in R^k} \rho^{kN}_\sigma(\boldsymbol{x}).$$

When $\sigma = 1$ or $\boldsymbol{v} = \boldsymbol{0}$, they are omitted. When $\boldsymbol{x}$ is sampled according to $\mathcal{N}_\sigma$ (see Section 2.1 in [8]), then,

$$\mathsf{Pr}[\|\boldsymbol{x}\|_\infty > \gamma\sigma] \leq 2e^{-\gamma^2/2} \text{ and } \mathsf{Pr}[\|\boldsymbol{x}\|_2 > \sqrt{2\gamma}\sigma] < 2^{-\gamma/4}.$$

A.2 Rejection Sampling

In lattice-based cryptography in general, and in our zero-knowledge protocols in particular, we would like to output vectors $\boldsymbol{z} = \boldsymbol{y} + \boldsymbol{v}$ such that $\boldsymbol{z}$ is independent of $\boldsymbol{v}$, and hence, $\boldsymbol{v}$ is masked by the vector $\boldsymbol{y}$. Here, $\boldsymbol{y}$ is sampled according to a Gaussian distribution $\mathcal{N}^k_\sigma$ with standard deviation σ, and we want the output vector $\boldsymbol{z}$ to be from the same distribution. The procedure is shown in Figure 2.

Here, $1/M$ is the probability of success, and M is computed as

$$\max \frac{\mathcal{N}^k_\sigma(\boldsymbol{z})}{\mathcal{N}^k_{\boldsymbol{v},\sigma}(\boldsymbol{z})} \leq \exp\left[\frac{24\sigma\|\boldsymbol{v}\|_2 + \|\boldsymbol{v}\|_2^2}{2\sigma^2}\right] = M \tag{2}$$

where we use the tail bound from Section A.1, saying that $|\langle \boldsymbol{z}, \boldsymbol{v}\rangle| < 12\sigma\|\boldsymbol{v}\|_2$ with probability at least $1 - 2^{-100}$. Hence, for $\sigma = 11\|\boldsymbol{v}\|_2$, we get $M \approx 3$. This is the standard way to choose parameters, see e.g. [16]. However, if the procedure is only done once for the vector $\boldsymbol{v}$, we can decrease the parameters slightly, to the cost of leaking only one bit of information about $\boldsymbol{v}$ from the given $\boldsymbol{z}$.

In [38], Lyubashevsky et al. suggest to require that $\langle \boldsymbol{z}, \boldsymbol{v}\rangle \geq 0$, and hence, we can set $M = \exp(\|v\|_2/2\sigma^2)$. Then, for $\sigma = 0.675\|\boldsymbol{v}\|_2$, we get $M \approx 3$. In Figure 2, we use the pre-determined bit b to denote if we only use $\boldsymbol{v}$ once or not, with the effect of rejecting about half of the vectors before the sampling of uniform value μ in the case $b = 1$ but allowing a smaller standard deviation.

A.3 Knapsack Problems

We first define the Search Knapsack problem in the ℓ_2 norm, also denoted as SKS^2. The SKS^2 problem is exactly the Module-SIS problem in its Hermite Normal Form.

Definition 6. *The* $\mathsf{SKS}^2_{n,k,\beta}$ *problem is to find a short non-zero vector* $\boldsymbol{y}$ *satisfying* $[\ \boldsymbol{I}_n \quad \boldsymbol{A}'\ ] \cdot \boldsymbol{y} = \boldsymbol{0}^n$ *for a random matrix* $\boldsymbol{A}'$. *An algorithm* Adv *has advantage* ϵ *in solving the* $\mathsf{SKS}^2_{n,k,\beta}$ *problem if the following probability is equal to* ϵ*:*

$$\Pr\left[\begin{array}{c|c} \|y_i\|_2 \leq \beta \wedge & \boldsymbol{A}' \xleftarrow{\$} q^{n\times(k-n)}; \\ [\ \boldsymbol{I}_n \quad \boldsymbol{A}'\ ] \cdot \boldsymbol{y} = \boldsymbol{0}^n & 0 \neq [y_1, \ldots, y_k]^\top \leftarrow \mathsf{Adv}(\boldsymbol{A}') \end{array}\right].$$

Additionally, we define the Decisional Knapsack problem in the ℓ_∞ norm (DKS^∞). The DKS^∞ problem is equivalent to the Module-LWE problem when the number of samples is limited.

Definition 7. *The* $\mathsf{DKS}^\infty_{n,k,\beta}$ *problem is to distinguish the distribution* $[\ \boldsymbol{I}_n \quad \boldsymbol{A}'\ ] \cdot \boldsymbol{y}$ *for a short* $\boldsymbol{y}$ *from a bounded distribution* S_B *when given* $\boldsymbol{A}'$. *An algorithm* Adv *has advantage* ϵ *in solving the* $\mathsf{DKS}^\infty_{n,k,\beta}$ *problem if*

$$\left|\Pr[b = 1 \mid \boldsymbol{y} \xleftarrow{\$} S^k_B; b \leftarrow \mathsf{Adv}(\boldsymbol{A}', [\ \boldsymbol{I}_n \quad \boldsymbol{A}'\ ] \cdot \boldsymbol{y})] - \Pr[b = 1 \mid \boldsymbol{u} \xleftarrow{\$} R^n_q; b \leftarrow \mathsf{Adv}(\boldsymbol{A}', \boldsymbol{u})]\right| = \epsilon,$$

for a uniformly sampled $\boldsymbol{A}' \xleftarrow{\$} R_q^{n\times(k-n)}$.

See [34] 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 $\mathcal{A}$, after choosing two messages m_0 and m_1 and receiving an encryption c of either m_0 or m_1 (chosen at random), cannot distinguish which message c is an encryption of. Hence, we want that*

$$|\Pr\left[b = b' \ : \ \begin{matrix} (\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\kappa) \\ (m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk}) \\ b \xleftarrow{\$} \{0,1\}, c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b) \\ b' \leftarrow \mathcal{A}(c, \mathsf{st}) \end{matrix} \right] - \frac{1}{2}| \leq \epsilon(\kappa),$$

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 $P_{\mathsf{sk}}(\cdot)$ if the following probability equals* 1*:*

$$\Pr\left[\begin{matrix} \mathsf{Comb}(\{c_i\}_{i\in[\tau]}, \{\mathsf{ds}_{i,j}\}_{i\in[\tau]}^{j\in[\xi_2]}) \\ = \\ \mathsf{Dec}(\mathsf{sk}, \{c_i\}_{i\in[\tau]}) \end{matrix} \ : \ \begin{matrix} (\mathsf{pp}, \mathsf{pk}, \mathsf{sk}, \{\mathsf{sk}_j\}_{j\in[\xi_2]}) \leftarrow \mathsf{KGen}(1^\kappa, \xi_2) \\ \{c_1, \ldots, c_\tau\} \leftarrow \mathsf{Adv}(\mathsf{pp}, \mathsf{pk}) \\ \forall i \in [\tau] : \ P_{\mathsf{sk}}(c_i) = 1, \forall j \in [\xi_2] : \\ \{\mathsf{ds}_{i,j}\}_{i\in[\tau]} \leftarrow \mathsf{DDec}(\mathsf{sk}_j, \{c_i\}_{i\in[\tau]}) \end{matrix} \right],$$

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 $\mathcal{A}$, 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*

$$|\Pr\left[b = b' \ : \ \begin{matrix} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa) \\ (m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ b \xleftarrow{\$} \{0,1\}, c \leftarrow \mathsf{Com}(m_b) \\ b' \leftarrow \mathcal{A}(c, \mathsf{st}) \end{matrix} \right] - \frac{1}{2}| \leq \epsilon(\kappa),$$

where the probability is taken over Setup *and* Com.

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

$$\Pr\left[\begin{matrix} m \neq \hat{m} \\ \mathsf{Open}(m, c, r) = 1 \\ \mathsf{Open}(\hat{m}, c, \hat{r}) = 1 \end{matrix} \ : \ \begin{matrix} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa) \\ (c, m, r, \hat{m}, \hat{r}) \leftarrow \mathcal{A}(\mathsf{pp}) \end{matrix} \right] \leq \epsilon(\kappa),$$

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 $\mathcal{V}$ outputs* 1 *when $\mathcal{P}$ knows a witness w and both parties follow the protocol. Hence, for any efficient sampling algorithm $\mathcal{P}_0$ we want that*

$$\Pr\left[\langle \mathcal{P}(\mathsf{sp}, x, w), \mathcal{V}(\mathsf{sp}, x) \rangle = 1 \ : \ \begin{matrix} \mathsf{sp} \leftarrow \mathsf{Setup}(1^\kappa) \\ (x, w) \leftarrow \mathcal{P}_0(\mathsf{sp}) \\ (x, w) \in \mathsf{R} \end{matrix} \right] = 1,$$

where the probability is taken over Setup, $\mathcal{P}$ *and* $\mathcal{V}$.

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

$$\Pr\left[(x, w) \in \mathsf{R} \ : \ \begin{matrix} \mathsf{sp} \leftarrow \mathsf{Setup}(1^\kappa) \\ \langle \mathcal{P}^*(\mathsf{sp}, x, \cdot), \mathcal{V}(\mathsf{sp}, x) \rangle = 1 \\ w \leftarrow \mathcal{E}^{\mathcal{P}^*(\cdot)}(\mathsf{sp}, x) \end{matrix} \right] \geq 1 - \epsilon(\kappa),$$

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

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

$$|\Pr\left[b = b' \ : \ \begin{matrix} \mathsf{sp} \leftarrow \mathsf{Setup}(1^\kappa) \\ \mathsf{T}_0 = \mathsf{T}_{\langle \mathcal{P}(\mathsf{sp},x,w), \mathcal{V}(\mathsf{sp},x) \rangle} \leftarrow {}''(\mathsf{sp}, x, w) \\ \mathsf{T}_1 = \mathsf{S}_{\langle \mathcal{P}(\mathsf{sp},x,\cdot), \mathcal{V}(\mathsf{sp},x) \rangle} \leftarrow \mathcal{S}(\mathsf{sp}, x) \\ b \xleftarrow{\$} \{0,1\}, b' \leftarrow \mathcal{V}^*(\mathsf{sp}, x, \mathsf{T}_b) \end{matrix} \right] - \frac{1}{2}| \leq \epsilon(\kappa),$$

where the probability is taken over Setup, $\mathcal{S}$ *and* $\mathcal{V}^*$.

B PROOFS OF LINEAR RELATIONS AND AMORTIZED NON-EXACT ZKPOPK

B.1 Zero-Knowledge Proof of Linear Relations

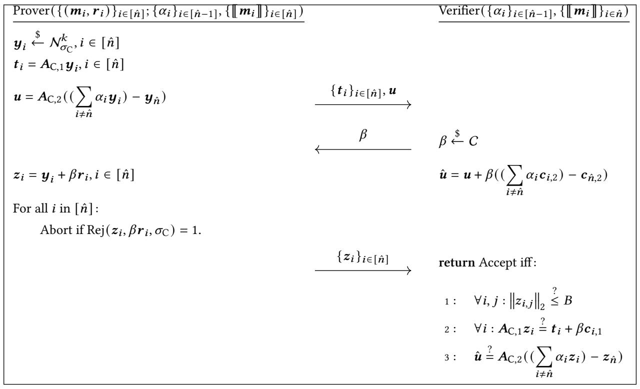
$\mathsf{Prover}(\{(\boldsymbol{m}_i, \boldsymbol{r}_i)\}_{i\in[\hat{n}]}; \{\alpha_i\}_{i\in[\hat{n}-1]}, \{[\![\boldsymbol{m}_i]\!]\}_{i\in[\hat{n}]})$
$\mathsf{Verifier}(\{\alpha_i\}_{i\in[\hat{n}-1]}, \{[\![\boldsymbol{m}_i]\!]\}_{i\in\hat{n}})$
$\boldsymbol{y}_i \xleftarrow{\$} \mathcal{N}^k_{\sigma_C}, i \in [\hat{n}]$
$\boldsymbol{t}_i = A_{C,1}\boldsymbol{y}_i, i \in [\hat{n}]$
$\boldsymbol{u} = A_{C,2}((\sum_{i\neq\hat{n}} \alpha_i \boldsymbol{y}_i) - \boldsymbol{y}_{\hat{n}})$
$\xrightarrow{\{\boldsymbol{t}_i\}_{i\in[\hat{n}]}, \boldsymbol{u}}$
$\xleftarrow{\beta}$
$\beta \xleftarrow{\$} C$
$\boldsymbol{z}_i = \boldsymbol{y}_i + \beta \boldsymbol{r}_i, i \in [\hat{n}]$
$\hat{\boldsymbol{u}} = \boldsymbol{u} + \beta((\sum_{i\neq\hat{n}} \alpha_i \boldsymbol{c}_{i,2}) - \boldsymbol{c}_{\hat{n},2})$
For all i in $[\hat{n}]$:
Abort if $\mathrm{Rej}(\boldsymbol{z}_i, \beta \boldsymbol{r}_i, \sigma_C) = 1$.
$\xrightarrow{\{\boldsymbol{z}_i\}_{i\in[\hat{n}]}}$
return Accept iff:
1 : $\forall i, j : \|z_{i,j}\|_2 \overset{?}{\leq} B$
2 : $\forall i : A_{C,1}\boldsymbol{z}_i \overset{?}{=} \boldsymbol{t}_i + \beta \boldsymbol{c}_{i,1}$
3 : $\hat{\boldsymbol{u}} \overset{?}{=} A_{C,2}((\sum_{i\neq\hat{n}} \alpha_i \boldsymbol{z}_i) - \boldsymbol{z}_{\hat{n}})$

Figure 3: $\Pi_{\mathbf{LIN}}$ is a Sigma-protocol to prove the relation $\mathcal{R}_{\mathbf{LIN}}$.

