Lattice-Based Proof of Shuffle and Applications to Electronic Voting

Diego F. Aranha[1], Carsten Baum[1⋆], Kristian Gjøsteen[2], Tjerand Silde[2], and Thor Tunge[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. A verifiable shuffle of known values is a method for proving that a collection of commitments opens to a given collection of known messages, without revealing a correspondence between commitments and messages. We propose the first practical verifiable shuffle of known values for lattice-based commitments.
Shuffles of known values have many applications in cryptography, and in particular in electronic voting. We use our verifiable shuffle of known values to build a practical lattice-based cryptographic voting system that supports complex ballots. Our scheme is also the first construction from candidate post-quantum secure assumptions to defend against compromise of the voter's computer using return codes.
We implemented our protocol and present benchmarks of its computational runtime. The size of the verifiable shuffle is 22τ KB and takes time 33τ ms for τ voters. This is around 5 times faster and 40 % smaller per vote than the lattice-based voting scheme by del Pino et al. (ACM CCS 2017), which can only handle yes/no-elections.

Keywords: Lattice-Based Cryptography, Proof of Shuffle, Verifiable Encryption, Return Codes, Electronic Voting, Implementation

1 Introduction

A *verifiable shuffle of known values* is a method for proving that a collection of commitments opens to a given collection of known messages, without revealing exactly which commitment corresponds to which message.

One well-known approach is due to Neff [Nef01]: Define two polynomials, one that has the known messages as its roots and another that has the values committed to as its roots. Since polynomials are stable under permutation of

⋆ This work was funded by the European Research Council (ERC) under the European Unions' Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO). Part of this work was done while visiting NTNU in Trondheim.

their roots, it is sufficient to prove that these two polynomials have the same evaluation at a randomly chosen point.

Proving that the second polynomial has a given evaluation at a given point could be done using multiplication and addition proofs on the commitments. Usually multiplication proofs for committed values are quite expensive, while it is somewhat cheap to do proofs of linear combinations of committed values with public coefficients. Following the idea of Neff, the determinant of a particular band matrix is the difference of the two polynomials, and we show that the polynomials are equal by showing that the columns of the matrix are linearly dependent.

1.1 Our Contribution

Verifiable shuffle of known values. Our main contribution is a verifiable shuffle of known values for lattice-based commitments. This is the first efficient construction from a candidate post-quantum secure assumption of such a primitive. As discussed above, our construction is based on techniques originating with Neff [Nef01], although there are a number of obstacles with this approach in the lattice-based setting, where we use the commitments of Baum *et al.* [BDL+18].

First of all, many group-homomorphic commitment schemes allow either direct or very simple verification of arbitrary linear relations. No known commitment scheme secure under an assumption considered as post-quantum secure has a similar structure, which means that we must use adaptations of existing proofs for linear relations. Secondly, the underlying algebraic structure is a ring, not a field. Since we need certain elements to be invertible, we need to choose challenges from special sets of invertible elements, and carefully adapt the proof so that the correctness of the shuffle is guaranteed.

In order to make our construction practical, we use the Fiat-Shamir transform to make the underlying Zero-Knowledge proofs non-interactive. We want to stress that our proof of security only holds in the conventional Random Oracle Model, which is not a sound model when considering quantum adversaries. Constructing a post-quantum secure verifiable shuffle of known values is an interesting open problem.

Voting from lattices. Our second contribution is the first construction of a practical voting system that is suitable for more general ballots (such as various forms of ranked choice voting, perhaps in various non-trivial combinations with party lists and candidate slates) and that is secure under lattice-based assumptions.

We adopt an architecture very similar to deployed cryptographic voting systems [HR16, Gjø11]. The protocol works as follows:

- The voter's computer commits to the voter's ballot and encrypts an opening of the ballot. The commitment and ciphertext are sent to a ballot box.
- When counting starts, the ballot box removes any identifying material from the ciphertext and sends this to the shuffle server.

– The shuffle server decrypts the openings, verifies the commitments and outputs the ballots. It uses our verifiable shuffle of known values to prove that the ballots are consistent with the commitments.
– One or more auditors inspect the ballot box and the shuffle server.

For this to work, the voter's ciphertext must contain a valid opening of the voter's commitment. To achieve this, we use the verifiable encryption scheme of Lyubashevsky and Neven [LN17].

This architecture seems to be an acceptable trade-off between security and practicality. It achieves privacy for voters under the usual threat models, it provides cast-as-intended verification via return codes, it achieves coercion-resistance via revoting, and it achieves integrity as long as at least one auditor is honest. However, the architecture makes it difficult to simultaneously achieve privacy and universal verifiability. (We cannot simply publish the ballot box, the decrypted ballots and the shuffle proofs, because the shuffle server then learns who submitted which ballot, breaking privacy.) This is often not a significant problem, because coercion resistance requires keeping the decrypted ballots secret when so-called Italian attacks apply, and it is usually quite expensive to achieve universal verifiability without publishing the decrypted ballots. If Italian attacks do not apply or coercion resistance is otherwise not an issue, if one is willing to pay the price, it would be possible to distribute the decryption among two (or more) players by using nested encryption and nested commitments, after which everything could be published and universal verifiability is achieved. The cost is significant, though. Limited verifiability can be achieved in cheaper ways.

Voting with return codes. Our third contribution is the first construction of a voting system that supports so-called return codes for verifying that ballots have been cast as intended and that is based on a candidate post-quantum assumption.

One of the major challenges in using computers for voting is that computers can be compromised. Countermeasures such as Benaloh challenges do not work very well in practice, since they are hard to understand[3]. Return codes can provide integrity for voters with a fairly high rate of fraud detection [GL16]. Return codes do not work well with complex ballots, but our scheme could be modified to use return codes only for parts of a complex ballot.

We again use the commitments and verifiable encryption. The voter's computer commits to a pre-code and proves that this pre-code has been correctly computed from the ballot and some key material. It also verifiably encrypts an opening of this commitment. The pre-code is later decrypted and turned into a return code, which the voter can inspect.

Implementation of our voting scheme. Our fourth contribution is a concrete choice of parameters for the system along with a prototype implementation, demonstrating that the scheme is fully practical. We choose parameters in such

[3] Very few members of the International Association for Cryptologic Research use Benaloh challenges when casting ballots in their elections.

a way that arithmetic in the used algebraic structures can be efficiently implemented. This gives a fairly low computational cost for the scheme, so the limiting factor seems to be the size of the proofs. For elections with millions of voters, the total proof size will be measured in gigabytes, while systems based on discrete logarithms would produce much smaller proofs. Since we do not try to achieve universal verifiability, which means that proofs in our architecture are only handled by well-resourced infrastructure players, the proof size is unlikely to matter much. (If ordinary voters were to verify all the shuffle proofs, this would still not be infeasible, but it would be more of an issue.)

1.2 Related Work

Verifiable shuffles. The idea for a verifiable shuffle of known values that we use was introduced by Neff [Nef01]. Since [Nef01], there has been a huge body of work improving verifiable shuffles of ciphertexts, but not for constructions that use post-quantum assumptions.

Costa *et al.* [CMM19] use ZK proofs for lattice commitments to show a correct shuffle and re-randomization of a collection of ciphertexts. They also adopt some of the techniques from Neff, but instead of using a linear algebra argument they use multiplication proofs. This is conceptually simpler than our approach, but turns out to be less efficient even with the newer, improved multiplication proofs of [ALS20]. A related concept to the verifiable shuffle of known values is the decrypting mix-net [Cha81], which proves that the decryption of a collection of ciphertexts equals a given collection of messages. Decryption mix-nets can be very fast [BHM20], but these constructions provide guarantees of correct decryption only if at least one participant in the mix-net is honest at the time of decryption, unlike our approach which provides proper soundness even if both the ballot box and shuffle server are compromised at the time of decryption.

Candidate post-quantum cryptographic voting systems. There is a large body of academic work on cryptographic voting systems, and several systems have been deployed in practice in Europe in e.g. Estonia [HR16] and Norway [HR16,Gjø11], while Switzerland [LPT19] also planned to use an e-voting system. All of these systems make significant efforts to provide so-called cast-as-intended verification, to defend against compromise of the voter's computer. For lower-stakes elections, Helios [Adi08] has seen significant use. All of these systems have roughly the same architecture, and offer varying levels of verifiability. None of these systems are secure against quantum computers.

Many real-world political elections have ballots that are essentially very simple, such as a single yes/no question, or a t-out-of-n structure (even though many such races can be combined to form a visually and cognitively complex ballot). However, real-world voting systems can also have more complicated ballots that cannot be decomposed to a series of simple, independent races. For example, the Australian parliamentary ballot may encode a total order on all candidates in a district, and transferable votes make counting quite complex. While work has

been done on homomorphic counting for such elections, the usual approach is to recover cleartext ballots and count them.

While it is a simple exercise to use existing theoretical constructions to build a candidate quantum-safe voting system similar to the above deployed systems, the problem is that these constructions are practically inefficient, either because they are too computationally expensive or the proofs used are too large to make verification of many such proofs practical.

del Pino *et al.* [dLNS17] gives a feasible construction that uses homomorphic counting, but it is only applicable to yes/no-elections (though it can be extended to 1 out of n elections, at some cost). The scheme also does not try to defend against compromise of the voter's computer, limiting its applicability. Chillotti *et al.* [CGGI16] proposed a system based on homomorphic counting, but using fully homomorphic encryption. Again, this only supports 1 out of n elections, and practical efficiency is unclear. Gjøsteen and Strand [GS17] proposed a method for counting a complex ballot using homomorphic encryption. However, their scheme is not complete and the size of the circuit makes the system barely practical.

As discussed above, existing verifiable shuffles for candidate post-quantum secure cryptosystems could be used for generic constructions. Costa et al. [CMM19] uses certain ZK proofs for lattice commitments to show a correct shuffle and re-randomization of a collection of ciphertexts. The bottleneck of their approach are the underlying rather inefficient ZK proofs. The faster construct by Strand [Str19] is too restrictive in the choice of plaintext domain. Even given that shuffle, these schemes still require a verifiable (distributed) decryption for lattice-based constructions. These, currently, do not exist.

1.3 Using Verifiable Shuffles of Known Values

There are many other applications of a verifiable shuffle of known values than the one that is presented in this work. We give a brief overview of three such applications.

Verifiable shuffles. One application is to build a verifiable shuffle of ciphertexts of a homomorphic encryption scheme. The idea is to provide a method for proving that one collection of ciphertexts is a re-randomization of a second collection of ciphertexts, without revealing the correspondence of the ciphertexts.

The ciphertexts of such schemes can be re-randomized by adding an encryption of 0. The idea is then to commit to each homomorphic encryption separately and use a proof of linearity to show that the committed value is the ciphertext, plus an encryption of 0. Depending on the commitment and homomorphic encryption scheme, this proof can be very efficient - in particular if proofs of correct encryption are "cheap" using the commitment scheme. Then, one can perform a proof of shuffle of known openings on these auxiliary commitments, which succeeds if a permutation of the re-randomized ciphertexts is revealed.

Verifiable shuffled decryption. A related concept is *verifiable shuffled decryption*, to prove that a collection of ciphertexts decrypt to a collection of messages.

For this, one would create auxiliary commitments to the decryptions of each ciphertext as well as make a commitment to the secret key. Using the homomorphism on the commitments, one can now show that each commitment indeed contains the correct plaintext. Then, one can apply the proof of verifiable shuffle of known values to reveal the openings of the commitments in shuffled form.

The efficiency of this approach depends again on the choice of commitment and encryption scheme, and how good they fit together in terms of homomorphically evaluating the decryption function.

Voting with less trust. The above two ideas can be used to construct a cryptographic voting system that does not rely on a single server which performs decryptions and shuffles together. Each voter would directly encrypt his ballot, instead of committing to it. These votes are then shuffled and re-randomized multiple times, before they are verifiably decrypted (using threshold decryption). Such a system would be less susceptible to attacks against the secrecy of the votes. The downside is that it needs efficient threshold verifiable decryption to protect the privacy of the ballot.

2 Preliminaries

2.1 Notation

If Φ is a probability distribution, then $z \xleftarrow{\$} \Phi$ denotes that z was sampled according to Φ. If S is a finite set, then $s \xleftarrow{\$} S$ denotes that s was sampled uniformly from the set S. The expressions $z \leftarrow xy$ and $z \leftarrow \texttt{Func}(x)$ denote that z is assigned the product of x and y and the value of the function `Func` evaluated on x, respectively.

For two matrices $\boldsymbol{A} \in S^{\alpha\times\beta}, \boldsymbol{B} \in S^{\gamma\times\delta}$ over an arbitrary ring S, we denote by $\boldsymbol{A} \otimes \boldsymbol{B} \in S^{(\alpha\cdot\gamma)\times(\beta\cdot\delta)}$ their tensor product, i.e. the matrix

$$\boldsymbol{B} = \begin{pmatrix} b_{1,1} & \dots & b_{1,\delta} \\ \vdots & \ddots & \vdots \\ b_{\gamma,1} & \dots & b_{\gamma,\delta} \end{pmatrix}, \qquad \boldsymbol{A} \otimes \boldsymbol{B} := \begin{pmatrix} b_{1,1} \cdot \boldsymbol{A} & \dots & b_{1,\delta} \cdot \boldsymbol{A} \\ \vdots & \ddots & \vdots \\ b_{\gamma,1} \cdot \boldsymbol{A} & \dots & b_{\gamma,\delta} \cdot \boldsymbol{A} \end{pmatrix}.$$

2.2 The Rings R and R_p

Let $p, r \in \mathbb{N}^+$ and $N = 2^r$. Then we define the rings $R = \mathbb{Z}[X]/\langle X^N + 1\rangle$ and $R_p = R/\langle p\rangle$, that is, R_p is the ring of polynomials modulo $X^N + 1$ with integer coefficients modulo a prime p. If p is congruent to 1 mod 2δ, for $N \geq \delta > 1$ a power of 2, then $X^N + 1$ splits into δ irreducible factors.

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 $\bar{f} \in R_p$ we choose coefficients as the representatives in $\left[-\frac{p-1}{2}, \frac{p-1}{2}\right]$, 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 2-norm to be $\|\boldsymbol{a}\|_2 = \sqrt{\sum \|a_i\|^2}$, and analogously for the ∞-norm. We omit the subscript in the case of the 2-norm.

One can show that sufficiently short elements in the ring R_p (with respect to the aforementioned norms) are invertible.

Lemma 1 ([LS18], Corollary 1.2). *Let $N \geq \delta > 1$ be powers of 2 and p a prime congruent to $2\delta+1$ mod 4δ. Then X^N+1 factors into δ irreducible factors $X^{N/\delta} + r_j$, for some r_j's in R_p. Additionally, any non-zero y such that*

$$\|y\|_\infty < p^{1/\delta}/\sqrt{\delta} \quad \text{or} \quad \|y\| < p^{1/\delta}$$

is invertible in R_p.

For the remaining part of this paper we will assume that the parameters p, δ and N are chosen such that Lemma 1 is satisfied. We define a set of short elements

$$D_{\beta_\infty} = \{x \in R_p \mid \|x\|_\infty \leq \beta_\infty\}.$$

We furthermore define

$$\mathcal{C} = \{c \in R_p \mid \|c\|_\infty = 1, \|c\|_1 = \nu\},$$

which consists of all elements in R_p that have trinary coefficients and are non-zero in exactly ν positions, and we denote by

$$\bar{\mathcal{C}} = \{c - c' \mid c \neq c' \in \mathcal{C}\}$$

the set of differences of distinct elements in $\mathcal{C}$. The size of $\mathcal{C}$ is $2^\nu\binom{N}{\nu}$. It can be seen from Lemma 1 that, for a suitable choice of parameters, we can ensure that all non-zero elements from the three sets are invertible.

We need a bound on how many roots a polynomial can have over the ring R_p. The total number of elements in the ring is $|R_p| = p^N$.

