Return Codes from Lattice Assumptions*

Audhild Høgåsen and Tjerand Silde

Department of Mathematical Sciences
Norwegian University of Science and Technology
audhildh@stud.ntnu.no, tjerand.silde@ntnu.no

Abstract. We present an approach for creating return codes for lattice-based electronic voting. For a voting system with four control components and two rounds of communication our scheme results in a total of 2.3 MB of communication per voter, taking less than 1 s of computation. Together with the shuffle and the decryption protocols by Aranha et al. [1,2], the return codes presented can be used to build a post-quantum secure cryptographic voting scheme.

Keywords: Lattice Cryptography · Return Codes · Electronic Voting

1 Introduction

In 2019, Switzerland put their electronic voting project on hold after having run electronic voting trials for 15 years. Now, electronic voting trials with a new and improved protocol [6] are in the planning. The new protocol offers individual and universal verifiability. Individual verifiability is achieved by using return codes, giving each voter a confirmation that the correct vote was registered by the system. The protocol does not assume a trustworthy voting server but does assume that at least one so-called control component is trustworthy.

The protocol [6] is based on discrete log-type assumptions, whose security could in a decade or two be broken by quantum computers. This is not only a future threat of integrity, but also a threat of privacy of votes cast today.

We present a lattice-based voting phase suitable for electronic voting with return codes, extending the framework by Aranha et al. [1,2]. While [1] includes return codes, but assumes a trustworthy voting server, [2] allows for an untrustworthy voting server, but does not include return codes. We fill this gap.

2 Lattice-Based Building Blocks

Let $R_q = \mathbb{Z}_q[X]/\langle X^N + 1\rangle$ where N a power of 2, and $p << q$ primes. We recall the setup in [2, Sec 3].

BGV Encryption [4] of message $m \in \mathbb{Z}_p^N$ with public key $\mathsf{pk} = (a, b) = (a, as+pe)$ with short uniform secret key $\mathsf{sk} = (s, e)$ is computed with short uniform r, e', e'':

$$c = \mathsf{Enc}(m, \mathsf{pk}) = (u, v) = (ar + pe', br + pe'' + m) \tag{1}$$

* This short paper is a compressed version of the master thesis of Audhild Høgåsen, which is available at ntnuopen.ntnu.no and tjerandsilde.no/academic.

Commitments [3] to messages $m \in R_q$ are computed with public matrices $A_1 = \left[I_n \; A_1' \right], A_2 = \left[0^{\ell \times n} \; I_\ell \; A_2' \right]$ where A_1' and A_2' are sampled uniformly random and a short uniform random vector d in the following way:

$$[\![m]\!] = \mathsf{Com}(m, d) = (c_1, c_2) = (A_1 d, A_2 d + m) \tag{2}$$

The described ciphertexts and commitments are additively homomorphic.

Zero-Knowledge Proofs are used to prove properties of commitments without revealing the openings. π_{LIN} [2, Sec 3.3] proves a linear relation $\alpha_1 m_1 + \cdots + \alpha_n m_n = \alpha_{n+1}$ with respect to commitments $[\![m_1]\!], \ldots [\![m_n]\!]$ and public scalars α_i. π_{AEx} [2, Sec 3.4] is an amortized exact proof of short openings. π_{NEx} [5, Section 5.2] is a proof of bounded opening. All these zero-knowledge proofs are proved secure in the random oracle model, but not in the quantum random oracle model.

3 The Swiss Post Voting Protocol

The Swiss Post Voting Protocol [6] is a return code-based electronic voting protocol. The voting phase consists of a SendVote protocol and a ConfirmVote protocol, with the following parties: voter (V), voting client (VC), voting server (VS) and several return code control components (CCR). The voter receives in advance of the election a voting card including return codes cc for each possible voting option of the election and a confirmation return code VCC. The setup and printing component making the voting cards are assumed to be trustworthy. It is assumed that at least one control component is trustworthy and that at least one honest auditor verifies the results using a trustworthy verifier. The voting client is trusted for privacy. $\mathsf{EL_{pk}}$ is the public election key. The SendVote Protocol shown in Figure 1 consists of the following steps:

1. V enters to VC the start voting key k from the voting card and selects voting options v corresponding to return codes cc.
2. VC computes the ballot b containing the encrypted vote ρ and encrypted partial return codes pCC. VC sends b to VS which forwards to CCR. Both verifies the ballot. CCR conducts a distributed decryption to retrieve pCC.
3. CCR generates return code shares $\mathsf{lCC_j}$ and sends them to VS.
4. VS combines the shares from CCR. With a mapping table it extracts return codes $\mathsf{cc}^\star$ that are sent to VC and shown to V.
5. V verifies $\mathsf{cc}^\star$ shown on the screen by checking that they are equal to cc.

