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

Diego F. Aranha[1], Carsten Baum[1],
Kristian Gjøsteen[2], and Tjerand Silde[2]*

[1] Aarhus University, Denmark
{dfaranha,cbaum}@cs.au.dk
[2] Norwegian University of Science and Technology, Norway
{kristian.gjosteen,tjerand.silde}@ntnu.no

Abstract. Cryptographic voting protocols have recently seen much interest from practitioners due to their (planned) use in countries such as Estonia, Switzerland and Australia. Many organizations also use Helios for elections. While many efficient protocols exist from discrete log-type assumptions, the situation is less clear for post-quantum alternatives such as lattices. This is because previous voting protocols do not carry over easily due to issues such as noise growth and approximate relations. In particular, this is a problem for tested designs such as verifiable mixing and decryption of ballot ciphertexts.

In this work, we make progress in this direction. We propose a new verifiable secret shuffle for BGV ciphertexts as well as 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 for BGV decryption, proving correctness of decryption steps in zero-knowledge.

We give concrete parameters for our system, estimate the size of each component and provide an implementation of all sub-protocols. Together, the shuffle and the decryption protocol are suitable for use in real-world cryptographic voting schemes, which we demonstrate with a prototype voting protocol design.

Keywords: lattice cryptography · verifiable mix-nets · distributed decryption · zero-knowledge proofs · cryptographic voting · implementation

1 Introduction

Mix-nets were originally proposed for anonymous communication [Cha81], but have since been used extensively for cryptographic voting systems. A mix-net is a multi-party protocol which gets as input a collection of ciphertexts and outputs another collection of ciphertexts whose decryption is the same set, up to order.

* Work done in part while visiting Aarhus University.

The mix-net will mix the ciphertexts so that the permutation between input and output ciphertexts is hidden if at least one party is honest.

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 must then be sent through a mix-net before decryption, to break the identity-ballot correlation.

In addition to breaking the correlation between input and output ciphertexts, the correctness of the mix-net must be verifiable. In some cases, it is sufficient that some auditor (possibly distributed) operating at the same time as the mix-net can verify correctness, but for other applications the mix-net should provide a proof of correctness that can be verified by anyone at any later point in time.

A *shuffle* of a set of ciphertexts is another set of ciphertexts whose decryption is the same as the original set, up to order. A shuffle is *secret* if it is hard to correlate input and output ciphertexts. A shuffle is *verifiable* if there is some proof for the claim that the decryptions are the same.

If we have a verifiable secret shuffle for some cryptosystem, building a mix-net is trivial. The nodes of the mix-net receive a set of ciphertexts as input, shuffle them sequentially and provide a proof of correctness. The mix-net proof then consists of the sets of intermediate ciphertexts along with the shuffle proofs. If at least one node in the mix-net is honest, it is hard to correlate the inputs and outputs.

For applications in cryptographic voting, we also need verifiable decryption to ensure that the correct result can be obtained. The design of the voting system must ensure that nobody has both the decryption key and the original ciphertexts. One simple strategy for this is to use verifiable threshold decryption, where the decryption key is secret-shared among a committee of decryption parties.

Verifiable shuffling and verifiable distributed decryption protocols are well-known for cryptosystems based on discrete log-type assumptions. For example, Neff [Nef01] proposed the first efficient verifiable secret shuffle for ElGamal-like cryptosystems. Distributed verifiable decryption can be achieved by giving uniformly random shares of the secret key to a set decryption nodes, and have each of them compute a partial decryption together with a proof of equality of discrete logarithms [CP93] with respect to the secret key share.

Quantum-safety is critical for cryptographic voting systems, since elections have a long-term need for privacy, and there is an urgent need for progress. Verifiable secret shuffles and verifiable distributed decryption are two long-standing obstacles to delivering practical cryptographic voting schemes based on quantum-safe computational problems such as lattice assumptions.

1.1 Our contributions

In this paper, we design a verifiable secret shuffle for BGV ciphertexts [BGV12] that is suitable for cryptographic voting systems. The main obstacle to simply adopting the ideas used by Neff for discrete logarithms to lattices is the lack of suitable underlying proofs and techniques. We overcome these obstacles by designing an extended version of the shuffle of commitments to known values by Aranha *et al.* [ABG+21]. In our protocol, the shuffler gets input ciphertexts $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_\tau$. We let the shuffler commit to re-randomization ciphertexts $\hat{\boldsymbol{c}}_1, \ldots, \hat{\boldsymbol{c}}_\tau$ using a suitable linearly homomorphic commitment scheme $\texttt{Com}$. Together with an amortized proof of shortness of the randomness used for the committed re-randomization ciphertexts, this gives us a verifiable shuffle:

1. First, the shuffler commits to the re-randomization ciphertexts $\hat{\boldsymbol{c}}_1, \ldots, \hat{\boldsymbol{c}}_\tau$ as $\texttt{Com}(\hat{\boldsymbol{c}}_i)$ and shows that they are well-formed.
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 $\boldsymbol{c}_i + \texttt{Com}(\hat{\boldsymbol{c}}_i)$.

Towards implementing this, we use highly efficient lattice-based commitments [BDL+18] together with a version of recent amortized proofs of shortness [BLNS21].

As explained, a verifiable secret shuffle on its own is usually not enough to build a cryptographic voting system. We also need some way to decrypt the output of the mix-net, without compromising the input ciphertexts or allowing a decryption server to cheat. 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, called noise drowning [BD10]. Finally they publish the partial decryptions together with a proof of correctness of the decryption (and boundedness of the noise used), and the plaintexts are computed in public by combining all the partial decryptions.

Lattice-based cryptography is very delicate with respect to noise-levels, bounds and dimensions, and we have to be cautious when combining the sub-protocols mentioned into a larger construction. Each shuffle adds extra noise to each ciphertext, which means that to ensure correctness of decryption we need to choose specific parameters based on the number of shuffles and the norm of the noise added in each shuffle. Here, the norm is guaranteed by the zero-knowledge proofs of shortness accompanying each shuffle. Furthermore, 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 both the bounds of the amortized zero-knowledge proof and the choice of parameters for the overall cryptosystem. 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 provide a sketch of a full cryptographic voting protocol. A simplified variant could be used as a quantum-safe Helios [Adi08] variant. We give example parameters suitable for this protocol with 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 tens of thousands.

To summarize our implementation results, a ciphertext ballot is of size 80 KB, 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 2.5 ms to encrypt a ballot, while the mixing proof takes 1024τ ms and the decryption proof takes 81.4τ ms. Note that the shuffle proof only takes 15.1τ ms, so the time it takes to mix the ciphertexts is dominated by the time it takes to prove correct re-randomization, which is 1009τ ms. Verification is much faster, only 20τ ms. These results improve on the state of the art considerably, see details in Section 7.

1.2 Related work

Aranha *et al.* [ABG$^+$21] provide a verifiable shuffle of known commitment openings together with concrete parameters and an 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 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.* [CMM19] design a more general shuffle with a straight-forward approach similar to Neff [Nef01] based on roots of polynomials. Their protocol requires committing to two evaluations of a polynomial, and then prove the correctness of evaluation using a sequence of multiplication proofs which are quite costly in practice. Farzaliyev *et al.* [FWK21] implements the mix-net by Costa *et al.* [CMM19] using the amortization techniques by Attema *et al.* [ALS20] for the commitment scheme by Baum *et al* [BDL$^+$18]. Here, the proof size is approximately 15 MB per voter, a factor 40 larger than our shuffle proof, even for a smaller parameter set that does not take into account distributed decryption afterwards. We expect our shuffle proof to be an additional factor 10 smaller than what we presented above with optimal parameters for the shuffle only ($q \approx 2^{32}$ and $N = 1024$). Furthermore, their proof generation takes approximately 1.5 seconds per vote, which is approximately 40 % faster than it takes to produce our shuffle proof (when normalizing for clock frequency), with parameters that do not take decryption into account.

Recently, Herranz *et al.* [HMS21] gave a new proof of correct shuffle based on Benes networks and sub-linear lattice-based proofs for arithmetic circuit satisfiability. However, the scheme is not implemented and the example parameters do not take the soundness slack of the amortized zero-knowledge proofs into account. Moreover, [HMS21] does not consider decryption of ballots, which would heavily impact the parameters of their protocol in practice.

A completely different approach to mix-nets is 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.* [BHM20] 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 done, but no auditor is available. Fast or generic decryption mix-nets such as Boyen *et al.* [BHM20] need an auditor (that can be distributed) to verify the mix-net, but the auditor must be trusted during the mix-net operation. This conflicts with universal verifiability.

del Pino *et al.* [dLNS17] 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.* [BHM21] that the protocol in [dLNS17] is not end-to-end verifiable unless all tallying authorities and all voters' voting devices are honest. This problem is solved by [BHM21], but their construction still has the downside of only supporting homomorphic tallying. Strand [Str19] built a verifiable shuffle for the GSW cryptosystem, but this construction is too restrictive for practical use. Chillotti *et al.* [CGGI16] uses fully homomorphic encryption, which for the foreseeable future is most likely not efficient enough to be considered for practical deployment.

2 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) [LN16]. 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 $\mathbb{Z}^N$:

$$||f||_1 = \sum |\alpha_i|, \qquad ||f||_2 = \left(\sum \alpha_i^2\right)^{1/2}, \qquad ||f||_\infty = \max_{i \in [1,\ldots,n]} \{|\alpha_i|\}.$$

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:

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

We furthermore define the sets $S_{\beta_\infty} = \{x \in R_q \mid \|x\|_\infty \leq \beta_\infty\}$ as well as

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

2.1 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

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

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 [BBC+18]), then,

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

2.2 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 1.

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

$$\max \frac{\mathcal{N}^k_\sigma(\boldsymbol{z})}{\mathcal{N}^k_{\boldsymbol{v},\sigma}(\boldsymbol{z})} = \exp\left[\frac{-2\langle \boldsymbol{z}, \boldsymbol{v}\rangle + \|\boldsymbol{v}\|_2^2}{2\sigma^2}\right] \leq \exp\left[\frac{24\sigma\|\boldsymbol{v}\|_2 + \|\boldsymbol{v}\|_2^2}{2\sigma^2}\right] = M, \qquad (1)$$

where we use the tail bound from Section 2.1, 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. [BLS19]. However, if the procedure is only done once for the vector $\boldsymbol{v}$, we can decease the parameters slightly, to the cost of leaking only one bit of information about $\boldsymbol{v}$ from the given $\boldsymbol{z}$.

In [LNS21], 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 1, 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 u in the case $b = 1$, but allowing a smaller standard deviation.

```
$\mathrm{Rej}(\boldsymbol{z}, \boldsymbol{v}, b, M, \sigma)$
if $b = 1$ and $\langle \boldsymbol{z}, \boldsymbol{v} \rangle < 0$: return $1$
$\mu \leftarrow\!\$\; [0, 1)$
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 $0$
```

Fig. 1. Rejection sampling

2.3 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 1 (Search Knapsack problem). *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 $\mathcal{A}$ has advantage ϵ in solving the $\mathsf{SKS}^2_{n,k,\beta}$ problem if*

$$\Pr\left[\begin{array}{c} \|y_i\|_2 \le \beta \,\wedge \\ [\ \boldsymbol{I}_n \quad \boldsymbol{A}'\ ] \cdot \boldsymbol{y} = \boldsymbol{0}^n \end{array} \;\middle|\; \begin{array}{c} \boldsymbol{A}' \leftarrow\!\$\; R_q^{n \times (k-n)}; \\ \boldsymbol{0} \ne \boldsymbol{y} = [y_1, \ldots, y_k]^\top \leftarrow \mathcal{A}(\boldsymbol{A}') \end{array}\right] \ge \epsilon.$$

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 2 (Decisional Knapsack problem). *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_{β_∞} when given $\boldsymbol{A}'$. An algorithm $\mathcal{A}$ has advantage ϵ in solving the $\mathsf{DKS}^\infty_{n,k,\beta}$ problem if*

$$\Big| \Pr[b = 1 \mid \boldsymbol{A}' \leftarrow\!\$\; R_q^{n \times (k-n)}; \boldsymbol{y} \leftarrow\!\$\; S^k_{\beta_\infty}; b \leftarrow \mathcal{A}(\boldsymbol{A}', [\ \boldsymbol{I}_n \quad \boldsymbol{A}'\ ] \cdot \boldsymbol{y})]$$
$$- \Pr[b = 1 \mid \boldsymbol{A}' \leftarrow\!\$\; R_q^{n \times (k-n)}; \boldsymbol{u} \leftarrow\!\$\; R_q^n; b \leftarrow \mathcal{A}(\boldsymbol{A}', \boldsymbol{u})] \Big| \ge \epsilon.$$

See [LS15] for more details about hardness problems over module lattices.

2.4 Public Key Encryption

We present definitions inspired by Goldwasser and Micali [GM82] for the security of a (slightly additively homomorphic) public key encryption scheme. We only present chosen plaintext (CPA) security here, as we need to randomize ciphertexts in our main protocol. To ensure full security one often requires chosen ciphertext security, which can be achieved by combining a CPA secure scheme with zero-knowledge proofs of correct encryption.

Definition 3 (Public Key Encryption Scheme). *A public key encryption scheme consists of three algorithms: key generation* (KeyGen)*, encryption* (Enc) *and decryption* (Dec)*, where*

- KeyGen*, on input security parameter* 1^λ*, outputs public parameters* pp*, a public key* pk*, and a secret key* sk*,*
- Enc*, on input the public key* pk *and a message m, outputs a ciphertext c,*
- Dec*, on input the secret key* sk *and a ciphertext c, outputs a message m,*

and the public parameters pp *are implicit inputs to* Enc *and* Dec*.*

Definition 4 (τ-Correctness). *We say that the public key encryption scheme is τ-*correct *if a sum of τ honestly generated ciphertext with overwhelming probability decrypts to the sum of the τ encrypted messages. Hence, we want that*

$$\Pr\left[\mathtt{Dec}(\mathsf{sk}, \sum_{i\in[\tau]} c_i) = \sum_{i\in[\tau]} m_i \;:\; \begin{matrix}(\mathsf{pp},\mathsf{pk},\mathsf{sk}) \leftarrow \mathtt{KeyGen}(1^\lambda) \\ \{c_i\}_{i\in[\tau]} \leftarrow \mathtt{Enc}(\mathsf{pk}, \{m_i\}_{i\in[\tau]})\end{matrix}\right] \geq 1 - \epsilon(\lambda),$$

where the probability is taken over the random coins of KeyGen *and* Enc*.*

Definition 5 (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*

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

where the probability is taken over the random coins of KeyGen *and* Enc*.*

2.5 Public Key Distributed Decryption

We now present a definition of a secure public key distributed decryption protocol that is suitable for our voting application.

Definition 6 (Distributed Decryption Scheme). *A public key distributed decryption scheme consists of five algorithms: key generation* (KeyGen)*, encryption* (Enc)*, decryption* (Dec)*, distributed decryption* (DistDec) *and combine* (Comb)*, where*

- KeyGen*, on input security parameter* 1^λ *and number of key-shares ξ, outputs public parameters* pp*, a public key* pk*, a secret key* sk*, and key-shares* $\{\mathsf{sk}_j\}$*,*
- Enc*, on input the public key* pk *and messages $\{m_i\}$, outputs ciphertexts $\{c_i\}$,*
- Dec*, on input the secret key* sk *and ciphertexts $\{c_i\}$, outputs messages $\{m_i\}$,*
- DistDec*, 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 the public parameters $\mathtt{pp}$ *are implicit inputs to* $\mathtt{Enc}$, $\mathtt{Dec}$, $\mathtt{DistDec}$ *and* $\mathtt{Comb}$.

Let $P_{\mathsf{sk}}(u, v)$ be an efficiently computable predicate that on input secret key $\mathsf{sk} = s$ and a ciphertext $c = (u, v)$ outputs 1 or 0. For example, it outputs 1 if $\|v - su\|_\infty < B_{\mathtt{Dec}} \ll \lfloor q/2 \rfloor$ and otherwise 0 for the BGV scheme in Section 3.1.

Definition 7 (Threshold Correctness). *We say that the public key distributed encryption scheme is* threshold correct *with respect to to* $P_{\mathsf{sk}}(\cdot)$ *if*

$$\Pr\left[\begin{matrix} \mathtt{Comb}(\{c_i\}_{i\in[\tau]}, \{\mathtt{ds}_{i,j}\}_{i\in[\tau],j\in[\xi]}) \\ = \\ \mathtt{Dec}(\mathsf{sk}, \{c_i\}_{i\in[\tau]}) \end{matrix} \quad : \quad \begin{matrix} (\mathtt{pp}, \mathtt{pk}, \mathsf{sk}, \{\mathtt{sk}_j\}_{j\in[\xi]}) \leftarrow \mathtt{KeyGen}(1^\lambda, \xi) \\ \{c_1, \ldots, c_\tau\} \leftarrow \mathcal{A}(\mathtt{pp}, \mathtt{pk}) \\ \forall i \in [\tau] : \;\; P_{\mathsf{sk}}(c_i) = 1 \\ \forall j \in [\xi] : \;\; \{\mathtt{ds}_{i,j}\}_{i\in[\tau]} \leftarrow \mathtt{DistDec}(\mathtt{sk}_j, \{c_i\}_{i\in[\tau]}) \end{matrix}\right] = 1,$$

where the probability is taken over the random coins of $\mathtt{KeyGen}$ *and* $\mathtt{DistDec}$.

Definition 8 (Threshold Verifiability). *We say that the public key distributed encryption scheme is* threshold verifiable *with respect to* $P_{\mathsf{sk}}(\cdot)$ *if an adversary* $\mathcal{A}$ *corrupting* $J \subseteq [\xi]$ *secret key-shares* $\{\mathtt{sk}_j\}_{j\in J}$ *cannot convince* $\mathtt{Comb}$ *to accept maliciously created decryption-shares* $\{\mathtt{ds}_{i,j}\}_{i\in[\tau],j\in J}$. *More concretely:*

$$\Pr\left[\begin{matrix} \mathtt{Dec}(\mathsf{sk}, \{c_i\}_{i\in[\tau]}) \\ \neq \\ \mathtt{Comb}(\{c_i\}_{i\in[\tau]}, \{\mathtt{ds}_{i,j}\}_{i\in[\tau],j\in[\xi]}) \\ \neq \\ \bot \end{matrix} \;:\; \begin{matrix} (\mathtt{pp}, \mathtt{pk}, \mathsf{sk}, \{\mathtt{sk}_j\}_{j\in[\xi]}) \leftarrow \mathtt{KeyGen}(1^\lambda, \xi) \\ (\{c_1, \ldots, c_\tau\}, \mathsf{st}) \leftarrow \mathcal{A}(\mathtt{pp}, \mathtt{pk}, \{\mathtt{sk}_j\}_{j\in J}) \\ \forall i \in [\tau] : \;\; P_{\mathsf{sk}}(c_i) = 1 \\ \forall j \notin J\colon \{\mathtt{ds}_{i,j}\}_{i\in[\tau]} \leftarrow \mathtt{DistDec}(\mathtt{sk}_j, \{c_i\}_{i\in[\tau]}) \\ \{\mathtt{ds}_{i,j}\}_{i\in[\tau],j\in J} \leftarrow \mathcal{A}(\{\mathtt{ds}_{i,j}\}_{i\in[\tau],j\notin J}, \mathsf{st}) \end{matrix}\right] \leq \epsilon(\lambda),$$

where the probability is taken over the random coins of $\mathtt{KeyGen}$ *and* $\mathtt{DistDec}$.