Assume that there are $\hat{n}$ commitments

$$[\![\boldsymbol{m}_i]\!] = \begin{bmatrix} \boldsymbol{c}_{i,1} \\ \boldsymbol{c}_{i,2} \end{bmatrix}, \text{ for } 1 \leq i \leq \hat{n} \text{ where } \boldsymbol{c}_{i,2} \in R_q^{l_c}.$$

For the public scalar vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{\hat{n}-1}) \in R_q^{\hat{n}-1}$ the prover wants to prove that the following relation holds:

$$\mathcal{R}_{\mathrm{LIN}} = \left\{ (x, w) \ \middle| \ \begin{matrix} x = (\mathsf{pk}, \{[\![\boldsymbol{m}_i]\!]\}_{i\in[\hat{n}]}, \boldsymbol{\alpha}) \ \wedge \\ w = (f, \{\boldsymbol{m}_i, \boldsymbol{r}_i\}_{i\in[\hat{n}]}) \ \wedge \\ \forall i \in [\hat{n}] : \mathsf{Open}_{\mathsf{pk}}([\![\boldsymbol{m}_i]\!], \boldsymbol{m}_i, \boldsymbol{r}_i, f) = 1 \\ \wedge\, \boldsymbol{m}_{\hat{n}} = \sum_{i=1}^{\hat{n}-1} \alpha_i \boldsymbol{m}_i \end{matrix} \right\}.$$

Π_{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 [9]). 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 [9].

The relation $\mathcal{R}_{\mathrm{LIN}}$ 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 $B = 2\sigma_C\sqrt{N}$ and the protocol produces a proof transcript of the form $\pi_{\mathrm{LIN}} = ((\{\boldsymbol{t}_i\}_{i\in[\hat{n}]}, \boldsymbol{u}), \beta, (\{\boldsymbol{z}_i\}_{i\in[\hat{n}]}))$.

The following is a direct adaption of Baum et al. [9, Lemma 8].

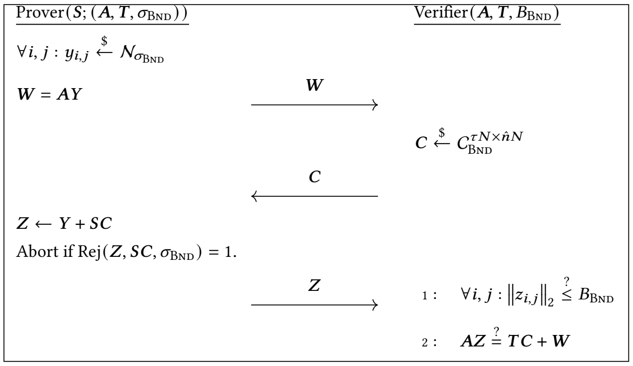
$\text{Prover}(\boldsymbol{S}; (\boldsymbol{A}, \boldsymbol{T}, \sigma_{\text{BND}}))$
$\text{Verifier}(\boldsymbol{A}, \boldsymbol{T}, B_{\text{BND}})$
$\forall i, j : y_{i,j} \xleftarrow{\$} \mathcal{N}_{\sigma_{\text{BND}}}$
$\boldsymbol{W} = \boldsymbol{A}\boldsymbol{Y}$
$\boldsymbol{W}$
$\boldsymbol{C} \xleftarrow{\$} \mathcal{C}_{\text{BND}}^{\tau N \times \hat{n} N}$
$\boldsymbol{C}$
$\boldsymbol{Z} \leftarrow \boldsymbol{Y} + \boldsymbol{S}\boldsymbol{C}$
Abort if $\text{Rej}(\boldsymbol{Z}, \boldsymbol{S}\boldsymbol{C}, \sigma_{\text{BND}}) = 1$.
$\boldsymbol{Z}$
$1 : \quad \forall i, j : \|z_{i,j}\|_2 \overset{?}{\leq} B_{\text{BND}}$
$2 : \quad \boldsymbol{A}\boldsymbol{Z} \overset{?}{=} \boldsymbol{T}\boldsymbol{C} + \boldsymbol{W}$

Figure 4: Π_{BND} is an approximate amortized zero-knowledge proof of knowledge of bounded preimages for $\mathbb{Z}_q$-matrices.

Theorem 8. *The zero-knowledge proof of linear relations is* complete *if the randomness $\boldsymbol{r}_i$ is bounded by $B_{\texttt{Com}}$ in the ℓ_∞ norm, it is* 2-special sound *if the* $\text{SKS}^2_{n,k,4\sigma_{\text{C}}\sqrt{N}}$ *problem is hard, and it is statistical* honest-verifier zero-knowledge. *The probability of abort is* $1-(1/M)^{\hat{n}}$ *(where M is a parameter showing up in Rejection Sampling as defined in Equation 2).*

Even though the success probability is $(1/M)^{\hat{n}}$, we will only use this protocol for small values $\hat{n}$ 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 [29] to make the proof non-interactive, we let β be the output of a hash function applied to the first message and x. Then, the proof transcript is reduced to $\pi_L = (\beta, \{z_i\}_{i \in [\hat{n}]})$. To estimate the size of the transcript, we assume that due to the Gaussian distribution of $\boldsymbol{y}_i$ the size of $\boldsymbol{z}_i$ is within 6 standard deviations of the distribution. Using this, one obtains that each $\boldsymbol{z}_i$ is of size $kN \log_2(6\sigma_{\text{C}})$ bits. To improve the efficiency of the proof one can compress $\boldsymbol{z}_i$ to $\hat{n}(k-n)N \log_2(6\sigma_{\text{C}})$ 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 $\boldsymbol{A}$ be a publicly known $r \times v$-matrix over R_q, let $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_\tau$ be bounded elements in R_q^v and let $\boldsymbol{A}\boldsymbol{s}_i = \boldsymbol{t}_i$ for $i \in [\tau]$. Letting $\boldsymbol{S}$ be the matrix whose columns are $\boldsymbol{s}_i$ and $\boldsymbol{T}$ be the same matrix for $\boldsymbol{t}_i$, but defined over $\mathbb{Z}_q^N$ instead of R_q as in the previous subsection, then Baum et al. [8] give an efficient amortized zero-knowledge proof of knowledge for the relation

$$\mathcal{R}_{\text{BND}} = \left\{ (x, w) \;\middle|\; \begin{array}{c} x = (\boldsymbol{A}, \boldsymbol{T}) \wedge w = \boldsymbol{S} \quad \wedge \forall i \in [\tau] : \\ \boldsymbol{t}_i = \boldsymbol{A}\boldsymbol{s}_i \wedge ||s_{i,j}||_2 \leq 2 \cdot B_{\text{BND}} \end{array} \right\}.$$

The protocol Π_{BND} is depicted in Figure 4. We can use a challenge matrix $\boldsymbol{C}$ with entries sampled from the set $\mathcal{C}_{\text{BND}} = \{0, 1\}$, then this allows us to choose the parallel[6] protocol instances $\hat{n} \geq \kappa + 2$ for security parameter κ. Let

$$\pi_{\text{BND}} \leftarrow \Pi_{\text{BND}}(\boldsymbol{S}; (\boldsymbol{A}, \boldsymbol{T}, \sigma_{\text{BND}})), \text{ and}$$
$$0 \vee 1 \leftarrow \Pi_{\text{BNDV}}((\boldsymbol{A}, \boldsymbol{T}, B_{\text{BND}}); \pi_{\text{BND}}),$$

denote the run of the proof and verification protocols, respectively, where the Π_{BND}-protocol, using Fiat-Shamir, produces a proof of the form $\pi_{\text{BND}} = (\boldsymbol{C}, \boldsymbol{Z})$, where $\boldsymbol{C}$ is the output of a hash function, and the Π_{BNDV}-protocol consists of the two checks in the last step in Figure 4. $\mathcal{N}_{\sigma_{\text{BND}}}$ is a Gaussian distribution over $\mathbb{Z}$ with standard deviation σ_{BND}, and the verification bound is $B_{\text{BND}} = \sqrt{2N}\sigma_{\text{BND}}$. Note that σ_{BND}, and hence B_{BND}, depends on the norm of $\boldsymbol{S}$ (see Section A.2). This means that the bound we can prove for each $\|s_{i,j}\|_2$ depends on the number of equations τ in the statement.

The following theorem follows from Baum et al. [8, Lemma 3].

Theorem 9. *The amortized zero-knowledge proof of bounded openings is* complete *if the secrets in $\boldsymbol{S}$ are bounded by $B_{\texttt{Com}}$ in the ℓ_∞ norm, it is* special sound *if the* $\text{SKS}^2_{n,k,2kB_{\text{BND}}}$ *problem is hard and is statistical* honest-verifier zero-knowledge. *The probability of an abort is* $1 - 1/M$.

Each amortized proof is of size $v\hat{n}N \log_2(6\sigma_{\text{BND}})$ bits, excluding the challenge $\boldsymbol{C}$ which can be compressed into 2κ bits for a NIZK using Fiat-Shamir. Note that the protocol can be generalized to check for different norms in each row of $\boldsymbol{S}$ depending on varying norms of the secrets, as the protocol is entirely linear, see [10].

C FIGURE AND PROOFS FOR AMORTIZED ZKPOPK

The full protocol Π_{SMALL} is defined in Figure 5.

Perfect completeness of the protocol is straightforward, so we focus on soundness and special honest-verifier zero-knowledge.

C.1 Soundness

Lemma 2. *Let* $\texttt{Encode}$ *be the encoding of a Reed-Solomon code of dimension $g' = vN + \eta$ and length l. Furthermore, let $g' \leq g \leq l < q$. Suppose that there is an efficient deterministic prover $\mathcal{P}^*$ convincing an honest verifier in the protocol in Figure 5 on input $\boldsymbol{A}, \boldsymbol{t}_1, \ldots, \boldsymbol{t}_\tau$ with probability*

$$\epsilon > 2 \cdot \max \left\{ 2 \left(\frac{g}{l - \eta} \right)^\eta, \frac{1}{q - \tau} + \left(1 - \frac{g - g'}{6l} \right)^\eta, \right.$$
$$\left. 2 \cdot \left(1 - \frac{2(g - g')}{3l} \right)^\eta, \frac{18\tau}{q - \tau} \right\}.$$

Then there exists an efficient probabilistic extractor $\texttt{Ext}$ *which, given access to $\mathcal{P}^*$ either produces vectors $\boldsymbol{s}_i \in \{-1, 0, 1\}^{vN}$ such that $\boldsymbol{t}_i = \boldsymbol{A}\boldsymbol{s}_i$ for all $i \in [\tau]$, or breaks the binding property of the commitment scheme* $(\texttt{Com}_{\texttt{Aux}}, \texttt{Open}_{\texttt{Aux}})$*, or finds a hash collision in expected time at most $64T$ where*

$$T := \frac{3}{\epsilon} + \frac{g - \eta}{\epsilon/2 - (g/(l - \eta))^\eta}$$

and running $\mathcal{P}^*$ takes unit time.