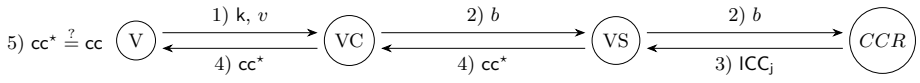

Fig. 1. The SendVote protocol of the Swiss Post Voting System [6, Figure 21].

In step 2, VC maps the selected voting options v of the voter to encodings $\{p_i\}$, then computes the vote $\rho = \prod p_i$ and the partial return codes $\{\mathsf{pCC}_i\} = \{p_i^k\}$. VC computes b consisting of two ciphertexts: an ElGamal encryption of

ρ using $\mathsf{EL}_{\mathsf{pk}}$ and a multi-recipient ElGamal encryption of $\{\mathsf{pCC}_i\}$ using the public key of CCR. b also includes one additional ciphertext and zero-knowledge proofs of correct exponentiation and of plaintext equality, proving that the initial ciphertexts were computed correctly, leaving no options for an untrustworthy VC to compute the two ciphertexts using different vote encodings $\{p_i\}$. Finally, b includes the identity of the voter and a signature [6, Sec 12.2.1.2].

In step 3, each component of CCR computes a return code share $\mathsf{lCC}_\mathsf{j} = \mathcal{H}(\mathsf{pCC})^{\mathsf{k}_\mathsf{j}}$ using a hash function and a secret user-specific key, and provides a zero-knowledge proof of correct exponentiation [6, Sec 12.2.1.6].

The ConfirmVote protocol [6, Sec 12.2.2] is only initiated by V if the verification from SendVote step 5 is successful. The steps of ConfirmVote are similar to the steps of SendVote. V types another key k' from the voting card. VC sends a confirmation key $\mathsf{CK}=(\mathsf{k}')^k$ to CCR. The CCR components compute shares $\mathsf{lCC}_\mathsf{j}{}' = \mathcal{H}(\mathsf{CK})^{\mathsf{k}'_\mathsf{j}}$ and a zero-knowledge proof of exponentiation. VS computes $\mathsf{VCC}^\star$ using the shares and a mapping table. Only after successfully verifying $\mathsf{VCC}^\star$ by comparing it with VCC from the voting card, V has completed the voting process.

We observe that in the ConfirmVote phase, VC gives no exponentiation proof for the computation of the confirmation key. An incorrect exponentiation would result in an unsuccessful confirmation attempt, but could not change the vote. The VC can always block the communication from the voter, thus an exponentiation proof would not change the security analysis.

4 Our Voting Protocol

Cryptographic primitives based on discrete log-type assumptions are used in the Swiss Post voting protocol [6] in steps 2 and 3 of the SendVote protocol of Figure 1, and similarly for the ConfirmVote protocol. The hash-functions used are considered post-quantum secure.

For privacy, the partial return codes are the weakest part of the protocol [6] as they are based on the ESGSP assumption. In the protocol we present, these partial return codes are uniformly random and therefore not an issue for long-term privacy. Still, these partial return codes must somehow be linked to the encrypted vote to avoid attacks from a cheating voting client. The ZK-proofs needed must be post-quantum secure to achieve long-time privacy. Therefore, when constructing a post-quantum secure voting system, we need to consider the voting phase as well, not only the tally phase as already described by [2].

Figure 2 presents a SendVote protocol using primitives based on lattice assumptions. In our protocol, VC does not encrypt the partial return codes pCC as the protocol security reductions for privacy [6, Sec 19.4] omit this encryption (but it could, if required). Commitments and shortness proofs to the polynomials k, k', k_j and k'_j are public information. The vote ρ is a bit-string which represents the voting options v chosen by V. There is a natural mapping from bit-strings to polynomials in R_q with coefficients modulo $p = 2$.

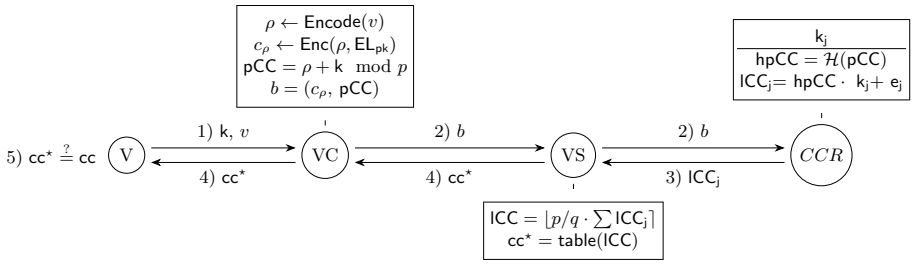

Fig. 2. Our SendVote protocol for lattice-based electronic voting.