Definition 9 (Distributed Decryption Simulatability). *We say that the public key distributed decryption scheme is* simulatable *with respect to* $P_{\mathsf{sk}}(\cdot)$ *if an adversary* $\mathcal{A}$ *corrupting* $J \subsetneq [\xi]$ *secret key-shares* $\{\mathtt{sk}_j\}_{j\in J}$ *cannot distinguish the transcript of the decryption protocol from a simulation by a simulator* $\mathcal{S}$ *which only gets* $\{\mathtt{sk}_j\}_{j\in J}$ *as well as correct decryptions as input. More concretely:*

$$|\Pr\left[b = b' \;\;:\;\; \begin{matrix} (\mathtt{pp}, \mathtt{pk}, \mathsf{sk}, \{\mathtt{sk}\}_{j\in[\xi]}) \leftarrow \mathtt{KeyGen}(1^\lambda, \xi) \\ (\{c_1, \ldots, c_\tau\}, \mathsf{st}) \leftarrow \mathcal{A}(\mathtt{pp}, \mathtt{pk}, \{\mathtt{sk}_j\}_{j\in J}) \\ \forall i \in [\tau] : \;\; P_{\mathsf{sk}}(c_i) = 1 \\ \{\mathtt{ds}^0_{i,j}\} \leftarrow \mathtt{DistDec}(\{\mathtt{sk}_j\}_{j\in[\xi]}, \{c_i\}_{i\in[\tau]}) \\ \{\mathtt{ds}^1_{i,j}\} \leftarrow \mathcal{S}(\mathtt{pp}, \{\mathtt{sk}_j\}_{j\in J}, \{c_i, \mathtt{Dec}(\mathsf{sk}, c_i)\}_{i\in[\tau]}) \\ b \xleftarrow{\$} \{0,1\}, b' \leftarrow \mathcal{A}(\{\mathtt{ds}^b_{i,j}\}_{i\in[\tau],j\in[\xi]}, \mathsf{st}) \end{matrix}\right] - \frac{1}{2}| \leq \epsilon(\lambda),$$

where the probability is taken over the random coins of $\mathtt{KeyGen}, \mathtt{DistDec}$ *and* $\mathcal{S}$.

2.6 Commitments

Commitment schemes were first introduced by Blum [Blu84], and have since become an essential component in many advanced cryptography protocols.

Definition 10 (Commitment Scheme). *A commitment scheme consists of three algorithms: key generation* ($\mathtt{KeyGen}$), *commitment* ($\mathtt{Com}$) *and opening* ($\mathtt{Open}$), *where*

- $\mathtt{KeyGen}$, *on input security parameter* 1^λ, *outputs public parameters* $\mathtt{pp}$,

- $\mathtt{Com}$*, on input message* m*, outputs commitment* c *and opening* r*,*
- $\mathtt{Open}$*, on input* m*,* c *and* r*, outputs either* 0 *or* 1*,*

and the public parameters $\mathtt{pp}$ *are implicit inputs to* $\mathtt{Com}$ *and* $\mathtt{Open}$*.*

Definition 11 (Completeness). *We say that the commitment scheme is* complete *if an honestly generated commitment is accepted by the opening algorithm. Hence, we want that*

$$\Pr\left[\mathtt{Open}(m,c,r)=1 \;:\; \begin{matrix}\mathtt{pp}\leftarrow \mathtt{KeyGen}(1^{\lambda})\\ (c,r)\leftarrow \mathtt{Com}(m)\end{matrix}\right]=1,$$

where the probability is taken over the random coins of $\mathtt{KeyGen}$ *and* $\mathtt{Com}$*.*

Definition 12 (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*

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

where the probability is taken over the random coins of $\mathtt{KeyGen}$ *and* $\mathtt{Com}$*.*

Definition 13 (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*

$$\Pr\left[\begin{matrix}m\neq \hat{m}\\ \mathtt{Open}(m,c,r)=1\\ \mathtt{Open}(\hat{m},c,\hat{r})=1\end{matrix} \;:\; \begin{matrix}\mathtt{pp}\leftarrow \mathtt{KeyGen}(1^{\lambda})\\ (c,m,r,\hat{m},\hat{r})\leftarrow \mathtt{A}(\mathtt{pp})\end{matrix}\right]\leq \epsilon(\lambda),$$

where the probability is taken over the random coins of $\mathtt{KeyGen}$*.*

2.7 Zero-Knowledge Proofs

These definitions are based on Goldwasser et al. [GMR85]. Let $\mathtt{L}$ be a language, and let $\mathtt{R}$ be a NP-relation on $\mathtt{L}$. Then, x is an element in $\mathtt{L}$ if there exists a witness w such that $(x,w)\in\mathtt{R}$. We let $\mathtt{P}$, $\mathtt{P}^*$, $\mathtt{V}$ and $\mathtt{V}^*$ be polynomial time algorithms.

Definition 14 (Interactive Proofs). *An interactive proof protocol* Π *consists of two parties: a prover* $\mathtt{P}$ *and a verifier* $\mathtt{V}$*, and a setup algorithm* ($\mathtt{Setup}$)*, where* $\mathtt{Setup}$*, on input the security parameter* 1^{λ}*, outputs public setup parameters* $\mathtt{sp}$*. The protocol consists of a transcript* $\mathtt{T}$ *of the communication between* $\mathtt{P}$ *and* $\mathtt{V}$*, with respect to* $\mathtt{sp}$*, and the conversation terminates with* $\mathtt{V}$ *outputting either* 1 *or* 0*. Let* $\langle \mathtt{P}(\mathtt{sp},x,w),\mathtt{V}(\mathtt{sp},x)\rangle$ *denote the output of* $\mathtt{V}$ *on input* x *after its interaction with* $\mathtt{P}$*, who holds a witness* w*.*

Definition 15 (Completeness). *We say that a proof protocol* Π *is* complete *if* $\mathtt{V}$ *outputs* 1 *when* $\mathtt{P}$ *knows a witness* w *and both parties follows the protocol. Hence, for any efficient sampling algorithm* $\mathtt{P}_0$ *we want that*

$$\Pr\left[\langle \mathtt{P}(\mathtt{sp}, x, w), \mathtt{V}(\mathtt{sp}, x)\rangle = 1 \;:\; \begin{array}{c} \mathtt{sp} \leftarrow \mathtt{Setup}(1^\lambda) \\ (x, w) \leftarrow \mathtt{P}_0(\mathtt{sp}) \\ (x, w) \in \mathtt{R} \end{array}\right] = 1,$$

where the probability is taken over the random coins of $\mathtt{Setup}, \mathtt{P}$ *and* $\mathtt{V}$*.*

Definition 16 (Knowledge Soundness). *We say that a proof protocol* Π *is* knowledge sound *if, when a cheating prover* $\mathtt{P}^*$ *that does not know a witness* w *is able to convince a honest verifier* $\mathtt{V}$*, there exists a polynomial time algorithm extractor* $\mathcal{E}$ *which, give black-box access to* $\mathtt{P}^*$*, can output a witness* w *such that* $(x, w) \in \mathtt{R}$*. Hence, we want that*

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

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

Definition 17 (Honest-Verifier Zero-Knowledge). *We say that a proof protocol* Π *is* honest-verifier zero-knowledge *if a honest but curious verifier* $\mathtt{V}^*$ *that follows the protocol cannot learn anything beyond the fact that* $x \in \mathtt{L}$*. Hence, we want for real accepting transcripts* $\mathtt{T}_{\langle \mathtt{P}(\mathtt{sp}, x, w), \mathtt{V}(\mathtt{sp}, x)\rangle}$ *between a prover* $\mathtt{P}$ *and a verifier* $\mathtt{V}$*, and a accepting transcript* $\mathtt{S}_{\langle \mathtt{P}(\mathtt{sp}, x, \cdot), \mathtt{V}(\mathtt{sp}, x)\rangle}$ *generated by simulator* $\mathcal{S}$ *that only knows* x*, that*

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

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

An interactive honest-verifier zero-knowledge proof protocol can be made non-interactive using the Fiat-Shamir transform [FS87].

3 Background: Lattice-Based Cryptography

We start this section by presenting the BGV encryption scheme by Brakerski *et al.* [BGV12], and continue by presenting the commitment scheme by Baum *et al.* [BDL$^+$18] and it's respective zero-knowledge proofs of linear relations. Then, we present two amortized zero-knowledge proofs of knowledge of short preimages. First, we present a modification of the protocol due to Bootle *et al.* [BLNS21] for proving knowledge of many instances when the secrets are very short, say, all coefficients are ternary, and the proof must be exact. We also provide the protocol due to Baum *et al.* [BBC$^+$18] for proving knowledge of many bounded instances where the proof is approximate.

3.1 BGV Encryption

Let $p \ll q$ be primes, let R_q and R_p be defined as above for a fixed N, let $\mathtt{D}$ be a bounded distribution over R_q, let $B_\infty \in \mathbb{N}$ be a bound and let λ be the security parameter. The (plain) BGV encryption scheme follows Definition 3 (in Section 2) and consists of three algorithms: key generation ($\mathtt{KeyGen}$), encryption ($\mathtt{Enc}$) and decryption ($\mathtt{Dec}$), where:

- $\mathtt{KeyGen}$ samples an element $a \leftarrow\!\!\$\, R_q$ uniformly at random, samples a short $s \leftarrow\!\!\$\, R_q$ such that $\|s\|_\infty \leq B_\infty$ and samples noise $e \leftarrow \mathtt{D}$. The algorithm outputs public key $\mathtt{pk} = (a, b) = (a, as + pe)$ and secret key $\mathtt{sk} = s$.

- $\mathtt{Enc}$, on input the public key $\mathtt{pk} = (a, b)$ and an element m in R_p, samples a short $r \leftarrow\!\!\$\, R_q$ such that $\|r\|_\infty \leq B_\infty$, samples noise $e', e'' \leftarrow \mathtt{D}$, and outputs the ciphertext $c = (u, v) = (ar + pe', br + pe'' + m)$.

- $\mathtt{Dec}$, on input the secret key $\mathtt{sk} = s$ and a ciphertext $c = (u, v)$ in R_q^2, outputs the message $m = (v - su \mod q) \mod p$.

The following theorem for the BGV encryption scheme follows from Brakerski *et al.* [BGV12, Section 3] and Lyubashevsky *et al.* [LPR13, Lemma 8.3].

Theorem 1 (Correctness and CPA Security of BGV). *The BGV encryption scheme is* 1-correct *(Definition 4 in 2.4) if $\|v - su\|_\infty \leq B_{\mathtt{Dec}} < \lfloor q/2 \rfloor$, and the scheme is secure against* chosen plaintext attacks *(Definition 5 in 2.4) if the $\mathsf{DKS}^\infty_{N,2,\beta}$ problem is hard for some $\beta = \beta(N, q, B_\infty, \sigma, p)$. More general, the scheme is* τ-correct *if $\tau \cdot B_{\mathtt{Dec}} < \lfloor q/2 \rfloor$.*

Furthermore, we present the passively secure distributed decryption technique used in the MPC-protocols by Damgård *et al.* [BD10, DKL$^+$13, DPSZ12]. Here the $\mathtt{KeyGen}$ algorithm for $1 \leq j \leq \xi$ outputs uniformly random shares $\mathtt{sk}_j = s_j$ of the secret key $\mathtt{sk} = s$ such that $s = s_1 + s_2 + \cdots + s_\xi$. This can be used to define a passively secure threshold decryption algorithm as follows:

- $\mathtt{DistDec}$, on input a secret key-share $\mathtt{sk}_j = s_j$ and a ciphertext $c = (u, v)$ in R_q^2, computes $m_j = s_j u$, sample some uniform noise $E_j \leftarrow\!\!\$\, R_q$ such that $\|E_j\|_\infty \leq 2^{\mathtt{sec}}(B_{\mathtt{Dec}}/p\xi)$ for statistical security parameter $\mathtt{sec}$ and noise-bound $B_{\mathtt{Dec}} = \max\|v - su\|_\infty$, then outputs $\mathtt{ds}_j = t_j = m_j + pE_j$.

- $\mathtt{Comb}$, on input the ciphertext (u, v) and the set of decryption shares $\{\mathtt{ds}_j\}_{j \in [\xi]}$, outputs the message $m = (v - t \mod q) \mod p$, where $t = t_1 + t_2 + \cdots + t_\xi$.

The following theorem for the distributed decryption protocol follows from the works by Damgård *et al.* [DPSZ12, Theorem 4] and [DKL$^+$13, Appendix G].

Theorem 2 (Correctness and Simulatability of Distributed BGV). *Let* sec *be the statistical security parameter. The distributed BGV encryption scheme is* correct *(Definition 7 in 2.5) if $\|v - t\|_\infty \leq (1 + 2^{\mathtt{sec}})B_{\mathtt{Dec}} < \lfloor q/2 \rfloor$, and is* decryption simulatable *against passive adversaries (Definition 9 in 2.5).*

3.2 Lattice-Based Commitments

Let R_q be defined as above for a fixed N and let $\mathcal{N}_{\sigma_{\mathrm{C}}}$ be a Gaussian distribution over R_q with standard deviation σ_{C}. The commitment scheme follows Definition 10 and consists of three algorithms: key generation ($\mathtt{KeyGen}_{\mathrm{C}}$), committing ($\mathtt{Com}$) and opening ($\mathtt{Open}$), where:

$\mathtt{KeyGen}_{\mathrm{C}}$ outputs a public key $\mathtt{pk}$ which allows to commit to messages in R_q^{ℓ} using randomness in $S^k_{B_{\mathtt{Com}}}$. We define

$$\begin{aligned} \boldsymbol{A}_1 &= \begin{bmatrix} \boldsymbol{I}_n & \boldsymbol{A}'_1 \end{bmatrix} && \text{where } \boldsymbol{A}'_1 \leftarrow\!\!\$\; R_q^{n\times(k-n)} \\ \boldsymbol{A}_2 &= \begin{bmatrix} \boldsymbol{0}^{\ell\times n} & \boldsymbol{I}_\ell & \boldsymbol{A}'_2 \end{bmatrix} && \text{where } \boldsymbol{A}'_2 \leftarrow\!\!\$\; R_q^{l\times(k-n-\ell)}, \end{aligned}$$

for height $n+\ell$ and width k and let $\mathtt{pk}$ be $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_1 \\ \boldsymbol{A}_2 \end{bmatrix}$.

$\mathtt{Com}$ commits to messages $\boldsymbol{m} \in R_q^{\ell}$ by sampling an $\boldsymbol{r_m} \leftarrow\!\!\$\; S^k_{B_{\mathtt{Com}}}$ and computes

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

$\mathtt{Com}$ outputs commitment $[\![\boldsymbol{m}]\!]$ and opening $\boldsymbol{d} = (\boldsymbol{m}, \boldsymbol{r_m}, 1)$.

$\mathtt{Open}$ verifies whether an opening $(\boldsymbol{m}, \boldsymbol{r_m}, f)$, with $f \in \bar{\mathcal{C}}$, is a valid opening of $[\![\boldsymbol{m}]\!]$ for the public key $\mathtt{pk}$ by checking that $\|\boldsymbol{r_m}[i]\| \le 4\sigma_{\mathrm{C}}\sqrt{N}$, for $i \in [k]$, and if

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

$\mathtt{Open}$ outputs 1 if all these conditions holds, and 0 otherwise.

For any commitment generated with $\mathtt{Com}$, the algorithm $\mathtt{Open}$ will accept (except with negligible probability) by setting $f = 1$.

The following theorem for the security of the commitment scheme follows from Baum *et al.* [BDL+18, Lemma 6 and Lemma 7].

Theorem 3 (Hiding and Binding of the Commitment Scheme). *The commitment scheme is* hiding *(Definition 12 in 2.6) if the* $\mathsf{DKS}^{\infty}_{n+\ell,k,\beta_\infty}$ *problem is hard, and the scheme is* binding *(Definition 13 in 2.6) if the* $\mathsf{SKS}^{2}_{n,k,16\sigma_{\mathrm{C}}\sqrt{\nu N}}$ *problem is hard.*

The commitments [BDL+18] have a weak additively homomorphic property:

Proposition 1. *Let* $[\![\boldsymbol{m}]\!] = \mathtt{Com}(\boldsymbol{m};\boldsymbol{r_m})$ *be a commitment with opening* $(\boldsymbol{m}, \boldsymbol{r_m}, f)$ *and let* $[\![\boldsymbol{m}']\!] = \mathtt{Com}(\boldsymbol{m}';\boldsymbol{0})$*. Then* $[\![\boldsymbol{m}]\!] - [\![\boldsymbol{m}']\!]$ *has the opening* $(\boldsymbol{m}-\boldsymbol{m}', \boldsymbol{r_m}, f)$*.*

The proof follows from the linearity of the verification algorithm.