Proof. First, we construct an extractor Ext as in [15] which does the following 8 times:

[6] This bound only applies to the interactive version of the proof. To apply the Fiat-Shamir transform, it has to be increased to at least 2κ

```
$\mathcal{P}$ has input $\{\boldsymbol{s}_i\}_{i=1}^{\tau}$; $(\boldsymbol{A}, \{\boldsymbol{t}_i\}_{i=1}^{\tau})$ while $\mathcal{V}$ has input $\boldsymbol{A}, \{\boldsymbol{t}_i\}_{i=1}^{\tau}$
$\mathcal{P}$ samples $\boldsymbol{s}_0 \xleftarrow{\$} \mathbb{Z}_q^{vN}$, $\boldsymbol{h} \xleftarrow{\$} \mathbb{Z}_q^{2vN}$, $\boldsymbol{r}_0 \xleftarrow{\$} \mathbb{Z}_q^{\eta}$, $\boldsymbol{r}_h \xleftarrow{\$} \mathbb{Z}_q^{\eta}$ and
    $\boldsymbol{r}_{i,j} \xleftarrow{\$} \mathbb{Z}_q^{\eta}$ for each $i \in [\tau], j \in \{0,1,2\}$.
$\mathcal{P}$ computes $\boldsymbol{f}(X) = \sum_{i=0}^{\tau} \boldsymbol{s}_i \ell_i(X)$ and $\boldsymbol{d} = -\boldsymbol{A}\boldsymbol{s}_0$ and defines $\boldsymbol{v}_{i,j}$ using
    $\frac{1}{\ell_0(X)} \cdot \prod_{h \in \{-1,0,1\}} \left[\boldsymbol{f}(X) - h \cdot \boldsymbol{1}\right] = \sum_{j=0}^{2} \sum_{i=1}^{\tau} \boldsymbol{v}_{i,j} \ell_i(X) \ell_0(X)^j$
$\mathcal{P}$ computes $\boldsymbol{H}_0 \leftarrow \mathsf{Encode}(\boldsymbol{s}_0, \boldsymbol{0}, \boldsymbol{r}_0)$, $\boldsymbol{H} \leftarrow \mathsf{Encode}(\boldsymbol{h}, \boldsymbol{r}_h)$ as well as
    $\boldsymbol{H}_{i,j} = \mathsf{Encode}(\delta_j \cdot \boldsymbol{s}_i, \boldsymbol{v}_{i,j}, \boldsymbol{r}_{i,j})$ for each $i \in [\tau], j \in \{0,1,2\}$.
$\mathcal{P}$ computes $\boldsymbol{E} = \mathtt{RowsToMatrix}(\boldsymbol{H}, \boldsymbol{H}_0, \boldsymbol{H}_{1,0}, \boldsymbol{H}_{1,1}, \ldots, \boldsymbol{H}_{\tau,2})$,
    $M = \mathtt{MerkleTree}(\mathtt{CommitToColums}(\boldsymbol{E}))$ and $(c_{\boldsymbol{d}}, r_{\boldsymbol{d}}) \leftarrow \mathsf{Com}_{\mathsf{Aux}}(\boldsymbol{d})$.
$\mathcal{P}$ sends $M, c_{\boldsymbol{d}}$ to $\mathcal{V}$
$\mathcal{V}$ samples $x \xleftarrow{\$} \mathbb{Z}_q^* \setminus \{a_1, \ldots, a_\tau\}$ and $\beta_0, \beta_{1,0}, \ldots, \beta_{\tau,2} \xleftarrow{\$} \mathbb{Z}_q^*$ and
    sends $x, \beta_0, \beta_{1,0}, \ldots, \beta_{\tau,2}$ to $\mathcal{P}$
$\mathcal{P}$ computes $\bar{\boldsymbol{f}} = \boldsymbol{f}(x)$, $\bar{\boldsymbol{r}}_f = \ell_0(x)\boldsymbol{r}_0 + \sum_{i=1}^{\tau} \sum_{j=0}^{2} \ell_i(x)\ell_0(x)^j \boldsymbol{r}_{i,j}$,
    $\bar{\boldsymbol{h}} = \boldsymbol{h} + \beta_0 \boldsymbol{s}_0 + \sum_{j=0}^{2} \sum_{i=1}^{\tau} \beta_{i,j}(\delta_j \cdot \boldsymbol{s}_i, \boldsymbol{v}_{i,j})$, $\bar{\boldsymbol{r}}_h = \boldsymbol{r}_h + \beta_0 \boldsymbol{r}_0 + \sum_{j=0}^{2} \sum_{i=1}^{\tau} \beta_{i,j} \boldsymbol{r}_{i,j}$
$\mathcal{P}$ sends $\bar{\boldsymbol{f}}, \bar{\boldsymbol{r}}_f, \bar{\boldsymbol{h}}, \bar{\boldsymbol{r}}_h, r_{\boldsymbol{d}}$ to $\mathcal{V}$
$\mathcal{V}$ samples $I \xleftarrow{\$} [l]^{\eta}$, $|I| = \eta$ and sends $I$ to $\mathcal{P}$
$\mathcal{P}$ computes and sends $\boldsymbol{E}|_I, \mathtt{MerklePaths}_I$ to $\mathcal{V}$
$\mathcal{V}$ runs $\mathtt{Verify}(\boldsymbol{E}|_I, M, \mathtt{MerklePaths}_I)$ and checks that
    $\mathsf{Open}_{\mathsf{Aux}}(c_{\boldsymbol{d}}, \frac{1}{\ell_0(x)} \cdot \left(\sum_{i=1}^{\tau} \boldsymbol{t}_i \ell_i(x) - \boldsymbol{A}\bar{\boldsymbol{f}}\right), r_{\boldsymbol{d}}) = 1$,
    $\mathsf{Encode}\left(\bar{\boldsymbol{f}}, \frac{1}{\ell_0(x)} \cdot \prod_{h \in \{-1,0,1\}} \left[\bar{\boldsymbol{f}} - h \cdot \boldsymbol{1}\right], \bar{\boldsymbol{r}}_f\right)\Big|_I$
        $\stackrel{?}{=} \ell_0(x) \cdot \boldsymbol{H}_0\big|_I + \sum_{i=1}^{\tau} \sum_{j=0}^{2} \ell_i(x)\ell_0(x)^j \boldsymbol{H}_{i,j}|_I$
    $\mathsf{Encode}\left(\bar{\boldsymbol{h}}, \bar{\boldsymbol{r}}_h\right)\Big|_I \stackrel{?}{=} \boldsymbol{H} + \beta_0 \boldsymbol{H}_0\big|_I + \sum_{i=1}^{\tau} \sum_{j=0}^{2} \beta_{i,j} \boldsymbol{H}_{i,j}\big|_I$
```

Figure 5: The protocol Π_{SMALL} is an exact amortized zero-knowledge proof of knowledge of ternary openings. δ_x is 1 if $x = 0$ and 0 otherwise. ($\mathsf{Com}_{\mathsf{Aux}}, \mathsf{Open}_{\mathsf{Aux}}$) is an arbitrary commitment scheme.

(1) First, run $\mathcal{P}^*$ on random challenges from $\mathcal{V}$ until an accepting transcript is found. Abort if none is found after $8/\epsilon$ steps.

(2) Let I_1 be the challenge set where $\mathcal{P}^*$ responded and let $\boldsymbol{E}|_{I_1}, \mathtt{MerklePaths}_{I_1}$ be the opened columns and Merkle tree paths. Fixing the other challenges, Ext adaptively reruns 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 Ext outputs the hash collision and terminates, otherwise it continues until it has collected a set J of at least g columns, or until $8\frac{g-\eta}{\epsilon/2-(g/(l-\eta))^{\eta}}$ time passed.

(3) Finally, Ext re-runs $\mathcal{P}^*$ $16/\epsilon$ times with completely fresh challenges, obtaining new $\boldsymbol{E}|_I, \mathtt{MerklePaths}_I$. If for some of these instances, the accepting transcript contains a hash collision in the Merkle tree (colliding with J) or $c_{\boldsymbol{d}}$ is opened with a different opening than in the first step, then output the hash collision or the respective two different openings of $c_{\boldsymbol{d}}$.

Using a standard heavy-row argument, [15] shows that Ext's total runtime is bounded by the term mentioned in the Lemma. Moreover, define S as the event that $\mathcal{P}^*$ outputs a valid proof in the last step and C be the event that the values $\mathcal{P}^*$ outputs to 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 [15] show that it must hold that $\Pr[S \wedge C] > \epsilon/2$. We now show that if we cannot use J to decode 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 Encode) when restricted to the indices of J. As $|J| = g$, C' has length g and minimum distance $d' = g - g' + 1$. For any $\boldsymbol{x} \in \mathbb{Z}_q^g$ we define the minimum distance of $\boldsymbol{x}$ to C' as $d'(C', \boldsymbol{x}) = \min_{\boldsymbol{c} \in C'} d(\boldsymbol{c}, \boldsymbol{x})$.

Let $\boldsymbol{E}^* := \boldsymbol{E}|_J$ be the matrix that was extracted by the extractor and let $\boldsymbol{d}^*$ be the opening message of the commitment. Assume that there exists $\boldsymbol{x} \in \mathbb{Z}_q^{3\tau+4}$ such that $d'(C', \boldsymbol{x}\boldsymbol{E}^*) \geq d'/3$. Then by Bootle et al. [14, Appendix B], any random linear combination of $\boldsymbol{E}^*$ (in particular, we compute and output such a combination in the proof as $\overline{\boldsymbol{h}}$) has distance $\geq d'/6$ from C' except with probability $1/(q-\tau)$. Similar to in [15] we can use this to deduce that in such a case, it must hold that

$$\epsilon/2 < \frac{1}{q-\tau} + \left(1 - \frac{g-g'}{6l}\right)^{\eta}$$

which contradicts the bound on ϵ in this lemma. Therefore, each row of $\boldsymbol{E}^*$ must be within $d'/3$ of C', meaning that it is efficiently decodable. Let $\boldsymbol{h}^*, \boldsymbol{s}_0^*, \boldsymbol{s}_{i,j}^*, \boldsymbol{v}_0^*, \boldsymbol{v}_{i,j}^*$ be the respective decoded values and $\boldsymbol{r}_h^*, \boldsymbol{r}_0^*, \boldsymbol{r}_{i,j}*$ be the randomness. We consider the composition of the aforementioned row values as $\boldsymbol{V}$ and the randomness used in the encoding as $\boldsymbol{R}$, By applying another result from Bootle et al. [14, Appendix B] we have that for any vector $\boldsymbol{y} \in \mathbb{Z}_q^{3\tau+4}$ it holds that $d'(\mathsf{Encode}_J(\boldsymbol{y}\boldsymbol{V}, \boldsymbol{y}\boldsymbol{R}), \boldsymbol{y}\boldsymbol{E}^*) < d'/3$. In other words, any linear transformation $\boldsymbol{y}$ when applied to the possibly noisy codewords $\boldsymbol{E}^*$ is within distance $d'/3$ of the codeword obtained from encoding $\boldsymbol{V}, \boldsymbol{R}$ after applying the same transformation $\boldsymbol{y}$. That means that $\overline{\boldsymbol{f}}$ is constructed from $\boldsymbol{s}_0^*, \boldsymbol{s}_{i,j}^*$ as we would expect.

