Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Audhild Høgåsen

Return Codes from Lattice Assumptions

Master's thesis in Industrial Mathematics
Supervisor: Professor Kristian Gjøsteen
Co-supervisor: Tjerand Silde
July 2022

NTNU
Norwegian University of Science and Technology

Audhild Høgåsen

Return Codes from Lattice Assumptions

Master's thesis in Industrial Mathematics
Supervisor: Professor Kristian Gjøsteen
Co-supervisor: Tjerand Silde
July 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Abstract

We present an approach for creating return codes for lattice-based electronic voting. For a voting system with four control components and two-round communications, our scheme results in a total of 2.3 MB of communication per voter, taking less than 1 s of computation.

The Swiss Post electronic voting protocol [Swi21], planned for use in elections and referendums in Switzerland, assumes an untrustworthy voting server and offers individual verifiability, universal verifiability, and privacy. The protocol is based on discrete log-type assumptions, whose security could be broken by quantum computers in a decade or two. This is not only a future threat against integrity, but also a threat against privacy of votes cast today.

Together with the shuffle and the decryption protocols by Aranha et al. [Ara+21; Ara+22], the return codes presented can be used to build a post-quantum secure cryptographic voting scheme compatible with the trust assumptions of [Swi21]. While [Ara+21] includes return codes, but assumes a trustworthy voting server, [Ara+22] allows for an untrustworthy voting server but does not include return codes. We fill this gap.

Sammendrag

Vi presenterer en fremgangsmåte for å lage returkoder for et gitter-basert elektronisk valgsystem. For et valgsystem med fire kontrollkomponenter og kommunikasjon i to runder oppnår protokollen vår en total kommunikasjonsstørrelse på 2.3 MB per velger, og bruker under 1 s for utregningene.

Den elektroniske valgprotokollen til Swiss Post [Swi21], som Sveits planlegger å bruke i valg og folkeavstemninger, antar en upålitelig valgserver og tilbyr individuell verifiserbarhet, universell verifiserbarhet og hemmelighold av stemmesedler. Protokollen bruker sikkerhets-antagelser basert på diskret logaritme, og disse antagelsene vil kunne trues av kvantedatamaskiner om et tiår eller to. Dette er ikke bare en fremtidig trussel mot integritet, men også en trussel mot langvarig hemmelighold av stemmesedler avgitt i dag.

Vi presenterer en gitter-basert valgprotokoll passende for valg med returkoder, og utvider med dette rammeverket til Aranha et al. [Ara+21; Ara+22]. [Ara+21] inkluderer returkoder, men antar en pålitelig valgserver, mens [Ara+22] antar en upålitelig valgserver, men inkluderer ikke returkoder. Våre returkoder kan sammen med miks- og dekrypterings-protokollene til Aranha et al. brukes til å lage et post-kvante-sikkert kryptografisk valgsystem som er kompatibelt med tillitsantagelsene til [Swi21].

Preface

This work is the result of the TMA4900 Industrial Mathematics Master's Thesis at the Norwegian University of Science and Technology (NTNU). The thesis concludes my studies at the 5-years master's degree programme Applied Physics and Mathematics with specialization in cryptography.

A compressed version of this master thesis has been accepted as a short paper for the conference E-Vote-ID which will be held in Bregenz, Austria in October 2022. The short paper is joint work with Tjerand Silde. The short paper will be published in University of Tartu Press Proceedings and will be available at e-vote-id.org/proceedings.

This thesis is available at ntnuopen.ntnu.no and tjerandsilde.no/academic.

Acknowledgements

First, I want to thank my supervisors Kristian Gjøsteen and Tjerand Silde for making the last year of my studies also the most interesting!

Thank you Kristian for motivation and guidance in the start phase of the master project to find the topic for my thesis, which I have very much enjoyed. Thank you for your comments and advice during the year. Thank you for being so flexible and allowing for online meetings between Norway and Switzerland.