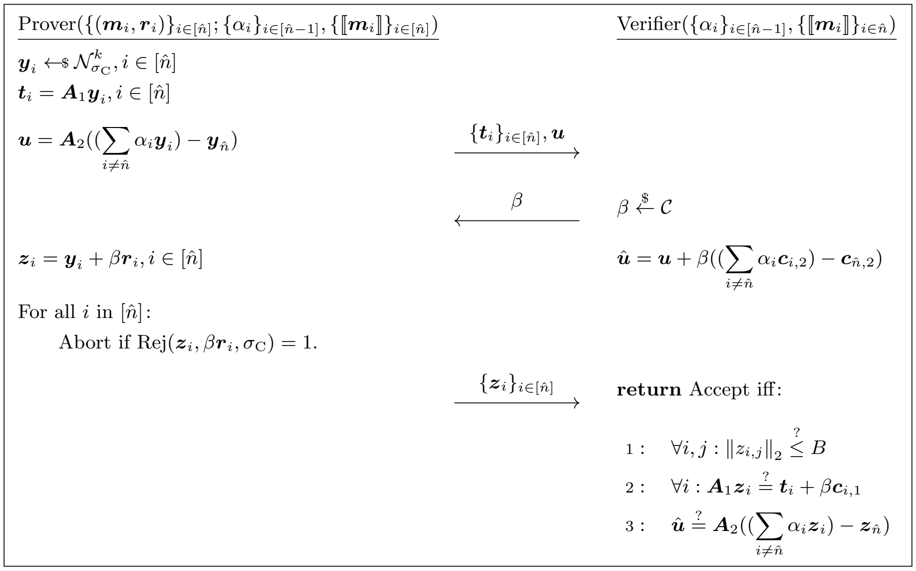
Prover$(\{(\boldsymbol{m}_i, \boldsymbol{r}_i)\}_{i\in[\hat{n}]}; \{\alpha_i\}_{i\in[\hat{n}-1]}, \{[\![\boldsymbol{m}_i]\!]\}_{i\in[\hat{n}]})$
Verifier$(\{\alpha_i\}_{i\in[\hat{n}-1]}, \{[\![\boldsymbol{m}_i]\!]\}_{i\in\hat{n}})$
$\boldsymbol{y}_i \leftarrow\!\$\, \mathcal{N}^k_{\sigma_{\mathrm{C}}}, i \in [\hat{n}]$
$\boldsymbol{t}_i = \boldsymbol{A}_1\boldsymbol{y}_i, i \in [\hat{n}]$
$\boldsymbol{u} = \boldsymbol{A}_2((\sum_{i\neq\hat{n}} \alpha_i \boldsymbol{y}_i) - \boldsymbol{y}_{\hat{n}})$
$\xrightarrow{\{\boldsymbol{t}_i\}_{i\in[\hat{n}]}, \boldsymbol{u}}$
$\xleftarrow{\beta}$
$\beta \overset{\$}{\leftarrow} \mathcal{C}$
$\boldsymbol{z}_i = \boldsymbol{y}_i + \beta\boldsymbol{r}_i, i \in [\hat{n}]$
$\hat{\boldsymbol{u}} = \boldsymbol{u} + \beta((\sum_{i\neq\hat{n}} \alpha_i \boldsymbol{c}_{i,2}) - \boldsymbol{c}_{\hat{n},2})$
For all i in $[\hat{n}]$:
Abort if $\mathrm{Rej}(\boldsymbol{z}_i, \beta\boldsymbol{r}_i, \sigma_{\mathrm{C}}) = 1$.
$\xrightarrow{\{\boldsymbol{z}_i\}_{i\in[\hat{n}]}}$
return Accept iff:
$1: \quad \forall i, j : \|z_{i,j}\|_2 \overset{?}{\leq} B$
$2: \quad \forall i : \boldsymbol{A}_1\boldsymbol{z}_i \overset{?}{=} \boldsymbol{t}_i + \beta\boldsymbol{c}_{i,1}$
$3: \quad \hat{\boldsymbol{u}} \overset{?}{=} \boldsymbol{A}_2((\sum_{i\neq\hat{n}} \alpha_i \boldsymbol{z}_i) - \boldsymbol{z}_{\hat{n}})$

Fig. 2. Protocol $\mathit{\Pi}_{\mathrm{LIN}}$ is a Sigma-protocol to prove the relation R_{LIN}.

3.3 Zero-Knowledge Proof of Linear Relations

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

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

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

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

$\mathit{\Pi}_{\mathrm{LIN}}$ in Figure 2 is a zero-knowledge proof of knowledge (ZKPoK) of this relation (it is a directly extended version of the proof of linearity in [BDL$^+$18]). The relation R_{LIN} is relaxed because of the additional factor f in the opening, which appears in the soundness proof. It does not show up in the protocol $\mathit{\Pi}_{\mathrm{LIN}}$, because an honest prover will implicitly use $f = 1$.

The bound is $B = 2\sigma_{\mathrm{C}}\sqrt{N}$ and the $\mathit{\Pi}_{\mathrm{LIN}}$-protocol produces a proof transcript of the form $\pi_{\mathrm{LIN}} = ((\{\boldsymbol{t}_i\}_{i\in[\hat{n}]}, u), \beta, (\{\boldsymbol{z}_i\}_{i\in[\hat{n}]}))$.

The following theorem for the security of zero-knowledge proof of linear relations is a direct adaption of Baum *et al.* [BDL$^+$18, Lemma 8].

Theorem 4 (Security of Zero-Knowledge Proof of Linear Relations). *The zero-knowledge proof of linear relations is* complete *(Definition 15 in 2.7) if the randomness $\boldsymbol{r}_i$ is bounded by $B_{\mathtt{Com}}$ in the ℓ_∞ norm, it is* special sound

(Definition 16 in 2.7) if the $\mathsf{SKS}^2_{n,k,4\sigma_C\sqrt{N}}$ *problem is hard, and it is statistical* honest-verifier zero-knowledge *(Definition 17 in 2.7). The probability of success is* $(1/M)^{\hat{n}}$*, where the constant* M *is computed as in Equation 1.*

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

When applying the Fiat-Shamir transform [FS87], we let β be the output of a hash-function applied to the first message and x. Then, the proof transcript is reduced to $\pi_L = (\beta, \{\boldsymbol{z}_i\}_{i\in[\hat{n}]})$ where β is of 2λ bits and each $\boldsymbol{z}_i$ is of size $kN\log_2(6\sigma_C)$ bits. We can compress $\boldsymbol{z}_i$ to $\hat{n}(k-n)N\log_2(6\sigma_C)$ bits by checking an approximate equality instead, as described in [ABG⁺21, Section 3.2]. We denote by

$$\begin{aligned} \pi_L &\leftarrow \Pi_{\text{LIN}}(\{(\boldsymbol{m}_i, \boldsymbol{r}_i)\}_{i\in[\hat{n}]}; (\{\alpha_i\}_{i\in[\hat{n}-1]}, \{[\![\boldsymbol{m}_i]\!]\}_{i\in[\hat{n}]})), \text{ and} \\ 0 \vee 1 &\leftarrow \Pi_{\text{LINV}}(\{\alpha_i\}_{i\in[\hat{n}-1]}, \{[\![\boldsymbol{m}_i]\!]\}_{i\in[\hat{n}]}; \pi_L), \end{aligned}$$

the run of the proof and verification protocols, respectively, where the verification protocol Π_{LinV} reconstructs the first message using β, performs the verification as in the last step in Figure 2 and then checks that β was computed correctly with respect to the statement and the first message.

3.4 Exact Amortized Zero-Knowledge Proof of Short Openings

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 $\hat{\boldsymbol{A}}$ in $\mathbb{Z}_q^{N\times N}$. Let $\boldsymbol{A}$ be a $r \times v$ matrix over R_q, that is, a $rN \times vN$ matrix over $\mathbb{Z}_q$. We will now consider how to prove generically that $\boldsymbol{t}_i = \boldsymbol{A}\boldsymbol{s}_i$ for a bounded $\boldsymbol{s}_i$, which is the same as proving correct multiplication over the ring R_q of a public polynomial a and a secret and bounded polynomials s_i resulting in public polynomials t_i.

Bootle *et al.* [BLNS21] 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 $\boldsymbol{A}\boldsymbol{s}_i + \boldsymbol{e}_i = \boldsymbol{t}_i$. Here we adapt their techniques for the case where $\boldsymbol{e}_i$ is zero, and prove that $\|\boldsymbol{s}_i\|_\infty \leq 1$. We further modify their protocol for amortized proofs to benefit from smaller parameters due to the small bound on $\boldsymbol{s}_i$, while their amortized protocol was optimized for larger bounds[3].

We first explain the main idea of [BLNS21] for proving knowledge of one preimage $\boldsymbol{s}$ of $\boldsymbol{t} = \boldsymbol{A}\boldsymbol{s}$ and then how it generalizes to an amortized proof for τ elements with sublinear communication.

The approach follows an ideal linear commitments-technique with vector commitments $\texttt{Com}_{\text{L}}(\cdot)$ 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

[3] The authors of [BLNS21] mention that this optimization is possible, but neither present the modified protocol nor a proof.

```
1 :  P has input $\{\boldsymbol{s}_i\}_{i=1}^{\tau}; (\boldsymbol{A}, \{\boldsymbol{t}_i\}_{i=1}^{\tau})$ while V has input $\boldsymbol{A}, \{\boldsymbol{t}_i\}_{i=1}^{\tau}$
2 :  P samples $\boldsymbol{s}_0 \leftarrow\!\!\$\ \mathbb{Z}_q^{vN}$, $\boldsymbol{h} \leftarrow\!\!\$\ \mathbb{Z}_q^{2vN}$, $\boldsymbol{r}_0 \leftarrow\!\!\$\ \mathbb{Z}_q^{\eta}$, $\boldsymbol{r}_h \leftarrow\!\!\$\ \mathbb{Z}_q^{\eta}$ and
     $\boldsymbol{r}_{i,j} \leftarrow\!\!\$\ \mathbb{Z}_q^{\eta}$ for each $i \in [\tau], j \in \{0, 1, 2\}$.
3 :  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
       $\frac{1}{\ell_0(X)} \cdot \prod_{h \in \{-1,0,1\}} \Big[\boldsymbol{f}(X) - h \cdot \boldsymbol{1}\Big] = \sum_{j=0}^{2} \sum_{i=1}^{\tau} \boldsymbol{v}_{i,j} \ell_i(X) \ell_0(X)^j$
4 :  P computes $\boldsymbol{H}_0 \leftarrow \texttt{Encode}(\boldsymbol{s}_0, \boldsymbol{0}, \boldsymbol{r}_0)$, $\boldsymbol{H} \leftarrow \texttt{Encode}(\boldsymbol{h}, \boldsymbol{r}_h)$ as well as
     $\boldsymbol{H}_{i,j} = \texttt{Encode}(\delta_j \cdot \boldsymbol{s}_i, \boldsymbol{v}_{i,j}, \boldsymbol{r}_{i,j})$ for each $i \in [\tau], j \in \{0, 1, 2\}$.
5 :  P computes $\boldsymbol{E} = \texttt{RowsToMatrix}(\boldsymbol{H}, \boldsymbol{H}_0, \boldsymbol{H}_{1,0}, \boldsymbol{H}_{1,1}, \ldots, \boldsymbol{H}_{\tau,2})$,
     $\mathcal{M} = \texttt{MerkleTree}(\texttt{CommitToColums}(\boldsymbol{E}))$ and $(c_{\boldsymbol{d}}, r_{\boldsymbol{d}}) \leftarrow \texttt{Com}_{\texttt{Aux}}(\boldsymbol{d})$.
6 :  P sends $\mathcal{M}, c_{\boldsymbol{d}}$ to V
7 :  V samples $x \leftarrow\!\!\$\ \mathbb{Z}_q^* \setminus \{a_1, \ldots, a_\tau\}$ and $\beta_0, \beta_{1,0}, \ldots, \beta_{\tau,2} \leftarrow\!\!\$\ \mathbb{Z}_q^*$ and
       sends $x, \beta_0, \beta_{1,0}, \ldots, \beta_{\tau,2}$ to P
8 :  P computes $\bar{\boldsymbol{f}} = \boldsymbol{f}(x)$, $\bar{\boldsymbol{r}}_f = \ell_0(x)\boldsymbol{r}_0 + \sum_{i=1}^{\tau} \sum_{j=0}^{2} \ell_i(x)\ell_0(x)^j \boldsymbol{r}_{i,j}$,
     $\bar{\boldsymbol{h}} = \boldsymbol{h} + \beta_0 \boldsymbol{s}_0 + \sum_{j=0}^{2} \sum_{i=1}^{\tau} \beta_{i,j}(\delta_j \cdot \boldsymbol{s}_i, \boldsymbol{v}_{i,j})$, $\bar{\boldsymbol{r}}_h = \boldsymbol{r}_h + \beta_0 \boldsymbol{r}_0 + \sum_{j=0}^{2} \sum_{i=1}^{\tau} \beta_{i,j} \boldsymbol{r}_{i,j}$
9 :  P sends $\bar{\boldsymbol{f}}, \bar{\boldsymbol{r}}_f, \bar{\boldsymbol{h}}, \bar{\boldsymbol{r}}_h, r_{\boldsymbol{d}}$ to V
10 : V samples $I \leftarrow\!\!\$\ [l]^{\eta}, |I| = \eta$ and sends $I$ to P
11 : P computes and sends $\boldsymbol{E}|_I, \texttt{MerklePaths}_I$ to V
12 : V runs $\texttt{Verify}(\boldsymbol{E}|_I, \mathcal{M}, \texttt{MerklePaths}_I)$ and checks that
     $\texttt{Open}_{\texttt{Aux}}\left(c_{\boldsymbol{d}}, \frac{1}{\ell_0(x)} \cdot \left(\sum_{i=1}^{\tau} \boldsymbol{t}_i \ell_i(x) - \boldsymbol{A}\bar{\boldsymbol{f}}\right), r_{\boldsymbol{d}}\right) = 1,$
     $\texttt{Encode}\left(\bar{\boldsymbol{f}}, \frac{1}{\ell_0(x)} \cdot \prod_{h \in \{-1,0,1\}} \Big[\bar{\boldsymbol{f}} - h \cdot \boldsymbol{1}\Big], \bar{\boldsymbol{r}}_f\right)\Big|_I$
         $\stackrel{?}{=} \ell_0(x) \cdot \boldsymbol{H}_0\big|_I + \sum_{i=1}^{\tau} \sum_{j=0}^{2} \ell_i(x)\ell_0(x)^j \boldsymbol{H}_{i,j}|_I$
     $\texttt{Encode}\left(\bar{\boldsymbol{h}}, \bar{\boldsymbol{r}}_h\right)\Big|_I \stackrel{?}{=} \boldsymbol{H} + \beta_0 \boldsymbol{H}_0\big|_I + \sum_{i=1}^{\tau} \sum_{j=0}^{2} \beta_{i,j} \boldsymbol{H}_{i,j}\big|_I$
```

Fig. 3. The protocol Π_{AEx} is an exact amortized zero-knowledge proof of knowledge of ternary openings. δ_x is 1 if $x = 0$ and 0 otherwise. $(\texttt{Com}_{\texttt{Aux}}, \texttt{Open}_{\texttt{Aux}})$ is an arbitrary commitment scheme.

of polynomials $\boldsymbol{f}(X) = \boldsymbol{s}_0(X) + \boldsymbol{s}$ for the prover. Now consider the vector of polynomials $\boldsymbol{f}(X) \circ (\boldsymbol{f}(X) - \mathbf{1}) \circ (\boldsymbol{f}(X) + \mathbf{1})$, where $\circ$ denote the coordinate-wise product, then we can see that the coefficients of X^0 are exactly $\boldsymbol{s} \circ (\boldsymbol{s} - \mathbf{1}) \circ (\boldsymbol{s} + \mathbf{1})$ and therefore $\mathbf{0}$ if and only if the aforementioned bound on $\boldsymbol{s}$ holds. In that case, each aforementioned polynomial in $\boldsymbol{f}(X) \circ (\boldsymbol{f}(X) - \mathbf{1}) \circ (\boldsymbol{f}(X) + \mathbf{1})$ is divisible by X. Therefore, the prover computes the coefficient vectors $\boldsymbol{v}_2, \boldsymbol{v}_1, \boldsymbol{v}_0$ of

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

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 $\overline{\boldsymbol{f}} = \boldsymbol{f}(x)$. Additionally, the prover uses the linear property of the commitment scheme to show that:

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

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 $\overline{f}$ 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 indeed the preimage of $\boldsymbol{t}$ under $\boldsymbol{A}$.

The ideal linear commitments in [BLNS21] get realized using an Encode-then-Hash commitment scheme. In this commitment scheme, the prover commits to vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{Z}_q^g$ as follows:

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

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. The prover 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 $\mathcal{M}$ by revealing the path.
4. The verifier checks that $\mathtt{Encode}(\boldsymbol{x} \| \boldsymbol{r})$ 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[4], but they do not sum up to $\boldsymbol{x}$, then the linear combination of the codewords in $\boldsymbol{E}$ must differ from $\texttt{Encode}(\boldsymbol{x}\|\boldsymbol{r})$ 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 $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.

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. Here, the goal is to establish the latter for all $\boldsymbol{t}_i$ simultaneously while verifying only one equation and sending only one vector $\overline{\boldsymbol{f}}$. Towards this, the prover as before 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 interpolation 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 $\{\ell_i(X)\ell_0(X)^j\}_{j=0}^{b-1}$. If we now, more generally, define the polynomial

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

then we observe that $\boldsymbol{f}(X) \circ (\boldsymbol{f}(X) - \boldsymbol{1}) \circ (\boldsymbol{f}(X) + \boldsymbol{1})$ is divisible by $\ell_0(X)$ iff 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 3. As our construction substantially deviates from that of [BLNS21] we show that the protocol indeed is a ZKPoK. Perfect completeness is straightforward, so we focus on soundness and special honest-verifier zero-knowledge.

Lemma 1 (Soundness in $\Pi_{\mathbf{AEx}}$). *Let* $\texttt{Encode}$ *be the encoding function of a Reed-Solomon code of dimension $k' = vN + \eta$ and length l. Furthermore, let $k' \leq k \leq l < q$. Suppose that there is an efficient deterministic prover* $\texttt{P}^*$ *convincing an honest verifier in the protocol in Figure 3 on input $\boldsymbol{A}, \boldsymbol{t}_1, \ldots, \boldsymbol{t}_\tau$*

[4] 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. This will be more clear in the soundness proof of our protocol.

with probability

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

Then there exists an efficient probabilistic extractor $\mathcal{E}$ which, given access to $\mathtt{P}^*$ 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 $(\mathtt{Com}_{\mathtt{Aux}}, \mathtt{Open}_{\mathtt{Aux}})$, or finds a hash collision in expected time at most $64T$ where

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

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

The proof for Lemma 1 as well as a proof of the zero-knowledge property are presented in Appendix C. The size of the amortized zero-knowledge proof in Figure 3 in terms of prover-to-verifier communication is

$$|c_{\boldsymbol{d}}| + |r_{\boldsymbol{d}}| + |\mathcal{M}| + (2vN + (3\tau+4)\eta)\log_2 q + \lambda\eta(1+\log_2 l) \text{ bits.} \qquad (2)$$

Theorem 5 (Security of Amortized Zero-Knowledge Proof of Exact Openings). *The amortized zero-knowledge proof of exact openings is* complete *(Definition 15 in 2.7) when the secrets $\boldsymbol{s}_i$ has ternary coefficients, it is* special sound *(Definition 16 in 2.7) if the $\mathsf{SKS}^2_{r,v,1}$ problem is hard (see Lemma 1), and it is statistically* honest-verifier zero-knowledge *(Definition 17 in 2.7).*

3.5 Amortized Zero-Knowledge Proof of Bounded Openings

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.* [BBC+18] give a efficient amortized zero-knowledge proof of knowledge for the relation

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

The protocol $\mathit{\Pi}_{\text{ANEx}}$ is depicted in Figure 4. We can use a challenge matrix $\boldsymbol{C}$ with entries sampled from the set $\mathcal{C}_{\text{ANEx}} = \{0,1\}$, then this allows us to choose the parallel protocol instances $\hat{n} \ge \lambda + 2$ for security parameter λ. Denote by

$$\pi_{\text{ANEx}} \leftarrow \mathit{\Pi}_{\text{ANEx}}(\boldsymbol{S}; (\boldsymbol{A}, \boldsymbol{T}, \sigma_{\text{ANEx}})), \text{ and } 0 \vee 1 \leftarrow \mathit{\Pi}_{\text{ANExV}}((\boldsymbol{A}, \boldsymbol{T}, B_{\text{ANEx}}); \pi_{\text{ANEx}}),$$

the run of the proof and verification protocols, respectively, where the $\mathit{\Pi}_{\text{ANEx}}$-protocol, using Fiat-Shamir, produces a proof of the form $\pi_{\text{ANEx}} = (\boldsymbol{C}, \boldsymbol{Z})$, where $\boldsymbol{C}$ is the output of a hash-function, and the $\mathit{\Pi}_{\text{ANExV}}$-protocol consists of the two checks in the last step in Figure 4. $\mathcal{N}_{\sigma_{\text{ANEx}}}$ is a Gaussian distribution over $\mathbb{Z}$ with

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

Fig. 4. Protocol Π_{ANEx} is the approximate amortized zero-knowledge proof of knowledge of bounded preimages for matrices and vectors over $\mathbb{Z}_q$.

standard deviation σ_{ANEx}, and the verification bound is $B_{\text{ANEx}} = \sqrt{2N}\sigma_{\text{ANEx}}$. Note that σ_{ANEx}, and hence B_{ANEx}, depends on the norm of $\boldsymbol{S}$ (see Section 2.2). This means that the bound we can prove for each $\|s_{i,j}\|_2$ depends on the number of equations τ in the statement to be proved.