Similar as in Bootle et al. [15, Corollary 3.7] one can show that if there are $\leq (\epsilon/4)(q-\tau)$ choices of x such that

$$\overline{\boldsymbol{f}} = \ell_0(x)\boldsymbol{s}_0^* + \sum_{i=1}^{\tau} \sum_{j=0}^{2} \ell_i(x)\ell_0(x)^j \boldsymbol{s}_{i,j}^* \tag{3}$$

$$\frac{1}{\ell_0(x)} \cdot \bar{\boldsymbol{f}} \circ \left[\bar{\boldsymbol{f}} - \boldsymbol{1}\right] \circ \left[\bar{\boldsymbol{f}} + \boldsymbol{1}\right] \tag{4}$$

$$= \ell_0(x)\boldsymbol{v}_0^* + \sum_{i=1}^{\tau} \sum_{j=0}^{2} \ell_i(x)\ell_0(x)^j \boldsymbol{v}_{i,j}^* \tag{5}$$

then $\epsilon < 4(1 - \frac{2(g-g')}{3l})^{\eta}$, contradicting the bound on ϵ in the lemma. By multiplying 4 with $\ell_0(X)$ we obtain the equation

$$\bar{\boldsymbol{f}} \circ (\bar{\boldsymbol{f}} - \boldsymbol{1}) \circ (\bar{\boldsymbol{f}} + \boldsymbol{1}) - \ell_0(X)^2 \boldsymbol{v}_0^* - \sum_{i=1}^{\tau} \sum_{j=0}^{2} \ell_i(X)\ell_0(X)^{j+1} \boldsymbol{v}_{i,j}^*, \tag{6}$$

which equals $\mathbf{0}$. Replacing $\bar{\boldsymbol{f}}$ 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 $\epsilon > 36\tau/(q-\tau)$, 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 $\boldsymbol{s}^*_{i,0}$ is in $\{-1,0,1\}^{vN}$.

Additionally, we have that

$$\boldsymbol{d}^* = \frac{1}{\ell_0(X)} \cdot \left(\sum_{i=1}^{\tau} \boldsymbol{t}_i \ell_i(X) - \boldsymbol{A}\bar{\boldsymbol{f}} \right)$$

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

$$-\boldsymbol{A}\boldsymbol{s}^*_0 + \frac{1}{\ell_0(X)} \cdot \left(\sum_{i=1}^{\tau} \boldsymbol{t}_i \ell_i(X) - \boldsymbol{A} \left[\sum_{i=1}^{\tau} \sum_{j=0}^{2} \ell_i(x)\ell_0(x)^j \boldsymbol{s}^*_{i,j} \right] \right)$$

Since $\boldsymbol{d}^*$ has been committed to before x is chosen, it must be that $\boldsymbol{d}^*$ is the constant of the polynomial on the right, so we have that $\boldsymbol{d}^* = -\boldsymbol{A}\boldsymbol{s}^*_0$ and therefore

$$\sum_{i=1}^{\tau} \sum_{j=0}^{2} \ell_i(x)\ell_0(x)^j \boldsymbol{A}\boldsymbol{s}^*_{i,j} = \sum_{i=1}^{\tau} \boldsymbol{t}_i \ell_i(X).$$

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

$$\sum_{i=1}^{\tau} \ell_i(x) \boldsymbol{A}\boldsymbol{s}^*_{i,0} = \sum_{i=1}^{\tau} \boldsymbol{t}_i \ell_i(X)$$

and by the independence of the $\ell_i(X)$ modulo $\ell_0(X)$ we have that $\boldsymbol{t}_i = \boldsymbol{A}\boldsymbol{s}^*_{i,0}$ for all $i \in [\tau]$, which proves the claim. □

C.2 Zero-Knowledge

Lemma 3. *There exists an efficient simulator* Sim *which, given* $x, \beta_0, \beta_{1,0}, \ldots, \beta_{\tau,2}, I$ *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.*

Proving Honest-Verifier Zero-Knowledge is sufficient for our application, as we will use the Fiat-Shamir transform to generate the challenges in Π_{Small}.

Proof. Towards constructing a simulation, observe that

(1) M and its opened paths do not reveal any information about the unopened columns as the commitment scheme used in creating M is hiding.
(2) $\bar{\boldsymbol{f}}, \bar{\boldsymbol{r}}_f, \bar{\boldsymbol{h}}, \bar{\boldsymbol{r}}_h$ are uniformly random due to the uniform choice of $\boldsymbol{s}_0, \boldsymbol{r}_0, \boldsymbol{h}, \boldsymbol{r}_h$.
(3) Each encoded row of $\boldsymbol{E}$ uses η bits of randomness, so revealing η columns does not leak any information about the message being committed in the respective row.

Thus, for the proof we let Sim choose uniformly random $\bar{\boldsymbol{f}}, \bar{\boldsymbol{r}}_f, \bar{\boldsymbol{h}}, \bar{\boldsymbol{r}}_h$. This allows the prover also to compute $\boldsymbol{d}$ consistently as $\frac{1}{\ell_0(x)} \cdot (\sum_{i=1}^{\tau} \boldsymbol{t}_i \ell_i(x) - \boldsymbol{A}\bar{\boldsymbol{f}})$, which has the same uniform distribution as in the real protocol and thereby fix $c_{\boldsymbol{d}}$. Next, we let Sim choose all but the first two rows of $\boldsymbol{E}|_I$ uniformly at random. The second row will be computed according to x thus fulfilling the check on the encoding of $\bar{\boldsymbol{f}}, \bar{\boldsymbol{r}}_f$, while the first row is computed according to $\beta_0, \beta_{i,j}$ for the encoding of $\bar{\boldsymbol{h}}, \bar{\boldsymbol{r}}_h$. Sim now fixes the remaining columns of $\boldsymbol{E}$ as $\mathbf{0}$ and commits honestly to these as in the protocol. □

D BGV MIXING SECURITY

First, we define *completeness*, *soundness* and *simulatability* for a mixing protocol Π_{Mix} executed by a prover $\mathcal{P}$, with respect to a generic encryption scheme $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$.

Definition 15 (Mixing Completeness). *We say that the mixing protocol* Π_{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* $1 - \epsilon(\kappa)$*:*

$$\Pr\left[\begin{matrix} \{m_i\}_{i\in[\tau]} = \mathsf{Dec}(\mathsf{sk}, \{\hat{c}_i\}_{i\in[\tau]} \\ 1 \leftarrow \mathcal{V}(\mathsf{pp}, \mathsf{pk}, \{c_i\}_{i\in[\tau]}, \{\hat{c}_i\}_{i\in[\tau]}, \pi) \end{matrix} : \begin{matrix} (\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\kappa) \\ \{c_i\}_{i\in[\tau]} \leftarrow \mathsf{Enc}(\mathsf{pk}, \{m_i\}_{i\in[\tau]}) \\ (\{\hat{c}_i\}_{i\in[\tau]}, \pi) \leftarrow \mathcal{P}(\mathsf{pp}, \mathsf{pk}, \{c_i\}_{i\in[\tau]}) \end{matrix} \right],$$

where the probability is taken over KGen, Enc *and* $\mathcal{P}$.

Definition 16 (Mixing Soundness). *We say that the mixing protocol* Π_{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* $\epsilon(\kappa)$*:*

$$\Pr\left[\begin{matrix} \{m_i\}_{i\in[\tau]} \neq \mathsf{Dec}(\mathsf{sk}, \{\hat{c}_i\}_{i\in[\tau]} \\ 1 \leftarrow \mathcal{V}(\mathsf{pp}, \mathsf{pk}, \{c_i\}_{i\in[\tau]}, \{\hat{c}_i\}_{i\in[\tau]}, \pi) \end{matrix} : \begin{matrix} (\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\kappa) \\ \{c_i\}_{i\in[\tau]} \leftarrow \mathsf{Enc}(\mathsf{pk}, \{m_i\}_{i\in[\tau]}) \\ (\{\hat{c}_i\}_{i\in[\tau]}, \pi) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk}, \{c_i\}_{i\in[\tau]}) \end{matrix} \right],$$

where the probability is taken over KGen, Enc *and* $\mathcal{A}$.

Definition 17 (Mixing Simulatability). *We say that the mixing protocol* Π_{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*

$$|\Pr\left[b = b' : \begin{matrix} (\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\kappa); b \overset{\$}{\leftarrow} \{0,1\} \\ \{c_i\}_{i\in[\tau]} \leftarrow \mathsf{Enc}(\mathsf{pk}, \{m_i\}_{i\in[\tau]}) \\ (\{\hat{c}_i\}_{i\in[\tau]}, \pi^{(0)}) \leftarrow \mathcal{P}(\mathsf{pp}, \mathsf{pk}, \{c_i\}_{i\in[\tau]}) \\ (\pi^{(1)}) \leftarrow \mathcal{S}(\mathsf{pp}, \mathsf{pk}, \{c_i\}_{i\in[\tau]}, \{\hat{c}_i\}_{i\in[\tau]}) \\ b' \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk}, \{c_i\}_{i\in[\tau]}, \{\hat{c}_i\}_{i\in[\tau]}, \pi^{(b)}) \end{matrix} \right] - \frac{1}{2}| \leq \epsilon(\kappa),$$

where the probability is taken over KGen, Enc, $\mathcal{P}$, $\mathcal{S}$ *and* $\mathcal{A}$.

Next, we state the security of Π_{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* Π_{Small} *and* $\Pi^{l_c}_{Shuf}$ *are complete, then* Π_{Mix} *always terminates. Moreover, if the input ciphertexts* $\mathbf{c}_i$ *have noise bounded by* B_{Dec}*, and the total noise added in* Π_{Mix} *is bounded by* B_{Mix} *such that* $(B_{\mathsf{Dec}} + B_{\mathsf{Mix}}) < \lfloor q/2 \rfloor$*, then the output ciphertexts* $\hat{\mathbf{c}}_i$ *decrypt to the same set of messages as* $\mathbf{c}_i$*.*

We sketch the argument. Since Π_{Small} and $\Pi^{l_c}_{\text{Shuf}}$ are complete, the protocol instance will finish and any verifier will accept the mixing outputs. Since $(B_{\mathsf{Mix}} + B_{\mathsf{Dec}}) < \lfloor q/2 \rfloor$, it follows that decryption is correct. We conclude that Π_{Mix} is complete.

Theorem 11 (Knowledge Soundness). *Let* Ext_1 *be a knowledge extractor for the protocol* Π_{SMALL} *with success probability* ϵ_1 *and let* Ext_2 *be a knowledge extractor for the protocol* $\Pi^{l_c}_{SHUF}$ *with success probability* ϵ_2*. Then we can construct a knowledge extractor* Ext_0 *that succeeds with probability* $\epsilon_0 \leq \epsilon_1 \cdot \epsilon_2$ *at extracting a witness for* $\mathcal{R}_{MIX}$ *for the outputs of* Π_{MIX}*. The runtime of* Ext_0 *is essentially the product of that of* Ext_1 *and* Ext_2*.*

We sketch the argument. The main observation is that it is necessary that both extractors Ext_1 and Ext_2 succeed.

If the extractor Ext_1 succeeds, we are able to extract τ randomness vectors $\boldsymbol{r}'_i$ bounded by B_{Mix}, which gives us the randomness for both the commitments and ciphertexts used in the protocol. However, if the adversary is able to cheat in $\Pi^{l_c}_{\mathrm{SHUF}}$ then the output ciphertexts will be different than the ciphertexts we extract from Π_{SMALL}, and hence, we have not yet extracted a witness for Π_{MIX}.

If the extractor Ext_2 succeeds, we are able to extract both the permutation π and τ randomness vectors $\boldsymbol{r}_i$ used in the commitments. However, if the adversary is able to cheat in Π_{SMALL} then the output ciphertexts might have more noise than $(B_{\mathsf{Dec}} + B_{\mathsf{Mix}})$ and lead to decryption failures, and hence, we have not yet extracted a witness for Π_{MIX}.

We conclude that it is both necessary and sufficient that both extractors succeed at the same time to extract witnesses with respect to the same set of output ciphertexts and proofs to extract both the randomness used to encrypt, the randomness used to commit, and the permutation used to shuffle, and hence, to extract a witness for the relation $\mathcal{R}_{\mathrm{MIX}}$.

Theorem 12 (Simulatability). *Suppose the protocol* Π_{SMALL} *and* $\Pi^{l_c}_{SHUF}$ *are honest-verifier zero-knowledge, that* $\mathtt{Com}$ *is hiding and that* Enc *is CPA secure. Then there exists a simulator for* Π_{MIX} *such that for any distinguisher* Adv_0 *for this simulator with advantage* ϵ_0*, there exists an adversary* Adv_3 *against hiding of the commitment scheme with advantage* ϵ_3*, an adversary* Adv_4 *against CPA security of the encryption scheme with advantage* ϵ_4*, and distinguishers* $\mathsf{Adv}_1, \mathsf{Adv}_2$ *for the simulators of* Π_{SMALL} *and* $\Pi^{l_c}_{SHUF}$*, respectively, with advantages* ϵ_1, ϵ_2*, such that* $\epsilon_0 \leq \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4$*. The runtime of* Adv_1*,* Adv_2*,* $\mathsf{Adv}_3, \mathsf{Adv}_4$ *are essentially the same as of* Adv_0*.*

We sketch the argument. The simulator is given a set of input ciphertexts and a set of output ciphertexts from an honest shuffle. The simulator simulates the zero-knowledge proofs Π_{SMALL} and $\Pi^{l_c}_{\mathrm{SHUF}}$ using the appropriate simulators. It replaces the commitments to the ciphertexts with commitments to zero and the shuffled output ciphertexts by the given ciphertexts to the correct messages.