Lemma 2. *Let $N \geq \delta \geq 1$ be powers of 2, p a prime congruent to $2\delta+1$ mod 4δ and $T \subseteq R_p$. Let $g \in R_p[X]$ be a polynomial of degree τ. Then, g has at most τ^δ roots in T, and $\Pr[g(\rho) = 0 | \rho \xleftarrow{\$} T] \leq \tau^\delta/|T|$.*

Proof. First, by Lemma 1, we divide X^N+1 into δ irreducible factors $X^{N/\delta}+r_j$. Each of the irreducible factors contributes at most τ roots to a polynomial $g \in R_p[X]$ of degree τ. Using the Chinese remainder theorem to combine the roots, we get that g has at most τ^δ roots in R_p. If we choose $\rho \xleftarrow{\$} R_p$ uniformly at random, the probability that this is a root of g is the total number of roots divided by the size of the ring. Since T is a subset of R_p, it can contain at most as many roots as R_p itself. □

2.3 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

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

When sampling randomness for our lattice-based commitment and encryption schemes, we'll 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.

2.4 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. *The $\mathsf{SKS}^2_{n,k,\beta}$ problem is to find a short vector $\boldsymbol{y}$ satisfying $[\,\boldsymbol{I}_n \quad \boldsymbol{A}'\,]\cdot \boldsymbol{y} = \boldsymbol{0}^n$ for a given random matrix $\boldsymbol{A}'$. An algorithm $\mathcal{A}$ has advantage ϵ in solving the $\mathsf{SKS}^2_{n,k,\beta}$ problem if*

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

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. *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 the uniform distribution when given $\boldsymbol{A}'$. An algorithm $\mathcal{A}$ has advantage ϵ in solving the $\mathsf{DKS}^\infty_{n,k,\beta}$ problem if*

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

3 Lattice-Background: Commitments and ZK Proofs

We first introduce the commitments of Baum et al. [BDL+18], and continue with a zero-knowledge proof protocol of linear relation over the ring R_p using these commitments. The protocol is implicitly mentioned in [BDL+18].

3.1 Lattice-Based Commitments

Algorithms. The scheme consists of three algorithms: $\texttt{KeyGen}_{\mathrm{C}}$, $\texttt{Com}$, and $\texttt{Open}$ for key generation, commitments and verifying an opening, respectively. We describe these algorithms for committing to one message, and refer to [BDL+18] for more details.

$\texttt{KeyGen}_{\mathrm{C}}$ outputs a public matrix $\boldsymbol{B}$ over R_p of the form

$$\begin{aligned} \boldsymbol{B}_1 &= \begin{bmatrix} \boldsymbol{I}_n & \boldsymbol{B}'_1 \end{bmatrix} && \text{where } \boldsymbol{B}'_1 \xleftarrow{\$} R_p^{n\times(k-n)} \\ \boldsymbol{b}_2 &= \begin{bmatrix} \boldsymbol{0}^n & 1 & \boldsymbol{b}'_2 \end{bmatrix} && \text{where } (\boldsymbol{b}'_2)^\top \xleftarrow{\$} R_p^{(k-n-1)}, \end{aligned}$$

for width k and height $n+1$ of the public key $\texttt{pk} := \boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_1 \\ \boldsymbol{b}_2 \end{bmatrix}$.

$\texttt{Com}$ commits to messages $m \in R_p$ by sampling an $\boldsymbol{r}_m \xleftarrow{\$} D^k_{\beta_\infty}$ and computing

$$\texttt{Com}(m; \boldsymbol{r}_m) = \boldsymbol{B} \cdot \boldsymbol{r}_m + \begin{bmatrix} \boldsymbol{0} \\ m \end{bmatrix} = \begin{bmatrix} \boldsymbol{c}_1 \\ c_2 \end{bmatrix} = [\![m]\!].$$

$\texttt{Com}$ outputs $[\![m]\!]$ and $d = (m; \boldsymbol{r}_m, 1)$.

$\texttt{Open}$ verifies whether an opening $(m; \boldsymbol{r}_m, f)$ with $f \in \bar{\mathcal{C}}$ is a valid opening of $\boldsymbol{c}_1, c_2$ by checking if

$$f \cdot \begin{bmatrix} \boldsymbol{c}_1 \\ c_2 \end{bmatrix} \stackrel{?}{=} \boldsymbol{B} \cdot \boldsymbol{r}_m + f \cdot \begin{bmatrix} 0 \\ m \end{bmatrix},$$

and that $\|\boldsymbol{r}_m[i]\| \leq 4\sigma_{\mathrm{C}}\sqrt{N}$ for $i \in [k]$ with $\sigma_{\mathrm{C}} = 11 \cdot \beta_\infty \cdot \nu \cdot \sqrt{kN}$. $\texttt{Open}$ outputs 1 if all these conditions hold, and 0 otherwise.

Baum et al. [BDL+18] proved the security properties of the commitment scheme with respect to the knapsack problems (which in turn are versions of standard Module-SIS/Module-LWE problems) defined in Section 2.4. More concretely, they showed that any algorithm $\mathcal{A}$ that efficiently solves the hiding property can be turned into an algorithm $\mathcal{A}'$ solving $\mathsf{DKS}^{\infty}_{n+1,k,\beta_\infty}$ with essentially the same runtime and success probability. Furthermore, any algorithm $\mathcal{A}$ that efficiently solves the binding problem can be turned into an algorithm $\mathcal{A}''$ solving $\mathsf{SKS}^{2}_{n,k,16\sigma_{\mathrm{C}}\sqrt{\nu N}}$ with the same success probability.

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

Proposition 1. *Let $\boldsymbol{z}_0 = \texttt{Com}(m; \boldsymbol{r}_m)$ be a commitment with opening $(m; \boldsymbol{r}_m, f)$ and let $\boldsymbol{z}_1 = \texttt{Com}(\rho; \boldsymbol{0})$. Then $\boldsymbol{z}_0 - \boldsymbol{z}_1$ is a commitment with opening $(m-\rho; \boldsymbol{r}_m, f)$.*

The proof follows from the linearity of the verification algorithm.

3.2 Zero-Knowledge Proof of Linear Relations

Let $[\![x]\!], [\![x']\!]$ be commitments as above such that $x' = \alpha x + \beta$ for some public $\alpha, \beta \in R_p$. Then Π_{Lin} in Figure 1 shows a zero-knowledge proof of knowledge (ZKPoK) of this fact (it is an adapted version of the linearity proof in [BDL+18]). The proof is a Σ protocol that aborts[4] with a certain probability to achieve the zero-knowledge property. For the protocol in Figure 1 we define

$$[\![x]\!] = \texttt{Com}(x; \boldsymbol{r}) = \begin{bmatrix} \boldsymbol{c}_1 \\ c_2 \end{bmatrix} \quad , \quad [\![x']\!] = \texttt{Com}(x'; \boldsymbol{r}') = \begin{bmatrix} \boldsymbol{c}'_1 \\ c'_2 \end{bmatrix}.$$

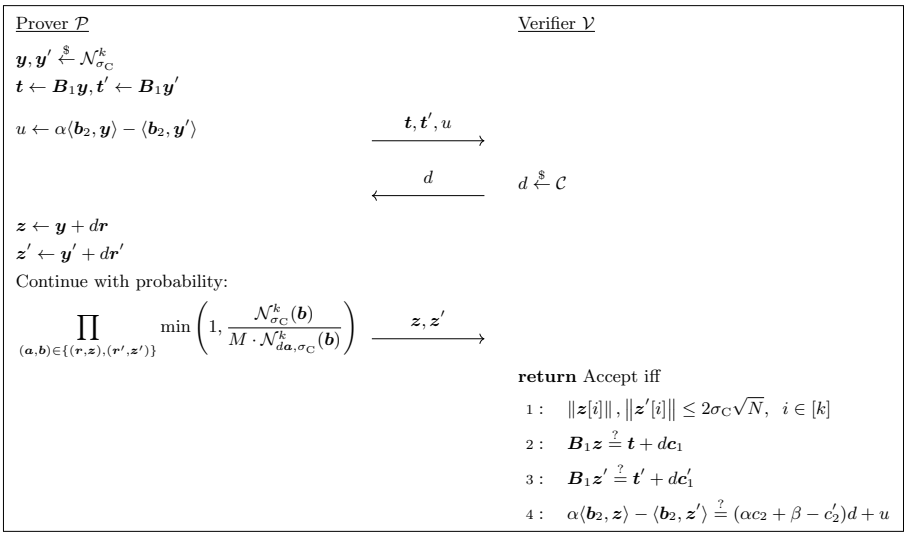

Fig. 1: Protocol Π_{Lin} is a Sigma-protocol to prove the relation $x' = \alpha x + \beta$, given the commitments $[\![x]\!], [\![x']\!]$ and the scalars α, β.

In [BDL+18] the authors show that a version of Π_{Lin} is a Honest-Verifier Zero-Knowledge Proof of Knowledge for the aforementioned commitment scheme. This can directly be generalized to relations of the form $\alpha \cdot \tilde{x} + \beta$ as follows:

Lemma 3. *Let $\alpha, \beta, [\![x]\!], [\![x']\!]$ be defined as above. Then Π_{Lin} is a HVZK proof of the relation*

$$R_{\text{Lin}} = \left\{ (s, w) \middle| \begin{array}{l} s = (\alpha, \beta, [\![x]\!], [\![x']\!], \boldsymbol{B}_1, \boldsymbol{b}_2), w = (\tilde{x}, \tilde{\boldsymbol{r}}, \tilde{\boldsymbol{r}}', f), \\ \texttt{Open}([\![x]\!], \tilde{x}, \tilde{\boldsymbol{r}}, f) = \texttt{Open}([\![x']\!], \alpha \cdot \tilde{x} + \beta, \tilde{\boldsymbol{r}}', f) = 1 \end{array} \right\}$$

[4] This approach is usually referred to as *Fiat Shamir with Aborts* (see e.g. [Lyu09, Lyu12] for a detailed description). If the proof is compiled with a random oracle into a NIZK, then these aborts only increase the prover time by a constant factor.

The proof for this is exactly the same as in [BDL+18], and we do only sketch it now: Assume that we can rewind an efficient poly-time prover and obtain two accepting transcripts with the same first message $\boldsymbol{t}, \boldsymbol{t}', u$ but differing $d, \overline{d}$ (as well as responses $\boldsymbol{z}, \boldsymbol{z}', \overline{\boldsymbol{z}}, \overline{\boldsymbol{z}}'$). Then one can extract valid openings $(\tilde{x}; \tilde{\boldsymbol{r}}, f)$ and $(\alpha\tilde{x} + \beta; \tilde{\boldsymbol{r}}', f)$ for $[\![x]\!], [\![x']\!]$ respectively as follows: From the two accepting transcripts and the equations checked by the verifier we can set $f = d - \overline{d}$, $\tilde{\boldsymbol{r}} = \boldsymbol{z} - \overline{\boldsymbol{z}}, \tilde{\boldsymbol{r}}' = \boldsymbol{z}' - \overline{\boldsymbol{z}}'$ where it must hold that

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

By setting $\tilde{x} = c_2 - f^{-1}\langle \boldsymbol{b}_2, \tilde{\boldsymbol{r}}\rangle$ and $\tilde{x}' = c_2' - f^{-1}\langle \boldsymbol{b}_2, \tilde{\boldsymbol{r}}'\rangle$, we then have that $\alpha x + \beta = x'$ by the aforementioned equation. The validity and bounds of the opening follow from the same arguments as in [BDL+18].

Compression. Using the techniques from [GLP12,BG14], as already mentioned in [BDL+18, Section 5.3], allows to compress the non-interactive version of the aforementioned zero-knowledge proof. The main idea is that the prover only hashes the parts of the proof that got multiplied by the uniformly sampled part $\boldsymbol{B}_1'$ of $\boldsymbol{B}_1$, and that the verifier only checks an approximate equality with these when recomputing the challenge. We do the following changes to the protocol.

The prover samples vectors $\boldsymbol{y}, \boldsymbol{y}'$ of dimension $k - n$ according to σ_{C}, then computes $\boldsymbol{t} = \boldsymbol{B}_1'\boldsymbol{y}$ and $\boldsymbol{t}' = \boldsymbol{B}_1'\boldsymbol{y}'$. Note that u is computed as before, as the n first values of $\boldsymbol{b}_2$ are zero. Then $\boldsymbol{z}$ and $\boldsymbol{z}'$ are computed as earlier, but are of dimension $k - n$ instead of k. The prover computes the challenge d as

$$d = \mathtt{H}(\boldsymbol{B}, [\![x]\!], [\![x']\!], \alpha, \beta, u, \lfloor \boldsymbol{t} \rceil_\gamma, \lfloor \boldsymbol{t}' \rceil_\gamma)$$

where $\gamma \in \mathbb{N}$ and $\lfloor \cdot \rceil_\gamma$ denotes rounding off the least γ bits.

To make sure that the non-interactive proof can be verified, we must ensure that d can be re-computed from the public information. Let $\hat{\boldsymbol{t}} = \boldsymbol{B}_1'\boldsymbol{z} - d\boldsymbol{c}_1$ and $\hat{\boldsymbol{t}'} = \boldsymbol{B}_1'\boldsymbol{z}' - d\boldsymbol{c}_1'$ and observe that $\hat{\boldsymbol{t}}[i] - \boldsymbol{t}[i] = d\boldsymbol{r}[i]$, for each coordinate $i \in [n]$, and similar for $\hat{\boldsymbol{t}'}$ and $\boldsymbol{t}'$. For honestly generated randomness, for each $i \in [k]$, we have that $\|\boldsymbol{r}[i]\| \leq \beta_\infty\sqrt{N}$, and since $d \in \bar{\mathcal{C}}$, we have that $\|d\| = \sqrt{\nu}$. It follows that $\|d\boldsymbol{r}[i]\|_\infty \leq \beta_\infty\sqrt{\nu N}$, and similar for $d\boldsymbol{r}'[i]$. When hashing $\boldsymbol{t}$ and $\boldsymbol{t}'$ to get the challenge d, we then remove the $\gamma = \lceil \log \beta_\infty\sqrt{\nu N} \rceil$ lower bits of each coordinate first, to ensure that both the prover and the verifier compute on the same value. Hence, before outputting the proof, the prover will also test that

$$\begin{aligned} d' &= \mathtt{H}(\boldsymbol{B}, [\![x]\!], [\![x']\!], \alpha, \beta, \hat{u}, \lfloor \boldsymbol{B}_1'\boldsymbol{z} - d\boldsymbol{c}_1 \rceil_\gamma, \lfloor \boldsymbol{B}_1'\boldsymbol{z}' - d\boldsymbol{c}_1' \rceil_\gamma), \text{ where} \\ \hat{u} &= \alpha\langle [1 \quad \boldsymbol{b}_2'], \boldsymbol{z}\rangle - \langle [1 \quad \boldsymbol{b}_2'], \boldsymbol{z}'\rangle - (\alpha c_2 + \beta - c_2')d. \end{aligned}$$

The prover then outputs the proof $(d, \boldsymbol{z}, \boldsymbol{z}')$ if $d = d'$ and $\|\boldsymbol{z}[i]\|, \|\boldsymbol{z}'[i]\| \leq 2\sigma_{\mathrm{C}}\sqrt{N}$ (when setting up the check as in [GLP12, BG14], then the test will fail with probability at most $1/2$), and the verifier will make the same checks to validate it. The proof size is reduced from k to $k - n$ Gaussian-distributed ring-elements, making the proof size a total of $2(k - n)\log(6\sigma_{\mathrm{C}})$ bits.

4 Protocol: Zero-Knowledge Proof of Correct Shuffle

In this section we present the shuffle protocol for openings of commitments. We construct a public-coin $4 + 3\tau$-move protocol[5] such that the commit-challenge-response stages require the prover to solve a system of linear equations in order to prove a correct shuffle. Our construction extends Neff's construction [Nef01] to the realm of post-quantum assumptions.