The following theorem for the security of the amortized zero-knowledge proof of bounded openings follows from Baum *et al.* [BBC+18, Lemma 3].

Theorem 6 (Security of Amortized Zero-Knowledge Proof of Bounded Openings). *The amortized zero-knowledge proof of bounded openings is* complete *(Definition 15 in 2.7) if the secrets in* $\boldsymbol{S}$ *are bounded by* $B_{\texttt{Com}}$ *in the* ℓ_∞ *norm, it is* special sound *(Definition 16 in 2.7) if the* $\mathsf{SKS}^2_{n,k,2kB_{\text{ANEx}}}$ *problem is hard, and is statistical* honest-verifier zero-knowledge *(Definition 17 in 2.7). The probability of success is* $1/M$*, as computed in Equation 1.*

Finally, we note that this protocol can be generalized to check for different norms (and using different standard deviation) in each row or column of $\boldsymbol{Y}$ and $\boldsymbol{Z}$ depending on varying norms of the secrets $s_{i,j}$, see [BEPU+20, Section 5] for details.

4 Verifiable Shuffle of BGV Ciphertexts

The recent work by Aranha *et al.* [ABG+21] presents an efficient protocol Π_{SHUF} for a shuffle of openings of lattice-based commitments using zero-knowledge proofs of linear relations.

More concretely, the authors present a proof for the following relation R_{Shuf}:

$$R_{\text{Shuf}} = \left\{ (x,w) \,\middle|\, \begin{array}{l} x = (\llbracket m_1 \rrbracket, \dots, \llbracket m_\tau \rrbracket, \hat{m}_1, \dots, \hat{m}_\tau), \\ w = (\pi, f_1, \dots, f_\tau, \boldsymbol{r}_1, \dots, \boldsymbol{r}_\tau), \pi \in S_\tau, \hat{m}_i \in R_q \\ \forall i \in [\tau]:\ f_i \cdot \llbracket m_{\pi^{-1}(i)} \rrbracket = f_i \cdot \begin{bmatrix} \boldsymbol{c}_{1,\pi^{-1}(i)} \\ c_{2,\pi^{-1}(i)} \end{bmatrix} = \boldsymbol{A}\boldsymbol{r}_i + f_i \cdot \begin{bmatrix} \boldsymbol{0} \\ \hat{m}_i \end{bmatrix} \\ \wedge\ ||\boldsymbol{r}_i[j]|| \le 4\sigma_C\sqrt{N} \end{array} \right\}.$$

In their work, they show the following result:

Theorem 7 (Security of $\Pi_{\textbf{Shuf}}$). *Assume that* $(\texttt{KeyGen}_{\text{C}}, \texttt{Com}, \texttt{Open})$ *is a secure commitment scheme with* Π_{Lin} *as a HVZK Proof of Knowledge of linear relation with soundness error* ϵ*. Then there exists a protocol* Π_{Shuf} *that is an HVZK PoK for the relation* $\mathcal{R}_{\text{Shuf}}$ *with soundness error* $(\tau^\delta + 1)/|R_q| + 4\tau\epsilon$*.*

We now extend their protocol, allowing to verifiably shuffle elements that are vectors in R_q^ℓ instead of the original elements from R_q.

4.1 The Extended Shuffle Protocol

We are now in a situation where both prover and verifier are given a list of commitments $\llbracket \boldsymbol{m}_1 \rrbracket, \dots, \llbracket \boldsymbol{m}_\tau \rrbracket$ as well as potential messages $(\hat{\boldsymbol{m}}_1, \dots, \hat{\boldsymbol{m}}_\tau)$ from R_q^ℓ. The prover additionally obtains openings $\boldsymbol{m}_i, \boldsymbol{r}_i, f_i$ and wants to prove that the set of plaintext elements are the same set as the underlying elements of the commitments for some secret permutation π of the indices in the lists. The protocol of Aranha *et al.* does not work in this setting, as their technique crucially requires that the plaintexts are from R_q. Thus, our goal is to prove

$$R^\ell_{\text{Shuf}} = \left\{ (x,w) \,\middle|\, \begin{array}{l} x = (\llbracket \boldsymbol{m}_1 \rrbracket, \dots, \llbracket \boldsymbol{m}_\tau \rrbracket, \hat{\boldsymbol{m}}_1, \dots, \hat{\boldsymbol{m}}_\tau), \\ w = (\pi, f_1, \dots, f_\tau, \boldsymbol{r}_1, \dots, \boldsymbol{r}_\tau), \pi \in S_\tau, \hat{\boldsymbol{m}}_i \in R_q^\ell \\ \forall i \in [\tau]:\ f_i \cdot \llbracket \boldsymbol{m}_{\pi^{-1}(i)} \rrbracket = f_i \cdot \begin{bmatrix} \boldsymbol{c}_{1,\pi^{-1}(i)} \\ \boldsymbol{c}_{2,\pi^{-1}(i)} \end{bmatrix} = \boldsymbol{A}\boldsymbol{r}_i + f_i \cdot \begin{bmatrix} \boldsymbol{0} \\ \hat{\boldsymbol{m}}_i \end{bmatrix} \\ \wedge\ ||\boldsymbol{r}_i[j]|| \le 4\sigma_C\sqrt{N} \end{array} \right\}.$$

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

Since each commitment $\llbracket \boldsymbol{m} \rrbracket$ can be written as

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

we can write $\boldsymbol{c}_1 = \boldsymbol{A}_1\boldsymbol{r}$ and $\boldsymbol{c}_2 = \boldsymbol{A}_{\boldsymbol{2}}\boldsymbol{r} + \boldsymbol{m}$. From this we can create a new commitment $\llbracket \langle \boldsymbol{\rho}, \boldsymbol{m} \rangle \rrbracket$ under the new commitment key $pk' = (\boldsymbol{A}_1, \rho\boldsymbol{A}_{\boldsymbol{2}})$ where

$\boldsymbol{c}_1' = \boldsymbol{c}_1$ remains the same, while we set $c_2' = \langle \rho, \boldsymbol{c}_2 \rangle$. Note that this does not increase the bound of the randomness of the commitment. Since

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

it holds that

$$\boldsymbol{a}_2' = \boldsymbol{\rho} \boldsymbol{A}_2 = \begin{bmatrix} \boldsymbol{0}^n & \boldsymbol{\rho}^\top & \boldsymbol{\rho} \boldsymbol{A}_2' \end{bmatrix}.$$

It is easy to see that breaking the binding property for pk' is no easier than breaking the binding property for pk.

Proposition 2. *If there exists an efficient attacker $\mathcal{A}$ that breaks the binding property on commitments under the key pk' with probability ϵ, then there exists an efficient algorithm $\mathcal{A}'$ that breaks the binding property on pk with the same probability.*

Proof. Assume that $\mathcal{A}$ outputs two valid $(\boldsymbol{r}_1, m_1, f_1), (\boldsymbol{r}_2, m_2, f_2)$ for pk'. This in particular implies that $f_1 c_2' = \langle \boldsymbol{a}_2', \boldsymbol{r}_1 \rangle + f_1 m_1$ and $f_2 c_2' = \langle \boldsymbol{a}_2', \boldsymbol{r}_2 \rangle + f_2 m_2$. Multiplying the first term with f_2 and the second with f_1 yields that

$$0 = \langle \boldsymbol{a}_2', (f_2 \boldsymbol{r}_1 - f_1 \boldsymbol{r}_2) \rangle + f_1 f_2 (m_1 - m_2).$$

Since $m_1 \neq m_2$ we have that $\langle \boldsymbol{a}_2', f_2 \boldsymbol{r}_1 - f_1 \boldsymbol{r}_2 \rangle \neq 0$.

Let $\boldsymbol{m}_1 = (f_1 c_2 - \boldsymbol{A}_2 r_1)/f_1$ and $\boldsymbol{m}_2 = (f_2 c_2 - \boldsymbol{A}_2 r_2)/f_2$. It is clear that $(\boldsymbol{r}_1, \boldsymbol{m}_1, f_1)$ and $(\boldsymbol{r}_2, \boldsymbol{m}_2, f_2)$ are valid openings for pk. We need to show that indeed $\boldsymbol{m}_1 \neq \boldsymbol{m}_2$.

Towards this, assume that $\boldsymbol{m}_1 = \boldsymbol{m}_2$. By multiplying the definition with $f_1 f_2$ we get that $\boldsymbol{0} = \boldsymbol{A}_2 (f_1 \boldsymbol{r}_2 - f_2 \boldsymbol{r}_1)$. This implies that

$$\langle \boldsymbol{\rho}, \boldsymbol{A}_2 (f_1 \boldsymbol{r}_2 - f_2 \boldsymbol{r}_1) \rangle = \langle \boldsymbol{\rho} \boldsymbol{A}_2, f_1 \boldsymbol{r}_2 - f_2 \boldsymbol{r}_1 \rangle = \langle \boldsymbol{a}_2', f_1 \boldsymbol{r}_2 - f_2 \boldsymbol{r}_1 \rangle = 0$$

which is a contradiction. □

Shuffle protocol. Given the above, we can now construct the protocol Π_{SHUF}^{ℓ}:

1. Initially, $\mathtt{P}$ and $\mathtt{V}$ hold $\{[\![\boldsymbol{m}_i]\!], \hat{\boldsymbol{m}}_i\}_{i \in [\tau]}$ for a public key $pk = (\boldsymbol{A}_1, \boldsymbol{A_2})$ while the prover additionally has $\{\boldsymbol{m}_i, \boldsymbol{r}_i\}_{i \in [\tau]}, \pi \in S_\tau$.
2. $\mathtt{V}$ chooses $h \leftarrow\$ R_q$ and sends it to $\mathtt{P}$. Both parties compute $\boldsymbol{\rho} \leftarrow (1, h, \ldots, h^{\ell-1})^\top$.
3. $\mathtt{P}$ and $\mathtt{V}$ for each $[\![\boldsymbol{m}_i]\!] = (\boldsymbol{c}_{1,i}, \boldsymbol{c}_{2,i})$ compute $[\![\langle \boldsymbol{\rho}, \boldsymbol{m}_i \rangle]\!] = (\boldsymbol{c}_{1,i}, \langle \boldsymbol{\rho}, \boldsymbol{c}_{2,i} \rangle)$.
4. $\mathtt{P}, \mathtt{V}$ now run Π_{SHUF} on input commitments $\{[\![\langle \boldsymbol{\rho}, \boldsymbol{m}_i \rangle]\!]\}_{i \in [\tau]}$ and messages $\langle \boldsymbol{\rho}, \boldsymbol{m}_i \rangle$. $\mathtt{P}$ uses the same permutation π, randomness $\boldsymbol{r}_i$ as before. The commitment key $pk' = (\boldsymbol{A}_1, \boldsymbol{\rho} \boldsymbol{A}_2)$ is used by both parties.
5. If the protocol Π_{SHUF} accepts then $\mathtt{V}$ accepts, otherwise he rejects.

We now show the following:

Lemma 2 (Soundness in $\Pi_{\mathbf{Shuf}}^{\ell}$). *Assume that Π_{SHUF} that is an HVZK Proof of Knowledge for the relation $\mathcal{R}_{\text{SHUF}}$ with soundness error ϵ'. Then Π_{SHUF}^{ℓ} is a HVZK PoK for the relation R_{SHUF}^{ℓ} with soundness error $\epsilon = 2\epsilon' + 3 \left(\frac{\ell - 1}{q} \right)^N$.*

Proof. Completeness and Zero-Knowledge of Π^{ℓ}_{SHUF} follow immediately from the same properties of Π_{SHUF}. Thus, we focus now on knowledge soundness.

Let $\mathtt{P}^*$ be a prover that convinces a verifier on input x with probability $\nu > \epsilon$. For the proof, we will use the standard definition of proof of knowledge where there must exist an extractor $\mathcal{E}$ that succeeds with black-box access to $\mathtt{P}^*$ running in expected time $p(|x|)/(\nu - \epsilon)$ where p is a polynomial.

Towards constructing a simulator $\mathcal{E}$ we know that there exists an extractor $\mathcal{E}'$ for Π_{SHUF}. We construct $\mathcal{E}$ as the following loop, which restarts whenever the loop "aborts":

1. Run random protocol instances with $\mathtt{P}^*$ until a valid protocol instance with challenge h was generated. Do this at most $2/\epsilon$ steps, otherwise abort.
2. Run $\mathcal{E}'$ with the fixed h with $\mathtt{P}^*$ until it outputs $\pi, \{f_i, \boldsymbol{r}_i\}_{i\in[\tau]}$. If $\mathcal{E}'$ aborts, then abort. In parallel, start a new loop instance until $\mathcal{E}'$ finishes.
3. Let $\tilde{\boldsymbol{m}}_i = (f_i \boldsymbol{c}_{2,i} - \boldsymbol{A}_2 \boldsymbol{r}_i)/f_i$. If $\tilde{\boldsymbol{m}}_{\pi^{-1}(i)} = \hat{\boldsymbol{m}}_i$ for all $i \in [\tau]$ then output $\pi, \{f_i, \boldsymbol{r}_i\}_{i\in[\tau]}$, otherwise abort.

First, by the definition we observe that $\tilde{\boldsymbol{m}}_i = (f_i \boldsymbol{c}_{2,i} - \boldsymbol{A}_2 \boldsymbol{r}_i)/f_i$ is well-defined because f_i is invertible. If $\mathcal{E}$ outputs a value, then the output of $\mathcal{E}$ is a witness for the relation R^{ℓ}_{SHUF}. We now show a bound on the expected time per loop-instance, and that each loop with constant probability outputs a valid witness.

In the first step, we expect to find an accepting transcript after $1/\epsilon$ steps. Since we run this step for $2/\epsilon$ iterations, we will have found an accepting transcript with probability at least $1/2$ by Markov's inequality. Consider the matrix $\boldsymbol{H}$ where the rows are indexed by all choices h and the columns by the choices of the used shuffle proof. Then, by the heavy-row lemma [Dam10], with probability $\geq 1/2$ we will have chosen a value h such that the row of $\boldsymbol{H}$ contains $\epsilon/2 > \epsilon'$ 1s. In that case, $\mathcal{E}'$ will by definition output a valid witness in an expected number of $p(|x|)/(\nu - \epsilon') < p(|x|)/(\nu - \epsilon)$ steps, which is within the runtime budget. For the case it gets stuck, we start another loop which we run in parallel. Once $\mathcal{E}'$ has found an opening, then the computation in Step 3 is inexpensive. We now have to compute the abort probability of this step.

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

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

Assume that $\mathcal{E}'$ extracts a valid witness $\pi, \{f_i, \boldsymbol{r}_i\}_{i\in[\tau]}$ for input commitments $\{[\![\langle \boldsymbol{\rho}, \boldsymbol{m}_i\rangle]\!]\}_{i\in[\tau]}$ and messages $\langle \boldsymbol{\rho}, \hat{\boldsymbol{m}}_i\rangle$ while the extracted $\tilde{\boldsymbol{m}}_i = (f_i\boldsymbol{c}_{2,i} - \boldsymbol{A}_2\boldsymbol{r}_i)/f_i$ do not form a permutation on the $\hat{\boldsymbol{m}}_i$. Then there exists an $i \in [\tau]$ such that

$$f_i \cdot \langle \rho, \boldsymbol{c}_{2,\pi^{-1}(i)}\rangle = \langle \rho\boldsymbol{A}_2, \boldsymbol{r}_i\rangle + f_i\langle \boldsymbol{\rho}, \hat{\boldsymbol{m}}_i\rangle$$

but

$$f_i \cdot \boldsymbol{c}_{2,\pi^{-1}(i)} = \boldsymbol{A}_2\boldsymbol{r}_i + f_i(\hat{\boldsymbol{m}}_i + \boldsymbol{\delta})$$

where $\tilde{\boldsymbol{m}}_i = \hat{\boldsymbol{m}}_i + \boldsymbol{\delta}$ for a non-zero vector $\boldsymbol{\delta}$. Combining both equations, we get that $\boldsymbol{0} = \langle \boldsymbol{\rho}, \boldsymbol{\delta}\rangle$. This implies that the polynomial $\sum_{i=0}^{\ell-1} \boldsymbol{\delta}[i]X^i$ that has coefficients from $\boldsymbol{\delta}$ must be zero at point h whose powers generate the vector $\boldsymbol{\rho}$. Since this polynomial is of degree $\ell - 1$, by [ABG+21, Lemma 2] it can be 0 in at most $(\ell - 1)^N$ positions without being the 0-polynomial itself. But since the transcript is extractable and thus accepting for strictly more than $(\ell - 1)^N$ choices of h (by our choice of the "default witness" from above), we must have that $\boldsymbol{\delta}$ was $\boldsymbol{0}$ to begin with. Therefore, Step 3 only aborts if we stumble upon a witness $\pi, \{f_i, \boldsymbol{r}_i\}_{i\in[\tau]}$ for R_{SHUF} that is not the "default witness" which occurs with constant probability only. □

Notation and communication. We denote by

$$\begin{aligned}
&\pi_{\text{SHUF}} \leftarrow \mathit{\Pi}^{\ell}_{\text{SHUF}}((\{(\boldsymbol{m}_i, \boldsymbol{r}_i)\}_{i=1}^{\tau}, h); (\{[\![\boldsymbol{m}_i]\!]\}_{i=1}^{\tau}, \{\hat{\boldsymbol{m}}_i\}_{i=1}^{\tau})), \text{ and}\\
&0 \vee 1 \leftarrow \mathit{\Pi}^{\ell}_{\text{SHUFV}}((\{[\![\boldsymbol{m}_i]\!]\}_{i=1}^{\tau}, \{\hat{\boldsymbol{m}}_i\}_{i=1}^{\tau}); \pi_{\text{SHUF}})
\end{aligned}$$

the run of the proof and verification protocols of the shuffle, respectively.

We refer to Section 7 for sizes, parameters and a concrete instantiation of the shuffle protocol, as parameters depend on the full mix-net protocol and the decryption protocol, such as the number of servers involved.