Thank you Tjerand for the weekly meetings where I could ask all my questions. Thank you for being so patient and positive even when I often asked the same questions over and over again, and even when the meetings often exceeded the agreed time. I have learned so much from our discussions, and the frequent meetings have motivated me to work continuously with the thesis throughout the semester. Thank you for giving so much of your time and for sharing your knowledge with me.

Thank you to professors and study advisors at the Department of Mathematical Sciences and at the Department of Physics at NTNU, first for providing interesting studies in Trondheim and then for being so flexible and letting me exchange five semesters at the University of Innsbruck and one semester at the University of Bern. Thanks to friends in Trondheim, Innsbruck, Bern, Valencia, Grenoble and Thun for making my study years such a good time!

Thank you Damaris and the other organizers and participants at the Bern-Fribourg spring 2022 graduate seminar for letting me join your seminar, letting me hold a presentation about my master thesis, and for the interesting discussions.

Thank you Timo for patiently listening to my monologues, for discussing electronic voting and cryptography with me, and for proofreading the thesis. Thank you Anne Kirsti for proofreading. Thank you everyone who discussed electronic voting and cryptography with me the last year, at the university, online, in the park and in the mountains!

A compressed version of this master thesis has been accepted as a short paper for the conference E-Vote-ID. Thank you Tjerand for the invested time and effort in writing the short paper with me. Thank you to the anonymous reviewers who provided helpful comments. Thank you Diego F. Aranha for providing specific timings of the underlying ZK-protocols.

Contents

List of Figures

List of Tables

Keys and Codes

b The ballot made by VC in the SendVote protocol.

cc The return codes corresponding to the voting options. Part of the voting card of the voter. Used in the SendVote protocol.

CK The confirmation key, computed by VC in the ConfirmVote protocol.

$\mathsf{EL_{pk}}$ The election public key used to encrypt the vote ρ. The corresponding private key is held by a trusted election board.

k The start voting key. Part of the voting card of the voter. Used in the SendVote protocol.

k' The ballot casting key. Part of the voting card of the voter. Used in the ConfirmVote protocol.

$\mathsf{k_j}$ A user-specific key held by CCR component j, used to make a return code share in the SendVote protocol.

$\mathsf{k'_j}$ A user-specific key held by CCR component j, used to make a return code share in the ConfirmVote protocol.

$\mathsf{ICC_j}$ Return code share made by CCR component j in the SendVote protocol.

$\mathsf{IVCC_j}$ Return code share made by CCR component j in the ConfirmVote protocol.

pCC The partial return codes computed by VC in the SendVote protocol.

v The voter's selected voting options, chosen in the SendVote protocol.

VCC The vote cast return code. Part of the voting card of the voter. Used in the ConfirmVote protocol.

vcd The Voting Card ID is a unique random id for each voter.

ρ The vote which is a representation of the voter's selected voting options in the SendVote protocol.

1 Introduction

Every democratic country needs a voting system for their citizens. Traditionally this voting system is an in-person paper-based system, but in the last few decades some countries have explored the possibility of an online voting system, letting their inhabitants participate in an election from wherever they want, using their own devices. The supporters of online voting claim it could reduce costs, increase the speed of counting ballots, and increase availability for the voters. However, severe problems were found in every online voting system implemented to be used in elections, including the systems used in Switzerland [Hai+20], Australia [HT15], Estonia [Spr+14] [Per21], and the United States [SKW20]. Also, quantum computers compose not only a future threat against integrity of online voting, but also a threat against privacy of votes cast today.

1.1 Purpose

There are at least three obvious reasons why Switzerland is eager to implement an online voting system. First, the frequent voting events. While most democratic countries today are indirect democracies where citizens cast their ballots about every second year, Switzerland is a direct democracy where the citizens cast their ballots not only in the parliamentary elections held every fourth year, but also in referendums three or four times a year. Second, Switzerland does not consider coercion to be an issue in their elections and does not require an electronic voting system to be coercion resistant. Through their extensive use of postal voting, the possibility of vote selling is already present. Therefore, implementing an electronic voting system in Switzerland faces one issue less than in countries where coercion resistance is a requirement. Third, the high percentage of Swiss living abroad. Around 11% of the Swiss population live abroad,[1] and an electronic voting system could make Swiss elections independent of foreign postal services.