The proof of shuffle protocol will use the commitments defined in Section 3. For the shuffle proof to work, the prover $\mathcal{P}$ and verifier $\mathcal{V}$ receive commitments $\{[\![m_i]\!]\}_{i=1}^{\tau}$. $\mathcal{P}$ also receives the set of openings $\{(m_i, \boldsymbol{r}_i)\}_{i=1}^{\tau}$ as well as a permutation $\pi \in S_\tau$. Additionally, both parties also obtain $\{\hat{m}_i\}_{i=1}^{\tau}$.

The goal is to ensure that the following relation R_{Shuffle} holds:

$$R_{\text{Shuffle}} = \left\{ (s, w) \;\middle|\; \begin{array}{l} s = ([\![m_1]\!], \ldots, [\![m_\tau]\!], \hat{m}_1, \ldots, \hat{m}_\tau, \hat{m}_i \in R_p), \\ w = (\pi, f_1, \ldots, f_\tau, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_\tau), \pi \in S_\tau, \\ \forall i \in [\tau] : \ \texttt{Open}([\![m_{\pi^{-1}(i)}]\!], \hat{m}_i, \boldsymbol{r}_i, f_i) = 1 \end{array} \right\}$$

To use the idea of Neff, all $\hat{m}_i$ messages involved have to be invertible. However, this may not be the case for arbitrary ring elements. We start by showing that if $\mathcal{V}$ samples a random ρ in R_p then all $\hat{m}_i - \rho$ will be invertible with high probability:

Proposition 2. *Let $N \geq \delta \geq 1$ be powers of 2, p a prime congruent to $2\delta + 1$ mod 4δ. Then*

$$\Pr_{x_1,\ldots,x_\tau \in R_p}[x_1 - \rho, \ldots, x_\tau - \rho \text{ invertible in } R_p \mid \rho \xleftarrow{\$} R_p] \leq 1 - \max(1, \tau \cdot (1 - e^{-\delta/p})).$$

Plugging in realistic parameters ($p = 2^{32}, \delta = 2, \tau = 1,000,000$) we see that the probability of of all $\hat{m}_i - \rho$ being simultaneously invertible is essentially 1.

Proof. By assumption, R_p factors into δ irreducible factors. Therefore, the number of invertible elements in R_p is exactly

$$(p^{N/\delta - 1} \cdot (p-1))^\delta = (p^{N/\delta} - p^{N/\delta - 1})^\delta.$$

Let S_1 be the set of choices of $\rho \in R_p$ such that $x_1 - \rho$ is not invertible, and similarly define $S_2, \ldots, S_\tau$. We know that $|S_i| \leq p^N - (p^{N/\delta} - p^{N/\delta-1})^\delta$ and so, if all S_i are disjoint sets, $|S_1 \cup \cdots \cup S_\tau| \leq \tau p^N - \tau(p^{N/\delta} - p^{N/\delta-1})^\delta$. Dividing by the total number of elements in R_p we then get that the probability of ρ hitting $S_1 \cup \cdots \cup S_\tau$ can be bounded by

$$\begin{aligned} \tau \cdot \frac{p^N - (p^{N/\delta} - p^{N/\delta-1})^\delta}{p^N} &= \tau - \tau \cdot \left(\frac{p^{N/\delta} - p^{N/\delta-1}}{p^{N/\delta}} \right)^\delta \\ &= \tau \cdot \left(1 - \left(1 - \frac{1}{p} \right)^\delta \right) \leq \tau \cdot (1 - e^{-\delta/p}), \end{aligned}$$

where the inequality follows from $1 + x \leq e^x$. □

[5] This is only a theoretical problem as the protocol is public-coin and can therefore directly be transformed into NIZKs using the Fiat-Shamir transform.

The first step for our shuffle protocol will be that $\mathcal{V}$ picks a random appropriate $\rho \xleftarrow{\$} R_p$ and sends ρ to $\mathcal{P}$. $\mathcal{P}$ and $\mathcal{V}$ then locally compute the values $\hat{M}_i, M_i$ by setting $M_i = m_i - \rho$, $\hat{M}_i = \hat{m}_i - \rho$. The proof, on a high level, then shows that $\prod_i M_i = \prod_i \hat{M}_i$. This is in fact sufficient, as the $m_i, \hat{m}_i$ can be considered as roots of polynomials of degree τ. By subtracting ρ from each such entry and multiplying the results we obtain the evaluation of these implicit polynomials in the point ρ, and if the $\hat{m}_i$ are not a permutation of the m_i then these implicit polynomials will be different. At the same time, the number of points on which both polynomials can agree is upper-bounded as shown in Lemma 2.

Prover $\mathcal{P}$		Verifier $\mathcal{V}$
	$\xleftarrow{\rho}$	$\rho \xleftarrow{\$} R_p \setminus \{\hat{m}_i\}_{i=1}^{\tau}$
$\hat{M}_i = \hat{m}_i - \rho$		$\hat{M}_i = \hat{m}_i - \rho$
$M_i = m_i - \rho$		$[\![M_i]\!] = [m_i] - \rho$
$\theta_i \xleftarrow{\$} R_p, \forall i \in [\tau - 1]$		
Compute $[\![D_i]\!]$ as in Eq. (1), i.e.		
$[\![D_1]\!] = [\![\theta_1 \hat{M}_1]\!], [\![D_\tau]\!] = [\![\theta_{\tau-1} M_\tau]\!]$,		
$[\![D_i]\!] = [\![\theta_{i-1} M_i + \theta_i \hat{M}_i]\!]$ for $i \in [\tau - 1] \setminus \{1\}$	$\xrightarrow{\{[\![D_i]\!]\}_{i=1}^{\tau}}$	
	$\xleftarrow{\beta}$	$\beta \xleftarrow{\$} R_p$
Compute $s_i, \forall i \in [\tau - 1]$ as in (3).	$\xrightarrow{\{s_i\}_{i=1}^{\tau-1}}$	
		Use Π_{Lin} to prove that
		(1) $\beta [\![M_1]\!] + s_1 \hat{M}_1 = [\![D_1]\!]$
		(2) $\forall i \in [\tau - 1] \setminus \{1\} : \; s_{i-1} [\![M_i]\!] + s_i \hat{M}_i = [\![D_i]\!]$
		(3) $s_{\tau-1} [\![M_\tau]\!] + (-1)^\tau \beta \hat{M}_\tau = [\![D_\tau]\!]$
		i.e. all equations from (2)
		return accept iff all instances of Π_{Lin} are accepting

Fig. 2: The public-coin zero-knowledge protocol of correct shuffle Π_{Shuffle}.

Our public-coin zero-knowledge protocol proves this identity of evaluations of these two polynomials by showing that a particular set of linear relations (2) is satisfied (we will show later how it is related to the aforementioned product of M_i and $\hat{M}_i$).

As a first step, $\mathcal{P}$ draws $\theta_i \xleftarrow{\$} R_p$ uniformly at random for each $i \in \{1, \ldots, \tau\}$, and computes the commitments

$$\begin{aligned} & [\![D_1]\!] = [\![\theta_1 \hat{M}_1]\!] \\ \forall j \in \{2, \ldots, \tau - 1\} : \; & [\![D_j]\!] = [\![\theta_{j-1} M_j + \theta_j \hat{M}_j]\!] \\ & [\![D_\tau]\!] = [\![\theta_{\tau-1} M_\tau]\!]. \end{aligned} \tag{1}$$

$\mathcal{P}$ then sends these commitments $\{[\![D_i]\!]\}_{i=1}^{\tau}$ to the verifier[6] $\mathcal{V}$, which in turn chooses a challenge $\beta \in R_p$, whereupon $\mathcal{P}$ computes $s_i \in R_q$ such that the following equations are satisfied:

$$\begin{aligned} \beta M_1 + s_1 \hat{M}_1 &= \theta_1 \hat{M}_1 \\ \forall j \in \{2, \ldots, \tau - 1\} : \; s_{j-1} M_j + s_j \hat{M}_j &= \theta_{j-1} M_j + \theta_j \hat{M}_j \\ s_{\tau-1} M_\tau + (-1)^\tau \beta \hat{M}_\tau &= \theta_{\tau-1} M_\tau. \end{aligned} \tag{2}$$

To verify the relations, $\mathcal{P}$ uses the protocol Π_{Lin} from Section 3 to prove that the content of each commitment $[\![D_i]\!]$ is such that D_i, M_i and $\hat{M}_i$ satisfies the equations (2). The protocol ends when $\mathcal{V}$ has verified all the τ linear equations in (2). In order to compute the s_i values, we can use the following fact:

Lemma 4. *Choosing*

$$s_j = (-1)^j \cdot \beta \prod_{i=1}^{j} \frac{M_i}{\hat{M}_i} + \theta_j \tag{3}$$

for all $j \in 1, \ldots, \tau - 1$ yields a valid assignment for Equation (2).

Proof. The correctness of this choice follows directly by considering all three cases: For the first case, we have that

$$\begin{aligned} \beta M_1 + s_1 \hat{M}_1 &= \beta M_1 + (-\beta M_1 / \hat{M}_1 + \theta_1) \hat{M}_1 \\ &= \theta_1 \hat{M}_1. \end{aligned}$$

In the second case, it holds that

$$\begin{aligned} s_{j-1} M_j + s_j \hat{M}_j &= ((-1)^{j-1} \beta \prod_{i=1}^{j-1} M_i / \hat{M}_i + \theta_{j-1}) M_j + ((-1)^j \beta \prod_{i=1}^{j} M_i / \hat{M}_i + \theta_1) \hat{M}_j \\ &= \theta_{j-1} M_j + \theta_j \hat{M}_j \end{aligned}$$

where the β-terms cancel. For the third case, since $\dfrac{M_1 \cdots M_\tau}{\hat{M}_1 \cdots \hat{M}_{\tau-1}} = \hat{M}_\tau$ so

$$\begin{aligned} s_{\tau-1} M_\tau + (-1)^\tau \beta \hat{M}_\tau &= ((-1)^{\tau-1} \beta \prod_{i=1}^{\tau-1} M_i / \hat{M}_i + \theta_{\tau-1}) M_\tau + (-1)^\tau \beta \hat{M}_\tau \\ &= \theta_{\tau-1} M_\tau \end{aligned}$$

□

From Lemma 4 it is clear that the protocol is indeed complete. Interestingly, this choice of s_j also makes these values appear random: each s_j is formed by adding a fixed term to a uniformly random private value θ_j. This will be crucial to show the zero-knowledge property. For the soundness, we get the following:

[6] $\mathcal{P}$ does not show that these commitments are well-formed, this will not be necessary.

Lemma 5. *Assume that the commitment scheme is binding and that Π_{Lin} is a sound proof of knowledge for the relation R_{Lin} except with probability t. Then the protocol in Figure 2 is a sound proof of knowledge for the relation R_{Shuffle} except with probability $\epsilon \leq \frac{\tau^\delta+1}{|R_p|} + 4\tau t$.*

Proof. To prove the statement, we will construct a PPT algorithm $\mathcal{E}$ called *extractor* which interacts with $\mathcal{P}^*$ in a black-box manner and which will output a witness w for a statement s such that $(s, w) \in R_{\text{Shuffle}}$ given that $\mathcal{P}^*$ wins for a given s with more than the stated probability. The expected runtime of $\mathcal{E}$ is $poly(\tau, t)/\epsilon$. The extractor algorithm will proceed as follows:

1. Construct sub-extractors $\mathcal{E}_i$ which for each $i \in [\tau]$ do the following:
   (a) Run instances with an arbitrary randomness tape for $\mathcal{P}^*$ as well as arbitrary challenges until an accepting transcript is found.
   (b) Upon finding an accepting transcript, rewind $\mathcal{P}^*$ until after the first message in the ith instance of Π_{Lin} was sent. Then probe for a second challenge for the ith proof that leads to an accepting transcript[7].
2. Subtract ρ_i from all M_i where ρ_i is the value used by the extractor $\mathcal{E}_i$. Then for each $[\![m_i]\!]$ let the opening be $(m_i; \boldsymbol{r}_i, f_i)$. If the m_i are indeed a permutation of the $\hat{m}_i$ then output the respective $w = (\pi, f_1, \ldots, f_\tau, \boldsymbol{r}, \ldots, \boldsymbol{r}_\tau)$.

We will first argue why the above algorithm is expected polynomial-time: The runtime of each $\mathcal{E}_i$ is expected polynomial time by a standard heavy-row argument, as we only need to rewind on a specific instance only. Applying the heavy-row argument is possible as the success probability of $\mathcal{P}^*$ is above $4t$ (we lose a factor of 4 in the heavy-row argument). We now argue why also Step 2 is polynomial-time i.e. that $\mathcal{E}$ outputs a witness.

Let us assume we would create two transcripts for an identical ρ but differing β, β' where we rewind $\mathcal{P}^*$. We would then obtain different s_i, s_i' such that all the equations are proven by $\mathcal{P}^*$ with Π_{Lin}. From the soundness of Π_{Lin} we obtain:

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

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

$$\underbrace{\begin{bmatrix} M_1 & \hat{M}_1 & 0 & \ldots & 0 & 0 \\ 0 & M_2 & \hat{M}_2 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & M_{\tau-1} & \hat{M}_{\tau-1} \\ (-1)^\tau \hat{M}_\tau & 0 & 0 & \ldots & 0 & M_\tau \end{bmatrix}}_{\boldsymbol{M}} \underbrace{\begin{bmatrix} \beta - \beta' \\ s_1 - s_1' \\ \vdots \\ s_{\tau-2} - s_{\tau-2}' \\ s_{\tau-1} - s_{\tau-1}' \end{bmatrix}}_{\boldsymbol{c}} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \tag{4}$$

[7] One would additionally in parallel run a process that aborts $\mathcal{E}_i$ with very small probability, see e.g. [BBC+18, Lemma 3]. We leave this out for the sake of simplicity.

We can directly see from the above that all $M_i, s_i - s'_i$ must be non-zero and that therefore $s_i \neq s'_i$: From the last equation and the assumption that $\beta \neq \beta'$ we know that $M_\tau \times (s_{\tau-1} - s'_{\tau-1}) \neq 0$ as $\hat{M}_\tau$ is invertible. From the second-to-last equation and due to the invertibility of $\hat{M}_{\tau-1}$ the same holds for $M_{\tau-1}$ and $s_{\tau-2} - s'_{\tau-2}$. By induction, this applies to all values.

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

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

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

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

that is, $g(X)$ and $\hat{g}(X)$ are the polynomials which have m_i and $\hat{m}_i$ as their roots, respectively. If there exists no permutation such that $\hat{m}_i = m_{\pi(i)}, \forall i = 1, \dots, \tau$, then $g(\rho) = \hat{g}(\rho)$ (i.e. $\det(\boldsymbol{M}) = 0$) for at most τ^δ choices of ρ according to Lemma 2. Due to the lower-bound on the success probability of $\mathcal{P}^*$ there must exist accepting transcripts for more than τ^δ choices of ρ with more than one accepting choice of β where all instances of Π_{Lin} prove correct. Then, by the above argument and because the commitment scheme is binding, it must hold that the values extracted by $\mathcal{E}$ indeed are a permutation of the $\hat{m}_i$. □

From Lemmas 4 and 5 we get the following theorem:

Theorem 1. *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 the relation* $\mathcal{R}_{\text{Lin}}$ *with soundness error* t*. Then the protocol* Π_{Shuffle} *is an HVZK Proof of Knowledge for the relation* $\mathcal{R}_{\text{Shuffle}}$ *with soundness error* $(\tau^\delta + 1)/|R_p| + 4\tau t$*.*

The proof of completeness and HVZK can be found in Appendix A.

5 Applications to Electronic Voting

We now construct an e-voting protocol by combining the shuffle protocol from Section 4 with a verifiable encryption scheme and a return code mechanism.