4.2 Verifiable Shuffle of BGV Ciphertexts

The following mixing protocol is for the relation R_{MIX}:

$$R_{\text{MIX}} = \left\{ (x, w) \left| \begin{array}{l} x = (\boldsymbol{A}'', \boldsymbol{c}_1, \ldots, \boldsymbol{c}_\tau, \hat{\boldsymbol{c}}_1, \ldots, \hat{\boldsymbol{c}}_\tau, [\![\boldsymbol{c}'_1]\!], \ldots, [\![\boldsymbol{c}'_\tau]\!]),\\ w = (\pi, \boldsymbol{r}'_1, \ldots, \boldsymbol{r}'_\tau,), \pi \in S_\tau,\\ \forall i \in [\tau] : \boldsymbol{c}_i = \mathsf{Enc}(\mathsf{pk}, m_i), \boldsymbol{c}'_i = \mathsf{Enc}(\mathsf{pk}, 0),\\ [\![\boldsymbol{c}'_i]\!] = \boldsymbol{A}''\boldsymbol{r}'_i, \|\boldsymbol{r}'_i\|_\infty \leq B_\infty, \hat{\boldsymbol{c}}_{\pi(i)} = \boldsymbol{c}_i + \boldsymbol{c}'_i \end{array} \right. \right\}.$$

If the noise-level in all $\boldsymbol{c}_i$ and $\boldsymbol{c}'_i$ are bounded by $B_{\mathtt{Dec}}$, and $2B_{\mathtt{Dec}} < \lfloor q/2 \rfloor$, then all $\boldsymbol{c}_i$ and $\hat{\boldsymbol{c}}_{\pi(i)}$ will, for some permutation π, decrypt to the same message m_i.

Public parameters. Let $p \ll q$ be primes, let R_q and R_p be defined as above for a fixed N, let $\mathtt{D}$ be a bounded distribution over R_q, and let $B_\infty \in \mathbb{N}$ be a bound. We assume properly generated keys and ciphertexts according to the `KeyGen` and `Enc` algorithms in Section 3.1.

Let $\mathcal{S}$ be the shuffle algorithm. Then $\mathcal{S}$ takes as input a set of τ publicly known BGV ciphertexts $\{\boldsymbol{c}_i\}_{i=1}^{\tau}$, where each ciphertext is of the form

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

where m_i in R_p is the encrypted message, $r_i \leftarrow\!\!\$ \; R_q$ is a short element such that $\|r_i\|_\infty \leq B_\infty$, and $e_{i,1}, e_{i,2} \leftarrow \mathtt{D}$ is some bounded random noise ensuring that the total noise in the ciphertext is bounded by $B_{\mathtt{Dec}}$.

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

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

where $r_i', e_{i,1}', e_{i,2}'$ are chosen as above. This corresponds to creating fresh, independent encryptions of 0. Observe that $\mathcal{S}$ 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 3.2 for $\ell = 2$ and add the public key of the encryption scheme to get a $(n+2) \times (k+3)$ commitment matrix $\boldsymbol{A}''$, 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,

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

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

$$[\![(\hat{u}_i, \hat{v}_i)]\!] = [\![(u_i, v_i)]\!]_{\boldsymbol{0}} + [\![(u_i', v_i')]\!].$$

Thus, the commitments $[\![(\hat{u}_i, \hat{v}_i)]\!]$ 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 $\mathit{\Pi}_{\textsc{Shuf}}^{\ell}$. To ensure correctness we have to additionally show that each u_i', v_i' was created such that decryption is still correct.

Proving correctness of commitments. Let $\boldsymbol{A}''$ 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:

$$\boldsymbol{t}_i = \boldsymbol{A}''\boldsymbol{r}_i' = [\![(ar_i' + pe_{i,1}', br_i' + pe_{i,2}')]\!], \qquad \|\boldsymbol{r}_i'\|_\infty \leq B_\infty.$$

To show this, $\mathcal{S}$ runs the $\mathit{\Pi}_{\textsc{AEx}}$-protocol for the inputs $\boldsymbol{A}'', \{\boldsymbol{r}_i'\}_{i=1}^{\tau}, \{\boldsymbol{t}_i\}_{i=1}^{\tau}$. Here, $\mathcal{S}$ uses the Fiat-Shamir transform to ensure non-interactivity of the proof.

We summarize the aforementioned in the following protocol $\mathit{\Pi}_{\textsc{Mix}}$:

1. $\mathcal{S}$ obtains as input the 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 mentioned above. It then creates commitments $\{[\![u'_i, v'_i]\!] = [\![ar'_i + pe'_{i,1}, br'_i + pe'_{i,2}]\!]\}_{i\in[\tau]}$ using randomness $\boldsymbol{r}_i$ for each such commitment.
3. Let $\boldsymbol{t}_i = [\![u'_i, v'_i]\!]$ and $\boldsymbol{r}'_i = [\boldsymbol{r}_i^\top, r'_i, e_{i,1}, e_{i,2}]^\top$. Then $\mathcal{S}$ computes $\pi_{\text{AEx}} \leftarrow \mathit{\Pi}_{\text{AEx}}$ for matrix $\boldsymbol{A}''$, 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 $\pi_{\text{Shuf}} \leftarrow \mathit{\Pi}^{\ell}_{\text{Shuf}}$ with input commitments $\{[\![(\hat{u}_i, \hat{v}_i)]\!]\}_{i\in[\tau]}$, commitment messages $\{(u_i, v_i)\}_{i\in[\tau]}$, commitment randomness $\{\boldsymbol{r}_i\}_{i\in[\tau]}$ and openings L.
5. $\mathcal{S}$ outputs $(\{\boldsymbol{t}_i\}_{i\in[\tau]}, \pi_{\text{AEx}}, L, \pi_{\text{Shuf}})$.

Given such a string $(\{\boldsymbol{t}_i\}_{i\in[\tau]}, \pi_{\text{AEx}}, L, \pi_{\text{Shuf}})$ 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 $\mathit{\Pi}_{\text{MixV}}$ to verify the mix step:

1. Run the verification algorithm of $\mathit{\Pi}_{\text{AExV}}$ for π_{AEx} on inputs $\boldsymbol{A}'', \{\boldsymbol{t}_i\}_{i\in[\tau]}$ and B. If the verification fails, then output 0.
2. For all $i \in [\tau]$ set $[\![(\hat{u}_i, \hat{v}_i)]\!] = [\![(u_i, v_i)]\!]_{\mathbf{0}} + \boldsymbol{t}_i$ where $(u_i, v_i) = \boldsymbol{c}_i$.
3. Run the verification algorithm of $\mathit{\Pi}^{\ell}_{\text{ShufV}}$ for π_{Shuf} on input $\{[\![(\hat{u}_i, \hat{v}_i)]\!]\}_{i\in[\tau]}, L$. If the verification fails, then output 0.
4. Output 1.

The following theorems refer to definitions of correctness, soundness and honest-verifier zero-knowledge given in Section 2.7. The protocol $\mathit{\Pi}_{\text{Mix}}$ is essentially a re-randomization and shuffle of a set of ciphertexts augmented with some commitments and zero-knowledge proofs, carefully composed to give security.

In the following theorems, define the noise bound B_{Dec} to be the maximum level of noise in a ciphertexts $\boldsymbol{c}'_i = \mathsf{Enc}(\mathsf{pk}, m'_i)$ when the randomness $r'_i, e'_{i,1}, e'_{i,2}$ used to create the ciphertexts is bounded by B_∞ and m'_i is bounded by p in the ℓ_∞ norm. Let B_{Dec} satisfy $2B_{\mathsf{Dec}} < \lfloor q/2 \rfloor$.

Theorem 8 (Correctness). *Let input ciphertexts have noise bounded by B_{Dec}, and let the total noise added in $\mathit{\Pi}_{\text{Mix}}$ be bounded by B_{Dec}. Suppose the protocols $\mathit{\Pi}_{\text{AEx}}$ and $\mathit{\Pi}^{\ell}_{\text{Shuf}}$ are complete. Then the mixing protocol is correct.*

We sketch the argument. Since $2B_{\mathsf{Dec}} < \lfloor q/2 \rfloor$, it follows that decryption is correct. Furthermore, since $\mathit{\Pi}_{\text{AEx}}$ and $\mathit{\Pi}^{\ell}_{\text{Shuf}}$ are complete, the arguments will be accepted, which means that the mixing proof will be accepted.

Theorem 9 (Special Soundness). *Let $\mathcal{E}_1$ be a knowledge extractor for the protocol $\mathit{\Pi}_{\text{AEx}}$ with success probability ϵ_1 and let $\mathcal{E}_2$ be a knowledge extractor for the protocol $\mathit{\Pi}^{\ell}_{\text{Shuf}}$ with success probability ϵ_2. Then we can construct a knowledge extractor $\mathcal{E}_0$ for the mixing protocol that succeeds with probability $\epsilon_0 \geq \epsilon_1 + \epsilon_2$. The runtime of $\mathcal{E}_0$ is essentially the same as the runtime of $\mathcal{E}_1$ and $\mathcal{E}_2$.*

We sketch the argument. The main observation is that it is enough that either of the extractors $\mathcal{E}_1$ and $\mathcal{E}_2$ succeeds.

If the extractor $\mathcal{E}_1$ succeeds, we are able to extract τ randomness vectors $\boldsymbol{r}'_i$ bounded by B_∞, which gives us the randomness for both the commitments and ciphertexts used in the protocol. Then we can directly extract the permutation π by inspection, and hence, we have an extractor $\mathcal{E}_0$ for the mixing protocol Π_{MIX}.

If the extractor $\mathcal{E}_1$ succeeds, we are able to extract both the permutation π and τ randomness vectors $\boldsymbol{r}_i$ used in the commitments and, indirectly, also the committed randomness used to create the encryption of 0. Hence, we have an extractor $\mathcal{E}_0$ for the mixing protocol Π_{MIX}.

If neither of these strategies works, we have an attacker against the binding property of the commitment scheme.

Theorem 10 (Honest-Verifier Zero-Knowledge). *Suppose the protocol Π_{AEX} and Π^{ℓ}_{SHUF} are honest-verifier zero-knowledge, that* `Com` *is hiding and that* `Enc` *is CPA secure. Then there exists a simulator for the mixing protocol such that for any distinguisher $\mathcal{A}_0$ for this simulator with advantage ϵ_0, there exists an adversary $\mathcal{A}_3$ against hiding for the commitment scheme with advantage ϵ_3, an adversary $\mathcal{A}_4$ against CPA security for the encryption scheme with advantage ϵ_4, and distinguishers $\mathcal{A}_1$ and $\mathcal{A}_2$ for the simulators for Π_{AEX} and Π^{ℓ}_{SHUF}, respectively, with advantages ϵ_1 and ϵ_2, such that $\epsilon_0 \leq \epsilon_1+\epsilon_2+\epsilon_3+\epsilon_4$. The runtime of $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3, \mathcal{A}_4$ are essentially the same as $\mathcal{A}_0$.*

We sketch the argument. The simulator is given the set of messages encrypted by the input ciphertexts. The simulator simulates the zero-knowledge proofs Π_{AEX} and Π^{ℓ}_{SHUF} using the appropriate simulators. It replaces the commitments to the ciphertexts (u'_i, v'_i) by random commitments and the output ciphertexts by fresh 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 by simulated arguments, which gives us a distinguisher $\mathcal{A}_2$ for the Π_{LIN} honest verifier simulator.

Second, we replace the Π_{AEX} arguments by simulated arguments, which gives us a distinguisher $\mathcal{A}_1$ for the Π_{AEX} honest verifier simulator.

Third, we replace the commitments to ciphertexts by random commitments, which gives us an adversary $\mathcal{A}_3$ against hiding for the commitment scheme.

Fourth, we replace the output ciphertexts by fresh ciphertexts, which gives us an adversary $\mathcal{A}_4$ against CPA security.

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

5 Verifiable Distributed Decryption

In this section we provide a construction for verifiable distributed decryption. We combine the distributed decryption protocol from Section 3.1 with zero-knowledge proofs to achieve an actively secure decryption protocol. In this protocol, the set of ciphertexts is given to a number of decryption servers each holding a share of the secret key. All of the servers compute a partial decryption of each ciphertext. Finally, they use noise drowning to hide the secret key and their published shares are added and rounded to output the plaintext. The partial decryption is a linear operation, and we prove correctness using zero-knowledge proofs of linear relations from Section 3.3, and we use the amortized proof from Section 3.5 to prove that the noise is bounded to ensure correct decryption. Both proofs are computed using the commitment scheme from Section 3.2. We also provide an optimistic distributed decryption protocol given in Appendix A which is secure if not all decryption servers are colluding simultaneously.

5.1 The Actively Secure Protocol

Public parameters. Let the ring R_q, the bounded distribution $\mathtt{D}$ over R_q, the statistical security parameter $\mathtt{sec}$ and error bounds $B_{\mathtt{Com}}$ and $B_{\mathtt{Dec}}$ be public system information, together with the plaintext modulus p for the encryption system. Let $\boldsymbol{A}$ be the public commitment matrix for message size $\ell = 1$ over R_q.

- $\mathtt{KeyGen}_A(1^\lambda, \xi)$:
  1. Compute $(\mathsf{pk}, \mathsf{sk}, s_1, \ldots, s_\xi) \leftarrow \mathtt{KeyGen}(1^\lambda, \xi)$ as in the passive protocol.
  2. For each $j \in [\xi]$ compute $([\![s_j]\!], \boldsymbol{d}_j) \leftarrow \mathtt{Com}(s_j)$.
  3. Output $\mathsf{pk}_A = (\mathsf{pk}, [\![s_1]\!], \ldots, [\![s_\xi]\!])$, $\mathsf{sk}_A = \mathsf{sk}$ and $\mathsf{sk}_{A,j} = (s_j, \boldsymbol{d}_j)$.

- $\mathtt{Enc}_A$ and $\mathtt{Dec}_A$ works just like the original $\mathtt{Enc}$ and $\mathtt{Dec}$ in the passively secure threshold encryption scheme, ignoring additional information in pk_A.

- $\mathtt{DistDec}(\mathsf{sk}_{A,j}, \{c_i\}_{i\in[\tau]})$ where $c_i = (u_i, v_i)$:
  1. For each $i \in [\tau]$ do the following. First, compute $m_{i,j} = s_j u_i$.
  2. Sample uniform noise $E_{i,j} \leftarrow R_q$ such that $\|E_{i,j}\|_\infty \leq 2^{\mathtt{sec}}(B_{\mathtt{Dec}}/p\xi)$.
  3. Compute the message decryption share $t_{i,j} = m_{i,j} + pE_{i,j}$.
  4. Compute $([\![E_{i,j}]\!], \boldsymbol{r}''_{i,j}) \leftarrow \mathtt{Com}(E_{i,j})$ and use the $\mathit{\Pi}_{\text{LIN}}$-protocol to compute a proof for the linear relation $t_{i,j} = s_j u_i + pE_{i,j}$ by computing
  $$\pi_{L_{i,j}} \leftarrow \mathit{\Pi}_{\text{LIN}}(((s_j, \boldsymbol{r}_j), (E_{i,j}, \boldsymbol{r}''_{i,j})); ([\![s_j]\!], [\![E_{i,j}]\!], t_{i,j}), (u_i, p)).$$
  5. The commitment to $E_{i,j}$ is of the form
  $$[\![E_{i,j}]\!] = \begin{bmatrix} \boldsymbol{I}_n & \boldsymbol{a}_{1,1} & \boldsymbol{A}_{1,2} \\ \boldsymbol{0}^n & 1 & \boldsymbol{a}_{2,2} \end{bmatrix} \cdot \boldsymbol{r}''_{i,j} + \begin{bmatrix} 0 \\ E_{i,j} \end{bmatrix}$$
  $$= \underbrace{\begin{bmatrix} \boldsymbol{I}_n & \boldsymbol{a}_{1,1} & \boldsymbol{A}_{1,2} & 0 \\ \boldsymbol{0}^n & 1 & \boldsymbol{a}_{2,2} & 1 \end{bmatrix}}_{\boldsymbol{A}''} \begin{bmatrix} \boldsymbol{r}''_{i,j} \\ E_{i,j} \end{bmatrix},$$
  where $\left\|\boldsymbol{r}''_{i,j}\right\|_\infty \leq B_{\mathtt{Com}}$ is the randomness used in the commitments. Run the $\mathit{\Pi}_{\text{ANEx}}$-protocol, denoted as $\mathit{\Pi}_{\text{ANEx}}(\{(E_{i,j}, \boldsymbol{r}''_{i,j}\}_{i\in[\tau]}); (\boldsymbol{A}'', \{[\![E_{i,j}]\!]\}_{i\in[\tau]}))$, and obtain the amortized zero-knowledge proof of knowledge $\pi_{\text{ANEx}_j} = (\boldsymbol{C}'', \boldsymbol{Z}'')$ with binary challenge matrix $\boldsymbol{C}''$.
  6. Output $\mathtt{ds}_j = (\{t_{i,j}\}_{i=1}^\tau, \pi_{\mathcal{D}_j})$ where $\pi_{\mathcal{D}_j} = (\{[\![E_{i,j}]\!]\}_{i=1}^\tau, \{\pi_{L_{i,j}}\}_{i=1}^\tau, \pi_{\text{ANEx}_j})$.

– $\mathtt{Comb}_A(\{c_i\}_{i=1}^{\tau}, \{\mathtt{ds}_j\}_{j\in[\xi]})$:
  1. Parse $\mathtt{ds}_j$ as $(\{t_{i,j}\}_{i=1}^{\tau}, \pi_{\mathcal{D}_j})$.
  2. Verify the proofs $\pi_{L_{i,j}}$: $\quad 0 \vee 1 \leftarrow \mathit{\Pi}_{\text{LinV}}([\![s_j]\!], [\![E_{i,j}]\!], t_{i,j}), (u_i, p); \pi_{L_{i,j}})$.
  3. Verify the proofs π_{ANEx_j}: $\quad 0 \vee 1 \leftarrow \mathit{\Pi}_{\text{ANExV}}(\boldsymbol{A}'', \{[\![E_{i,j}]\!]\}_{i\in[\tau]}; \pi_{\text{ANEx}_j})$.
  4. If any verification protocol returned 0 then output $\perp$. Otherwise compute

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

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

Remark 1. The randomness $\boldsymbol{r}''_{i,j}$ has much smaller ℓ_∞ norm than $E_{i,j}$, and hence, we will run the $\mathit{\Pi}_{\text{ANEx}}$ protocol with small standard deviation σ_{ANEx} for rows 1 to k, while row $k+1$ will have large standard deviation $\hat{\sigma}_{\text{ANEx}}$, as noted in 3.5.

The following theorems refer to definitions of threshold correctness, threshold verifiability and distributed decryption simulatability given in Section 2.5. It is important to understand that the protocol is essentially a passively secure distributed decryption protocol augmented with some commitments and some zero-knowledge proofs, carefully composed to give active security.

In the following three theorems, let the noise bounds $B_{\mathtt{Dec}}$ and $\hat{B}_{\text{ANEx}}$ satisfy $(1 + B_{\mathtt{Dec}}) \cdot 2^{\text{sec}} < 2\hat{B}_{\text{ANEx}} < \lfloor q/2 \rfloor$.