The first trials with internet voting for federal votes in Switzerland were run in the canton of Geneva in 2004.[2] By 2010, trials in 12 of 26 cantons with three different systems were held and in 2012, 50% of Swiss living abroad could use internet voting for federal votes. The Swiss Post voting system came in use in 2016, and in 2019, the source code of this system was published. After severe failures were found in this source code [Hai+20], Switzerland decided in March 2019 to put the whole electronic voting project on hold. Now, electronic voting trials are again in the planning. Since the 1st of July 2022 a new legislation applies for the first stage of the redesign of e-voting trials.[3] This new legislation again allows the Swiss cantons to apply to the Federal Council to offer e-voting on a trial basis. Some cantons are planning to resume trials with the new and improved Swiss Post voting system.

The Swiss Post voting system [Swi21] uses cryptography whose security is based on discrete log-type assumptions. These mathematical assumptions are assumed to provide the necessary security against classical computers. However, quantum computers could be a threat against the security of these systems. A quantum computer uses qubits, which not only can be represented as 0 or 1, but also as a superposition of both. This allows to speed up the algorithms to compute discrete logarithms. It is hard to say when quantum computers will be strong enough to break discrete log-type assumptions. First then, they could tamper with the results of an ongoing election and be a threat against the integrity of online voting protocols. Still, the possibility of future quantum computers is also a threat against privacy of votes cast today. A future quantum computer could break the vote secrecy of a past election. Ballots cast today could be stored by those waiting for quantum computers to decrypt them in the future.

To ensure long-lasting secrecy of ballots, voting systems could use cryptography based on other mathematical assumptions than discrete logarithm. Lattice-based cryptography is a good candidate. The best algorithms known for breaking the assumptions of lattice-based cryptography cannot

[1] https://www.eda.admin.ch/eda/en/fdfa/living-abroad/schweizerinnen-und-schweizer-im-ausland/fifth-switzerland/statistics.html

[2] https://www.bk.admin.ch/bk/en/home/politische-rechte/e-voting/chronik.html

[3] https://www.bk.admin.ch/bk/en/home/dokumentation/medienmitteilungen.msg-id-89020.html

All links 1, 2, and 3: Accessed the 29th of May 2022

be speeded up by quantum computers. As of today, no voting system in use is post-quantum secure. In this thesis, we investigate how the Swiss voting protocol can be made post-quantum secure using lattice-based cryptography.

1.2 Return Code-Based Voting

In an electronic voting system, a voter can use her personal device, from now on called the voting client, to send her vote to a voting server. We could imagine a very simple voting system, where the voter types her preferred voting option to the voting client which encrypts it and sends it to the voting server. Finally, the voting client shows the voter a message on the screen saying "Congratulations, you have now voted in this election". Trusting the voting client to perform such an important task is risky; how does the voter know that her voting client really delivered the right vote to the voting server? A hacked voting client could try to cast another vote than the vote intended by the voter. We need to design the voting system in such a way that a voter can be assured that the right vote arrives to the voting server even if her voting client is hacked. This can be solved with so-called return codes, giving each voter a confirmation that the correct vote was received and recorded by the voting system.

A return code-based voting system consists of three phases. In the setup phase, everything is prepared for the election, the different components receive their keys, and the voters receive per postal mail a voting card consisting of different keys and return codes. In the voting phase, the voter types her preferred voting options to the voting client and receives some return codes back which she can compare with the return codes from her voting card. Finally in the tally phase, the election result is computed based on all the confirmed votes. In the following we focus on the voting phase of a return code-based voting system.

The voting phase consists of voter, voting client, voting server and return code server. The voting phase includes different cryptographic operations like encryption and zero-knowledge proofs. Because a voter cannot do any cryptographic operations, the voting client will do these operations for her. Figure 1 shows the 5 steps of the SendVote protocol of the voting phase. The protocol is initiated by the voter by typing some key and is ended after the voter successfully compared the received return codes with the return codes from her voting card.