Towards this end, consider the shuffled openings of commitments as the outcome of the election, meaning that each commitment will contain a vote. Commitments are not sufficient for a voting system, and we also need encryptions of the actual ballots and these must be tied to the commitments, so that the shuffling server can open the commitments without anyone else being able to. We use a version of the verifiable encryption scheme by Lyubashevsky and Neven [LN17] to verifiably encrypt openings under a public key that belongs to the shuffle server. We also reuse the verifiable encryption to get a system for return codes. The return code computation is done in two stages, where the first stage is done on the voter's computer, and the second stage is done by an infrastructure player. The voter's computer commits to its result and verifiably encrypts an opening of that commitment for the infrastructure player. Then it proves that the commitment contains the correct value.

We will now describe the verifiable encryption scheme that we use as well as the return code mechanism in more detail, before explaining how to construct the full e-voting protocol.

5.1 Verifiable Encryption

In a *verifiable encryption scheme*, anyone can verify that the encrypted plaintext has certain properties. We use a version of [LN17] where we use a generalization of the [LPR13, BGV12] encryption system. The reason is that in [LN17] the public key only consists of single polynomials of degree N, requiring that the plaintext vector must also be a multiple of N - which might not always be the case as in our setting.

In our setting, the goal is to show that the plaintext is a value $\boldsymbol{\mu} \in D^{\kappa}_{\beta_\infty}$ such that

$$\boldsymbol{T}\boldsymbol{\mu} = \boldsymbol{u} \bmod p, \tag{5}$$

for some fixed $\boldsymbol{T}, \boldsymbol{u}$ and where $\boldsymbol{T} \in R_p^{\lambda \times \kappa}$. Using the construction of [LN17], one can show a weaker version of the statement, namely that decryption yields a small $\bar{\boldsymbol{\mu}}$ and $\bar{c} \in \overline{\mathcal{C}}$ over R_p such that

$$\boldsymbol{T}\bar{\boldsymbol{\mu}} = \bar{c}\boldsymbol{u} \bmod p. \tag{6}$$

We will see that this will be sufficient for our voting scheme[8].

The [LN17] verifiable encryption scheme consists of 4 algorithms: Key generation $\texttt{KeyGen}_V$, encryption `Enc`, verification `Ver` and decryption `Dec`. We will first describe the underlying non-verifiable encryption scheme and then explain how it is made verifiable.

The encryption, verification and decryption algorithms are described in Figures 3, 4 and 5 respectively. Here, encryption follows [LPR13, BGV12] but additionally computes a NIZK that the plaintext is a valid preimage of Equation

[8] Recently, [ALS20] showed a more efficient HVZKPoK for the respective relation. Unfortunately, their proof cannot guarantee that $\bar{c}$ is invertible, which is crucial for the verifiability of the encryption scheme. Their optimization can therefore not be applied in our setting.

5 and also bounded. `Ver` validates the NIZK, while `Dec` decrypts to a short plaintext that is valid under Equation 6.

To generate a public key $(\boldsymbol{A}, \boldsymbol{t})$ for the verifiable encryption scheme one samples $\boldsymbol{A} \leftarrow R_q^{\ell \times \ell}$ uniformly at random as well as $\boldsymbol{s}_1, \boldsymbol{s}_2 \leftarrow D_1^\ell$, setting $\boldsymbol{t} \leftarrow \boldsymbol{A}\boldsymbol{s}_1 + \boldsymbol{s}_2$ and outputting $(\boldsymbol{A}, \boldsymbol{t})$ as public key as well as $\boldsymbol{s_1}$ as private key. To encrypt a single message $\mu \in D_{\beta_\infty}$, first sample $\boldsymbol{r}, \boldsymbol{e} \leftarrow D_1^\ell, e' \leftarrow D_1$ and then compute

$$\begin{bmatrix} \boldsymbol{v} \\ w \end{bmatrix} = \begin{bmatrix} p(\boldsymbol{A}^\top \boldsymbol{r} + \boldsymbol{e}) \bmod q \\ p(\langle \boldsymbol{t}, \boldsymbol{r} \rangle + e') + \mu \bmod q \end{bmatrix}.$$

To decrypt a ciphertext $(\boldsymbol{v}, w) \in R_q^\ell \times R_q$ compute

$$w - \langle \boldsymbol{v}, \boldsymbol{s}_1 \rangle \bmod q \bmod p = (p(\langle \boldsymbol{r}, \boldsymbol{s}_2 \rangle + e' - \langle \boldsymbol{e}, \boldsymbol{s}_1 \rangle) + \mu \bmod q) \bmod p = \mu,$$

where the last equality holds if $\|p(\langle \boldsymbol{r}, \boldsymbol{s}_2 \rangle + e' - \langle \boldsymbol{e}, \boldsymbol{s}_1 \rangle + \mu)\|_\infty < q/2$.

To encrypt a vector $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_\kappa) \in D_{\beta_\infty}^\kappa$ we can abbreviate our equations in matrix notation in the following way:

$$\begin{bmatrix} \boldsymbol{A}^\top \otimes (p\boldsymbol{I}_\kappa) & p\boldsymbol{I}_{\ell \cdot \kappa} & \boldsymbol{0}^{(\ell \cdot \kappa) \times \kappa} & \boldsymbol{0}^{(\ell \cdot \kappa) \times \kappa} \\ \boldsymbol{t}^\top \otimes (p\boldsymbol{I}_\kappa) & \boldsymbol{0}^{\kappa \times (\ell \cdot \kappa)} & p\boldsymbol{I}_\kappa & \boldsymbol{I}_\kappa \end{bmatrix} \begin{bmatrix} \boldsymbol{r} \\ \boldsymbol{e} \\ \boldsymbol{e}' \\ \boldsymbol{\mu} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \end{bmatrix} \bmod q, \tag{7}$$

for $\boldsymbol{r}, \boldsymbol{e} \xleftarrow{\$} D_1^{\ell \cdot \kappa}, \boldsymbol{e}' \xleftarrow{\$} D_1^\kappa$ and where $\boldsymbol{v} \in R_q^{\ell \cdot \kappa}, \boldsymbol{w} \in R_q^\kappa$. For decryption we get:

$$\boldsymbol{\mu}[i] = \boldsymbol{w}[i] - \langle \boldsymbol{s}_1, \hat{\boldsymbol{v}}_i \rangle \bmod q \bmod p$$

where $\hat{\boldsymbol{v}}_i = [\boldsymbol{v}[(i-1) \cdot \ell + 1], \boldsymbol{v}[(i-1) \cdot \ell + 2], \ldots, \boldsymbol{v}[i \cdot \ell]]^\top$ which follows from the definition of the tensor product. For correctness, the aforementioned decryption bound generalizes to

$$\|p(\langle \hat{\boldsymbol{r}}_i, \boldsymbol{s}_2 \rangle + \boldsymbol{e}'[i] - \langle \hat{\boldsymbol{e}}, \boldsymbol{s}_1 \rangle) + \boldsymbol{\mu}[i]\|_\infty < q/2$$

where $\hat{\boldsymbol{r}}_i, \hat{\boldsymbol{e}}_i$ are analogously defined to $\hat{\boldsymbol{v}}_i$. Using standard bounds on ∞-norms of products of elements in R_p, this directly translates into the requirement that $q > 2p(2\ell \cdot N^2 \cdot {\beta_\infty}^2 + N + 1)$.

To make this verifiable, we will use a NIZK which shows for a ciphertext $\boldsymbol{v}, \boldsymbol{w}$ that the sender knows $\boldsymbol{r}, \boldsymbol{e}, \boldsymbol{e}', \boldsymbol{\mu}$ that are bounded such that

- $\boldsymbol{r}, \boldsymbol{e}, \boldsymbol{e}', \boldsymbol{\mu}$ are a preimage of $\boldsymbol{v}, \boldsymbol{w}$ modulo q as in Equation 7.
- $\boldsymbol{\mu}$ fulfills equation (5) modulo p.

To prove both relations simultaneously we use a standard lattice-based zero-knowledge proof for the specific relation, and by using the Fiat-Shamir transform [FS87] this then becomes non-interactive. As mentioned before, the output of `Dec` will be a witness for Equation 6, i.e. it will also contain the additional factor $\bar{c}$.

We now argue that our modification of the scheme is still secure.

First of all, the encryption scheme (as the authors of [BGV12] show) can be generalized to work with the generalized M-LWE assumption[9]. As the actual matrix dimensions for the encryption scheme do not change between our instance of the verifiable encryption scheme and [LN17] the same security proof still applies with respect to privacy.

The verifiability and thus decryption of the above construction directly follows from the original proof, as neither of the conditions of [LN17, Lemma 3.1] are altered by changing the matrix structure. Furthermore, as we will choose $\ell = 2$ in our setting even this R_q of smaller dimension than in the original work has a large enough challenge space necessary for the (non-)interactive proofs to be sound. We will therefore be able to basically rely on the same security analysis as [LN17] and can essentially re-use their parameters (with some slightly increased p, q).

There are multiple parameter restrictions in [LN17] in order to achieve security. These also apply to our setting:

1. The underlying encryption scheme must safely be able to encrypt and decrypt messages from R_p^κ. For this, we obviously need that message and noise, upon decryption, do not "overflow" $\bmod q$ while the noise at the same time must be large enough such that the underlying MLWE-problem is hard. For concrete parameters, the latter can be established by e.g. using the LWE Estimator [APS15]. For correctness of the decryption alone, we require that the decryption of a correct encryption must yield[10] a value $< q/2$. This also means that the decryption algorithm will always terminate for $\bar{c} = 1$ in case the encryptor is honest.
2. The NIZK requires "quasi-unique responses", which (as the authors of [LN17] argue) it will have with overwhelming probability over the choice of $\boldsymbol{A}$ as long as $24\sigma_{\mathrm{E}}^2 < q$.

Encrypting openings of commitments. We want to make sure that the voter actually knows his vote, and that the commitment and the opening of the commitment are well-formed. We also want to ensure that the ciphertext actually contains a valid opening of the commitment. This can be achieved if the voter creates a proof that the underlying plaintext is an opening of the commitment. Then the ballot box can ensure that the shuffle server will be able to decrypt the vote and use it in the shuffle protocol. Note that the voter may send a well-formed but invalid vote, but then the shuffle server can publicly discard that vote later, and everyone can check that the vote indeed was invalid.

Recall that the commitment is of the form

$$\mathtt{Com}(m; \boldsymbol{r}_m) = \begin{bmatrix} \boldsymbol{c}_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{B}_1 \\ \boldsymbol{b}_2 \end{bmatrix} \cdot \boldsymbol{r}_m + \begin{bmatrix} \boldsymbol{0} \\ m \end{bmatrix}.$$

[9] M-LWE generalizes the Ring-LWE assumption [LS15] and is a more conservative security assumption for the same dimensions of the matrix.

[10] This translates into the requirement that $q > 2p(2\ell \cdot N^2 \cdot {\beta_\infty}^2 + N + 1)$.

Input: Public key $\mathtt{pk} = (\boldsymbol{A}, \boldsymbol{t}, p, q)$, pair $(\boldsymbol{T}, \boldsymbol{u}), \boldsymbol{\mu} \in D_\beta^\kappa$ such that $\boldsymbol{T}\boldsymbol{\mu} = \boldsymbol{u}$, hash function $H : \{0,1\}^* \to \mathcal{C}$,

$\sigma_{\mathrm{E}} = 11 \cdot \max_{c \in \mathcal{C}} \|c\| \cdot \sqrt{\kappa N(3+\beta)}$

Output: ciphertext $(\boldsymbol{v}, \boldsymbol{w}, c, \boldsymbol{z}) \in R_q^{\ell\cdot\kappa} \times R_q^\kappa \times \mathcal{C} \times R^{(2\ell+2)\kappa}$

1 : $\boldsymbol{r}, \boldsymbol{e} \xleftarrow{\$} D_1^{\ell\cdot\kappa}, \boldsymbol{e'} \xleftarrow{\$} D_1^\kappa$

2 : $\begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \end{bmatrix} \leftarrow \begin{bmatrix} \boldsymbol{A}^\top \otimes (p\boldsymbol{I}_\kappa) & p\boldsymbol{I}_{\ell\cdot\kappa} & \boldsymbol{0}^{(\ell\cdot\kappa)\times\kappa} & \boldsymbol{0}^{(\ell\cdot\kappa)\times\kappa} \\ \boldsymbol{t}^\top \otimes (p\boldsymbol{I}_\kappa) & \boldsymbol{0}^{\kappa\times(\ell\cdot\kappa)} & p\boldsymbol{I}_\kappa & \boldsymbol{I}_\kappa \end{bmatrix} \begin{bmatrix} \boldsymbol{r} & \boldsymbol{e} & \boldsymbol{e'} & \boldsymbol{\mu} \end{bmatrix}^\top$

3 : $\boldsymbol{y} \leftarrow \begin{bmatrix} \boldsymbol{y_r} & \boldsymbol{y_e} & \boldsymbol{y_{e'}} & \boldsymbol{y_\mu} \end{bmatrix}^\top \xleftarrow{\$} D_{R^{(2\ell+2)\kappa}, \boldsymbol{0}, \sigma_{\mathrm{E}}}$

4 : $\boldsymbol{Y} \leftarrow \begin{bmatrix} \boldsymbol{A}^\top \otimes (p\boldsymbol{I}_\kappa) & p\boldsymbol{I}_{\ell\cdot\kappa} & \boldsymbol{0}^{(\ell\cdot\kappa)\times\kappa} & \boldsymbol{0}^{(\ell\cdot\kappa)\times\kappa} \\ \boldsymbol{t}^\top \otimes (p\boldsymbol{I}_\kappa) & \boldsymbol{0}^{\kappa\times(\ell\cdot\kappa)} & p\boldsymbol{I}_\kappa & \boldsymbol{I}_\kappa \\ \boldsymbol{0}^{\lambda\times(\ell\cdot\kappa)} & \boldsymbol{0}^{\lambda\times(\ell\cdot\kappa)} & \boldsymbol{0}^{\lambda\times\kappa} & \boldsymbol{T} \end{bmatrix} \begin{bmatrix} \boldsymbol{y_r} & \boldsymbol{y_e} & \boldsymbol{y_{e'}} & \boldsymbol{y_\mu} \end{bmatrix}^\top \begin{matrix} \bmod q \\ \bmod q \\ \bmod p \end{matrix}$

5 : $c \leftarrow H\left(\begin{bmatrix} \boldsymbol{A}^\top \otimes (p\boldsymbol{I}_\kappa) & p\boldsymbol{I}_{\ell\cdot\kappa} & \boldsymbol{0}^{(\ell\cdot\kappa)\times\kappa} & \boldsymbol{0}^{(\ell\cdot\kappa)\times\kappa} \\ \boldsymbol{t}^\top \otimes (p\boldsymbol{I}_\kappa) & \boldsymbol{0}^{\kappa\times(\ell\cdot\kappa)} & p\boldsymbol{I}_\kappa & \boldsymbol{I}_\kappa \\ \boldsymbol{0}^{\lambda\times(\ell\cdot\kappa)} & \boldsymbol{0}^{\lambda\times(\ell\cdot\kappa)} & \boldsymbol{0}^{\lambda\times\kappa} & \boldsymbol{T} \end{bmatrix}, \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \\ \boldsymbol{u} \end{bmatrix}, \boldsymbol{Y} \right)$

6 : $\boldsymbol{s} \leftarrow \begin{bmatrix} \boldsymbol{r} & \boldsymbol{e} & \boldsymbol{e'} & \boldsymbol{\mu} \end{bmatrix}^\top c$

7 : $\boldsymbol{z} \leftarrow \boldsymbol{s} + \boldsymbol{y}$

8 : With probability $1 - \min\left(1, \dfrac{D_{R^{(2\ell+2)\kappa}, \boldsymbol{0}, \sigma_{\mathrm{E}}}(\boldsymbol{z})}{3 \cdot D_{R^{(2\ell+2)\kappa}, \boldsymbol{s}, \sigma_{\mathrm{E}}}(\boldsymbol{z})}\right)$ goto 3

9 : **if** $\|\boldsymbol{z}\|_\infty \geq 6\sigma_{\mathrm{E}}$ goto 3, **else return** $\boldsymbol{e} = (\boldsymbol{v}, \boldsymbol{w}, c, \boldsymbol{z})$

Fig. 3: The verifiable encryption algorithm `Enc`.