Theorem 11 (Threshold Correctness). *Let ciphertexts have noise bounded by $B_{\mathtt{Dec}}$, and let the total noise added in* $\mathtt{DistDec}$ *be bounded by $2^{\text{sec}} B_{\mathtt{Dec}}$. Suppose the passively secure protocol is threshold correct and the protocols $\mathit{\Pi}_{\text{Lin}}$ and $\mathit{\Pi}_{\text{ANEx}}$ are complete. Then the actively secure protocol is threshold correct.*

We sketch the argument. Since $B_{\mathtt{Dec}} + 2^{\text{sec}} B_{\mathtt{Dec}} < q/2$, it follows that decryption is correct. Furthermore, since $(1 + B_{\mathtt{Dec}}) \cdot 2^{\text{sec}} < 2\hat{B}_{\text{ANEx}} < q/2$ and $\mathit{\Pi}_{\text{Lin}}$ and $\mathit{\Pi}_{\text{ANEx}}$ are complete, the arguments will be accepted, which means that the decryption proof will be accepted.

Theorem 12 (Threshold Verifiability). *Let $\mathcal{A}_0$ be an adversary against threshold verifiability for the actively secure protocol with advantage ϵ_0. Then there exists adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ against soundness for $\mathit{\Pi}_{\text{Lin}}$ and $\mathit{\Pi}_{\text{ANEx}}$, respectively, with advantages ϵ_1 and ϵ_2, such that $\epsilon_0 \leq \epsilon_1 + \epsilon_2$. The runtime of $\mathcal{A}_1$ and $\mathcal{A}_2$ are essentially the same as the runtime of $\mathcal{A}_0$.*

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

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 norm at most $2\hat{B}_{\text{ANEx}}$.

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 $\mathcal{A}_1$ against soundness for $\mathit{\Pi}_{\text{Lin}}$. Similar for the latter case and $\mathit{\Pi}_{\text{ANEx}}$.

Theorem 13 (Distributed Decryption Simulatability). *Suppose the passively secure protocol is simulatable and $\mathit{\Pi}_{\text{LIN}}$ and $\mathit{\Pi}_{\text{ANEx}}$ are honest-verifier zero-knowledge. Then there exists a simulator for the actively secure protocol such that for any distinguisher $\mathcal{A}_0$ for this simulator with advantage ϵ_0, there exists an adversary $\mathcal{A}_4$ against hiding for the commitment scheme*[5]*, with advantage ϵ_4, and distinguishers $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ for the simulators for the passively secure protocol, $\mathit{\Pi}_{\text{LIN}}$ and $\mathit{\Pi}_{\text{ANEx}}$, respectively, with advantages ϵ_1, ϵ_2 and ϵ_3, such that $\epsilon_0 \leq \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4$. The runtime of $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ and $\mathcal{A}_4$ are essentially the same as the runtime of $\mathcal{A}_0$.*

We sketch the argument. The simulator simulates the arguments and the passively secure distributed decryption algorithm, using the appropriate simulators. Also, it replaces the commitments to the noise $E_{i,j}$ by random commitments.

The claim about the simulator follows from a straight-forward hybrid argument. We begin with the distributed decryption algorithm.

First, we replace the $\mathit{\Pi}_{\text{LIN}}$ arguments by simulated arguments, which gives us a distinguisher $\mathcal{A}_2$ for the $\mathit{\Pi}_{\text{LIN}}$ honest verifier simulator.

Second, we replace the $\mathit{\Pi}_{\text{ANEx}}$ arguments by simulated arguments, which gives us a distinguisher $\mathcal{A}_3$ for the $\mathit{\Pi}_{\text{ANEx}}$ honest verifier simulator.

Third, we replace the commitments to the noise $E_{i,j}$ by random commitments, which gives us an adversary $\mathcal{A}_4$ against hiding for the commitment scheme.

Fourth, we replace the passively secure distributed decryption algorithm by its simulator, which gives us a distinguisher $\mathcal{A}_1$ for the simulator.

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

We refer to Section 7 for sizes, parameters and a concrete instantiation of the distributed decryption, combining it with the mix-net described in Section 4.

6 A Cryptographic Voting System

Our voting protocol follows a fairly natural design paradigm of mixing and threshold decryption. Common voting scheme security definitions such as [BCG+15] do not model shuffles and distributed decryption. Following other works such as e.g. [Scy, Gjø11] we therefore define *ad hoc* security definitions. The high-level architecture for the counting phase of our protocol is shown in Figure 5.

We follow the standard approach for voting system analysis, which is to consider a voting system to be fairly simple cryptographic protocol built on top of a *cryptographic voting scheme.*

The *verifiable* cryptographic voting scheme *with return codes*, *shuffles* and *distributed decryption* is described in Appendix B. It uses our shuffle and verifiable decryption as described previously as well as other primitives. We now explain how our voting system can be described as a simple protocol on top of

[5] 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.

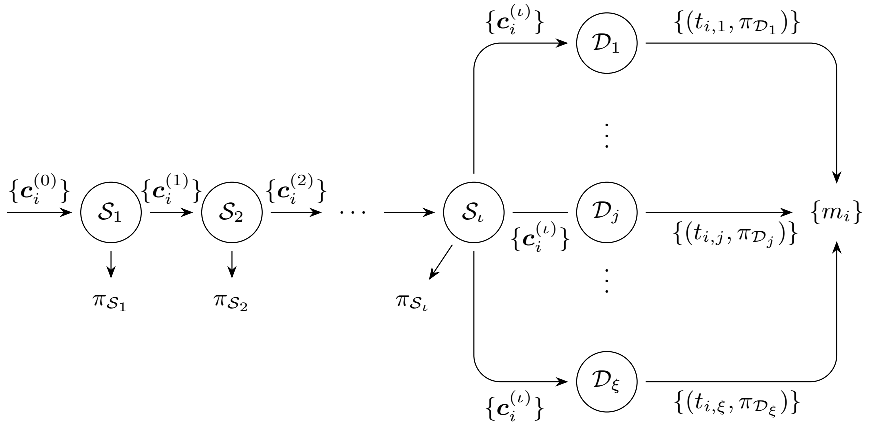
$\{\boldsymbol{c}_i^{(0)}\}$
$\mathcal{S}_1$
$\{\boldsymbol{c}_i^{(1)}\}$
$\mathcal{S}_2$
$\{\boldsymbol{c}_i^{(2)}\}$
$\cdots$
$\mathcal{S}_\iota$
$\pi_{\mathcal{S}_1}$
$\pi_{\mathcal{S}_2}$
$\pi_{\mathcal{S}_\iota}$
$\{\boldsymbol{c}_i^{(\iota)}\}$
$\mathcal{D}_1$
$\{(t_{i,1}, \pi_{\mathcal{D}_1})\}$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

Fig. 5. 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 $\{t_{i,j}\}$ together with a proof of correctness $\pi_{\mathcal{D}_j}$. All votes can be reconstructed to $\{m_i\}$ from the partial decryptions. The full voting protocol also includes proofs of known messages from voters and a return code protocol for verifiability, see details in the Appendix.

this cryptographic voting scheme. We define security notions for this cryptosystem, sketch the security proof, and then informally discuss the voting protocol's security properties in terms of the cryptosystem's security in the Appendix.

6.1 Voting Protocol

The voting protocol requires a *trusted set of players* to run setup and registration, a set of *voters* $\mathcal{V}_i$ and their *computers* P_i, a *ballot box* $\mathcal{B}$, a *return code generator* $\mathcal{R}$, a collection of *shuffle servers* $\mathcal{S}_k$, a collection of *decryption servers* $\mathcal{D}_j$ and one or more *auditors* $\mathcal{A}$.

In the *setup phase*, a trusted set of players runs the setup algorithm. The key generation can either be done in a trusted fashion, or in a distributed fashion using the protocol by Rotaru *et al.* [RST+22] to get an actively secure robust threshold sharing of the secret decryption key. The derived public parameters are given to every participant, while the decryption key shares are given to the decryption servers $\mathcal{D}_j$. The code key is given to the return code generator $\mathcal{R}$, who will use the key to derive so-called *return codes* [CES02,HR16] that are sent to the voter. (As detailed in the Appendix, these codes are human-verifiable and can allow the voter to detect a cheating computer tampering with ballots.)

In the *registration phase*, a set of trusted players run the register algorithm to generate verification and casting keys for each voter $\mathcal{V}_i$, making every verification key public and giving the voter casting key to the voter's computer. Then the return code generator chooses a PRF-key, and a set of trusted players compute the return code table. The voter gets the return code table.

In the *casting phase*, each voter $\mathcal{V}_i$ instructs its computer P_i which ballot to cast. The computer runs the casting algorithm, signs the encrypted ballot and the ballot proof on the voter's behalf, and sends the encrypted ballot, the ballot proof and the signature to the ballot box $\mathcal{B}$. The ballot box $\mathcal{B}$ sends the encrypted ballot, the ballot proof and the signature to the return code generator $\mathcal{R}$, who runs the code algorithm. It uses the result to compute the return code and sends it to the voter's phone $\mathcal{F}_i$, which sends it on to the voter $\mathcal{V}_i$.

Both the ballot box and the return code generator will verify the voter's signature. After sending the return code, the return code generator countersigns the encrypted ballot, the ballot proof and the voter's signature, and sends the countersignature to the ballot box, which in turns sends the countersignature to the voter's computer. The computer verifies the countersignature and only then accepts, showing the encrypted ballot, the ballot proof, the signature and the countersignature to the voter, which constitutes the voter's *receipt*.

The voter $\mathcal{V}_i$ accepts the ballot as cast if and only if the computer accepts with a receipt, and the voter's phone shows a return code such that the pair is in the return code table.

In the *counting phase*, the ballot box $\mathcal{B}$ and the return code generator $\mathcal{R}$ send the encrypted ballots, ballot proofs and voter signatures they have seen to the auditor $\mathcal{A}$ as well as every decryption server. The ballot box $\mathcal{B}$ then sorts the list of encrypted ballots $\{\boldsymbol{c}_i\}$ and sends this to the first shuffle server $\mathcal{S}_1$ and every decryption server. In the event that some voter has cast more than one ballot, only the encrypted ballot seen last is included in this list.

The shuffle servers $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_\iota$ use the shuffle algorithm on the input encrypted ballots, passing the shuffled and re-encrypted ballots to the next shuffle server. They also pass the shuffled re-encrypted ballots and the shuffle proof to the auditor and every decryption server.

Each decryption server verifies that the data from $\mathcal{B}$ and $\mathcal{R}$ is consistent (similar to the auditor $\mathcal{A}$), and that every shuffle proof verifies. Only then they run the distributed decryption algorithm with their decryption key share and send their partial decryption shares of each ballot as well as proofs of correct decryption to the auditor.

If the data is consistent (that is, the same encrypted ballots, ballot proofs and signatures appear in the same order in the data from both $\mathcal{B}$ and $\mathcal{R}$, and the signatures and the ballot proofs verify), the auditor $\mathcal{A}$ approves.

The auditor verifies the data from $\mathcal{B}$ and $\mathcal{R}$ (the same encrypted ballots, ballot proofs and signatures appear in the same order in the two data sets, and the voter signatures and the ballot proofs verify), that the encrypted ballots received by the first shuffle are consistent with the data from $\mathcal{B}$ and $\mathcal{R}$, that every shuffle proof verifies, and then runs the combining algorithm on the received ballot decryption shares. If the combining algorithm accepts then the auditor accepts, otherwise it rejects. Finally, $\mathcal{A}$ outputs the complete list of messages received, including the public key material, as its transcript.

There is 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 result. If a receipt is present, it also verifies the countersignature and the voters' signatures in the receipt, that the encrypted ballot, the ballot proof and the voters' signatures are present in the ballot box data set, and that the encrypted ballots are present in the first shuffle server's input.

This concludes the description of the voting protocol in terms of a verifiable cryptographic voting scheme with return codes.

Note that there are many variations of this protocol. It can be used without return codes, simply by omitting return codes. Also, depending on the exact setting and security required, the return code generator can be merged with the ballot box.

Many comparable schemes are phrased in terms of an ideal (web) 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 some network.

It is also 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.

7 Performance

7.1 Size of the Submission Phase

Each BGV ciphertexts are of size $2N \log_2 q$ bits. We assume that the set of input-ciphertexts to the mixing network are honestly generated. We can ensure this by using e.g. the exact proofs by Beullens [Beu20] or Baum and Nof [BN20], the efficient range proofs by Attema *et al.* [ALS20], or the new techniques from Lyubashevsky *et al.* [LNS21,ENS20] to prove that the noise is bounded and that the user knows the message.

7.2 Size of the Mixing Network

Let ι denote the number of mixing servers and let τ be the number of ciphertexts. Each ciphertext consists of two elements in R_q, where each element can be represented by $N \log_2 q$ bits. The transcript of the mixing phase will contain ι sets of τ new ciphertexts, and is of size $2\iota\tau N \log_2 q$ bits.

For each shuffle the servers must provide a proof of shuffle and an amortized proof of shortness. Both proofs prove a relation about commitments to the randomization factors added to the old ciphertexts to get the new ciphertexts, and each commitment is of size $(n + 2)N \log_2 q$ bits.

The shuffle proof consists of τ commitment of size $(n+1)N \log_2 q$ bits, τ ring elements of size $N \log_2 q$ bits and a proof of linearity per ciphertext. The proof

Parameter	Explanation	Constraints
λ	Security parameter	At least 128 bits
N	Degree of polynomial $X^N + 1$ in R	N a power of two
p	Plaintext modulus	p a small prime
q	Ciphertext and commitment modulus	Prime $q = 1 \mod 2N$ s.t. $\max\lVert v - su \rVert \ll q/2$
k	Width (over R_q) of commitment matrix	
n	Height (over R_q) of commitment matrix	
ν	Maximum l_1-norm of elements in $\mathcal{C}$	
$\mathtt{D}$	Bounded distribution for noise in ciphertexts	Chosen to be the uniform ternary distribution
σ_{C}	Standard deviation for one-time commitments	Chosen to be $\sigma_{\mathrm{C}} = 0.954 \cdot \nu \cdot \beta_\infty \cdot \sqrt{kN}$
$\hat{\sigma}_{\mathrm{C}}$	Standard deviation for reusable commitments	Chosen to be $\hat{\sigma}_{\mathrm{C}} = 22 \cdot \nu \cdot \beta_\infty \cdot \sqrt{kN}$
σ_{ANEx}	Standard deviation for the one-time amortized proof	Chosen to hide the commitment randomness $\boldsymbol{r}''_{i,j}$
$\hat{\sigma}_{\mathrm{ANEx}}$	Standard deviation for the one-time amortized proof	Chosen to hide the noise-drowning values $E_{i,j}$
$\mathcal{C}$	Challenge space	$\mathcal{C} = \left\{ c \in R_p \mid \lVert c \rVert_\infty = 1, \lVert c \rVert_1 = \nu \right\}$
$\bar{\mathcal{C}}$	The set of differences $\mathcal{C} - \mathcal{C}$ excluding 0	$\bar{\mathcal{C}} = \{ c - c' \mid c \neq c' \in \mathcal{C} \}$
S_{β_∞}	Set of elements of ∞-norm at most β_∞	$S_{\beta_\infty} = \{ x \in R_p \mid \lVert x \rVert_\infty \leq \beta_\infty \}$
$\hat{n}$	Dimension of proof in $\mathit{\Pi}_{\mathrm{ANEx}}$	$\hat{n} \geq \lambda + 2$
ι, ξ	Number of shuffle- and decryption-servers	
τ	Total number of messages	For soundness we need $(\tau^\delta + 1)/\lvert R_q \rvert < 2^{-128}$

Table 1. System parameters and constraints.

of linearity is of size $2(k-n)N\log_2(6\sigma_{\mathrm{C}})$ bits. The shuffle proof is then of size $((n+2)N\log_2 q + 2(k-n)N\log_2(6\sigma_{\mathrm{C}}))\tau$ bits.

The amortized proof of shortness depends on the bound, in our case $B = 1$, but also on the ratio between the number of commitments and the size of the modulus. We denote the proof by π_{AEx}, and discuss it in more detail later based on concrete parameters. The total bit-size of the mixing is

$$\iota((2n+6)N\log_2 q + 2(k-n)N\log_2(6\sigma_{\mathrm{C}}))\tau + \iota|\pi_{\mathrm{AEx}}|.$$

7.3 Size of the Distributed Decryption

Let ξ denote the number of decryption servers and let τ be the number of ciphertexts. Each partial decryption consists of one element from R_q, which means that the output of the decryption is of size $\xi\tau N\log_2 q$ bits.

Additionally, each decryption server outputs a commitment to the added noise and a proof of linearity per ciphertext, and an amortized proof of shortness for all the added noise values. Also, each server has a public commitment of their decryption key-share to be used in the proof of linearity. Each commitment is of size $(n+1)N\log_2 q$ bits, and each proof of linearity is of size $(k-n)N(\log_2(6\sigma_{\mathrm{C}}) + \log_2(6\hat{\sigma}_{\mathrm{C}})$ 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 $k\hat{n}N\log_2(6\sigma_{\mathrm{ANEx}}) + \hat{n}\log_2(6\hat{\sigma}_{\mathrm{ANEx}})$ bits because of the different norms of the secret values as noted in Remark 1. As the bounds in the amortized proof depends on the number of commitments in the statement, we compute batched proofs of N equations at once to control the growth.

The total size of the distributed decryption is

$$\xi((n+2)N\log_2 q + (k-n)N(\log_2(6\sigma_{\mathrm{C}}) + \log_2(6\hat{\sigma}_{\mathrm{C}})) \\ + k\hat{n}\log_2(6\sigma_{\mathrm{ANEx}}) + \hat{n}\log_2(6\hat{\sigma}_{\mathrm{ANEx}}))\tau \text{ bits.}$$

7.4 Concrete Parameters and Total Size

Standard deviation. We let the success probability of each of the zero-knowledge protocols to be $1/M \approx 1/3$. The algorithm in Section 2.2 is used for rejection sampling. 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, which means that each of them must have a success probability of $1/\sqrt{3}$. This gives $\sigma_{\mathrm{C}} = 0.954\nu\beta_\infty\sqrt{kN}$. For the re-usable commitments we get $\hat{\sigma}_{\mathrm{C}} = 22\nu\beta_\infty\sqrt{kN}$. The amortized proof also have two checks, and we get standard deviation $0.954\left\|\boldsymbol{S}'\boldsymbol{C}'\right\|_2$, where σ_{ANEx} and $\hat{\sigma}_{\mathrm{ANEx}}$ are depending on the norm of the elements in the rows of $\boldsymbol{S}'$.

For the encryption, we let $\mathtt{D}$ be the ternary distribution over R_q, where each polynomial has coefficients in $\{-1,0,1\}$ sampled uniformly at random. We let the commitment randomness be bounded by $B_{\mathtt{Com}} = 1$.