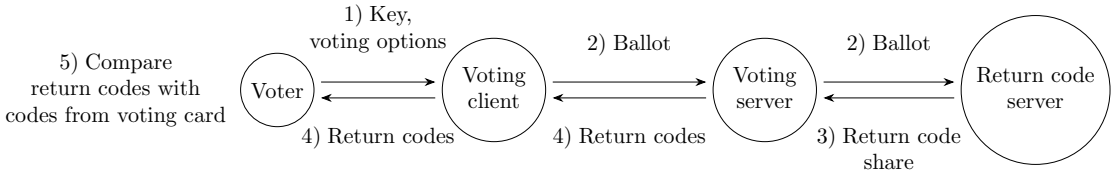


Figure 1: The SendVote protocol of a return code-based voting scheme

1. To start the voting process, the voter types the start voting key from her voting card to the voting client. Then she types her preferred voting options.
2. The voting client encrypts the voting options and computes the ballot. The voting client sends the ballot to the voting server which forwards it to the return code server.
3. Based on the ballot, the return code server computes a return code share and sends it to the voting server.
4. Based on the return code share, the voting server computes the return codes. The return codes are sent back to the voter.
5. The voter verifies the return codes shown on the screen by comparing them with the corresponding return codes from the voting card.

When the return codes shown on the screen correspond with the return codes from the voting card, the voter is assured that their intended vote was correctly sent from the voting client to the voting

server. This property is called sent-as-intended individual verifiability. As already mentioned, the vote is encrypted by the voting client before being sent over the internet to the voting server. This encryption is done so that nobody eavesdropping the internet communication can learn the vote and assures confidentiality of the vote against the "outside world". Making return codes is simple if we only care about this type of privacy. We simply let the voting server decrypt the vote, and send return codes based on the plaintext vote. However, for a voting system we do not only want confidentiality of the vote against the "outside world" but also against the voting system itself.

Absolute privacy where no information is leaked to no one cannot coexist with verifiability where sufficient info is available to verify the result. There will always be some compromise. We must choose someone to trust, to assume someone to be trustworthy. Trust assumptions must be chosen to achieve high levels of privacy and verifiability, but still allow for an efficient and voter-friendly system. Voters might not agree upon which authorities are trustworthy. By using distribution of trust, multiple authorities perform crucial tasks together. By choosing enough heterogeneous authorities, collaboration is unlikely.

The Swiss Post voting system does not trust the voting server, but does trust that at least one of several return code components is honest. Therefore, the return code server shown in Figure 1 does in fact consist of several independent components controlled by independent authorities. Each return code component makes one return code share. Then, all the return code shares are combined by the voting server to compute the return codes. The Swiss Post voting system achieves individual verifiability if at least one of the return code components is honest. If one return code component is honest, it is not possible for other parts of the system to change or drop a vote while showing the right return codes to the voter. Even if the voting client, the voting server, and all but one control component are untrustworthy and try to cooperate to change or drop a vote, they are not able to do this unnoticed. The voter would notice because wrong return codes would be shown.

The voting phase of the Swiss Post voting system consists of two rounds. We remember from step 5 of Figure 1 that the voter verifies the return codes shown on the screen by comparing them with the corresponding return codes from the voting card. A second round of the voting phase lets the voters actively confirm their vote. If the return codes shown on the screen do not match those on the voting card, the voter can try again with another device or go to a physical voting place. If the return codes do match, the voter initiates the second round of the voting phase, called the ConfirmVote protocol. The ConfirmVote protocol is initiated by the voter by typing another key from the voting card. The ConfirmVote protocol precedes similarly to the SendVote protocol and is ended when the voter receives the confirmation return code that she can compare with the confirmation return code from her voting card. While the SendVote protocol assures that the voting client sends the vote as intended, the ConfirmVote protocol lets the voter actively confirm their vote and assures that the voting server records the vote as confirmed. This property is called recorded-as-confirmed individual verifiability.