Input: Secret key $\mathtt{sk} = (\boldsymbol{s}_1)$, pair $x = (\boldsymbol{T}, \boldsymbol{u})$, ciphertext $t = (\boldsymbol{v}, \boldsymbol{w}, c, \boldsymbol{z}), C = \max_{c \in \mathcal{C}} ||c||_\infty$

1 : **if** $\mathtt{Ver}(t, x, \mathtt{pk}) = 1$ **then**

2 : **while**

3 : $c' \xleftarrow{\$} \mathcal{C}$

4 : $\overline{c} \leftarrow c - c'$

5 : $\overline{\boldsymbol{m}}[i] \leftarrow (\boldsymbol{w} - \langle \boldsymbol{s}_1, \boldsymbol{v}_i \rangle)\overline{c} \bmod q$ for all $i \in [\kappa]$

6 : **if** $||\overline{\boldsymbol{m}}||_\infty \leq q/2C$ and $||\overline{\boldsymbol{m}} \bmod p||_\infty < 12\sigma_E$ **then**

7 : **return** $(\overline{\boldsymbol{m}} \bmod p, \overline{c})$

Fig. 4: Algorithm `Dec` for decryption of a ciphertext.

Input: ciphertext $t = (\boldsymbol{v}, \boldsymbol{w}, c, \boldsymbol{z}) \in R_q^{\ell\cdot\kappa} \times R_q^\kappa \times R_q \times R^{(2\ell+2)\kappa}$, language element $x = (\boldsymbol{T}, \boldsymbol{u})$, public key $\mathtt{pk} = (\boldsymbol{A}, \boldsymbol{t}, p, q)$

1 : **if** $\|\boldsymbol{z}\|_\infty > 6 \cdot \sigma_{\mathrm{E}}$ **then return** 0

2 : $\boldsymbol{Z} \leftarrow \begin{bmatrix} \boldsymbol{A}^\top \otimes (p\boldsymbol{I}_\kappa) & p\boldsymbol{I}_{\ell\cdot\kappa} & \boldsymbol{0}^{(\ell\cdot\kappa)\times\kappa} & \boldsymbol{0}^{(\ell\cdot\kappa)\times\kappa} \\ \boldsymbol{t}^\top \otimes (p\boldsymbol{I}_\kappa) & \boldsymbol{0}^{\kappa\times(\ell\cdot\kappa)} & p\boldsymbol{I}_\kappa & \boldsymbol{I}_\kappa \\ \boldsymbol{0}^{\lambda\times(\ell\cdot\kappa)} & \boldsymbol{0}^{\lambda\times(\ell\cdot\kappa)} & \boldsymbol{0}^{\lambda\times\kappa} & \boldsymbol{T} \end{bmatrix} \boldsymbol{z} - c \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \\ \boldsymbol{u} \end{bmatrix} \begin{matrix} \bmod q \\ \bmod q \\ \bmod p \end{matrix}$

3 : **if** $c \neq \mathrm{H}\left(\begin{bmatrix} \boldsymbol{A}^\top \otimes (p\boldsymbol{I}_\kappa) & p\boldsymbol{I}_{\ell\cdot\kappa} & \boldsymbol{0}^{(\ell\cdot\kappa)\times\kappa} & \boldsymbol{0}^{(\ell\cdot\kappa)\times\kappa} \\ \boldsymbol{t}^\top \otimes (p\boldsymbol{I}_\kappa) & \boldsymbol{0}^{\kappa\times(\ell\cdot\kappa)} & p\boldsymbol{I}_\kappa & \boldsymbol{I}_\kappa \\ \boldsymbol{0}^{\lambda\times(\ell\cdot\kappa)} & \boldsymbol{0}^{\lambda\times(\ell\cdot\kappa)} & \boldsymbol{0}^{\lambda\times\kappa} & \boldsymbol{T} \end{bmatrix}, \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \\ \boldsymbol{u} \end{bmatrix}, \boldsymbol{Z} \right)$ **then return** 0

4 : **return** 1

Fig. 5: Algorithm `Ver` for verification of a ciphertext.

The value $\boldsymbol{c}_1$ serves to bind the committer to a single choice of $\boldsymbol{r}_m$, while c_2 hides the actual message using the unique $\boldsymbol{r}_m$. Fixing $\boldsymbol{r}_m$ fixes m uniquely, and m can indeed be recovered using $\boldsymbol{r}_m$ only. The idea is to use the verifiable encryption scheme to encrypt the opening $\boldsymbol{r}_m$, and prove that the voter knows a witness for the relation $\boldsymbol{c}_1 = \boldsymbol{B_1 r}_m \bmod p$ where $\boldsymbol{r}_m$ is bounded. Any such randomness could then be used to uniquely open the commitment.

To encrypt the opening $\boldsymbol{r}_m$ verifiably, Step 4 in Figure 3 is now the system

$$\begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \\ \boldsymbol{c}_1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}^\top \otimes (p\boldsymbol{I}_\kappa) & p\boldsymbol{I}_{\ell\cdot\kappa} & \boldsymbol{0}^{(\ell\cdot\kappa)\times\kappa} & \boldsymbol{0}^{(\ell\cdot\kappa)\times\kappa} \\ \boldsymbol{t}^\top \otimes (p\boldsymbol{I}_\kappa) & \boldsymbol{0}^{\kappa\times(\ell\cdot\kappa)} & p\boldsymbol{I}_\kappa & \boldsymbol{I}_\kappa \\ \boldsymbol{0}^{\lambda\times(\ell\cdot\kappa)} & \boldsymbol{0}^{\lambda\times(\ell\cdot\kappa)} & \boldsymbol{0}^{\lambda\times\kappa} & \boldsymbol{B}_1 \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{r} \\ \boldsymbol{e} \\ \boldsymbol{e}' \\ \boldsymbol{r}_m \end{bmatrix}.$$

Note that the encryption of the opening $\boldsymbol{r}_m$ now contains two parts: $\boldsymbol{v}$ and $\boldsymbol{w}$ correspond to the ciphertext of the encryption while $\boldsymbol{c}_1$ corresponds to the verification of the opening of the commitment.

5.2 Return Codes

In the case of a malicious computer, we need to make sure that the voter can detect if the encrypted vote being sent to the ballot box is not an encryption of the correct ballot. We achieve this by giving the voter a pre-computed table of return codes which he can use for verification. The return codes are generated per voter, using a voter-unique blinding-key and a system-wide PRF-key.

A commitment to the blinding key is made public. The computer gets the blinding-key and must create a pre-code by blinding the ballot with the blinding-key. The computer also generates commitments to the ballot and the pre-code, along with a proof that the pre-code has been generated correctly. Anyone with an opening of the pre-code commitment and the PRF-key can now generate the correct return code without learning anything about the ballot.

Defining the Return Code. Assume that the voters have ω different options in the election. Let $\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_\omega \in R_p$ be ballots, let $a_j \in R_p$ be a blinding-key for a voter V_j and let $\texttt{PRF}_k : \{0,1\}^* \times R_p \to \{0,1\}^n$ be a pseudo-random function, instantiated with a PRF-key k, from pairs of binary strings and elements from R_p to the set of binary strings of length n. The *pre-code* $\hat{r}_{ij}$ corresponding to the ballot $\hat{v}_i$ is $\hat{r}_{ij} = a_j + \hat{v}_i \mod p$. The *return code* r_{ij} corresponding to the ballot $\hat{v}_i$ is $r_{ij} = \texttt{PRF}_k(V_j, \hat{r}_{ij})$.

Let $\boldsymbol{c}_{a_j}$, $\boldsymbol{c}_j$ and $\boldsymbol{c}_{\hat{r}_j}$ be commitments to the blinding key a_j, the ballot $v_j \in \{\hat{v}_1, \ldots, \hat{v}_\omega\}$ and the pre-code $\hat{r}_j = a_j + v_j \mod p$. It is now clear that we can prove that a given $\hat{r}_j$ value has been correctly computed by giving a proof of linearity that $a_j + v_j = \hat{r}_j$. This can be done either by adding the commitments $\boldsymbol{c}_{a_j}$ and $\boldsymbol{c}_j$ together directly to get a commitment $\boldsymbol{c}_{a_j+v_j}$ with larger randomness (if the choice of parameters allows for the sum of the randomness to be a valid opening of the commitment) and then prove the equality of the

committed messages, or to extend the proof of linearity to handle three terms. Our return code construction is now straight-forward:

A commitment $\boldsymbol{c}_{a_j}$ to voter V_j's voter-unique blinding key a_j is public. The voter V_j's *computer* will get the voter-unique blinding-key a_j together with the randomness used to create $\boldsymbol{c}_{a_j}$. It has already created a commitment $\boldsymbol{c}_j$ to the ballot v_j. It will compute the pre-code $\hat{r}_j = a_j + v_j$, a commitment $\boldsymbol{c}_{\hat{r}_j}$ to $\hat{r}_j$ and a proof $\Pi_{\hat{r}_j}$ of knowledge of the opening of that sum. Finally, it will verifiably encrypt as $\boldsymbol{e}_{\hat{r}_j}$ the opening of $\boldsymbol{c}_{\hat{r}_j}$ with the return code generator's public key.

The *return code generator* receives V_j, $\boldsymbol{c}_{\hat{r}_j}$, $\boldsymbol{c}_j$, $\boldsymbol{e}_{\hat{r}_j}$ and $\Pi_{\hat{r}_j}$. It verifies the proof and the encryption, and then decrypts the ciphertext to get $\hat{r}_j$. It computes the return code as $r_j = \mathtt{PRF}_k(V_j, \hat{r}_j)$.

Note that if a voter re-votes (such as when exposed to coercion), the return code generator would be able to learn something about the ballots involved. The return code mechanism can be extended for re-voting using higher degree polynomials to hide the vote.

5.3 The Voting Scheme

We get our e-voting protocol by combining the shuffle protocol with the verifiable encryption scheme and the return code construction. A complete description of this protocol can be found in Figure 6 and in Appendix B. All communication happens over secure channels. We discuss the *privacy*, *integrity* and *coercion resistance* of the voting scheme in detail in Appendix B.

Registration phase. The only thing that happens in this phase is key generation. Every player generates their own key material and publishes the public keys and any other commitments.

The voter's computer, the return code generator and a *trusted printer* then use a multi-party computation protocol[11] to compute the ballot-return code pairs for the voter, such that only the trusted printer learns the pairs. The trusted printer then sends these pairs to the voter through a secure channel. We emphasize that for many voters, the registration phase likely requires significant computational resources for the return code generator and the trusted printer. In practice, the voter's computer will usually play a minor role in this key generation.

Casting a ballot. The voter begins the ballot casting by giving the ballot v_j to the voter's computer.

The voter's computer has the per-voter secret key material, and gets the ballot v_j to be cast from the voter. It computes the pre-code $\hat{r}$ and generates commitments $\boldsymbol{c}_j$ and $\boldsymbol{c}_{\hat{r}}$ to the ballot and the pre-code, respectively. It creates

[11] Since this happens before the election, speed is no longer essential. Even so, for the computations involved here, ordinary MPC is sufficiently practical. In a practical deployment, the voter's computer is unlikely to be part of this computation. It would instead be delegated to a set of trusted key generation players.

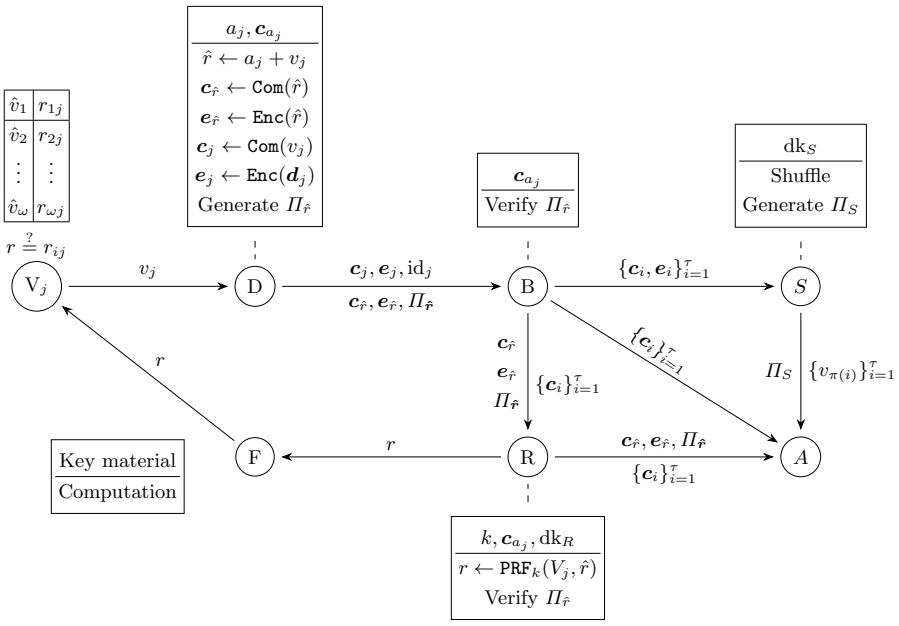

Fig. 6: Complete voting protocol. A voter V_j gives a vote v_j to their computer D. The value $\boldsymbol{d}_j$ is the opening of the commitment $\boldsymbol{c}_j$. The public keys for commitments and encryption are assumed known to all parties. Signatures are omitted: D signs the vote to be verified by the ballot box B and the return code server R, while R signs the incoming votes and sends the signature in return, via B, to D to confirm that the vote is received. Both B and R sends the commitments of the votes to authorities A to verify consistent views. After all votes are cast, B forwards them to the shuffle server S, stripping away the voters id's and signatures.

a proof $\Pi_{\hat{r}}$ that the pre-code has been correctly computed (with respect to the commitments). It creates a verifiable encryption $\boldsymbol{e}_j$ of an opening of $\boldsymbol{c}_j$ to v_j under the shuffle server's public key, and a verifiable encryption $\boldsymbol{e}_{\hat{r}}$ of an opening of $\boldsymbol{c}_{\hat{r}}$ under the return code generator's public key. It then signs all of these values, together with its identity and every public key and commitment used to create the proofs. We note that the voter's identity and relevant keys and commitments are included in the proofs used. (This is not an artifact of making the security proof work, but it prevents real-world attacks.)

The computer sends the commitments, encryptions, proofs and signature to the ballot box. The ballot box verifies the signature and the proofs. Then it sends everything to the return code generator.

The return code generator verifies the signature and the proofs. Then it decrypts the opening of $\boldsymbol{e}_{\hat{r}}$ and computes the return code from $\hat{r}$. It hashes everything and creates a signature on the hash. It sends the return code to the voter's phone and the signature to the ballot box. The ballot box verifies the

return code generator's signature on the hash. Then it sends the return code generator's signature to the voter's computer.

The voter's computer verifies the return code generator's signature and then shows the hash and the signature to the voter as the transcript. The voter checks that the computer accepts the ballot as cast. When the phone displays the return code r, the voter accepts the ballot as cast if (v_j, r) is in the return code table.

Tallying. When the tally phase begins, the ballot box sends everything from every successful ballot casting to the auditors. It extracts the commitments to the ballots and the encrypted openings (without any proofs), organizes them into a sorted list of commitment-ciphertext pairs and sends the sorted list to the shuffle server. The return code generator sends everything from every successful ballot casting to the auditors. The shuffle server receives a sorted list of commitment-ciphertext pairs from the ballot box. It hashes the list and sends a hash to the auditors. The shuffle server waits for the auditors to accept provisionally.

An auditor receives data from the ballot box and the return code generator, and a hash from the shuffle server. If the data from the ballot box and the return code generator agree, the auditor extracts the sorted commitment-ciphertext pairs from the data, hashes it and compares the result with the hash from the shuffle server. If it matches, the auditor provisionally accepts.