In step 2, when VC computes $\mathsf{pCC} \mod p$, this might produce some computational overflow which is stored in a secret overflow binary vector z. VC computes commitments to z and to the randomness used in c_ρ. A proof π_{LIN} proves correct computation of pCC by proving that $\mathsf{pCC} + 2\ z = \rho + \mathsf{k} \mod q$. Proofs π_{LIN} prove correct computation of c_ρ as in Equation (1). A proof π_{AEx} proves that z and the randomness used in c_ρ are binary. Together, these proofs leave no options for an untrustworthy VC to compute c_ρ and pCC with different values of ρ or too much noise.

In step 3, each CCR component computes $\mathsf{ICC_j}$, a commitment to the added noise $\mathsf{e_j}$, a proof π_{LIN} proving that $\mathsf{ICC_j}$ was computed correctly with respect to hpCC, and a proof π_{NEx} proving that the noise value is bounded.

For the ConfirmVote protocol, VC computes $\mathsf{CK} = \mathsf{k}' + \mathsf{k} \mod p$. Each CCR component computes $\mathsf{ICC_j}' = \mathcal{H}(\mathsf{CK}) \cdot \mathsf{k_j}' + \mathsf{e_j}'$, a commitment to $\mathsf{e_j}'$, and proofs π_{LIN} and π_{NEx}.

5 Performance

We use equations, parameters and computed values from [2]. Sizes of ciphertexts, commitments, π_{LIN} are found in [Table 3]. The size of π_{AEx} for binary secrets and τ commitments is computed to $(443 + 6.3\tau)$ KB by [Equation 2] with parameters from [Sec 7.4]. The size of π_{NEx} for only one commitment proving that both the randomness and the message is computed correctly is estimated to 30 KB using [5, Section 5.2] with Gaussian standard deviation for one-time commitments like in [Table 1]. Timings of cryptographic operations to encrypt and commit are found in [Table 4]. Protocol timings from [Table 5] are given for an input of 1000 commitments. We use the given timings of π_{LIN} and assume the timings of π_{NEx} are at most the given timings of π_{ANEx}. By contacting the authors of [2] we received the following timings for an input of 10 commitments: 90τ ms for π_{AEx} and 60τ ms for π_{AExV}.

For the SendVote protocol, VC computes 1 ciphertext, 5 commitments, π_{LIN} for 8 commitments, and π_{AEx} for 5 commitments. Each CCR component computes 1 commitment, π_{LIN} for 2 commitments, and π_{NEx} for 1 commitment. For the ConfirmVote protocol, each CCR component computes 1 commitment, π_{LIN} for 2 commitments, and π_{NEx} for 1 commitment.

For the SendVote protocol we achieve 1095 KB of communication from VC, and 145 KB from each CCR component. For the ConfirmVote protocol we achieve

another 145 KB from each CCR component. As a concrete example having four CCR components the total communication size of the two round voting phase is 2.3 MB.

For the SendVote protocol we achieve timings of 498 ms for VC and 404 ms for each CCR component, computing in parallel, including verifying the proofs from VC. This results in total timings of 902 ms. For the ConfirmVote protocol we achieve timings of 65 ms for each CCR component.

The estimates of communication sizes and timings are meant to give an indication of the performance of the presented protocol, and not an exact performance of an actual implemented system. The waiting time for V until return codes are shown could be reduced if VC starts computing commitments and proofs while V is typing the voting options. We emphasize that the waiting time is not only dependent on the timing of the cryptographic operations, but would in practice be dominated by human operations and network-latency. Among the cryptographic operations, the proofs of exact shortness are the most expensive, both in terms of size and timings. Because exact proofs keep the overall parameters of the system low, they are to be preferred over relaxed proofs of boundedness. We expect that future work on more efficient lattice-based zero-knowledge proofs of exact shortness will improve the concrete efficiency of our protocol.

Acknowledgements

We thank Diego F. Aranha for providing timings of the underlying protocols.

References

1. Aranha, D.F., Baum, C., Gjøsteen, K., Silde, T., Tunge, T.: Lattice-based proof of shuffle and applications to electronic voting. In: CT-RSA (2021)
2. Aranha, D.F., Baum, C., Gjøsteen, K., Silde, T.: Verifiable mix-nets and distributed decryption for voting from lattice-based assumptions. Cryptology ePrint Archive, Report 2022/422 (2022), https://ia.cr/2022/422
3. Baum, C., Damgård, I., Lyubashevsky, V., Oechsner, S., Peikert, C.: More efficient commitments from structured lattice assumptions. In: International Conference on Security and Cryptography for Networks. Springer (2018)
4. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (2014)
5. Lyubashevsky, V.: Basic lattice cryptography: Encryption and fiat-shamir signatures (2019), https://drive.google.com/file/d/1JTdW5ryznp-dUBBjN12QbvWz9R41NDGU/view
6. SwissPost: Protocol of the swiss post voting system – computational proof of complete verifiability and privacy – version 0.9.11 (2021), https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/97c83a77c9ebda4c3a47fca022c60cbcb006d452/Protocol/Swiss_Post_Voting_Protocol_Computational_proof.pdf