After all voters have cast their votes, the encrypted votes must be mixed and then decrypted to compute the voting results. For this, we can use a mix-net. If the mix-net would take as input a list of encrypted votes, change the order of the list, and then output the new list, it would be easy to spot the permutation and break privacy. Thus, the output must look different than the input. Making this possible without risking that the mix-net changes votes in the process is a delicate task. We need the mix-net to be verifiable, that is, it must prove that the votes contained in the output are the same as in the input, although they look different.

Because of the desired properties and trust assumptions of the Swiss Post voting system, the ballot from step 2 of Figure 1 cannot only include the encrypted vote, it must include several components. First, the encrypted vote that will be the input to the mix-net. Second, some code that can be sent to the return code components so that they can compute the return codes without learning the vote itself. Third, the voter's identity so that the voting server can check that the voter is eligible to vote. Finally, we include proofs proving that the preferred voting options of the voter were in fact used both in the computation of the encrypted vote sent to the mix-net and in the computation of the codes sent to the return code components. To make the components of the ballot, different cryptographic building blocks are needed. We need encryption, commitments, and zero-knowledge proofs.

1.3 Cryptographic Building Blocks

A public key encryption scheme can be compared with a box with a key lock given to you by the decryptor. Only the decryptor has the secret key that can open the box. When you receive the box, it is open and empty. You can put a paper with a secret message in the box, and when you close the box, it automatically locks. Now you can send the box back to the decryptor. The decryptor can use the secret key to open the box and read the message.

A commitment scheme can also be compared with a box with a key lock. However, this time, you make the box yourself and only you have the secret randomness to open it. Now you can put a paper with a secret message in the box, lock it, and send it to a verifier, while keeping the randomness for yourself. Now the verifier is in possession your message but cannot open the box to read it because she does not have the randomness. However, as the verifier now possesses the box, you are not anymore able to change the message contained in it. Later, you could give the randomness to the verifier, and she can open the box and read the message. Both the fact that the verifier is not able to read the message before you give them the randomness and the fact that you are not able to change the message after you gave the box away, are important properties of the commitment scheme. We call these properties hiding and binding. We can think about a commitment as a one-time encryption of a given message. After you revealed or re-used the randomness, the commitment will not be hiding anymore.

Zero-Knowledge proofs, also called ZK-proofs, are proofs that do not reveal more than necessary about some secret message. If we combine such ZK-proof with a commitment box as above, we can prove properties about the secret message contained in the box without revealing the secret message itself. For example, we can prove that the secret message in the box is short. If we send two commitment boxes, a ZK-proof can prove that the secret messages contained fulfil some linear relation with each other. In the voting system, ZK-proofs can be used by the voting client to prove that the ballot was correctly computed, and be used by the return code servers to prove that the return codes were correctly computed.

1.4 Related Work

Our goal is to make the voting protocol of [Swi21] post-quantum secure. For this we need a voting protocol that (1) respects the security properties of [Swi21], in particular it should provide (1a) individual verifiability, (1b) universal verifiability and (1c) privacy. (2) These security properties should be assured with the given trust assumptions, in particular the protocol should allow for several control components for making return codes and mixing votes, where only one of these control components is assumed trustworthy. (3) Vote privacy should be preserved even against an attack using quantum computers in order to guarantee long-term privacy. (4) The protocol should be efficient so that it could be used in practice also in larger elections.

[Swi21] uses a so-called decryption mix-net, where the input ciphertexts are nested encryptions and each node in the mix-net is responsible for decrypting one layer of each ciphertext. Decryption mix-nets based on dlog-like assumptions can achieve high efficiency with strong security properties including universal verifiability, while existing efficient schemes for lattice-based primitives do not provide universal verifiability. In 2020, Boyen, Haines, and Müller [BHM20] presented the first verifiable and practical post-quantum mix-net with external auditing which can be used as a drop-in replacement of existing constructions. Their construction is a decryption mix-net using only lattice-based primitives, including nested BGV ciphertexts. Their system is quite efficient, though the nested BGV ciphertexts are quite costly in terms of communication size. They formally proved that their mix-net provides a high level of verifiability. However, their use of an active auditor that must be assumed to be honest during the mixing process conflicts with universal verifiability.