When the shuffle server receives the provisional accept, it decrypts the commitment openings and verifies that the openings are valid. For any invalid opening, it sends the commitment-ciphertext pair and the opening to the auditors. It then sorts the ballots and creates a shuffle proof for the ballots and the commitments. It then counts the ballots to get the election result and sends the ballots, the shuffle proof and the result to the auditors.

An auditor receives the ballots, the shuffle proof and the result from the shuffle server. It verifies the proof and that the election result is correct. It extracts the hashes (but not the signatures) signed by the return code generator from the ballot box data and creates a sorted list of hashes. It signs the hash list and the result and send both signature and hash list to the shuffle server. Once the shuffle server has received signatures and hash lists from every auditor, it verifies that the hash lists are identical and that the signatures verify. It then outputs the result, the hash list and the auditors' signatures as the transcript.

Verification. The voter has the transcript output by the voter's computer and the transcript output by the shuffle server. It first verifies that the hash from the computer's transcript is present in the shuffle server's hash list. Then it verifies all the signatures. If everything checks out, the voter accepts.

6 Performance

As outlined in our construction, we are nesting the commitment scheme of [BDL+18] into the encryption scheme of [LN17]. To determine secure while not enormously big parameters for our scheme, we need to first make sure that

Parameter	Explanation	Constraints
N, δ	Degree of polynomial $X^N + 1$ in R	$N \geq \delta \geq 1$, where N, δ powers of two
p	Modulus for commitments	Prime $p = 2\delta + 1 \mod 4\delta$
β_∞	∞-norm bound of certain elements	Choose β_∞ such that $\beta_\infty < p^{1/\delta}/\sqrt{\delta}$
σ_{C}	Standard deviation of discrete Gaussians	Chosen to be $\sigma_{\mathrm{C}} = 22 \cdot \nu \cdot \beta_\infty \cdot \sqrt{kN}$
k	Width (over R_p) of commitment matrix	
n	Height (over R_p) of commitment matrix	
ν	Maximum l_1-norm of elements in $\mathcal{C}$	
$\mathcal{C}$	Challenge space	$\mathcal{C} = \{c \in R_p \mid \|c\|_\infty = 1, \|c\|_1 = \nu\}$
$\bar{\mathcal{C}}$	The set of differences $\mathcal{C} - \mathcal{C}$ excluding 0	$\bar{\mathcal{C}} = \{c - c' \mid c \neq c' \in \mathcal{C}\}$
D_{β_∞}	Set of elements of ∞-norm at most β_∞	$D_{\beta_\infty} = \{x \in R_p \mid \|x\|_\infty \leq \beta_\infty\}$
σ_{E}	Standard deviation of discrete Gaussians	Chosen to be $\sigma_{\mathrm{E}} = 11 \cdot \nu \cdot \sqrt{\kappa N(3 + \beta_\infty)}$
κ	Dimension of message space in encryption	Equal to the length of randomness k
ℓ	Dimension the encryption matrix	Equal to the size of the commitments $k - n$
λ	Dimension of public $\boldsymbol{u}$ in $\boldsymbol{T\mu} = \boldsymbol{u}$	Equal to the height $n + 1$ of the commitment matrix
q	Modulus for encryption	Must choose prime q such that $q > 24\sigma_{\mathrm{E}}^2$ and $q > 2p(2\ell \cdot N^2 \cdot {\beta_\infty}^2 + N + 1)$ and $q = 2\delta + 1 \mod 4\delta$
τ	Total number of votes	For soundness we need $(\tau^\delta + 1)/\lvert R_p \rvert < 2^{-128}$

Table 1: Parameters for the commitment and verifiable encryption schemes.

we have sufficiently large parameters to ensure both binding and hiding of the commitments for which we will use the "optimal" parameter set of [BDL+18] (but with twice the standard deviation to keep the probability of abort in the rejection sampling down to 3 trials for the proofs of linearity) which is both computationally binding and hiding (see Table 2). The LWE-estimator [APS15] estimates at least 100 bits of security with these parameters. We then instantiate the verifiable encryption scheme with compatible parameters, which is possible due to our generalization of [LN17]. The verifiable encryption scheme will then yield decryptions with an ∞-norm that is way below the bound for which the commitment scheme is binding, so any valid decryption which differs from the original vote would break the binding of the commitment scheme. In general, the instantiation of the encryption scheme offers much higher security than the commitment scheme, but the choice of parameters are restricted by the constraints from combining it with the commitments.

6.1 Size

Size of the Votes. Note that each ciphertext $\boldsymbol{e}$ includes both the encrypted opening $(\boldsymbol{v}, \boldsymbol{w})$ and the proof of valid opening $(c, \boldsymbol{z})$. Using a lattice based signature scheme like Falcon-768 [PFH+17], we have signatures of size ≈ 1 KB. The voter verifiability protocol requires a commitment, an encryption + proof, and a proof of linearity. It follows that a vote $(\boldsymbol{c}_j, \boldsymbol{e}_j, \boldsymbol{c}_{\hat{r}}, \boldsymbol{e}_{\hat{r}}, \Pi_{\hat{r}})$ is of total size ≈ 240 KB, which means that, for τ voters, the ballot box $\mathcal{B}$ receives 240τ KB of data.

Size of the Shuffle Proof. Our shuffle protocol is a $4+3\tau$-move protocol with elements from R_p. Each element in R_p has at most N coefficients of size at most

Parameter	Commitment (I)	Encryption (III)
N	1024	1024
p	$\approx 2^{32}$	$\approx 2^{32}$
q	-	$\approx 2^{56}$
β_∞	1	1
σ	$\sigma_{\mathrm{C}} \approx 54000$	$\sigma_{\mathrm{E}} \approx 54000$
ν	36	36
δ	2	2
k	3	-
n	1	-
ℓ	-	2
κ	-	3
λ	-	2
Proof	9.4 KB	42.4 KB
Primitive	8.2 KB	64.5 KB

Table 2: Parameters for the commitments by Baum et al. [BDL+18] and verifiable encryption scheme by Lyubashevsky and Neven [LN17].

p, and hence, each R_p-element has size at most $N \log p$ bits. For every R_p-vector that follows a Gaussian distribution with standard deviation σ we assume that we can represent the element using $N \cdot \log(6\sigma)$ bits. Every element from $\mathcal{C}$ will be assumed to be representable using at most $2N$ bits.

We analyze how much data we have to include in each step of the shuffling protocol in Figure 2. Using the Fiat-Shamir transform [FS87], we can ignore the challenge-messages from the verifier. The prover ends up sending 1 commitment, 1 ring-element and 1 proof of linearity per vote. Using the parameters from Table 2, we get that the shuffle proof is of total size $\approx 22\tau$ KB.

6.2 Timings

We collected performance figures from our prototype implementation written in C to estimate the runtime of our scheme. Estimates are based on Table 2 and the implementation was benchmarked on an Intel Skylake Core i7-6700K CPU running at 4GHz without TurboBoost using `clang` 12.0 and FLINT 2.7.1 [HJP13]. Timings are available in Table 3, and the source code can be found on GitHub [12].

Elementary Operations. Multiplication in R_p and R_q is usually implemented when $p \equiv q \equiv 1 \bmod (2N)$ and $X^N + 1$ splits in N linear factors, for which the Number-Theoretic Transform is available. Unfortunately, Lemma 1 restricts

[12] https://github.com/dfaranha/lattice-voting-ctrsa21

Our Scheme:	Commit	Open	Encrypt	Verify	Decrypt	Shuffle
Time	1.1 ms	1.2 ms	208 ms	39 ms	6 ms	27τ ms

Table 3: Timings for 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.

parameters and we instead adopt $p \equiv q \equiv 5 \bmod 8$ [LN17]. In this case, $X^N + 1$ splits in two $N/2$-degree irreducible polynomials $(X^{N/2} \pm r)$ for r a modular square root of -1. This gives an efficient representation for $a = a_1 X^{N/2} + a_0$ using the Chinese Remainder Theorem: $CRT(a) = (a \pmod{X^{N/2} - r}, a \pmod{X^{N/2} + r})$. Even though the conversions are efficient due to the choice of polynomials, we sample ring elements directly in this representation whenever possible. As in [LS18], we implement the base case for degree $N/2$ using FLINT for polynomial arithmetic [HJP13]. We use SHA256 for hashing to generate challenges.

Commitment. A commitment is generated by multiplying the matrix $\boldsymbol{B}$ by a vector $\boldsymbol{r}_m$ over R_p and finally adding the message m to the second component in the *CRT* domain. Computing and opening a commitment takes 0.9 ms and 1.2 ms, respectively, and sampling randomness $\boldsymbol{r}_m$ takes only 0.2 ms.

Verifiable Encryption. Verifiable encryption needs to sample vectors according to a discrete Gaussian distribution. For an R_q element with standard deviation $\sigma_{\mathrm{E}} \approx 2^{15.7}$ (for the encryption scheme), the implementation from COSAC [ZSS20] made available for $\sigma = 2^{17}$ samples 1024 integers in 0.12 ms using very small precomputation tables. Each encryption iteration takes 69 ms and, because we expect to need 3 attempts to generate one valid encryption (line 8 in Figure 3), the total time of encryption is around 208 ms. For verification, 39 ms are necessary to execute a test; and 6 ms are required for the actual decryption.

Shuffle Proof. The shuffle proof operates over R_p and is thus more efficient. Sampling uses the same approach as above for σ_{C} from the commitment scheme. Benchmarking includes all samplings required inside the protocol, the commitment, the proof of linearity and, because we expect to need 3 attempts to generate each of the proofs of linearity to the cost of 7.5 ms, amounts to 27τ ms for the entire proof, omitting the communication cost.

6.3 Comparison

We briefly compare our scheme with the scheme by del Pino et al. [dLNS17] from CCS 2017 in Table 4. We note that the scheme in [dLNS17] requires at least $\xi \geq 2$ authorities to ensure ballot privacy, where at least one authority must be honest. The authorities run the proof protocol in parallel, and the time they need to process each vote is ≈ 5 times slower per vote than in our scheme. We only need one party to compute the shuffle proof, where we first decrypt all votes and then shuffle. Our proof size is at least 14 KB smaller per vote when

$\xi = 2$, that is, a saving of 40 %, and otherwise much smaller in comparison for $\xi \geq 3$. We further note that both implementations partially rely on FLINT for polynomial arithmetic and were benchmarked on Intel Skylake processors. A significant speedup persists after correcting for clock frequency differences.

Comparison	Vote Size	Voter Time	Proof Size	Prover Time
Our Scheme:	120 KB	209 ms	22τ KB	33τ ms
CCS 2017 [dLNS17]:	20ξ KB	9 ms	$18\xi\tau$ KB	150τ ms

Table 4: Comparing our scheme with the yes/no voting scheme in [dLNS17]

For a fair comparison, we only included the size and timings of the commitment of the vote and the encrypted openings from our scheme. In practice, the size and timings of the voter will be twice of what it is in the table, because of the return code mechanism, which is not a part of [dLNS17]. This has no impact on the decryption and shuffle done by the prover. The work done by the voter is still practical. For [dLNS17] to be used in a real world election, they would need to include an additional mechanism for providing voter verifiability, like the one we have constructed.

Finally, we note that [dLNS17] can be extended from yes/no voting to votes consisting of strings of bits. However, the size and timings of such an extension will be linear in the length of the bit-strings, and our scheme would do even better in comparison, as we can handle votes encoded as arbitrary ring-elements.

Thanks

We thank Andreas Hülsing and the anonymous reviewers for helpful comments.

References

Adi08. Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *USENIX Security 2008*, 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, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, 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, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, 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.

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*, volume 11035 of *LNCS*, pages 368–385. Springer, Heidelberg, September 2018.

BG14. Shi Bai and Steven D. Galbraith. An improved compression technique for signatures based on learning with errors. In Josh Benaloh, editor, *CT-RSA 2014*, volume 8366 of *LNCS*, pages 28–47. Springer, Heidelberg, February 2014.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, 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, Part II*, volume 12309 of *LNCS*, pages 336–356. Springer, Heidelberg, September 2020.

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 *LNCS*, pages 330–346. Springer, Heidelberg, February 2019.

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*, pages 1565–1581. ACM Press, October / November 2017.

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

Gjø11. Kristian Gjøsteen. The norwegian internet voting protocol. In *E-Voting and Identity - Third International Conference, VoteID 2011*, pages 1–18, 2011.

GL16. Kristian Gjøsteen and Anders Smedstuen Lund. An experiment on the security of the Norwegian electronic voting protocol. *Annals of Telecommunications*, pages 1–9, 2016.

GLP12. Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 530–547. Springer, Heidelberg, September 2012.

GS17. Kristian Gjøsteen and Martin Strand. A roadmap to fully homomorphic elections: Stronger security, better verifiability. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y. A. Ryan, Vanessa Teague,

Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson, editors, *FC 2017 Workshops*, volume 10323 of *LNCS*, pages 404–418. Springer, Heidelberg, April 2017.

HJP13. W. Hart, F. Johansson, and S. Pancratz. FLINT: Fast Library for Number Theory, 2013. Version 2.4.0, http://flintlib.org.

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

LN17. Vadim Lyubashevsky and Gregory Neven. One-shot verifiable encryption from lattices. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 293–323. Springer, Heidelberg, April / May 2017.

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

LPT19. Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. Trapdoor commitments in the SwissPost e-voting shuffle proof, 2019.

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

LS18. Vadim Lyubashevsky and Gregor Seiler. Short, invertible elements in partially splitting cyclotomic rings and applications to lattice-based zero-knowledge proofs. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 204–224. Springer, Heidelberg, April / May 2018.

Lyu09. Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 598–616. Springer, Heidelberg, December 2009.

Lyu12. Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 738–755. Springer, Heidelberg, April 2012.

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*, pages 116–125. ACM Press, November 2001.

PFH+17. Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions.

Scy. Scytl. Scytl sVote, complete verifiability security proof report - software version 2.1 - document 1.0. https://www.post.ch/-/media/post/evoting/dokumente/complete-verifiability-security-proof-report.pdf.

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 *LNCS*, 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 Security Proof of the Shuffle Protocol

We now show how to prove correctness and the zero-knowledge property of the protocol Π_{Shuffle} as described in Figure 2.

Theorem 2. *The shuffle protocol Π_{Shuffle} as described in Figure 2 is complete for the relation $\mathcal{R}_{\text{Shuffle}}$ if $\tau \ll |R_p|$ and Π_{Lin} is complete.*

Proof. Due to the size of R_p there must always exist such a value ρ. Furthermore, the instances of Π_{Lin} always succeed due to the completeness of the respective proof if the s_i indeed fulfill the required linear relations. As Lemma 4 shows that the respective values for s_i always exist, the claim follows. □

We prove Special Honest Verifier Zero-Knowledge (HVZK) by a series of games, as shown in Figure 7. In the first game, we swap the HVZK protocol Π_{Lin} from Figure 1 with its simulation. Then we change the way we calculate the initial commitments $[\![D_i]\!]$ by instead sampling s_i uniformly random and computing what D_i should be. In this step we make no use of the values θ_i. Next, we let the $[\![D_i]\!]$'s be commitments $[\![0]\!]$, and sample random s_i's. Thus, we do not use any permutation in the last game; the Simulator is indistinguishable from the Real game. We proceed with the proofs that an adversary cannot distinguish between any of these games.

Real		Game 1		Game 2		Simulator	
1 :	$\theta_i \xleftarrow{\$} R_p$	1 :	$\theta_i \xleftarrow{\$} R_p$	1 :	$\boxed{s_i \xleftarrow{\$} R_p}$	1 :	$\boxed{[\![D_i]\!] \leftarrow [\![0]\!]}$
2 :	$D_i \leftarrow (1)$	2 :	$D_i \leftarrow (1)$	2 :	$\theta_i \xleftarrow{\$} R_p$	2 :	$s_i \xleftarrow{\$} R_p$
3 :	Send $[\![D_i]\!]$	3 :	Send $[\![D_i]\!]$	3 :	$\boxed{D_i \leftarrow (8)}$	3 :	Send $[\![D_i]\!]$
4 :	$s_i \leftarrow (2)$	4 :	$s_i \leftarrow (2)$	4 :	Send $[\![D_i]\!]$	4 :	Send s_i
5 :	Send s_i	5 :	Send s_i	5 :	Send s_i	5 :	Sim
6 :	SHVZK	6 :	$\boxed{\textbf{Sim}}$	6 :	Sim		

Fig. 7: Honest Verifier Zero-Knowledge games. We denote by $x \leftarrow (i)$ that x was computed according to equation (i). The steps that changes from game to game are indicated with boxes.