The claim about the simulator follows from a hybrid argument. We begin with the verifiable shuffle protocol.

First, we replace the Π_{SHUF} arguments with simulated arguments, which gives us a distinguisher Adv_2 for the Π_{SHUF} honest verifier simulator. Second, we replace the Π_{SMALL} arguments by simulated arguments, which gives us a distinguisher Adv_1 for Π_{SMALL}. Third, we replace the commitments to ciphertexts with commitments to zero, which gives us an adversary 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 Adv_4 against CPA security.

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

E PROOF OF KNOWLEDGE SOUNDNESS OF EXTENDED SHUFFLE

The first observation that we make is that the protocol $\Pi^{l_c}_{\mathrm{SHUF}}$ does not increase the round complexity in comparison to Π_{SHUF}. Indeed, the first message in Π_{SHUF} is also a random element from the verifier to the prover, and all additional operations that both parties do in $\Pi^{l_c}_{\mathrm{SHUF}}$ after sending h are local computation only.

To prove that $\Pi^{l_c}_{\mathrm{SHUF}}$ is a ZK Proof of Knowledge, we rely on techniques recently developed by [6]. In the process, we will simplify the original proof from [4]. Using the formalism from [6] then allows us to apply known techniques to transform the interactive proof into a NIZK without incurring a huge loss due to the multi-round nature of $\Pi^{l_c}_{\mathrm{SHUF}}$.

We first introduce the necessary notation from [6]:

Definition 18 (Monotone Structure). *Let* C *be a non-empty finite set and let* $\Gamma \subset 2^C$ *be a family of subsets of* C*. Then,* (Γ, C) *is said to be a monotone structure if it is closed under taking supersets, i.e. if* $S \in \Gamma$ *and* $S \subset T \subseteq C$ *implies* $T \in \Gamma$*.*

Such monotone structures later allow us to model the challenge spaces of the different rounds of the proofs appropriately. We now also recap the notion of distance to a monotone structure, i.e. the minimal number of elements by which one has to extend a set S before it is in a monotone access structure Γ.

Definition 19 (Distance to a Monotone Structure). *For a non-empty monotone structure* (Γ, C)*, we define the following distance function*

$$d_\Gamma : 2^C \to \mathcal{N}_0, \qquad S \mapsto \min_{T \in \Gamma} |T \setminus S|.$$

If $\Gamma = \emptyset$ *then we define* d_Γ *to be identically equal to* ∞*.*

The authors of [6] provide the following definition of special soundness, adapted to general access structures. It's a natural generalization of k-special soundness, although now the challenges of the collected transcripts with identical first message have to be a permitted element in the monotone access structure.

Definition 20 (Γ-out-of-C special-soundness). *Let* (Γ, C) *be a monotone structure. A* 3*-round public coin interactive proof* $\mathcal{P}, \mathcal{V}$ *for relation* R*, with challenge set* C*, is* Γ*-out-of-*C *special-sound if there exists an algorithm that, on input a statement* x *and a set of accepting transcripts* $(a, c_1, z_1), \ldots, (a, c_k, z_k)$ *with common first message* a *such that* $\{c_1, \ldots, c_k\} \in \Gamma$*, runs in time polynomial in* $|x|$ *and outputs a witness* $w \in R(x)$*. We also say that* $\mathcal{P}, \mathcal{V}$ *is* Γ*-special sound.*

For the soundness proof, it is important that the reduction can find accepting transcripts so that the challenges form an element from Γ. For this, the authors introduce the notion of *Useful Elements*. These are elements which the reduction, given existing challenges, can try out while getting closer to an element from Γ, which allows to give a bound on the runtime of the extractor.

Definition 21 (Useful Elements). *For a monotone structure* (Γ, C)*, define the following function:*

$$U_\Gamma : 2^C \to 2^C, \qquad S \mapsto \{c \in C \setminus S \mid \exists A \in \Gamma . S \subset A \wedge A \setminus \{c\} \notin \Gamma\}.$$

When the extractor uses useful elements as challenges, it is important how often it has to repeat this process of choosing useful elements before ending up with an element from Γ. This is denoted by the so-called t-value of a set S. This can be thought of a generalization of the threshold k for normal k-special soundness.

Definition 22 (t-value). *Let (Γ, C) be a monotone structure and $S \subseteq C$. Then*

$$t_\Gamma(S) := \max \left\{ t \in \mathbb{N}_0 \mid \begin{matrix} \exists c_1, \ldots, c_t \in C. \\ \forall i : \; c_i \in U_\Gamma(S \cup \{c_1, \ldots, c_{i-1}\}) \end{matrix} \right\}$$

The authors of [6] then show that Γ-special sound protocols have an efficient knowledge extractor.

Lemma 4 ([6], Theorem 1.). *Let $\mathcal{P}, \mathcal{V}$ be a Γ-out-of-C special-sound Σ-protocol such that t_Γ is polynomial in the size $|x|$ of the public input statement x of $\mathcal{P}, \mathcal{V}$ and sampling from $U_\Gamma(S)$ takes polynomial time in $|x|$ for all S with $|S| < t_\Gamma$. Then $\mathcal{P}, \mathcal{V}$ is knowledge sound with knowledge error $\kappa_\Gamma = \max_{S \notin \Gamma} |S|/|C|$.*

The protocol $\Pi^{l_c}_{\text{SHUF}}$, whose security we will analyze, requires a generalization of the aforementioned definitions to multiple rounds of challenges.

The first definition considers a tree of challenges, which means that a collection of challenge vectors can be classified by how their challenges at each index form an element from a monotone access structure.

Definition 23 (Tree of Challenges). *Let (Γ_i, C_i) be monotone structures for $1 \leq i \leq \mu$. A set containing a single challenge vector $(c_1, \ldots, c_\mu) \in C_1 \times \cdots \times C_\mu$ is also referred to as a $(1, \ldots, 1)$-tree of challenges. Further, for $1 \leq t \leq \mu$, a $(1, ;1, \Gamma_t, \ldots, \Gamma_\mu)$-tree T_t of challenges is the union of several $(1, \ldots, 1, \Gamma_{t+1}, \ldots, \Gamma_\mu)$-trees such that*

- *The first $t-1$ coordinates of all $\mathbf{c} \in T_t \subseteq C_1 \times \cdots \times C_\mu$ are equal;*
- *The t-th coordinate of the tree elements form an element in Γ_t i.e.*

  $$\{c \in C_t \mid \exists (c_1, \ldots, c_{t-1}, c, c_{t+1}, \ldots, c_\mu) \in T_t\} \in \Gamma_t.$$

This naturally generalizes to also including prover messages, where same messages must have been sent by prover in previous rounds. This is necessary for extraction and the generalization of fixed first messages for 3-round special sound protocols.

Definition 24 (Tree of Transcripts). *Let (Γ_i, C_i) be monotone structures for $1 \leq i \leq \mu$. Let $\mathcal{P}, \mathcal{V}$ be a $2\mu + 1$-round public-coin interactive proof with challenge sets $C_1, \ldots, C_\mu$. A $(\Gamma_1, \ldots, \Gamma_\mu)$-tree of transcripts is a set of protocol transcripts such that*

- *The corresponding set of challenge vectors, obtained by ignoring the prover's messages, is a $(\Gamma_1, \ldots, \Gamma_\mu)$-tree of challenges;*
- *The prover's messages collide, i.e. if two transcripts $(a_0, c_1, a_1, \ldots, c_\mu, a_\mu)$ and $(a'_0, c'_1, a'_1, \ldots, c'_\mu, a'_\mu)$ are both in the tree, and $c_i = c'_i$ for all $i \leq j$, then also $a_i = a'_i$ for all $i \leq j$.*

Given these definitions, one can naturally extend Definition 20 to the setting with multiple challenge rounds.

Definition 25 ($(\Gamma_1, \ldots, \Gamma_\mu)$-out-of-$(C_1, \ldots, C_\mu)$ Special-Soundness). *Let (Γ_i, C_i) be monotone structures for $1 \leq i \leq \mu$. A $2\mu + 1$-round public-coin interactive proof $\mathcal{P}, \mathcal{V}$ for relation R, with challenge sets $C_1, \ldots, C_\mu$, is $(\Gamma_1, \ldots, \Gamma_\mu)$-out-of-$(C_1, \ldots, C_\mu)$ special-sound if there exists a polynomial time algorithm that, on input a statement x and a $(\Gamma_1, \ldots, \Gamma_\mu)$-tree of accepting transcripts, outputs a witness $w \in R(x)$. We also say that $\mathcal{P}, \mathcal{V}$ is $(\Gamma_1, \ldots, \Gamma_\mu)$-special sound.*

For this, the authors show the following analogue to Lemma 4:

Lemma 5 ([6], Theorem 2.). *Let $\mathcal{P}, \mathcal{V}$ be a $(\Gamma_1, \ldots, \Gamma_\mu)$-out-of-$(C_1, \ldots, C_\mu)$ special-sound interactive proof such that $T_\Gamma = \prod_{i=1}^{\mu} t_{\Gamma_i}$ is polynomial in the size $|x|$ of the public statement x of $\mathcal{P}, \mathcal{V}$ and sampling from $U_{\Gamma_i}(S_i)$ takes polynomial time in $|x|$ for all $1 \leq i \leq \mu$ and $S_i \subset C_i$ with $|S_i| < t_{\Gamma_i}$. Then $\mathcal{P}, \mathcal{V}$ is knowledge sound with knowledge error*

$$\kappa_\Gamma = 1 - \prod_{i=1}^{\mu} \left(1 - \max_{S_i \notin \Gamma_i} \frac{|S_i|}{|C_i|}\right).$$

$\Pi^{l_c}_{\text{SHUF}}$ is at first glance not a $2\mu + 1$-round protocol since the verifier sends the first message, in our version. This can easily be changed by letting the prover send the commitments from the first-round messages of the proofs of linearity before the verifier sends its challenges. In the following, we assume that this is the case, and that the verifier additionally sends the first message of $\Pi^{l_c}_{\text{SHUF}}$ and Π_{SHUF} together as argued before.

Let us quickly recap the full protocol $\Pi^{l_c}_{\text{SHUF}}$:

(1) Initially, prover $\mathcal{P}$ and verifier $\mathcal{V}$ hold $\{[\![\boldsymbol{m}_i]\!], \hat{\boldsymbol{m}}_i\}_{i \in [\tau]}$ for a public key $\text{pk} = (\boldsymbol{A}_{\text{C},1}, \boldsymbol{A}_{\text{C},2})$ while the prover additionally hold secrets $\{\boldsymbol{m}_i, \boldsymbol{r}_i\}_{i \in [\tau]}, \pi \in S_\tau$.

(2) The samples $h \xleftarrow{\$} R_q$ and sets $\boldsymbol{\alpha} \leftarrow (1, h, \ldots, h^{l_c - 1})^\top$. It then samples $\rho \xleftarrow{\$} R_q$ subject to the constraint that all $\langle \boldsymbol{\alpha}, \hat{\boldsymbol{m}}_i \rangle - \rho$ are invertible and sends h, ρ to the prover.

(3) The prover also computes $\boldsymbol{\alpha} \leftarrow (1, h, \ldots, h^{l_c - 1})^\top$ and checks that all $\langle \boldsymbol{\alpha}, \hat{\boldsymbol{m}}_i \rangle - \rho$ are invertible. Otherwise, it aborts.