For making efficent schemes that provide universal verifiability, we need another approach. In the following schemes, the mixing and the decryption are separated from each other. The first fully post-quantum proof of a shuffle for Ring-LWE encryption schemes was presented by Costa, Pinilla, and Bosch [CMM19] in 2019. This shuffle was in 2021 used by Farzaliyev, Willemson, and Kaasik [FWK21] to construct a mix-net, using the amortization techniques by Attema, Lyubashevsky,

and Seiler [ALS20] for the commitment scheme by Baum, Damgård, Lyubashevsky, Oechsner, and Peikert [Bau+18].

Herranz, Martínez, and Sánchez [HMS21] in 2021 gave the first sub-linear post-quantum zero-knowledge argument for the correctness of a shuffle. However, the scheme is not implemented, the example parameters do not take the soundness slack of the amortized zero-knowledge proofs into account and their scheme does not consider decryption of ballots, which would make an actual implementation of the scheme less efficient.

In 2021, Aranha, Baum, Gjøsteen, Silde, and Tunge [Ara+21] proposed a verifiable shuffle for lattice-based commitments to known values. Their scheme is much more efficient in terms of communication size and communication timings than the scheme of [FWK21]. Their shuffle can be used to prove that a collection of commitments opens to a given collection of known messages, without revealing a correspondence between commitments and messages. Their scheme was the first construction from candidate post-quantum secure assumptions using return codes to defend against compromise of the voting client. However, their trust model assumes a single trusted voting server to ensure privacy of the ballots, which is too restrictive for use in [Swi21].

In 2022, Aranha, Baum, Gjøsteen, and Silde [Ara+22] proposed a verifiable secret shuffle for BGV ciphertexts as well as a verifiable distributed decryption protocol for it. The protocol allows for an untrustworthy voting server. The shuffle is based on an extension of the shuffle of commitments to known values which is combined with an amortized proof of correct re-randomization. The verifiable distributed decryption protocol uses noise drowning for BGV decryption and proves decryption correctness in zero-knowledge. They give concrete parameters for their system and estimate communication size and timings of their protocol. An implementation of all sub-protocols is provided. They demonstrate with a prototype voting protocol design that the shuffle and the decryption protocol are suitable for use in real-world cryptographic voting schemes. However, their prototype voting protocol does not describe how to make return codes. The return codes from [Ara+21] do not fit with this shuffle protocol because of the different trust assumptions and because the voting phase of [Ara+21] gives commitments as input to the mix-net, while the mix-net of [Ara+22] requires BGV ciphertexts as input.

1.4.1 Our Contribution

We present a lattice-based voting phase suitable for electronic voting with return codes, extending the framework by Aranha et al. [Ara+21; Ara+22]. Like [Swi21], our return code-based voting protocol does not assume a trustworthy voting server but does assume that at least one so-called control component is trustworthy. With the given trust assumptions, our protocol achieves sent-as-intended integrity and privacy.

1.5 Outline

The first sections of the report present background material needed to build an electronic voting protocol. Some preliminaries are presented in Section 2. Hard problems that can be used to assure the security of cryptographic building blocks are presented in Section 3. Sections 4, 5, 6 present cryptographic building blocks needed in a voting system. Encryption schemes are presented in Section 4, commitment schemes in Section 5 and zero-knowledge proofs in Section 6. For each building block, we present both dlog-based and lattice-based schemes. Desired properties of a voting protocol are presented in Section 7.

The Swiss Post voting protocol [Swi21] from 2021 is presented in Section 8. We present syntax and the two-round return code-based voting protocol and we discuss the security of the protocol. The lattice-based voting protocol by Aranha et al. [Ara+21] from 2021 is presented in Section 9. We present the one-round return code-based voting protocol and discuss why this protocol cannot be used directly to make return-codes for the shuffle by Aranha et al. [Ara+22] from 2022.