Lemma 6. *If there exists an adversary that can distinguish between* Real *and* Game 1 *in Figure 7, then there exists an adversary that can break the HVZK property of Π_{Lin}.*

Proof. This follows because the only difference between Real and Game 1 is that we simulate the last step, which is exactly the zero-knowledge proof. □

Lemma 7. *The distribution of the transcripts* $(\{[\![D_i]\!]\}, \beta, \{s_i\})$ *produced by* Game 1 *and* Game 2 *are perfectly indistinguishable.*

Proof. We give the argument for one of the equations. The argument is identical for the remaining equations. In Game 1 we sample $\theta_1 \xleftarrow{\$} R_p$, compute

$$[\![D_1]\!] = [\![\theta_1 \hat{M}_1]\!],$$

and then compute s_1 according to (3). Since θ_1 is uniform, this is a commitment to a uniformly random value. Also, we know that s_1 is uniformly distributed.

In Game 2 we sample a uniformly random $s_1 \xleftarrow{\$} R_p$ and then compute

$$[\![D_1]\!] = [\![\beta \hat{M}_1 + s_1 M_1]\!]. \tag{8}$$

Now, s_1 is uniformly random, so $[\![D_1]\!]$ is a commitment to a uniformly random value. Hence, it follows that the transcripts are perfectly indistinguishable. □

Lemma 8. *If there exists an adversary* $\mathcal{A}$ *that can distinguish between the transcripts of* Game 2 *and* Simulator*, then there exists an adversary* $\mathcal{A}'$ *who breaks the hiding property of the commitment scheme.*

Proof. The proof is shown in Figure 8.

We use a standard distinguishing game. A commitment oracle will then commit to random values $m \in R_p$ or to 0. The oracle will send the challenges $\{\boldsymbol{c}_{b,i}\}$ to $\mathcal{A}'$ where b is a bit indicating commitments to 0 or m if b is 0 or 1, respectively. Then $\mathcal{A}'$ will pick random values s_i and send the transcript $(\{\boldsymbol{c}_{b,i}\}, \beta, \{s_i\})$ to the distinguishing adversary $\mathcal{A}$. We then use the distinguishing power of $\mathcal{A}$ to decide on a bit b' and send this back to the commitment challenger.

We now show that the $\boldsymbol{c}_{b,i}$'s are distributed as $\mathcal{A}$ expected. When querying $\mathcal{A}$ on the transcripts, we use the challenges $\boldsymbol{c}_{b,i} \leftarrow \mathcal{C}(m_i, r_i)$ for a uniformly random m_i, but $\mathcal{A}$ expects the commitments $[\![D_i]\!]$ from (8). For the first equation of (8), we note that the committed message is $\beta \hat{M}_1 + s_1 M_1$. Since each s_i is a uniform sample from R_p, this expression is a uniformly random sample from R_p. Hence, the distributions of $\boldsymbol{c}_{1,i}$ and $[\![D_i]\!]$ (from (8)) are identical. □

This means that we can simulate a transcript of the real protocol without knowing any of the private information known to $\mathcal{P}$. We summarize this result in a theorem.

Theorem 3. *Assuming that* Π_{Lin} *is an HVZK proof and that the used commitment scheme is hiding, then the transcripts of* Real *and* Simulator *in Figure 7 are indistinguishable.*

Proof. This is true by combining Lemma 6, 7 and 8. □

Adversary Breaking the Hiding Property

Commitment Oracle

for $i = 1, \dots \tau$ **do**

$m \xleftarrow{\$} R_p$

$\boldsymbol{r} \xleftarrow{\$} D^k_{\beta_\infty}$

$\boldsymbol{c}_{0,i} \leftarrow \mathcal{C}(0, \boldsymbol{r})$

$\boldsymbol{c}_{1,i} \leftarrow \mathcal{C}(m, \boldsymbol{r})$

$b \leftarrow \{0, 1\}$

$\xrightarrow{\{\boldsymbol{c}_{b,i}\}_{i=1}^{\tau}}$

Using Adversary

$\mathcal{A}'$		$\mathcal{A}$
$s_i \xleftarrow{\$} R_p$		
$\beta \xleftarrow{\$} R_p$		
	$\xrightarrow{\{\boldsymbol{c}_{b,i}\}, \beta, \{s_i\}}$	
		$b' \leftarrow \mathcal{A}(\{\boldsymbol{c}_{b,i}\}, \beta, \{s_i\})$
	$\xleftarrow{b'}$	

$\xleftarrow{b'}$

if $b' = b$ **then**

Return 1

Return 0

Fig. 8: Using an adversary $\mathcal{A}$ who can distinguish between Game 2 and Simulator in Figure 7, we can construct an adversary $\mathcal{A}'$ who can break the hiding property of the commitment scheme.

B Security of the Voting Protocol

Our return code mechanism does not fit the standard voting scheme syntax security definitions such as [BCG+15]. The usual solution is to define *ad hoc* security definitions (see e.g. [Scy] and [Gjø11]). Note that protocols with this [Gjø11] and similar architectures have been deployed in large-scale national elections.

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*, which is in some sense similar to a public key cryptosystem with some very specific functionality.

We begin by describing a *verifiable* cryptographic voting scheme *with return codes*, explain how our voting system can be described as a simple protocol on top of such a cryptographic voting scheme. We define security notions for this cryptosystem, sketch the security proof, and then informally discuss the voting protocol and its security properties in terms of the cryptosystem's security.

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 four tasks election setup, casting ballots, counting cast ballots and verifying the count. To support return codes, we need algorithms for two more tasks: voter registration and pre-code computation.

The *setup* algorithm $\texttt{Setup}$ outputs a *public key* $\texttt{pk}$, a *decryption key* $\texttt{dk}$ and a *code key* $\texttt{ck}$.

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

The *cast* algorithm $\texttt{Cast}$ takes a public key $\texttt{pk}$, a voter casting key $\texttt{vck}$ and a *ballot* v, and outputs an *encrypted ballot* ev and a *ballot proof* Π^v.

The *code* algorithm $\texttt{Code}$ takes a code key $\texttt{ck}$, an encrypted ballot ev and a proof Π^v as input and outputs a pre-code $\hat{r}$ or $\perp$.

The *count* algorithm $\texttt{Count}$ takes a decryption key $\texttt{dk}$ and a sequence of encrypted ballots $ev_1, ev_2, \ldots, ev_{l_t}$, and outputs a sequence of ballots $v_1, v_2, \ldots, v_{l_t}$ and a proof Π^c, or $\perp$.

The *verify* algorithm $\texttt{Verify}$ takes a public key $\texttt{pk}$, a sequence of encrypted ballots $ev_1, ev_2, \ldots, ev_{l_t}$, a sequence of ballots $v_1, v_2, \ldots, v_{l_t}$ and a proof Π^c, and outputs 0 or 1.

A cryptographic voting scheme is *correct* if for any $(\texttt{pk}, \texttt{dk}, \texttt{ck})$ output by $\texttt{Setup}$ and any $(\texttt{vvk}_1, \texttt{vck}_1, f_1), \ldots, (\texttt{vvk}_{l_V}, \texttt{vck}_{l_V}, f_{l_V})$ output by $\texttt{Reg}(\texttt{pk})$, any ballots $v_1, \ldots, v_{l_V}$, any (ev_i, Π_i^v) output by $\texttt{Cast}(\texttt{pk}, \texttt{vck}_i, v_i)$, $i = 1, 2, \ldots, l_V$, and any $(v'_1, \ldots, v'_{l_V}, \Pi^c)$ output by $\texttt{Count}(\texttt{dk}, ev_1, \ldots, ev_{l_V})$, then:

- $\texttt{Code}(\texttt{ck}, \texttt{vvk}_i, ev_i, \Pi_i^v) = f_i(v_i)$,
- $\texttt{Verify}(\texttt{pk}, ev_1, \ldots, ev_{l_V}, v'_1, \ldots, v'_{l_V}, \Pi^c) = 1$, and
- $v_1, \ldots, v_{l_V}$ equals $v'_1, dots, v'_{l_V}$, up to order.

We further require that the distribution of ev_i only depends on $\mathtt{pk}$ and v_i, not $\mathtt{vck}_i$. Also, whether $\mathtt{Count}$ outputs $\perp$ does not depend on the order of the encrypted ballots, and if $\mathtt{Count}$ outputs $\perp$ for some list $(ev_1, \ldots, ev_{l_V})$ of encrypted ballots, $\mathtt{Count}$ outputs $\perp$ for any other list with $(ev_1, \ldots, ev_{l_V})$ as a prefix.

B.2 Our Scheme

We summarize the scheme from Section 5 in the above terms. The commitment algorithms are $(\mathtt{KeyGen}_C, \mathtt{Com}(,)\mathtt{Open}())$, the verifiable encryption algorithms are $(\mathtt{KeyGen}_{VE}, \mathtt{Enc}_{VE}, \mathtt{Ver}_{VE}, \mathtt{Dec}_{VE})$.

- The *setup* algorithm computes $\mathtt{pk}_C \leftarrow \mathtt{KeyGen}_C$, $(\mathtt{pk}_V, \mathtt{dk}_V) \leftarrow \mathtt{KeyGen}_{VE}$, $(\mathtt{pk}_R, \mathtt{dk}_R) \leftarrow \mathtt{KeyGen}_{VE}$. The public key $\mathtt{pk} = (\mathtt{pk}_C, \mathtt{pk}_V, \mathtt{pk}_R)$, the decryption key is $\mathtt{dk} = (\mathtt{pk}_C, \mathtt{dk}_V)$ and the code key is $\mathtt{ck} = (\mathtt{pk}_C, \mathtt{pk}_V, \mathtt{dk}_R)$.
- The *register* algorithm takes $\mathtt{pk} = (\mathtt{pk}_C, \mathtt{pk}_V, \mathtt{pk}_R)$ as input. It samples $a \xleftarrow{\$} R_p$ and computes $(\boldsymbol{c}_a, d_a) \leftarrow \mathtt{Com}(\mathtt{pk}_C, a)$. 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 $v \mapsto v + a$.
- The *cast* algorithm takes $\mathtt{pk} = (\mathtt{pk}_C, \mathtt{pk}_V, \mathtt{pk}_R)$, $\mathtt{vck} = (a, \boldsymbol{c}_a, d_a)$ and v as input. It computes $(\boldsymbol{c}, d) \leftarrow \mathtt{Com}(\mathtt{pk}_C, v)$, $\hat{r} \leftarrow a + v$, $(\boldsymbol{c}_{\hat{r}}, d_{\hat{r}}) \leftarrow \mathtt{Com}(\mathtt{pk}_C, \hat{r})$, $\Pi^{lin}_{\hat{r}}$is a proof that $\boldsymbol{c} + \boldsymbol{c}_a$ (which is a commitment to $v + a$) and $\boldsymbol{c}_{\hat{r}}$ satisfy the relation $v + a = \hat{r}$. Then it encrypts $\boldsymbol{e} = (\boldsymbol{v}, \boldsymbol{w}, c, \boldsymbol{z}) \leftarrow \mathtt{Enc}_{VE}(\mathtt{pk}_V, d)$ and $\boldsymbol{e}_{\hat{r}} = (\boldsymbol{v}_{\hat{r}}, \boldsymbol{w}_{\hat{r}}, c_{\hat{r}}, \boldsymbol{z}_{\hat{r}}) \leftarrow \mathtt{Enc}_{VE}(\mathtt{pk}_R, d_{\hat{r}})$. The encrypted ballot is $ev = (\boldsymbol{c}, d, \boldsymbol{v}, \boldsymbol{w}, c)$, while the ballot proof is $\Pi^v = (\boldsymbol{z}, \boldsymbol{c}_{\hat{r}}, \boldsymbol{e}_{\hat{r}}, \Pi^{lin}_{\hat{r}})$.
- The *code* algorithm takes $\mathtt{ck} = (\mathtt{pk}_C, \mathtt{pk}_V, \mathtt{dk}_R)$, a voter verification key $\mathtt{vvk}$, an encrypted ballot $ev = (\boldsymbol{c}, d, \boldsymbol{v}, \boldsymbol{w}, c)$ and a ballot proof $\Pi^v = (\boldsymbol{z}, \boldsymbol{c}_{\hat{r}}, \boldsymbol{e}_{\hat{r}}, \Pi^{lin}_{\hat{r}})$ as input. It verifies $\Pi^{lin}_{\hat{r}}$, and then verifies $(\boldsymbol{v}, \boldsymbol{w}, c, \boldsymbol{z}))$ and $\boldsymbol{e}_{\hat{r}}$ using $\mathtt{Ver}_{VE}$. If any verification fails, it outputs $\perp$. It then decrypts $d_{\hat{r}} \leftarrow \boldsymbol{e}_{\hat{r}}$ and recovers $\hat{r}$ from $\boldsymbol{c}_{\hat{r}}$ and $d_{\hat{r}}$, and outputs $\hat{r}$.
- The *count* algorithm takes as input $\mathtt{dk} = (\mathtt{pk}_C, \mathtt{dk}_V)$ and encrypted ballots $(\boldsymbol{c}_1, \boldsymbol{v}_1, \boldsymbol{w}_1, c_1), \ldots, (\boldsymbol{c}_{l_t}, \boldsymbol{v}_{l_t}, \boldsymbol{w}_{l_t}, c_{l_t})$. It computes $d_i \leftarrow \mathtt{Dec}_{VE}(\mathtt{dk}_V, \boldsymbol{v}_i, \boldsymbol{w}_i, c_i)$ and recovers v'_i from $\boldsymbol{c}_i$ and d_i. If any decryption fails, it outputs $\perp$. Otherwise, it chooses a random permutation π on $\{1, 2, \ldots, l_t\}$, sets $v_{\pi(i)} = v'_i$, and creates a proof of shuffle of known values Π^c. It outputs $v_1, v_2, \ldots, v_{l_t}$ and Π^c.
- The *verify* algorithm takes as input $\mathtt{pk} = (\mathtt{pk}_C, \mathtt{pk}_V, \mathtt{pk}_R)$, encrypted ballots $(\boldsymbol{c}_1, \boldsymbol{v}_1, \boldsymbol{w}_1, c_1), \ldots, (\boldsymbol{c}_{l_t}, \boldsymbol{v}_{l_t}, \boldsymbol{w}_{l_t}, c_{l_t})$, ballots $v_1, v_2, \ldots, v_{l_t}$ and a count proof Π^c. It verifies that Π^c is a correct proof of shuffle of known values for $\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_{l_t}$ and $v_1, v_2, \ldots, v_{l_t}$. It outputs 1 if verification holds, otherwise it outputs 0.

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

B.3 Our Voting Protocol

Recall the protocol described in Fig. 6.

In the *setup phase*, a trusted set of players run the setup algorithm `Setup`. The derived public key `pk` is given to every player, the decryption key `dk` is given to the shuffler S and the code key `ck` is given to the return code generator R.

In the *registration phase*, a set of trusted players run the register algorithm `Reg` to generate per-voter keys $(\texttt{vvk}, \texttt{vck}, f)$ for the voter V, making every verification key public and giving `vck` to the voter's computer. Then the return code generator chooses key k for `PRF`, and a set of trusted players compute the return code table $\{(v_i, \texttt{PRF}_k(V, f(v_i))\}$ for a relatively small set of ballots $v_1, \ldots, v_\omega$. The voter gets the return code table.