Bounding the noise. To be able to choose concrete parameters for the mix-net, we need to estimate how much noise that 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:

- Original ciphertext: $B_{\mathtt{Start}} = p(\|er\|_\infty + \|e_{i,2}\|_\infty + \|-e_{i,1}s\|_\infty) + \|m\|_\infty$.
- Additional noise per shuffle: $B_{\mathrm{SHUF}} = p(\|er'\|_\infty + \left\|e'_{i,2}\right\|_\infty + \left\|-e'_{i,1}s\right\|_\infty)$.
- Additional noise in partial decryption: $B_{\mathtt{DistDec}} = p\xi\left\|E'_{i,j}\right\|_\infty \leq 2^{\mathtt{sec}}B_{\mathtt{Dec}}$,

where $B_{\mathtt{Dec}} = B_{\mathtt{start}} + \iota B_{\mathrm{SHUF}}$ 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:

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

Using the bounds from Section 2, we get upper bounds:

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

which for ι shuffles gives us

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

Finally, we need to make sure that $B_{\mathtt{Dec}} + B_{\mathtt{DistDec}} < q/2$, where $B_{\mathtt{DistDec}} = 2p\xi\hat{B}_{\mathrm{ANEx}}$ because of the soundness slack of the amortized proof of bounded values from Section 3.5. A honestly generated value $E_{i,j}$ is bounded by $2^{\mathtt{sec}}(B_{\mathtt{Dec}}/p\xi)$, but the proof can only guarantee that the values are shorter than some larger bound $2\hat{B}_{\mathrm{ANEx}}$ (following [BBC+18, Lemma 3]) that depends on the number of equation in the statement. Define $\boldsymbol{S}''$ to be the first k rows of $\boldsymbol{S}'$ and define $\boldsymbol{S}'''$ to be the last row of $\boldsymbol{S}'$. For batches of N equations we then get that:

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

and, similarly,

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

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

N	p	q	sec	ι	ξ	n	k	ν	$B_{\mathtt{Com}}$	$\hat{n}$	σ_{C}	$\hat{\sigma}_{\mathrm{C}}$	σ_{ANEx}	$\hat{\sigma}_{\mathrm{ANEx}}$	$\hat{B}_{\mathrm{ANEx}}$
4096	2	$\approx 2^{78}$	40	4	4	1	$\ell + 2$	36	1	130	$\approx 2^{12}$	$\approx 2^{16.5}$	$\approx 2^{13.5}$	$\approx 2^{66}$	$\approx 2^{72.5}$

Table 2. Concrete parameters estimated for $\lambda \approx 168$ bits of security using the LWE-estimator by Albrecht *et al.* [APS15].

We fix plaintext modulus $p = 2$, statistical security parameter $\mathtt{sec} = 40$, 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 $\iota = \xi = 4$. It follows that $B_{\mathtt{Dec}} < 2^{17}$ and $B_{\mathtt{DistDec}} < 2^{76.5}$. We then set $q \approx 2^{78}$, and verify that

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

Exact amortized proof. Finally, we must decide on parameters for the exact proof of shortness from Section 3.4. The soundness of the protocol depends on the ration between the number of equations and the size of the modulus, see Lemma 1. 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 $k \approx 2^{20}, l \approx 2^{20.3}, \eta = 325$ to keep the soundness $\approx 2^{-62}$. The total size of π_{AEx}, by instantiating 2, is $\approx 20\tau$ KB.

Total size. We give a complete set of parameters in Table 2, and the concrete sizes of each part of the protocol in Table 3. Each voter submit a ciphertext size 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 $\mathcal{D}_j$.

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

Table 3. Size of the ciphertexts, commitments and proofs.

7.5 Implementation

In order to estimate the efficiency of our protocols, we developed a proof-of-concept implementation to compare with previous results in the literature. Our performance figures were collected on an Intel Skylake Core i7-6700K CPU machine running at 4GHz without TurboBoost. The results can be found in Tables 4 and 5, and we intend to release the code publicly in the near future.

First, we compare performance of the main building blocks with an implementation of the shuffle proof protocol proposed in [ABG+21]. That work used the FLINT library to implement arithmetic involving polynomials of degree $N = 1024$ with 56-bit coefficients, fitting a 64-bit machine word. Their parameters were not compatible with the fast Number Theoretic Transform (NTT), so a CRT decomposition to two half-degree polynomials was used instead. The code was made available, so a direct comparison is possible.

In this work, the degree is much larger ($N = 4096$) and coefficients are multi-word ($q \approx 2^{78}$), but the parameters are compatible with the NTT. We implemented polynomial arithmetic with the efficient NFLlib [ABG+16] library using the RNS representation for coefficients. The arithmetic is accelerated with AVX2 instructions, especially the NTT transform and polynomial arithmetic. We observed that our polynomial multiplication is around 19 times more efficient than [ABG+21] ($61,314$ 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 division, but that incurred some non-trivial costs to convert representations between two libraries. For Gaussian sampling, we adapted COSAC [ZSS20] and adjusted the standard deviation σ accordingly.

Computing a commitment takes 0.45 ms on the target machine, which is 2x faster than [ABG+21]. Opening a commitment is slower due to conversions between libraries for performing the norm test. Our implementation of BGV encryption is much faster than the 69 ms reported for verifiable encryption in [ABG^{+}21], while decryption is improved by a factor of 7. Distributed decryption with passive security costs additional 1.51 ms per party, but the zero-knowledge proofs for active security increase the cost further. The shuffle proof performance is at 30 ms per vote, thus close to [ABG^{+}21].

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 5. In the case of $\Pi_{\textsc{AEx}}$, we only implement the performance-critical polynomial arithmetic and commitment scheme, since this is already representative of the overall performance. From the table, we can compute the cost of distributed decryption with active security as $(10.7 + 30 + 15.7 + 25.0) = 81.4$ ms per vote, the cost of $\Pi_{\textsc{Mix}}$ as $(0.45 + 1009 + 15.1) = 1024$ ms and the cost of $\Pi_{\textsc{MixV}}$ as $(20 + 16.1) = 36.1$ ms per vote. This result compares very favorably with the costs of 1.5 s and 1.49 s per vote to respectively generate/verify a proof in the lattice-based shuffle proof of [FWK21] in a Haswell processor running at approximately half the frequency. By considering space-time efficiency as a metric, our total numbers are 1.4 times higher after adjusting for clock frequency and negligible soundness, while storage overhead is much lower.

Primitive	Commit	Open	Encrypt	Decrypt	DistDec
Time	0.45 ms	2.3 ms	2.5 ms	0.83 ms	1.51 ms

Table 4. Timings for basic cryptographic operations. Numbers were obtained by computing the average of 10^4 consecutive executions of an operation measured using the cycle counter available in the platform.

Protocol	$\Pi_{\textsc{Lin}} + \Pi_{\textsc{LinV}}$	$\Pi^{\ell}_{\textsc{Shuf}} + \Pi^{\ell}_{\textsc{ShufV}}$	$\Pi_{\textsc{ANEx}} + \Pi_{\textsc{ANExV}}$	$\Pi_{\textsc{AEx}} + \Pi_{\textsc{AExV}}$
Time	$(10.7 + 15.7)\tau$ ms	$(15.1 + 16.1)\tau$ ms	$(30.0 + 25.0)\tau$ ms	$(1009 + 20)\tau$ ms

Table 5. Timings for cryptographic protocols, obtained by computing the average of 100 consecutive executions with $\tau = 1000$.

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.

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 key material will be secret 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 suitable distributed 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 protocol by Gjøsteen *et al.* [GHM+21] or by Silde [Sil22]). 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 have to add an additional proof of linearity for each new ciphertext in each shuffle.

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 preprint by Lyubashevsky *et al* [LNP22], continuously improves the state-of-the-art, and we expect future works to improve the concrete efficiency of our protocol.

References

ABG+16. Carlos Aguilar Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian, and Tancrède Lepoint. NFLlib: NTT-based fast lattice library. In Kazue Sako, editor, *Topics in Cryptology – CT-RSA 2016*, volume 9610 of *Lecture Notes in Computer Science*, pages 341–356. Springer, Heidelberg, February / March 2016.

ABG+21. Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, Tjerand Silde, and Thor Tunge. Lattice-based proof of shuffle and applications to electronic voting. In Kenneth G. Paterson, editor, *Topics in Cryptology – CT-RSA 2021*, volume 12704 of *Lecture Notes in Computer Science*, pages 227–251. Springer, Heidelberg, May 2021.

Adi08. Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *USENIX Security 2008: 17th USENIX Security Symposium*, pages 335–348. USENIX Association, July / August 2008.

ALS20. Thomas Attema, Vadim Lyubashevsky, and Gregor Seiler. Practical product proofs for lattice commitments. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 470–499. Springer, Heidelberg, August 2020.

APS15. Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

BBC$^+$18. Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 669–699. Springer, Heidelberg, August 2018.

BCG$^+$15. David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. SoK: A comprehensive analysis of game-based ballot privacy definitions. In *2015 IEEE Symposium on Security and Privacy*, pages 499–516. IEEE Computer Society Press, May 2015.

BCG$^+$17. Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part III*, volume 10626 of *Lecture Notes in Computer Science*, pages 336–365. Springer, Heidelberg, December 2017.

BD10. Rikke Bendlin and Ivan Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 201–218. Springer, Heidelberg, February 2010.

BDL$^+$18. Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner, and Chris Peikert. More efficient commitments from structured lattice assumptions. In Dario Catalano and Roberto De Prisco, editors, *SCN 18: 11th International Conference on Security in Communication Networks*, volume 11035 of *Lecture Notes in Computer Science*, pages 368–385. Springer, Heidelberg, September 2018.

BEPU$^+$20. Carsten Baum, Daniel Escudero, Alberto Pedrouzo-Ulloa, Peter Scholl, and Juan Ramón Troncoso-Pastoriza. Efficient protocols for oblivious linear function evaluation from ring-LWE. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20: 12th International Conference on Security in Communication Networks*, volume 12238 of *Lecture Notes in Computer Science*, pages 130–149. Springer, Heidelberg, September 2020.

Beu20. Ward Beullens. Sigma protocols for MQ, PKP and SIS, and Fishy signature schemes. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 183–211. Springer, Heidelberg, May 2020.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, pages 309–325. Association for Computing Machinery, January 2012.

BHM20. Xavier Boyen, Thomas Haines, and Johannes Müller. A verifiable and practical lattice-based decryption mix net with external auditing. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, *ESORICS 2020: 25th European Symposium on Research in Computer Security, Part II*, volume 12309 of *Lecture Notes in Computer Science*, pages 336–356. Springer, Heidelberg, September 2020.

BHM21. Xavier Boyen, Thomas Haines, and Johannes Müller. 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, pages 272–291. IEEE, 2021.

BLNS21. Jonathan Bootle, Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. More efficient amortization of exact zero-knowledge proofs for lwe. In Elisa Bertino, Haya Shulman, and Michael Waidner, editors, *Computer Security – ESORICS 2021*, pages 608–627, Cham, 2021. Springer International Publishing.

BLS19. Jonathan Bootle, Vadim Lyubashevsky, and Gregor Seiler. Algebraic techniques for short(er) exact lattice-based zero-knowledge proofs. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 176–202. Springer, Heidelberg, August 2019.

Blu84. Manuel Blum. How to exchange (secret) keys. *ACM Transactions on Computer Systems*, 1:175–193, 1984.

BN20. Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 12110 of *Lecture Notes in Computer Science*, pages 495–526. Springer, Heidelberg, May 2020.

CES02. e-voting security study. CESG, United Kingdom, July 2002. Issue 1.2.

CGGI16. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. A homomorphic LWE based E-voting scheme. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*, pages 245–265. Springer, Heidelberg, 2016.

Cha81. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

CMM19. Núria Costa, Ramiro Martínez, and Paz Morillo. Lattice-based proof of a shuffle. In Andrea Bracciali, Jeremy Clark, Federico Pintore, Peter B. Rønne, and Massimiliano Sala, editors, *FC 2019 Workshops*, volume 11599 of *Lecture Notes in Computer Science*, pages 330–346. Springer, Heidelberg, February 2019.

CP93. David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer, Heidelberg, August 1993.

Dam10. Ivan Damgård. On σ-protocols, 2010. https://cs.au.dk/~ivan/Sigma.pdf.

DKL$^+$13. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013: 18th European Symposium on Research in Computer Security*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18. Springer, Heidelberg, September 2013.

dLNS17. Rafaël del Pino, Vadim Lyubashevsky, Gregory Neven, and Gregor Seiler. Practical quantum-safe voting from lattices. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1565–1581. ACM Press, October / November 2017.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, Heidelberg, August 2012.

ENS20. Muhammed F. Esgin, Ngoc Khanh Nguyen, and Gregor Seiler. Practical exact proofs from lattices: New techniques to exploit fully-splitting rings. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 259–288. Springer, Heidelberg, December 2020.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, Heidelberg, August 1987.

FWK21. Valeh Farzaliyev, Jan Willemson, and Jaan Kristjan Kaasik. Improved lattice-based mix-nets for electronic voting. In *Information Security and Cryptology – ICISC 2021*. Springer International Publishing, 2021.

GHM$^+$21. Kristian Gjøsteen, Thomas Haines, Johannes Müller, Peter Rønne, and Tjerand Silde. Verifiable decryption in the head. Cryptology ePrint Archive, Report 2021/558, 2021.

Gjø11. Kristian Gjøsteen. The norwegian internet voting protocol. In Aggelos Kiayias and Helger Lipmaa, editors, *E-Voting and Identity - Third International Conference, VoteID 2011*, volume 7187 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011.

GM82. Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, page 365–377, New York, NY, USA, 1982. Association for Computing Machinery.

GMR85. S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, page 291–304, New York, NY, USA, 1985. Association for Computing Machinery.

HMS21. Javier Herranz, Ramiro Martínez, and Manuel Sánchez. Shorter lattice-based zero-knowledge proofs for the correctness of a shuffle. Cryptology ePrint Archive, Report 2021/488, 2021.

HR16. Feng Hao and Peter Y. A. Ryan, editors. *Real-World Electronic Voting: Design, Analysis and Deployment*. CRC Press, 2016.

LN16. Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In Sara Foresti and Giuseppe Persiano, editors, *CANS 16: 15th International Conference on Cryptology and Network Security*, volume 10052 of *Lecture Notes in Computer Science*, pages 124–139. Springer, Heidelberg, November 2016.

LNP22. Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Maxime Plancon. Lattice-based zero-knowledge proofs and applications: Shorter, simpler, and more general. Cryptology ePrint Archive, Report 2022/284, 2022. https://ia.cr/2022/284.

LNS21. Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. Shorter lattice-based zero-knowledge proofs via one-time commitments. In Juan Garay, editor, *PKC 2021: 24th International Conference on Theory and*

Practice of Public Key Cryptography, Part I, volume 12710 of *Lecture Notes in Computer Science*, pages 215–241. Springer, Heidelberg, May 2021.

LPR13. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 35–54. Springer, Heidelberg, May 2013.

LS15. Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptography*, 75(3):565–599, June 2015.

Nef01. C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001: 8th Conference on Computer and Communications Security*, pages 116–125. ACM Press, November 2001.

RST$^+$22. Dragos Rotaru, Nigel P. Smart, Titouan Tanguy, Frederik Vercauteren, and Tim Wood. Actively secure setup for spdz. *J. Cryptol.*, 35(1), jan 2022.

Scy. Scytl. Scytl sVote, complete verifiability security proof report - software version 2.1 - document 1.0.

Sil22. Tjerand Silde. Verifiable decryption for BGV. Workshop on Advances in Secure Electronic Voting, 2022. https://ia.cr/2021/1693.

Str19. Martin Strand. A verifiable shuffle for the GSW cryptosystem. In Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala, editors, *FC 2018 Workshops*, volume 10958 of *Lecture Notes in Computer Science*, pages 165–180. Springer, Heidelberg, March 2019.

ZSS20. Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. COSAC: COmpact and scalable arbitrary-centered discrete gaussian sampling over integers. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 284–303. Springer, Heidelberg, 2020.

Appendix

A Verifiable Decryption Secure Against $n-1$ Parties

A.1 Optimistic Decryption

We present a new verifiable decryption protocol secure against $n-1$ parties. The protocol is inspired by the information theoretic MAC used in the MPC-protocols by Damgård *et al.* [DPSZ12,DKL+13]. Here we start with an encrypted message m, sample a secret MAC α as the sum of independent shares α_j and check that the decryption of αm is correct. We use homomorphic encryption and standard commit-and-open techniques to ensure that the MAC value is honestly generated. Assuming at least one honest party, then the probability that the adversary can succeed is to guess the value of α before the decryption of αm is revealed. This happens with probability $1/|R_p| = 1/p^N$, which is negligible in the security parameter λ.

Note that this setting is usually not acceptable in an election, but it might still have value in at least the two following situations.

Firstly, as the protocol above is expensive both in computational complexity and memory consumption, it might be interesting to get a preliminary result of the election quickly by running the more lightweight protocol first and then heavier protocol afterwards. This way, we can within short time get a result with good confidence, and later get a proof that the first output was indeed correct.

Secondly, some smaller elections are organized such that each party voting is also computing a shuffle and a partial decryption, and hence, is both a voter and an infrastructure player in the election. In this case one has neither privacy nor integrity if everyone is colluding, and it makes sense to use a more lightweight protocol to compute the tally.

A.2 Homomorphic Multiplication

We describe additional algorithms to extend the BGV encryption protocol from Section 3.1 to include homomorphic multiplication of ciphertexts.

- `KeyGenExt`, runs `KeyGen` from Section 3 and outputs public key $\mathtt{pk} = (a, b) = (a, as + pe)$ and secret key $\mathtt{sk} = (s_1, s_2) = (s, s^2)$.

- `Mult`, on input two ciphertexts $\boldsymbol{c}_1 = (u_1, v_1)$ and $\boldsymbol{c}_2 = (u_2, v_2)$, computes $d_0 = v_1 \cdot v_2$, $d_1 = u_1 \cdot v_2 + u_2 \cdot v_1$ and $d_2 = u_1 \cdot u_2$, and outputs the product ciphertext $\boldsymbol{d} = (d_0, d_1, d_2)$.

- `DecExt`, takes as input a product ciphertext $\boldsymbol{d} = (d_0, d_1, d_2)$ and computes $m' = (d_0 - s_1 \cdot d_1 + s_2 \cdot d_2 \mod q) \mod p$. The algorithm outputs m'.

The output of the decryption algorithm is correct if $\|d_0 - s_1 \cdot d_1 + s_2 \cdot d_2\|_\infty = B_{\mathtt{Dec}} < \lfloor q/2 \rceil$. Furthermore, we present the extended passively secure distributed decryption technique used in the MPC-protocols by Damgård *et al.* [DPSZ12, DKL+13]. When decrypting, we assume that each decryption server $\mathcal{D}_j$, for $1 \leq j \leq \xi$, has a uniformly random share $\mathsf{sk}_j = (s_{1,j}, s_{2,j})$ of the secret key $\mathsf{sk} = (s_1, s_2)$ such that $s_1 = s_{1,1} + s_{1,2} + ... + s_{1,\xi}$ and $s_2 = s_{2,1} + s_{2,2} + ... + s_{2,\xi}$. Then they partially decrypt using the following algorithm:

- $\mathtt{DistDecExt}$, on input a secret key-share sk_j and a ciphertext $\boldsymbol{d} = (d_0, d_1, d_2)$, computes $m_j = s_{1,j} \cdot d_1 - s_{2,j} \cdot d_2$, sample some large noise $E_j \leftarrow\!\!\$\ R_q$ such that $\|E_j\|_\infty \leq 2^{\mathsf{sec}}(B_{\mathtt{Dec}}/p\xi)$, and then outputs $t_j = m_j + pE_j$.