Our voting protocol is presented in Section 10. We present the two-round return code-based voting

protocol and compare it with the protocols of [Swi21] and [Ara+21]. We argue why our protocol can be used to make return-codes for the shuffle by Aranha et al. [Ara+22] from 2022. We give an informal security analysis. In Section 11, we give concrete communication sizes and timings of our protocol and discuss its performance. Some final concluding remarks are given in Section 12.

2 Preliminaries

We define the rings R_q and R_p, the norms of elements and vectors, and we explain what we mean by "short" elements.

Let N be a power of 2 and $p \ll q$ primes. Let $R_q = \mathbb{Z}_q[X]/\langle X^N + 1\rangle$ and $R_p = \mathbb{Z}_p[X]/\langle X^N + 1\rangle$ be rings of polynomials modulo $X^N + 1$ with integer coefficients modulo q and p respectively.

We define the norms of elements $f(X) = \sum \alpha_i X^i \in R_q$ to be the norms of the coefficient vector as a vector in $\mathbb{Z}_q^N$

$$||f||_1 = \sum |\alpha_i|, \quad ||f||_2 = (\sum \alpha_i^2)^{1/2}, \quad ||f||_\infty = \max_{i\in[1,\dots,n]}\{|\alpha_i|\}$$

For an element $\bar{f} \in R_q$, we choose coefficients as the representatives in $[-\frac{q-1}{2}, \frac{q-1}{2}]$, and then we compute the norms as if $\bar{f}$ is an element in R. For vectors $a = (a_1, \dots, , a_k) \in R^k$ we define the norms ℓ_1, ℓ_2 and ℓ_∞ as

$$||a||_1 = \sum ||a_i||_1, \quad ||a||_2 = (\sum ||a_i||_2^2)^{1/2}, \quad ||a||_\infty = \max_{i\in[1,\dots,n]}\{||a_i||_\infty\}$$

For any positive integer β, we define the set $[\beta] = \{-\beta, \dots - 1, 0, 1, \dots, \beta\}$. We extend the notation to N-dimensional polynomials. We mean by short polynomial in $[\beta]$ a polynomial with N integer coefficients that all have maximum absolute value β. Then we extend the notation to polynomial vectors. We mean by short polynomial vector in $[\beta]^k$ a polynomial vector consisting of k polynomials each with N integer coefficients that each have maximum absolute value β.

3 Hard Problems

Cryptographic protocols can base their security on problems that are easy in one direction, while hard in the other direction. We briefly present a few such problems. First, we present the discrete logarithm problem (Dlog) and a few other dlog-based problems: the computational Diffie-Hellman problem (CDH), the decisional Diffie-Hellman problem (DDH), and the extended subgroup generated by small primes problem (ESGSP). Then we present a few lattice-based problems: the Short Integer Solution problem (SIS), which is a computational problem, then the decisional version of the Learning With Errors problem (LWE), and the decisional version of the Learning With Rounding problem (LWR).

3.1 Dlog-Based Problems

Let q be a prime and let $\mathbb{G}_q$ be a multiplicative cyclic group of order q with generator g.

The Discrete Logarithm Problem (Dlog) Given $(g, y \in \mathbb{G}_q)$, the dlog problem asks to compute a such that $g^a = y$.

The Computational Diffie Hellman Problem (CDH) Given (g, g^a, g^b) where $a, b \xleftarrow{\$} \mathbb{Z}_q$, the CDH problem asks to compute g^{ab}.

The Decisional Diffie-Hellman Problem (DDH) Given (g, g^a, g^b) where $a, b \xleftarrow{\$} \mathbb{Z}_q$, the DDH problem asks to distinguish g^{ab} from a random element of $\mathbb{G}_q$.

Let q be a prime such that $p = 2q + 1$ is also a prime, then the quadratic residues of the finite field with p elements is a finite cyclic group $\mathbb{Q}_p$ of prime order q. Let $l_1 \in \mathbb{Q}_p$ be the group element corresponding to the smallest prime that is a quadratic residue modulo p.

The Extended Subgroup Generated by Small Primes Problem (ESGSP) Given (g^s, g_1, l_1) where $s \xleftarrow{\$} \mathbb{Z}_q$, $g_1 \xleftarrow{\$} \mathbb{Q}_p$, the ESGSP problem asks to distinguish (g_1^s, l_1^s) from random elements of $\mathbb{Q}_p$.