(4) $\mathcal{P}$ and $\mathcal{V}$ for each $[\![\boldsymbol{m}_i]\!] = (\boldsymbol{c}_{1,i}, \boldsymbol{c}_{2,i})$ compute $[\![\langle \boldsymbol{\alpha}, \boldsymbol{m}_i \rangle]\!] = (\boldsymbol{c}_{1,i}, \langle \boldsymbol{\alpha}, \boldsymbol{c}_{2,i} \rangle) = (\boldsymbol{c}_{1,i}, \boldsymbol{c}'_{2,i})$.

(5) The parties for $i \in [\tau]$ set $[\![m_i]\!] := [\![\langle \boldsymbol{\alpha}, \boldsymbol{m}_i \rangle]\!]$ and messages $\hat{m}_i := \langle \boldsymbol{\alpha}, \hat{\boldsymbol{m}}_i \rangle$. They use the commitment key $\text{pk}' = (\boldsymbol{A}'_{\text{C},1}, \boldsymbol{A}'_{\text{C},2})$ for the rest of the protocol.

(6) Both parties set $\hat{M}_i = \hat{m}_i - \rho$. The prover additionally sets $M_i = m_i - \rho$ while the verifier sets $[\![M_i]\!] = [m_i] - \rho$

(7) The prover samples $\theta_i \xleftarrow{\$} R_q, \forall i \in [\tau - 1]$ and computes $[\![D_i]\!]$ such that $[\![D_1]\!] = [\![\theta_1 \hat{M}_1]\!]$, $[\![D_\tau]\!] = [\![\theta_{\tau-1} M_\tau]\!]$ and $[\![D_i]\!] = [\![\theta_{i-1} M_i + \theta_i \hat{M}_i]\!]$ for $i \in [\tau - 1] \setminus \{1\}$

(8) The prover sends $\{[\![D_i]\!]\}_{i=1}^{\tau}$ to the verifier.

(9) The verifier chooses $\beta \xleftarrow{\$} R_q$ and sends it to the prover.

(10) The prover computes $s_i, \forall i \in [\tau - 1]$ as $s_i = (-1)^i \cdot \beta \prod_{j=1}^{i} \frac{M_j}{\hat{M}_j} +$ θ_i and sends $\{s_i\}_{i=1}^{\tau-1}$ to the verifier

(11) Prover and Verifier use a 2-special sound proof Π_{Lin} to prove that
- $\beta [\![M_1]\!] + s_1 \hat{M}_1 = [\![D_1]\!]$
- $\forall i \in [\tau - 1] \setminus \{1\} : \; s_{i-1} [\![M_i]\!] + s_i \hat{M}_i = [\![D_i]\!]$
- $s_{\tau-1} [\![M_\tau]\!] + (-1)^\tau \beta \hat{M}_\tau = [\![D_\tau]\!]$

The verifier accepts iff all instances of Π_{Lin} are accepting, otherwise it rejects.

For the soundness argument, we will use the following simple Lemma:

LEMMA 6. *Let R_q be a finite ring as defined before and $F_1 \times \cdots \times F_\delta$ be its decomposition into fields. Moreover, let $\Phi : R_q \longrightarrow F_1 \times \cdots \times F_\delta$ be the unique isomorphism computing the CRT decomposition of every R_q element and let $\Phi|_1$ be its restriction to the projection into F_1. Let $g \in R_q[X]$ be a polynomial of degree $r > 0$.*

Then for any set $\{t_1, \dots, t_{r+1}\} = T \subset R_q$ such that $\Phi_1(t_i) \neq \Phi(t_j)$ if $i \neq j$, there is at lest one element $t \in T$ such that $g(t) \neq 0$ over R_q.

PROOF. Since F_1 is a field, $g(\cdot)$ has at most r roots in F_1 as it is not the 0-polynomial by assumption. By the Chinese Remainder Theorem, $g(s) = 0$ over R_q iff $g(s) = 0$ in each F_i simultaneously, in particular F_1. Since T contains more than r elements which are all different in their F_1-component, one of them must not be a root of g in F_1. But then, since Φ is an isomorphism, this element also cannot be a root of g in R_q. □

We moreover use another simple lemma to give a non-tight lower bound on the challenge space for ρ. It shows that by picking ρ at random, the chance that it makes all differences invertible is at least $1/2$.

LEMMA 7. *Let R_q be a ring such that it factors into N fields $F_1, \dots, F_N$. Moreover, let $x_1, \dots, x_\tau \in R_q$ be arbitrarily chosen. Let T define the set of elements ρ such that all $x_i - \rho$ are simultaneously invertible over R_q. If $\tau \leq \frac{1}{2-2\exp(-N/q)}$ then $|T| \geq |R_q|/2$.*

Observe that the extra condition on τ is easily met for our parameter choices.

PROOF. The proof is similar to Aranha et al. [4, Proposition 2]. The ring R_q contains q^N elements, out of which $(q-1)^N$ are invertible by a simple counting argument via the CRT decomposition of R_q into fields. Let S_1 be the set of elements ρ such that $x_1 - \rho$ is not invertible. Then $|S_1| = q^N - (q-1)^N$. Define $S_2, \dots, S_\tau$ accordingly, then $|S_1 \cup \cdots \cup S_\tau| \leq \tau q^N - \tau(q-1)^N$. This means that the set of ρ values which makes all differences invertible can be written as

$$q^N - \tau q^N(1 - (1 - 1/q)^N) \geq q^N - \tau q^N(1 - \exp(-N/q))$$

using $(1 + x) \leq e^x$. We ask when this has size $q^N/2$. For this, let

$$q^N - \tau q^N(1 - \exp(-N/q)) = q^N/2 \to \tau q^N(1 - \exp(-N/q)) = q^N/2$$

We can divide this by q^N and solve for τ, which gives the desired bound. □

We now show that $\Pi^{l_c}_{\text{SHUF}}$ fulfills the generalized special soundness definition as given in Definition 25. The protocol has $\mu = 3$ challenge rounds:

(1) First, the verifier sends challenges h, ρ.
(2) Next, it sends a challenge β.
(3) Finally, it sends all the challenges in each instance of Π_{Lin} in parallel.

We first define the challenge spaces and minimal sets for soundness as required by Definition 25.

- Let $C_1 \subseteq R_q \times R_q$ be such that if $(h, \rho) \in C_1$ then the respective M_i are all invertible. By Lemma 7 we have that there are always at least $q^N/2$ choices of ρ. We then define Γ_1 as sets of tuples (h, ρ) from C_1 such that there are $\tau \cdot (l_c - 1) + 1$ different choices of $\Phi_1(h)$ and $\tau + 1$ choices of $\Phi_1(\rho)$ in each subset of Γ_1, as defined in Lemma 6. Hence, each set in Γ_1 is of size at most $(\tau \cdot (l_c - 1) + 1) \cdot (\tau + 1)$. Sampling from Γ_1 is simple by always extending an input set S by a new random R_q-element, subject to its projection along Φ_1 being different from $\Phi_1(S)$.
- Let C_2 consist of all elements from R_q, while Γ_2 consists of all pairs of different values from C_2.
- Let C_3 consist of the τ cross-product of the challenge space of Π_{Lin}, which is C. Γ_3 consists of sets $\gamma_0, \dots, \gamma_\tau$ of size $\tau + 1$ where γ_i differs from γ_{i-1} only in position i. Observe that given any set $S \in C_3$, one can efficiently sample from its useful set by picking a coordinate that doesn't appear in flipped form in S and changing it to a random challenge from $|C|$.

PROOF OF 1. The Completeness and Zero-Knowledge property trivially follow from the same argument as in [4].

Assume that we have a $(\Gamma_1, \Gamma_2, \Gamma_3)$-tree of accepting transcripts.

Consider the two challenges β, β' that we obtained, then we also obtained messages s_i, s'_i from the prover. By the definition of Γ_3, we can use the 2-special soundness of Π_{Lin} we obtain:

(1) $\beta M_1 + s_1 \hat{M}_1 = D_1$ and $\beta' M_1 + s'_1 \hat{M}_1 = D_1$
(2) $\forall i \in [\tau - 1] \setminus \{1\} : \; s_{i-1} M_i + s_i \hat{M}_i = D_i$ and $s'_{i-1} M_i + s'_i \hat{M}_i = D_i$
(3) $s_{\tau-1} M_\tau + (-1)^\tau \beta \hat{M}_\tau = D_\tau$ and $s'_{\tau-1} M_\tau + (-1)^\tau \beta' \hat{M}_\tau = D_\tau$

Subtracting the equations with identical D_i on the right-hand side yields the following system of τ linear equations:

$$\underbrace{\begin{bmatrix} M_1 & \hat{M}_1 & 0 & \dots & 0 & 0 \\ 0 & M_2 & \hat{M}_2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & M_{\tau-1} & \hat{M}_{\tau-1} \\ (-1)^\tau \hat{M}_\tau & 0 & 0 & \dots & 0 & M_\tau \end{bmatrix}}_{\boldsymbol{M}} \underbrace{\begin{bmatrix} \beta - \beta' \\ s_1 - s'_1 \\ \vdots \\ s_{\tau-2} - s'_{\tau-2} \\ s_{\tau-1} - s'_{\tau-1} \end{bmatrix}}_{\boldsymbol{c}} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \tag{7}$$

Since $\beta \neq \beta'$ by assumption, we have that $\boldsymbol{c}$ is non-zero. By a standard rule from linear algebra, we know that if $\det(\boldsymbol{M}) \neq 0$ then $\boldsymbol{0}$ is the only element in the kernel of $\boldsymbol{M}$, while $\boldsymbol{c} \neq \boldsymbol{0}$. Therefore, $\det(\boldsymbol{M}) = 0$. We explicitly compute $\det(\boldsymbol{M})$ using the Laplace expansion obtained by removing the first column (and then all rows successively). As most of the cofactors are 0, we obtain $0 = \det(\boldsymbol{M}) = M_1 \cdot \det(\boldsymbol{M}_1) + (-1)^{2\tau+1} \cdot \det(\boldsymbol{M}_2)$ where

$$\boldsymbol{M}_1 = \begin{bmatrix} M_2 & \hat{M}_2 & \dots & 0 & 0 \\ 0 & M_3 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & M_{\tau-1} & \hat{M}_{\tau-1} \\ 0 & 0 & \dots & 0 & M_\tau \end{bmatrix}$$

$$\boldsymbol{M}_2 = \begin{bmatrix} \hat{M}_1 & 0 & \dots & 0 & 0 \\ M_2 & \hat{M}_2 & \dots & 0 & 0 \\ 0 & M_3 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & M_{\tau-1} & \hat{M}_{\tau-1} \end{bmatrix}.$$

N	p	q	sec	ξ_1	ξ_2	n	k
4096	2	$\approx 2^{78}$	40	4	4	1	$l_c + 2$

ν	B_{Com}	$\hat{n}$	σ_{C}	$\hat{\sigma}_{\mathrm{C}}$	σ_{BND}	$\hat{\sigma}_{\mathrm{BND}}$	$\hat{B}_{\mathrm{BND}}$
36	1	130	$\approx 2^{12}$	$\approx 2^{16.5}$	$\approx 2^{13.5}$	$\approx 2^{66}$	$\approx 2^{72.5}$

Table 5: Concrete parameters estimated for $\kappa \approx 168$ bits of DKS^{∞} security using the LWE-estimator and $\kappa \approx 262$ bits of SKS^{2} security (by computing the Hermite root value to be 1.00225 from the dimension, modulus, and 2-norm of the secret vector).

Clearly $\det(\boldsymbol{M}_1) = M_2 \cdots M_\tau$ and $\det(\boldsymbol{M}_2) = \hat{M}_1 \cdots \hat{M}_{\tau-1}$ and therefore $\det(\boldsymbol{M}) = \prod_{i=1}^{\tau} M_i - \prod_{i=1}^{\tau} \hat{M}_j$ and $\prod_{i=1}^{\tau} M_i = \prod_{i=1}^{\tau} \hat{M}_j \neq 0$. Define the two polynomials

$$g(X) = \prod\nolimits_{i=1}^{\tau} (m_i - X), \text{ and } \hat{g}(X) = \prod\nolimits_{i=1}^{\tau} (\hat{m}_i - X),$$

that is, $g(X)$ and $\hat{g}(X)$ are the polynomials which have m_i and $\hat{m}_i$ as their roots, respectively. If there exists no permutation such that $\hat{m}_i = m_{\pi(i)}, \forall i = 1, \ldots, \tau$, then $g(\rho) = \hat{g}(\rho)$ (i.e. $\det(\boldsymbol{M}) = 0$) for at most τ choices of ρ according to Lemma 6. But due to the structure of Γ_1 this cannot be the case (as there are $\tau + 1$ accepting choices of ρ) and assuming that the commitment scheme is binding, it must hold that the values m_i derived from the M_i extracted using Π_{Lin} indeed are a permutation of the $\hat{m}_i$. We additionally obtain values f_i and $\boldsymbol{r}_i$ as a side-product.