In the *casting phase*, the voter V instructs the voter's computer D which ballot to cast. The voter's computer D runs the casting algorithm `Cast` and sends the encrypted ballot and the ballot proof to the ballot box B. (In practice, the ballot box will verify the ballot proofs, but it is not necessary.) The ballot box B sends the encrypted ballot and the ballot proof to the return code generator R, who runs the code algorithm `Code` to get the precode $\hat{r}$. It computes the return code $r \leftarrow \texttt{PRF}_k(\hat{r})$ and sends r to the voter's phone F, which sends it on to the voter V. The voter V compares the return code to the code in its return code table, and accepts the ballot as cast if and only if the codes match.

In addition, the voter's computer will sign the encrypted ballot and ballot proof on behalf of the voter. The ballot box and the return code generator will verify this signature. The return code generator will countersign everything and return this signature to the voter's computer via the ballot box. The voter's computer will verify the countersignature. (This ensures that if the voter's computer accepts the ballot as cast, the ballot box and the return code generator agree that the ballot was indeed cast. If either of them is honest, this gives us a stronger form of integrity.)

In the casting phase, every player uses fixed-length encodings for messages, ensuring that every message of a given type has a fixed length that is public knowledge.

In the *counting phase*, the ballot box B and the return code generator R send the encrypted ballots and ballot proofs they have seen to the auditor A. If the data is consistent, the auditor A approves. The ballot box B then sorts the list of encrypted ballots and sends this to the shuffler S. (In the event that some voter has cast more than one ballot, only the encrypted ballot seen last is included.) The shuffler S uses the count algorithm `Count` to compute a list of ballots $v_1, \ldots, v_{l_t}$ and a shuffle proof, which is sent to the auditor A. The auditor A uses the verification algorithm `Verify` to verify the shuffle proof against the encrypted ballots received from B and R.

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

If some limited verifiability is desired, the ballot box B, return code generator R and auditor A may make commitments to the encrypted ballots received public. The voter's computer can be given the commitment and an opening to verify the correctness of the commitment to the voter's encrypted ballot, as well as its presence in the public record.

Note that more than one auditor can be used in the protocol.

B.4 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 and the decrypted ballots 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 and some voter casting keys, and insert maliciously generated ciphertexts into the ballot box.

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 and choose which ballots get counted. This experiment models both a left-or-right game for confidentiality, and a test query for integrity. The experiment works as follows:

- Sample $b, b'' \xleftarrow{\$} \{0,1\}$. Set L to be an empty list.
- $(\mathtt{pk}, \mathtt{dk}, \mathtt{ck}) \leftarrow \mathtt{Setup}$. For $i = 1, \ldots, l_V$: $(\mathtt{vvk}_i, \mathtt{vck}_i, f_i) \leftarrow \mathtt{Reg}(\mathtt{pk})$. Send $(\mathtt{pk}, \mathtt{vvk}_1, \ldots, \mathtt{vvk}_{l_V})$ to $\mathcal{A}$.
- On a *voter reveal query* i, send $(\mathtt{vck}_i, f_i)$ to $\mathcal{A}$. On a *decrypt reveal query*, send $\mathtt{dk}$ 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 and 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 *count query* $(j_1, \ldots, j_{l_s})$, with

$$L = ((i_1, v_{0,1}, v_{1,1}, ev_1, \Pi_1^v), \ldots, (i_{l_t}, v_{0,l_t}, v_{1,l_t}, ev_{l_t}, \Pi_{l_t}^v)),$$

  compute $result \leftarrow \mathtt{Count}(\mathtt{dk}, ev_{j_1}, \ldots, ev_{j_{l_s}})$ and send $result$ to $\mathcal{A}$.
- On a *test query* $(j_1, \ldots, j_{l_s}, v_1, \ldots, v_{l_s}, \Pi^c)$, compute $result \leftarrow \mathtt{Verify}(\mathtt{pk}, ev_{j_1}, \ldots, ev_{j_{l_s}}, v_1, \ldots, v_{l_s}, \Pi^c)$ and send $result$ to $\mathcal{A}$.

Eventually, the adversary outputs a bit b'.

Confidentiality fails trivially for the usual reasons, and in particular, the count trivially reveals the challenge bit 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.) For executions where confidentiality fails trivially, we should not count the adversary's answer towards the advantage, so we will compare the adversary's guess with a secret random bit. Integrity can fail, either if pre-codes are incorrect, if an outcome

verifies as correct but is inconsistent with the challenge ciphertexts, or if there is no unique decryption.

We define events related to confidentiality and integrity. Let E_g be the event that $b = b'$ and let E_r be the event that $b'' = b'$. Let E_f denote the event that an execution is *fresh*, which is true if the following are satisfied: there is no decrypt reveal query; for any i, there is either no challenge query, or at most one challenge query and no voter reveal query or chosen ciphertext query; if there is a count query where $result \neq \perp$, then the sequence $(v_{0,j_1}, \ldots, v_{0,j_{l_s}})$ equals $(v_{1,j_1}, \ldots, v_{1,j_{l_s}})$, up to order.

Let F_i (incorrect pre-code) be the event that for some chosen ciphertext query (i, ev, Π^v) where $\texttt{Code}(\texttt{ck}, \texttt{vvk}_i, ev, \Pi^v) = \hat{r} \neq \perp$, we have that either $\texttt{Count}(\texttt{dk}, ev) = \perp$ or $\texttt{Count}(\texttt{dk}, ev) = (v, \Pi^c)$ and $f_i(v) \neq \hat{r}$. Let F_c (count failure) be the event that a count query gets $result = \perp$. Let F_d (inconsistent decryption) be the event that a test query $(j_1, \ldots, j_{l_s}, v_1, \ldots, v_{l_s}, \Pi^c)$ with $L = ((i_1, v_{0,1}, v_{1,1}, ev_1, \Pi_1^v), \ldots, (i_{l_t}, v_{0,l_t}, v_{1,l_t}, ev_{l_t}, \Pi_{l_t}^v))$ gets $result = 1$ and there is no permutation π on $\{1, 2, \ldots, l_s\}$ such that $v_{b,k} = \perp$ or $v_{b,k} = v_{\pi(k)}$ for $k = 1, 2, \ldots, l_s$. Let F_u (no unique decryption) be the event that two test queries $(j_1, \ldots, j_{l_s}, v_1, \ldots, v_{l_s}, \Pi^c)$ and $(j_1, \ldots, j_{l_s}, v'_1, \ldots, v'_{l_s}, \Pi^{c'})$ both get $result = 1$, but there is no permutation π on $\{1, 2, \ldots, l_s\}$ such that $v_k = v'_{\pi(k)}$ for $k = 1, 2, \ldots, l_s$.

The advantage of the adversary is

$$\max\{2|\Pr[E_f \wedge E_g] + \Pr[\neg E_f \wedge E_r] - 1/2|, \Pr[F_i \vee F_c \vee F_d \vee F_u]\}.$$

B.5 Security Proof Sketch

We briefly sketch how a proof to bound the advantage of an adversary against the cryptographic voting scheme in terms of adversaries against the shuffle of known values, the underlying commitment scheme or the related linearity proofs, or the verifiable encryption scheme.

Confidentiality events We begin by analyzing the confidentiality events. Note that $\Pr[\neg E_f \wedge E_r] = (1 - \Pr[E_f])/2$. We must therefore analyze $\Pr[E_f \wedge E_g] = \Pr[E_g \mid E_f] \Pr[E_f]$.

The proof would proceed as a sequence of games, where the first 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. 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

create a real-or-random adversary against the commitment scheme by making a the challenge and getting a commitment that is either a or a random value.

Next, in the count query, instead of decrypting an encrypted ballot from a challenge query, we use the corresponding left cleartext (regardless of the value of b). Since the shuffle proof has been simulated, the execution is fresh and the ballots are permuted randomly, this change is not observable.

Next, in the count query, instead of decrypting an encrypted ballot from a challenge query, we instead compute the ballot from the corresponding pre-code $\hat{r}$ and voter casting key component a as $v = \hat{r} - a$. This change is only observable if the linear proof verified in the code algorithm was unsound, that is, if we have openings of $\boldsymbol{c}$ to v, of $\boldsymbol{c}_a$ to a, of $\boldsymbol{c} + \boldsymbol{c}_a$ to $v + a$ and $\boldsymbol{c}_{\hat{r}}$ to $\hat{r}$, but $\hat{r} \neq v + a$. Also, the change is observable if the encrypted ballot does not decrypt to an opening, but since the proof for the encrypted ballot was verified during the code query, it follows that this results in an adversary against the verifiability of the encryption scheme.

Observe that at this point, the decryption key $\mathtt{dk}_V$ is no longer used. Also, the pre-code encrypted in the challenge query is independent of the challenge ballots.

In the next game, we encrypt randomness instead of the correct opening of $\boldsymbol{c}$. If this change is observable, we get a real-or-random adversary against the verifiable encryption.

Finally, we commit to a random value instead of the challenge ballot. If this change is observable, we get a real-or-random adversary against the commitment scheme.

We can now observe that the challenge query processing is independent of the challenge bit. The adversary no longer has any information about the challenge bit, and therefore has no advantage in this game. The claim that the difference between $\Pr[E_f \wedge E_g] + \Pr[\neg E_f \wedge E_r]$ and $1/2$ is appropriately bounded follows.

Integrity events 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 like above either the ciphertext $\boldsymbol{e}$ does not decrypt to an opening of the commitment (in which case we get an adversary against the verifiability of the encryption scheme), or we have broken the linearity proof for commitments. It follows that the probability of F_i happening is appropriately bounded.

In the event that F_c happens, then since every encrypted ballot either originates with a challenge query or a chosen ciphertext query, we know that either the ciphertext $\boldsymbol{e}$ will decrypt to an opening of the commitment or we will have an adversary against the verifiability of the encryption scheme. In either case, it follows that the probability of F_c happening is appropriately bounded.

In the event that F_d happens, we have openings of the ciphertexts that originated with challenge queries. Since the shuffle is a proof of knowledge, we get an adversary against binding for the commitment scheme. It follows that the probability of F_d happening is appropriately bounded.

In the event that F_u happens, then since the shuffle is a proof of knowledge, we get an adversary against binding for the commitment scheme. 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.6 Voting System Security Properties

Coercion Resistance A coercer controls the voter during ballot casting. The appropriate mental image we should have in mind is that the coercer is a "helpful" person who observes the voter while the voter is casting a ballot of the coercer's choice. The coercer may "assist" the voter in the casting process. When the ballot casting is done, the coercer leaves and the voter is left to their own devices. (The coercer may return at a later point in time and redo the coercion process.)

The voter may choose to *resist* the coercion attempt. This may involve active steps during the coerced ballot casting (e.g. lying about authentication data). For remote voting, it will usually also involve *recovery* after the coercer has left.

The coercer plays a game with a set of voters, some of which may be corrupt. The coercer may coerce one or more voters, who will either all resist or all accept coercion. The coercer may also ask voters to cast ballots uncoerced. Eventually, there is a tally and the coercer receives the outcome (including any transcripts).

A system is *coercion resistant* if the coercer cannot decide if the voters resisted.

It is generally assumed that a coercer does not control any infrastructure players and cannot monitor networks, since for remote voting either capability usually allows trivial winning strategies for the coercer. In the real world, a coercer will always be able deduce some information about the success of the coercion attempt by looking at unavoidable public information such as the election result. This applies, regardless of the voting system used (so-called Italian attacks are an example of this). It is desirable to avoid this class of attacks, which can be done by having the coercer decide the voting intention of every voter. The consequence is that the coercer must organize coercion such that it has no effect on the election result, regardless of whether voters resist or not.

Analysis. Our voting system uses re-voting to resist coercion: after the coercer has left, the voter casts another ballot, this time according to their true voting intention. The use of re-voting for coercion resistance is well-understood and largely independent of the underlying cryptosystem. Since the coercer does not learn the contents of the ballot box (except possibly for the commitments), nor any network traffic, it is impossible for the coercer to discover the re-voting.

In summary, the coercer cannot decide if the voter re-voted or not.

Privacy Privacy is modeled as an indistinguishability game between 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.

Unlike for coercion resistance, the adversary can corrupt infrastructure players and also control the network. Like for coercion resistance, we want to avoid adversaries that deduce the honest voters' ballots from the cast ballots. Again, we require that the adversary organizes the pairs of ballots given to the honest voters in such a way that the ballots cast are independent of whether the voters cast the left or the right ballot. The difference between privacy and coercion is that for privacy, the adversary learns the ballots cast by compromising infrastructure players. For coercion, the coercer only has access to public information from the infrastructure players, but may have access to private information about the voter.

The adversary controls the network, but 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.

Analysis. If some honest voter's *computer* D is compromised, the adversary can trivially win the privacy game.

Next, consider the case that the *shuffler* S is compromised. If any of B, R or A is compromised, the adversary can trivially win the game, since B, R and A all know who submitted which encrypted ballot, while the shuffler S has the decryption key. If the shuffler S is the only compromised player, then since the ballot box B sorts the list of encrypted ballots before sending them to the shuffler and encrypted ballots are independent of the per-voter key material, the correspondence between the ciphertexts and the voters is lost, so the encrypted ballots alone reveal only the cast ballots. By assumption, these ballots are independent of the left-or-right choice of the voters.

If a voter casts more than one ballot, a compromised *return code generator* 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 (*e.g.* if the voter is resisting coercion), the return code generator will get information about which ballots were submitted, and typically learn both ballots.

Suppose the honest voters cast at most one ballot each. Then privacy against B, R and A follows from confidentiality of the cryptographic voting system, since the protocol execution can be interpreted as an interaction with the cryptosystem experiment and our assumptions ensure a fresh execution.

Note that cut-and-paste attacks against confidentiality, which commonly affect this type of voting protocol, do not work against this protocol because the ballot proof includes an encryption of the return code and a proof that the return code is correct, which means that the adversary must know the ballot to make a cut-and-paste attack work.

In summary, privacy holds if none of the honest voters' computers are compromised, and either only the decryption service is corrupted or no honest voter casts more than one ballot.

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

We can define a variant notion called ϵ-integrity where we allow a small error, and say that the adversary wins if the result is inconsistent with any $(1 - \epsilon)$ fraction of the ballots accepted as cast by 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 of collision in the PRF).

If the *return code generator* 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 A is honest, the count will only be accepted if the encrypted ballot has been included in the count by the shuffler S. By the integrity of the cryptographic voting system, all such ballots must then be included in the count.

If the voter's computer D, the ballot box B and the auditor A are honest, the count will only be accepted if the encrypted ballot has been included in the count by the shuffler S. By the integrity of the cryptographic voting system, all such ballots must then be included in the count.

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 the return code generator, or both the voters' computers and the ballot box are honest.

Limited Verifiability. In a *verifiable voting system* voters receive a *receipt*, the voting system provides an *election proof* in addition to the election outcome, and there is an additional *verification algorithm* that accepts or rejects the proof and a voter's receipt. Roughly speaking, the voting system is *verifiable* if when all the receipts accepted by the honest voters verify as accepted with the election proof, then the outcome is consistent with the honest voters' cast ballots.

We have *limited verifiability* if the same claim holds when certain players are honest during the election.

(Verifiability is a technical property. The practical idea is not that every voter verifies their ballot. But it can be shown that if a sufficiently large and random sample of voters separately accept their receipts together with the ballot proof, then ϵ-integrity holds for the voting system. In our particular system, if a sufficiently random selection of voters finds their commitments on the public

list, then with high probability almost all the voters would have found their commitments on the list if they had looked for them.)

Analysis. If the voters' *computers* are honest, then a voter's computer will not accept the ballot as cast unless it receives a commitment that opens to its encrypted ballot. A voter will not accept the result as verified unless the commitment has been made public.

An honest auditor will only make commitments public if both the ballot box and the return code generator agree on the presence of the encrypted ballot, and this ballot was given to the shuffler. By the integrity of the cryptographic voting system, almost all the ballots accepted as cast by honest voters will be among the ballots output by the decryption service.

In summary, for the variant voting system with limited verifiability, ϵ-integrity holds if the voters' computers and the auditor are honest.