We obtain the full decryption of the ciphertext $\boldsymbol{d} = (d_0, d_1, d_2)$ as $m = (d_0 - t \mod q) \mod p$, where $t = t_1 + t_2 + ... + t_\xi$. This will give the correct decryption as long as the noise $\|d_0 - t\|_\infty \leq (1 + 2^{\mathsf{sec}})B_{\mathtt{Dec}} < \lfloor q/2 \rfloor$.

A.3 Decryption Protocol

We use the same public parameters and ciphertexts as received in the decryption protocol in Section 5. Each decryption server has a secret key-share as described in $\mathtt{KeyGenExt}$. Our verifiable decryption protocol works as following.

Partial decryption. Each party $\mathcal{D}_j$ run $\mathtt{DistDec}$, as defined in Section 3.1, on each ciphertext $\boldsymbol{c}_i$ to produce partial decryptions $t_{i,j}$. Output $\{t_{i,j}\}_{i=1}^{\tau}$.

Commit to randomness. Each party samples a random $\alpha_j \leftarrow\!\!\$\ R_p$, run $\mathtt{Enc}$ with message α_j to produce ciphertext $\boldsymbol{c}_{\alpha_i} = (u_{\alpha_j}, v_{\alpha_j})$, and commit to the message and randomness as $h_j = \mathtt{H}(\alpha_j, r_{\alpha_j}, e_{\alpha_j}, e'_{\alpha_j})$, for a hash-function $\mathtt{H}$. Output h_j.

Output MAC shares. When each party $\mathcal{D}_j$ has published h_j, output $\boldsymbol{c}_{\alpha_j}$.

Compute MAC. Each party gather the encrypted MAC shares $\{\boldsymbol{c}_{\alpha_j}\}$ and sum them together as $\boldsymbol{c}_\alpha = \boldsymbol{c}_{\alpha_1} + ... + \boldsymbol{c}_{\alpha_\xi}$. Then, for each ciphertext $\boldsymbol{c}_i$, use $\mathtt{Mult}$ to compute the homomorphic multiplication with $\boldsymbol{c}_\alpha$ denoted $\boldsymbol{d}_i = (d_{i,0}, d_{i,1}, d_{i,2})$.

Commit to partial MAC decryption. Each party $\mathcal{D}_j$ runs the $\mathtt{DistDecExt}$ on each ciphertext $\boldsymbol{d}_i$ to produce partial decryptions $t_{\alpha_{i,j}}$. Commit to the partial decryption as $h_{\alpha_j} = \mathtt{H}(t_{\alpha_{1,j}}, ..., t_{\alpha_{\tau,j}})$. Output h_{α_j}.

Output partial MAC decryption. When each party $\mathcal{D}_j$ has published $h_{\alpha_{i,j}}$, output the partial MAC α_j, randomness $r_{\alpha_j}, e_{\alpha_j}, e'_{\alpha_j}$ and partial decryptions $\{t_{\alpha_{i,j}}\}_{i=1}^{\tau}$.

Verify correct decryption. A verifier computes $m_i = (v_{i,0} - t_i \mod q) \mod p$ for $t_i = t_{i,1} + ... + t_{i,\xi}$ and $m_{\alpha_i} = (d_{i,0} - t_{\alpha_i} \mod q) \mod p$ for $t_{\alpha_i} = t_{\alpha_{i,1}} + ... + t_{\alpha_{i,\xi}}$ for all $i = 1, ..., \tau$. Also compute $\alpha = \alpha_1 + ... + \alpha_\xi$. Then, check that $\boldsymbol{c}_{\alpha_i}$ was correctly computed, that h_j and $h_{\alpha_{i,j}}$ have valid openings and that $m_{\alpha_i} = m_i \cdot \alpha$ in R_p for all i. If all checks holds then output $\{m_i\}_{i=1}^{\tau}$ and otherwise output $\perp$.

A.4 Parameters and Size

Bounding the noise. We know from Section 7 that the amount of noise in each input ciphertext $\boldsymbol{c}_i$ is bounded by $B_{\boldsymbol{c}} = (\iota+1)p(2N+1)+\lceil(p-1)/2\rceil$. Furthermore, the noise in each ciphertext $\boldsymbol{c}_{\alpha_j}$ is bounded by $B_{\boldsymbol{c}_\alpha} = p(2N+1)+\lceil(p-1)/2\rceil$. It follows from Brakerski *et al.* [BGV12, Section 5.2] that the noise in each product ciphertext $\boldsymbol{d}_i$ is bounded by $B_{\boldsymbol{d}} = \sqrt{N}\cdot B_{\boldsymbol{c}}\cdot B_{\boldsymbol{c}_\alpha}$. Using `DistDecExt`, noise of size $2^{\mathsf{sec}}B_{\boldsymbol{d}}$ is added to $\boldsymbol{d}_i$. To ensure correct decryption, we must choose q such that $(1+2^{\mathsf{sec}})B_{\boldsymbol{d}} < q/2$. Inserting the parameters from Table 2 we get that $B_{\boldsymbol{d}} < 2^{38}$, and hence, we can choose $q \approx 2^{78}$ as in Section 7.

Total size. Each partial decryption consists of one ring-element, and each ring element can be represented with $N \log q$ bits. Each party $\mathcal{D}_j$ only sends two hashes of size 256 bits and the elements $\alpha_j, r_{\alpha_j}, e_{\alpha_j}, e'_{\alpha_j}$ which are of size $2N$ bits each to generate the MAC, which is essentially negligible compared to the ciphertexts and partial decryptions. The total size of the decryption protocol, not counting the input ciphertexts, is $\approx 2\xi\tau N\log q$. Using the parameters from above we get that the concrete size is $\approx 80\tau$ KB per decryption server.

B Security of the Voting Protocol

Here we provide a more formal description of the voting protocol described in Section 6, give security notions, sketch a security proof and discuss the security properties of the full voting protocol.

B.1 Verifiable Voting Schemes 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. To support return codes, we also need algorithms for voter registration and pre-code computation. Finally, to accurately model the counting process, we need algorithms for shuffling and distributed decryption.

The *setup* algorithm `Setup` outputs a *public key* `pk`, *decryption key shares* $\mathtt{dk}_i$ 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 $\mathtt{Verify}$ 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 $\mathtt{DistDec}$ 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 $\mathtt{Comb}$ takes a public key pk, a sequence of encrypted ballots $\boldsymbol{ev}$, ballot decryption share sequences $\boldsymbol{sv}_1, \boldsymbol{sv}_2, \dots, \boldsymbol{sv}_{l_d}$ with proofs $\pi_{d,1}, \pi_{d,2}, \dots, \pi_{d,l_d}$, and outputs either $\perp$ or a sequence of ballots $v_1, v_2, \dots, v_{l_t}$.

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

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

We also require that the distribution of ev_i only depends on pk and v_i, not $\mathtt{vck}_i$.

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

B.2 Our Scheme

Our voting scheme combines the BGV encryption together with our shuffle (Section 4) and distributed decryption (Section 5). We adapt the techniques from Aranha *et al.* [ABG+21] to get extractability and code voting, but omit the details.

- $\mathtt{Setup}$ computes $\mathsf{pk}_C \leftarrow \mathtt{KeyGen}_C$, $(\mathsf{pk}_V, \mathsf{dk}_V) \leftarrow \mathtt{KeyGen}_{VE}$, $(\mathsf{pk}_R, \mathsf{dk}_R) \leftarrow \mathtt{KeyGen}_{VE}$, 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 $\mathsf{ck} = (\mathsf{pk}_C, \mathsf{pk}_V, \mathsf{dk}_R)$.
- $\mathtt{Reg}$ takes $\mathsf{pk} = (\mathsf{pk}_C, \mathsf{pk}_V, \mathsf{pk}_R)$ as input. It samples $a \leftarrow\!\!\$\ R_p$ and computes $(\boldsymbol{c}_a, d_a) \leftarrow \mathtt{Com}(\mathsf{pk}_C, a)$. The voter verification key is $\mathtt{vvk} = \boldsymbol{c}_a$, the voter casting key is $(a, \boldsymbol{c}_a, d_a)$, and the function f is $v \mapsto v + a$.
- $\mathtt{Cast}$ takes $\mathsf{pk} = (\mathsf{pk}_C, \mathsf{pk}_V, \mathsf{pk}_R)$, $\mathtt{vck} = (a, \boldsymbol{c}_a, d_a)$ and v as input. It computes $\boldsymbol{v} \leftarrow \mathtt{Enc}_{VE}(\mathsf{pk}_V, v)$, $\hat{r} \leftarrow a + v$ and $\boldsymbol{w} \leftarrow \mathtt{Enc}_{VE}(\mathsf{pk}_R, \hat{r})$, 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 $\pi_v = (\boldsymbol{w}, \pi_{v,0})$.
- $\mathtt{Code}$ takes $\mathtt{ck} = (\mathtt{pk}_C, \mathtt{pk}_V, \mathtt{dk}_R)$, a voter verification key $\mathtt{vvk}$, 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 $\hat{r} \leftarrow \mathtt{Dec}_{VE}(\mathtt{dk}_R, \boldsymbol{w})$ and outputs $\hat{r}$. (If $\mathtt{ck} = \perp$, it outputs 1 if and only if it accepts $\pi_{v,0}$.)
- The *shuffle* algorithm $\mathtt{Shuffle}$ and the *verify* algorithm $\mathtt{Verify}$ are as described in Section 4. The *distributed decryption* algorithm $\mathtt{DistDec}$ and the *combining* algorithm $\mathtt{Comb}$ are as described in Section 5.

It is straight-forward to verify that the scheme is correct.

B.3 Security Notions

Our notion of confidentiality is similar to the usual ballot box privacy notions [BCG+15]. 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 for integrity. The experiment works as follows:

- Sample $b, \leftarrow\!\!\$ \{0,1\}$. Set L, L', L'' to be empty lists.
- $(\mathtt{pk}, \{\mathtt{dk}_i\}, \mathtt{ck}) \leftarrow \mathtt{Setup}$. For $i = 1, \dots, l_V$: $(\mathtt{vvk}_i, \mathtt{vck}_i, f_i) \leftarrow \mathtt{Reg}(\mathtt{pk})$. Send $(\mathtt{pk}, \mathtt{vvk}_1, \dots, \mathtt{vvk}_{l_V})$ to $\mathcal{A}$.
- On a *voter reveal query* i, send $(\mathtt{vck}_i, f_i)$ to $\mathcal{A}$. On a *decrypt reveal query* i, send $\mathtt{dk}_i$ to $\mathcal{A}$. On a *code reveal query*, send $\mathtt{ck}$ to $\mathcal{A}$.
- On a *challenge query* (i, v_0, v_1), compute $(ev, \pi_v) \leftarrow \mathtt{Cast}(\mathtt{pk}, \mathtt{vck}_i, v_b)$, $\hat{r} \leftarrow \mathtt{Code}(\mathtt{ck}, \mathtt{vvk}_i, ev, \pi_v)$, 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 $\hat{r} \leftarrow \mathtt{Code}(\mathtt{ck}, \mathtt{vvk}_i, ev, \pi_v)$. 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 $(\boldsymbol{ev}', \pi_s) \leftarrow \mathtt{Shuffle}(\mathtt{pk}, \boldsymbol{ev})$, 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 $\mathtt{Verify}(\mathtt{pk}, \boldsymbol{ev}, \boldsymbol{ev}', \pi_s) = 1$.

– On a *ballot decryption share query* $(i, \boldsymbol{ev})$, we then compute $(\boldsymbol{sv}_i, \pi_{d,i}) \leftarrow$ $\mathtt{DistDec}(\mathtt{dk}_i, \boldsymbol{ev})$, 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* $\leftarrow$ $\mathtt{Comb}(\mathsf{pk}, \boldsymbol{ev}, \boldsymbol{sv}_1, \dots, \boldsymbol{sv}_{l_d}, \pi_{d,1}, \dots, \pi_{d,l_d})$ 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 for the confidentiality game.

The integrity game follows the usual pattern where the adversary's goal is to achieve certain inconsistencies, either during a code query or during 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 ballots and the right-hand ballots are identical, 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 an 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 $\mathtt{Code}(\mathtt{ck}, \mathtt{vvk}_i, ev, \pi_v) = \hat{r} \neq \perp$, we have that either $\mathtt{Dec}(\{\mathtt{dk}_i\}, ev) = \perp$ or $\mathtt{Dec}(\{\mathtt{dk}_i\}, ev) = v$ and $f_i(v) \neq \hat{r}$.

Let F_c (count failure) be the event that a test query gets $\mathit{result} = \bot$ 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 $\mathit{result} = (v_1, \dots, v_{l_c})$, 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 are equal to $\bot$. The advantage of the adversary is

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

B.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 the further analysis we may assume that the execution remains fresh.

We next simulate all the zero knowledge proofs involved, which is straightforward 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 5 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 $\mathtt{dk}_R$ and subtract a to recover v, and then record (i, v, v, ev, π_v) instead of $(i, \bot, \bot, ev, \pi_v)$. By 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 $\texttt{Dec}(\texttt{dk}_V, ev) = v_b$.

Next, for any ballot decryption share query for an honest and balanced sequence $(ev_1, \ldots, ev_{l_c})$ deriving from $(\cdot, v_{01}, v_{11}, \cdot, \cdot), \ldots, (\cdot, v_{0l_c}, v_{1l_c}, \cdot, \cdot)$, sample a permutation π on $\{1, 2, \ldots, l_c\}$ 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 $\{\texttt{dk}_i\}$ are no longer used. Also, the pre-code encrypted in the challenge query is independent of the challenge ballots.

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

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 that 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.

B.5 Voting System Security Properties

Integrity. Integrity for a voting system is modeled 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 the honest voters. (We need this since return codes for a single voter must be human-comparable, and can therefore collide with some non-negligible probability.)

Analysis. The voter will only accept the ballot as cast if the correct return code is received. If the correct return code is received, then the correct pre-code must have been computed at some point (except with some small probability due to collisions in the PRF).

If the *return code generator* $\mathcal{R}$ is honest, 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 P and the ballot box $\mathcal{B}$ and the auditor $\mathcal{A}$ are honest, the 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.

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 an 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 correspond 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.

Privacy. Privacy for a voting protocol is modeled 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, and they will all either cast the left ballot or the 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, so we require that the adversary organizes the pairs of ballots given to the honest voters in such a way that the ballots cast by the honest voters are independent of whether the voters cast the left or the right ballot.

Analysis. If some honest voter's *computer* P 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 honest, but this requires adding a restricted challenge query that does not reveal the precode to 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 is honest. Then privacy follows from confidentiality of the cryptographic voting system, since the protocol execution can be interpreted as an interaction with the cryptosystem experiment and the protocol together with 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 a 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.

C Proofs for Amortized Exact ZKPoPK

C.1 Proof of Lemma 1.

Proof. First, we construct an extractor $\mathcal{E}$ as in [BLNS21] which does the following 8 times:

1. First, run $\mathtt{P}^*$ on random challenges from $\mathtt{V}$ until an accepting transcript is found. Abort if none is found after $8/\epsilon$ steps.
2. Let I_1 be the challenge set where $\mathtt{P}^*$ responded and let $\boldsymbol{E}|_{I_1}, \mathtt{MerklePaths}_{I_1}$ be the opened columns and Merkle tree paths. Fixing the other challenges, $\mathcal{E}$ adaptively reruns the proof for different challenges $I_2, I_3, \dots$ that contain so far unopened columns and collects these. If any collision in the Merkle tree is found then $\mathcal{E}$ outputs the hash collision and terminates, otherwise it continues until it collected a set J of at least k columns, or until $8\frac{k-\eta}{\epsilon/2-(k/(l-\eta))^\eta}$ time passed.
3. Finally, $\mathcal{E}$ re-runs $\mathtt{P}^*$ $16/\epsilon$ 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, [BLNS21] show that $\mathcal{E}$'s total runtime is bounded by the term mentioned in the Lemma. Moreover, define S as the event that $\mathtt{P}^*$ outputs a valid proof in the last step and C be the event that the values $\mathtt{P}^*$ outputs to $\mathcal{E}$ in the last step of the extractor are consistent (i.e. no hash collisions and the commitment was not opened differently than before). Then [BLNS21] 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 $\mathtt{P}^*$ must be lower than the given bound.

Define C' to be the RS code obtained from C (generated by $\mathtt{Encode}$) when restricted to the indices of J. As $|J| = k$, C' has length k and minimum distance $d' = k - k' + 1$. For any $\boldsymbol{x} \in \mathbb{Z}_q^k$ 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 [BCG$^+$17, 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 as in [BLNS21] we can use this to deduce that in such a case, it must hold that

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

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 [BCG$^+$17, Appendix B] we have that for any vector $\boldsymbol{y} \in \mathbb{Z}_q^{3\tau+4}$ it holds that $d'(\mathtt{Encode}_J(\boldsymbol{y}\boldsymbol{V}, \boldsymbol{y}\boldsymbol{R}), \boldsymbol{y}\boldsymbol{E}^*) < d'/3$. 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 [BLNS21, Corollary 3.7] one can show that if there are $\leq (\epsilon/4)(q-\tau)$ choices of x such that

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

$$\frac{1}{\ell_0(x)} \cdot \bar{\boldsymbol{f}} \circ \left[\bar{\boldsymbol{f}} - \mathbf{1}\right] \circ \left[\bar{\boldsymbol{f}} + \mathbf{1}\right] = \ell_0(x)\boldsymbol{v}_0^* + \sum_{i=1}^{\tau}\sum_{j=0}^{2} \ell_i(x)\ell_0(x)^j \boldsymbol{v}_{i,j}^* \tag{4}$$

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

$$\bar{\boldsymbol{f}} \circ (\bar{\boldsymbol{f}} - \mathbf{1}) \circ (\bar{\boldsymbol{f}} + \mathbf{1}) - \ell_0(X)^2 \boldsymbol{v}_0^* - \sum_{i=1}^{\tau} \sum_{j=0}^{2} \ell_i(X)\ell_0(X)^{j+1} \boldsymbol{v}_{i,j}^* = \mathbf{0}. \qquad (5)$$

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 5 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

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

and replacing again $\bar{\boldsymbol{f}}$ with Equation 3 yields

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

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 $\mathcal{S}$ which, given $x, \beta_0, \beta_{1,0}, \ldots, \beta_{\tau,2}, I$ outputs a protocol transcript of the the protocol in Figure 3 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 $\mathit{\Pi}_{\mathrm{AEx}}$.

Proof. Towards constructing a simulation, observe that

1. $\mathcal{M}$ and its opened paths do not reveal any information about the unopened columns as the commitment scheme used in creating $\mathcal{M}$ 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 $\mathcal{S}$ 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 $\mathcal{S}$ 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 $\bar{\boldsymbol{h}}, \bar{\boldsymbol{r}}_h$. $\mathcal{S}$ now fixes the remaining columns of $\boldsymbol{E}$ as $\boldsymbol{0}$ and commits honestly to these as in the protocol. □