Assume, for the sake of contradiction, that the extracted $\tilde{\boldsymbol{m}}_i = (f_i \boldsymbol{c}_{2,i} - \boldsymbol{A}_{\mathrm{C},2} \boldsymbol{r}_i)/f_i$ do not form a permutation on the $\hat{\boldsymbol{m}}_i$. The inner product with $\boldsymbol{\alpha}$ computed with each $\hat{\boldsymbol{m}}_i$ implicitly computes an evaluation of the polynomial with coefficients $\hat{\boldsymbol{m}}_i$ in point h, where the polynomial is of degree $l_c - 1$.

By assumption, there exists a $\tilde{\boldsymbol{m}}_i$ which is different from all $\hat{\boldsymbol{m}}_j$. So consider all the polynomials $\boldsymbol{\delta}_j = \tilde{\boldsymbol{m}}_i - \hat{\boldsymbol{m}}_j$, each of degree $l_c - 1$ and each non-zero by assumption. The non-permutation is only not detected if some $\boldsymbol{\delta}_j$ is always 0 during the check, when evaluated in h. By Lemma 6 any $\boldsymbol{\delta}_j$ has at most $l_c - 1$ different roots in the set containing choices of h, so there can only be $\tau \cdot (l_c - 1)$ such roots in total. By requiring that there must always be $\tau \cdot (l_c - 1) + 1$ accepting choices of h, the result follows. □

For each Γ_i, sampling from the image of the set $U_{\Gamma_i}(S)$ for any S is straightforward. Moreover, the t-value of Γ_1 is $(\tau \cdot (l_c - 1) + 1) \cdot (\tau + 1)$ while it is 2 for Γ_2 and $\tau + 1$ for Γ_3. Therefore, T_Γ is polynomial. Hence, we can compute an error-bound $\kappa_\Gamma = 2/q^N + 2\tau/|C|^\tau + 4(\tau + 1) \cdot (\tau \cdot (l_c - 1) + 1)/q^{2N}$.

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 $\sigma_{\mathrm{C}} = 0.954 \nu B_{\mathsf{Com}} \sqrt{kN}$. For the re-usable commitments we get $\hat{\sigma}_{\mathrm{C}} = 22 \nu B_{\mathsf{Com}} \sqrt{kN}$. The amortized proof also has two checks, and we get a standard deviation $0.954 \|\boldsymbol{S}' \boldsymbol{C}'\|_2$, where σ_{BND} and $\hat{\sigma}_{\mathrm{BND}}$ are depending on the norm of the elements in the rows of $\boldsymbol{S}'$.

We let the noise bounds $B_{\mathsf{Key}} = B_{\mathsf{Err}} = 1$ 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: $B_{\mathrm{START}} = p \left\| er + e_{i,2} - e_{i,1} s \right\|_\infty + \|m\|_\infty$.
- Noise per shuffle: $B_{\mathrm{SHUF}} = p(\|er'\|_\infty + \left\| e'_{i,2} \right\|_\infty + \left\| -e'_{i,1} s \right\|_\infty)$.
- Noise in partial decryption: $B_{\mathsf{DDec}} = p \xi_2 \left\| E'_{i,j} \right\|_\infty \leq 2^{\mathrm{sec}} B_{\mathsf{Dec}}$,

where $B_{\mathsf{Dec}} = B_{\mathrm{START}} + \xi_1 B_{\mathrm{SHUF}}$ is the upper bound of the noise added before the decryption phase. This means that we have the following bounds on each of the noise terms above, when using ternary noise:

$$\|e\|_1 \leq N, \quad \|r\|_\infty \leq 1, \quad \left\| e_{i,2} \right\|_\infty \leq 1, \quad \left\| e_{i,1} \right\|_1 \leq N,$$

$$\|s\|_\infty \leq 1, \quad \left\| r' \right\|_\infty \leq 1, \quad \left\| e'_{i,2} \right\|_\infty \leq 1, \quad \left\| e'_{i,1} \right\|_1 \leq N.$$

We get the following upper bounds:

$$B_{\mathrm{START}} = p(2N + 1) + \lceil (p - 1)/2 \rceil, \quad B_{\mathrm{SHUF}} = p(2N + 1),$$

which for ξ_1 shuffles gives us

$$B_{\mathsf{Dec}} = (\xi_1 + 1) p (2N + 1) + \lceil (p - 1)/2 \rceil.$$

Finally, we need to make sure that $B_{\mathsf{Dec}} + B_{\mathsf{DDec}} < q/2$, where $B_{\mathsf{DDec}} = 2 p \xi_2 \hat{B}_{\mathrm{BND}}$ because of the soundness slack of the amortized proof of bounded values. A honestly generated value $E_{i,j}$ is bounded by $2^{\mathrm{sec}} (B_{\mathsf{Dec}} / p \xi_2)$, but the proof can only guarantee that the values are shorter than some larger bound $2 \hat{B}_{\mathrm{BND}}$ (following Baum et al. [8, Lemma 3]) that depends on the number of equations in the statement. Define $\boldsymbol{S}'_{1,k}$ to be the first k rows of $\boldsymbol{S}'$ and define $\boldsymbol{S}'_{k+1}$ to be the last row of $\boldsymbol{S}'$. For batches of N equations, we then get that:

$$\begin{aligned} B_{\mathrm{BND}} &\leq \sqrt{2N} \cdot \sigma_{\mathrm{BND}} \leq \sqrt{2N} \cdot 0.954 \cdot \max \left\| \boldsymbol{S}'_{1,k} \boldsymbol{C}' \right\|_2 \\ &\leq 1.35 \cdot \sqrt{N} \cdot \max \left\| \boldsymbol{S}'_{1,k} \right\|_1 \cdot \max \left\| \boldsymbol{C}' \right\|_\infty \\ &\leq 1.35 \cdot k \cdot \sqrt{N} \cdot N \cdot B_{\mathsf{Com}}, \end{aligned}$$

and, similarly,

$$\hat{B}_{\mathrm{BND}} \leq \sqrt{2N} \cdot \hat{\sigma}_{\mathrm{BND}} \leq 1.35 \cdot \sqrt{N} \cdot N \cdot \left\| E_{i,j} \right\|_\infty,$$

with B_{BND} for rows 1 to k of $\boldsymbol{Z}$ and $\hat{B}_{\mathrm{BND}}$ for the last.

We fix plaintext modulus $p = 2$, statistical security parameter $\mathrm{sec} = 40$, 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 $B_{\mathsf{Dec}} < 2^{17}$ and $B_{\mathsf{DDec}} < 2^{76.5}$. We then set $q \approx 2^{78}$, and verify that

$$\max_{i \in [\tau]} \| v_i - s u_i \| < 2 \cdot (2^{17} + 2^{76.5}) < q.$$

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 τ commitments at once. Then we get $18N/(q-N) \approx 2^{-62}$, and hence, we must compute each proof twice in parallel to achieve negligible soundness. Furthermore, we choose $g \approx 2^{20}, l \approx 2^{20.3}, \eta = 325$ to keep the soundness $\approx 2^{-62}$. The total size of π_{SMALL}, by instantiating 1, is $\approx 20\tau$ 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 370τ KB per mixing node $\mathcal{S}_i$. The size of the decryption phase, including partial decryptions, commitments, proofs of linearity, and proofs of boundedness, is approximately 157τ KB per decryption node $\mathcal{D}_j$.

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 (Π_{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 [27] could be used to show that Π_{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 [26], which applies to Σ-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 Π_{BND} as it seems unlikely that [26] 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 Π_{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_i 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* v, and outputs an *encrypted ballot* ev and a *ballot proof* π_v.

The *code* algorithm Code takes a code key ck, an encrypted ballot ev and a proof π_v as input and outputs a pre-code $\hat{r}$ or $\perp$. (If the code key ck is $\perp$, the algorithm outputs 0 or 1.)

The *shuffle* algorithm Shuffle takes a public key pk and a sequence of encrypted ballots $\boldsymbol{ev}$, and outputs a sequence of encrypted ballots $\boldsymbol{ev}'$ and a proof of shuffle π_s.

The *verify* algorithm Verify takes a public key pk, two sequences of encrypted ballots $\boldsymbol{ev}$, and $\boldsymbol{ev}'$ and a proof π_s, and outputs 0 or 1.

The *distributed decryption* algorithm DistDec takes a decryption key dk_i and a sequence of encrypted ballots $\boldsymbol{ev}$, and outputs a sequence of ballot decryption shares $\boldsymbol{sv}_i$ and a decryption proof $\pi_{d,i}$.

The *combining* algorithm Comb takes a public key pk, a sequence of encrypted ballots $\boldsymbol{ev}$, ballot decryption share sequences $\boldsymbol{sv}_1, \boldsymbol{sv}_2, \ldots, \boldsymbol{sv}_{l_d}$ with proofs $\pi_{d,1}, \pi_{d,2}, \ldots, \pi_{d,l_d}$, and outputs either $\perp$ or a sequence of ballots $v_1, v_2, \ldots, v_{l_t}$.

A cryptographic voting scheme is l_s*-correct* if for any $(\mathsf{pk}, \{\mathsf{dk}_i\}, \mathsf{ck})$ output by Setup and any $(\mathsf{vvk}_1, \mathsf{vck}_1, f_1), \ldots, (\mathsf{vvk}_{l_V}, \mathsf{vck}_{l_V}, f_{l_V})$ output by Reg(pk), any ballots $v_1, \ldots, v_{l_V}$, any $(ev_i^{(0)}, \pi_{v,i})$ output by $\mathsf{Cast}(\mathsf{pk}, \mathsf{vck}_i, v_i)$, $i = 1, \ldots, l_V$, any sequence of l_s sequences of encrypted ballots $\boldsymbol{ev}^{(j)}$ with proofs $\pi_{s,j}$ output by $\mathsf{Shuffle}(\mathsf{pk}, \boldsymbol{ev}^{(j-1)})$, any ballot decryption shares $\boldsymbol{sv}_1, \ldots, \boldsymbol{sv}_{l_d}$ with proofs $\pi_{d,1}, \ldots, \pi_{d,l_d}$ output by $\mathsf{DistDec}(\mathsf{dk}_i, \boldsymbol{ev}^{(l_s)})$, $i = 1, 2, \ldots, l_d$ and any $(v'_1, \ldots, v'_{l_V})$ possibly output by $\mathsf{Comb}(\mathsf{pk}, \boldsymbol{ev}^{(l_s)}, \boldsymbol{sv}_1, \ldots, \boldsymbol{sv}_{l_d}), \pi_{d,1}, \ldots, \pi_{d,l_d})$, then:

- $\mathsf{Code}(\mathsf{ck}, \mathsf{vvk}_i, ev_i, \pi_{v,i}) = f_i(v_i)$, $\mathsf{Code}(\perp, \mathsf{vvk}_i, ev_i, \pi_{v,i}) = 1$,
- $\mathsf{Verify}(\mathsf{pk}, \boldsymbol{ev}^{(j-1)}, \boldsymbol{ev}^{(j)}, \pi_{s,j}) = 1$ for $j = 1, 2, \ldots, l_s$,
- $\mathsf{Comb}(\mathsf{pk}, \boldsymbol{ev}^{(l_s)}, \boldsymbol{sv}_1, \ldots, \boldsymbol{sv}_{l_d}), \pi_{d,1}, \ldots, \pi_{d,l_d})$ did not output $\perp$, and
- $v_1, \ldots, v_{l_V}$ equals $v'_1, \ldots, v'_{l_V}$, up to order.

We also require that the distribution of ev_i only depends on pk and v_i, not vck_i.

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 $\mathsf{pk}_C \leftarrow \mathsf{KeyGen}_C$, $(\mathsf{pk}_V, \mathsf{dk}_V) \leftarrow \mathsf{KeyGen}_{VE}$, $(\mathsf{pk}_R, \mathsf{dk}_R) \leftarrow \mathsf{KeyGen}_{VE}$, as well as key shares $\mathsf{dk}_{V,i}$ for every decryption server. The public key $\mathsf{pk} = (\mathsf{pk}_C, \mathsf{pk}_V, \mathsf{pk}_R)$, the decryption share is $\mathsf{dk} = (\mathsf{pk}_C, \mathsf{dk}_{V,i})$ and the code key is $\mathsf{ck} = (\mathsf{pk}_C, \mathsf{pk}_V, \mathsf{dk}_R)$.
- Reg takes $\mathsf{pk} = (\mathsf{pk}_C, \mathsf{pk}_V, \mathsf{pk}_R)$ as input. It samples $a \xleftarrow{\$} R_p$ and computes $(\boldsymbol{c}_a, d_a) \leftarrow \mathsf{Com}\mathsf{pk}_C, a$. The voter verification key is $\mathsf{vvk} = \boldsymbol{c}_a$, the voter casting key is $(a, \boldsymbol{c}_a, d_a)$, and the function f is $v \mapsto v + a$.
- Cast takes $\mathsf{pk} = (\mathsf{pk}_C, \mathsf{pk}_V, \mathsf{pk}_R)$, $\mathsf{vck} = (a, \boldsymbol{c}_a, d_a)$ and v as input. It computes $\boldsymbol{v} \leftarrow \mathsf{Enc}_{VE}(\mathsf{pk}_V, v)$, $\hat{r} \leftarrow a + v$ and $\boldsymbol{w} \leftarrow \mathsf{Enc}_{VE}(\mathsf{pk}_R, \hat{r})$, along with a proof $\pi_{v,0}$ that $\boldsymbol{v}$ and $\boldsymbol{w}$ are well-formed ciphertexts, and that $\boldsymbol{c}_a$ is a commitment to the difference of the decryptions. The encrypted ballot is $ev = \boldsymbol{v}$, while the ballot proof is $\pi_v = (\boldsymbol{w}, \pi_{v,0})$.
- Code takes $\mathsf{ck} = (\mathsf{pk}_C, \mathsf{pk}_V, \mathsf{dk}_R)$, a voter verification key vvk, an encrypted ballot $ev = \boldsymbol{v}$ and a ballot proof $\pi_v = (\boldsymbol{w}, \pi_{v,0})$ as input. It verifies $\pi_{v,0}$ and outputs $\perp$ if verification fails. Otherwise, it computes $\hat{r} \leftarrow \mathsf{Dec}_{VE}(\mathsf{dk}_R, \boldsymbol{w})$ and outputs $\hat{r}$. (If $\mathsf{ck} = \perp$, it outputs 1 if and only if it accepts $\pi_{v,0}$.)
- 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 [12]. 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 $b, \xleftarrow{\$} \{0, 1\}$. Set L, L', L'' to be empty lists.
- $(\mathsf{pk}, \{\mathsf{dk}_i\}, \mathsf{ck}) \leftarrow \mathsf{Setup}$. For $i = 1, \dots, l_V$: $(\mathsf{vvk}_i, \mathsf{vck}_i, f_i) \leftarrow \mathsf{Reg}(\mathsf{pk})$. Send $(\mathsf{pk}, \mathsf{vvk}_1, \dots, \mathsf{vvk}_{l_V})$ to $\mathcal{A}$.
- On a *voter reveal query* i, send (vck_i, f_i) to $\mathcal{A}$. On a *decrypt reveal query* i, send dk_i to $\mathcal{A}$. On a *code reveal query*, send ck to $\mathcal{A}$.
- On a *challenge query* (i, v_0, v_1), compute $(ev, \pi_v) \leftarrow \mathsf{Cast}(\mathsf{pk}, \mathsf{vck}_i, v_b)$, $\hat{r} \leftarrow \mathsf{Code}(\mathsf{ck}, \mathsf{vvk}_i, ev, \pi_v)$, append (i, v_0, v_1, ev, π_v) to L. Send (ev, π_v) to $\mathcal{A}$.
- On a *chosen ciphertext query* (i, ev, π_v), compute $\hat{r} \leftarrow \mathsf{Code}(\mathsf{ck}, \mathsf{vvk}_i, ev, \pi_v)$. If $\hat{r} \neq \perp$, append $(i, \perp, \perp, ev, \pi_v)$ to L. Send $\hat{r}$ to $\mathcal{A}$.
- On a *shuffle query* $\boldsymbol{ev}$, compute $(\boldsymbol{ev}', \pi_s) \leftarrow \mathsf{Shuffle}(\mathsf{pk}, \boldsymbol{ev})$, then record $(\boldsymbol{ev}, \boldsymbol{ev}', \pi_s)$ in L'. Send $(\boldsymbol{ev}', \pi_s)$ to $\mathcal{A}$.
- On a *chosen shuffle query* $(\boldsymbol{ev}, \boldsymbol{ev}', \pi_s)$, we record it in L' if and only if $\mathsf{Verify}(\mathsf{pk}, \boldsymbol{ev}, \boldsymbol{ev}', \pi_s) = 1$.
- On a *ballot decryption share query* $(i, \boldsymbol{ev})$, we then compute $(\boldsymbol{sv}_i, \pi_{d,i}) \leftarrow \mathsf{DistDec}(\mathsf{dk}_i, \boldsymbol{ev})$, record $(i, \boldsymbol{ev}, \boldsymbol{sv}_i, \pi_{d,i})$ in L'' and send $(\boldsymbol{sv}_i, \pi_{d,i})$ to $\mathcal{A}$.
- On a *test query* $(\boldsymbol{ev}, \boldsymbol{sv}_1, \dots, \boldsymbol{sv}_{l_d}, \pi_{d,1}, \dots, \pi_{d,l_d})$, then compute the *result* $\leftarrow \mathsf{Comb}(\mathsf{pk}, \boldsymbol{ev}, \boldsymbol{sv}_1, \dots, \boldsymbol{sv}_{l_d}, \pi_{d,1}, \dots, \pi_{d,l_d})$ 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 $\boldsymbol{ev}$ is *valid* if tuples $(i_1, v_{01}, v_{11}, ev_1, \pi_{v,1}), \dots, (i_{l_c}, v_{0l_c}, v_{1l_c}, ev_{l_c}, \pi_{v,l_c})$ in L and L' contains a sequence of tuples $(\boldsymbol{ev}^{(j-1)}, \boldsymbol{ev}^{(j)}, \pi_{s,j})$, $j = 1, 2, \dots, l_s$, such that $\boldsymbol{ev}^{(0)} = (ev_1, \dots, ev_{l_c})$

and $\boldsymbol{ev}^{(l_s)} = \boldsymbol{ev}$. In this case we also say that $\boldsymbol{ev}$ *derives* from $(i_1, v_{01}, v_{11}, ev_1, \pi_{v,1}), \dots, (i_{l_c}, v_{0l_c}, v_{1l_c}, ev_{l_c}, \pi_{v,l_c})$. A valid sequence $\boldsymbol{ev}$ is *honest* if at least one of the tuples $(\boldsymbol{ev}^{(j-1)}, \boldsymbol{ev}^{(j)}, \pi_{s,j})$ originated with a shuffle query. A valid sequence $\boldsymbol{ev}$ is *balanced* if the ballot sequence $(v_{01}, \dots, v_{0l_c})$ equals $(v_{11}, \dots, v_{1l_c})$, 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 at least one 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 $(\cdot, \boldsymbol{ev})$, the sequence $\boldsymbol{ev}$ is balanced and honest *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, π_v) where $\mathrm{Code}(\mathsf{ck}, \mathsf{vvk}_i, ev, \pi_v) = \hat{r} \neq \bot$, we have that either $\mathrm{Dec}(\{\mathsf{dk}_i\}, ev) = \bot$ or $\mathrm{Dec}(\{\mathsf{dk}_i\}, ev) = v$ and $f_i(v) \neq \hat{r}$.

Let F_c (count failure) be the event that a test query gets *result* $= \bot$ when $\boldsymbol{ev}$ is valid and $(\boldsymbol{ev}, \boldsymbol{sv}_i, \pi_{d,i})$ is in L'' for $i = 1, \dots, l_d$.

Let F_d (inconsistent decryption) be the event that a test query $(\boldsymbol{ev}, \boldsymbol{sv}_1, \dots, \boldsymbol{sv}_{l_d}, \pi_{d,1}, \dots, \pi_{d,l_d})$ with *result* $= (v_1, \dots, v_{l_c})$, where $\boldsymbol{ev}$ derives from $(i_1, v_{01}, v_{11}, ev_1, \pi_{v,1}), \dots, (i_{l_c}, v_{0l_c}, v_{1l_c}, ev_{l_c}, \pi_{v,l_c})$, there is no permutation π on $\{1, 2, \dots, l_c\}$ such that $v_{b,k} = \bot$ or $v_{b,k} = v_{\pi(k)}$ for $k = 1, 2, \dots, l_c$. Let F_u (no unique decryption) be the event that two test queries $(\boldsymbol{ev}, \boldsymbol{sv}_1, \dots, \boldsymbol{sv}_{l_d}, \pi_{d,1}, \dots, \pi_{d,l_d})$ and $(\boldsymbol{ev}, \boldsymbol{sv}'_1, \dots, \boldsymbol{sv}'_{l_d}, \pi_{d,1}{}', \dots, \pi_{d,l_d}{}')$ for some valid $\boldsymbol{ev}$ get results *result* and *result'* that are not equal up to order, and neither of which is equal to $\bot$. The advantage of the adversary is

$$\max\{2 \cdot |\Pr[E_g \wedge E_f] - \Pr[E_f]/2|, \Pr[F_i \vee F_c \vee F_d \vee F_u]\}.$$

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 $\Pr[E_g \wedge E_f]$.

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 $(ev_1, \dots, ev_{l_c})$, 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 ith ballot decryption share to give the correct result when combined.)

Next, for chosen ciphertext queries, we decrypt the $\boldsymbol{w}$ using dk_R and subtract a to recover v, and then record (i, v, v, ev, π_v) instead of $(i, \bot, \bot, ev, \pi_v)$. The soundness of the ballot proof (details omitted), we now have that every tuple (i, v_0, v_1, ev, π_v) in L satisfies $\mathtt{Dec}(\mathsf{dk}_V, ev) = v_b$.

Next, for any ballot decryption share query for an honest and balanced sequence $(ev_1, \dots, ev_{l_c})$ deriving from $(\cdot, v_{01}, v_{11}, \cdot, \cdot), \dots, (\cdot, v_{0l_c}, v_{1l_c}, \cdot, \cdot)$, sample a permutation π on $\{1, 2, \dots, l_c\}$ and use $v_i = v_{0\pi(i)}$ 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 $\{\mathsf{dk}_i\}$ 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 $\Pr[E_g \wedge E_f]$ 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_c 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_c 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_u 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_u happening is appropriately bounded.

The claim that $\Pr[F_i \vee F_c \vee F_d \vee F_u]$ 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 ϵ-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 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* $\mathcal{R}$ 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 $\mathcal{A}$ 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 $\mathcal{B}$ and the auditor $\mathcal{A}$ 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, ϵ-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 a result and a *transcript*. Also, there is an algorithm for verifying either a transcript, a result, and optionally a receipt.

Consider an execution of the voting protocol where the auditor outputs a result and a transcript. Then there is a set of honest voters with honest computers that accept their ballot as cast with some receipt, and for which the verification algorithm accepts the transcript, the result, and their receipt. We say that a system is *verifiable* if the result is consistent with the list of these voters' ballots being included in the result.

Note that verifiability in and of itself does not guarantee anything about the correctness of the result. Instead, verifiability is best thought of as a tool that can be used to achieve trust in election integrity under fairly weak trust assumptions. For instance, one can prove that if a sufficiently large and hard-to-guess subset of voters run the verification algorithm on the transcript, result, and their receipt, then the overall election has ϵ-integrity for some ϵ. (The "hard to guess" part is instrumental in proving this result. If the set of voters verifying an election is not hard to guess, achieving election integrity is much more difficult, at least without strong trust assumptions.)

Note also that we assume that the honest voter's computer is honest in the definition. If the voter's computer is corrupted, we are left with considering integrity as above, which can be achieved conditional on other players being honest.

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.

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.