Assuming the ESGSP problem is hard is a similar but stronger assumption than assuming that the DDH problem is hard. While for the DDH problem the base element is known, for the ESGSP problem we use the same exponent several times for different bases and we want to hide not only the exponent, but also the base.

3.2 Lattice-Based Problems

Let N be a power of 2 and q a prime. Let $R_q = \mathbb{Z}_q[X]/\langle X^N + 1\rangle$.

The Short Integer Solution Problem (SIS) Given $A \in R_q^{n \times m}$ where $m \geq 2n$ and $t \in R_q^n$, the SIS problem asks to compute short $s \in [\beta]^m$ so that $As = t$.

The Decisional Learning With Error Problem (LWE) Given $A \in R_q^{n \times m}$, the LWE problem asks to distinguish $As + e$, where $s \in [\beta]^m$ and $e \in [\beta]^n$, from a random polynomial vector from the ring.

LWE can also be defined with $s \in R_q^m$, and the presented definition is a special case.

Let p be a prime so that $p \ll q$. Divide the elements of R_q into p contiguous intervals of roughly q/p elements each. Define the rounding function $\lfloor \cdot \rceil : R_q \to R_p$ that maps $x \in R_q$ into the index of the interval that x belongs to in R_p. Let the rounding function be extended to vectors by applying it componentwise.

The Decisional Learning With Rounding Problem (LWR) Given $A \in R_q^{n \times m}$, the LWR problem asks to distinguish $\lfloor As \rceil$, where $s \in [\beta]^m$, from a random polynomial vector from the ring.

LWR can also be defined with $s \in R_q^m$, and the presented definition is a special case.

Knapsack Problems The Search Knapsack Problem is essentially the module SIS problem. The Decisional Knapsack Problem is essentially the module LWE problem.

4 Public Key Encryption

Let λ be a security parameter. A public-key encryption scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is a triple of efficient algorithms.

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\lambda)$ The key generation algorithm is a probabilistic algorithm that on input a security parameter λ, outputs the public/private key pair $(\mathsf{pk}, \mathsf{sk})$.
- $c \leftarrow \mathsf{Enc}(m, \mathsf{pk})$ The encryption algorithm is a probabilistic algorithm that on input the public key pk and a message m, outputs a ciphertext c.
- $m \leftarrow \mathsf{Dec}(c, \mathsf{sk})$ The decryption algorithm is a deterministic algorithm that on input the secret key sk and a ciphertext c, outputs a message m.

Messages are assumed to lie in some finite message space $\mathcal{M}$ and ciphertexts in some finite ciphertext space $\mathcal{C}$. We say that $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is defined over $(\mathcal{M}, \mathcal{C})$.

4.1 Properties

4.1.1 Correctness

Perfect correctness of an encryption scheme implies that decryption undoes encryption. That is, for all honestly generated keys $(\mathsf{pk}, \mathsf{sk})$ from KeyGen and for all messages m, we have the relation $\mathsf{Dec}(\mathsf{Enc}(m, \mathsf{pk}), \mathsf{sk}) = m$.

Not all encryption schemes have perfect correctness. Decryption does not perfectly undo encryption. This is the case if the encryption algorithm includes some noise in the ciphertext. We can define correctness less strictly and say that the encryption scheme is correct if decryption with overwhelming probability undoes encryption. That is, given honestly generated keys $(\mathsf{pk}, \mathsf{sk})$ from KeyGen, then we have $\Pr[\mathsf{Dec}(\mathsf{Enc}(\mathsf{pk}, m), \mathsf{sk}) = m] \geq 1 - \epsilon(\lambda)$.

We will in this thesis only describe encryption schemes with homomorphic properties. Additive homomorphic decryption implies that we can add two or more ciphertexts that are encrypted with the same public key and then decrypt the sum with the secret key, and get the same result as if we had first decrypted each ciphertext with the secret key and then taken the sum of the messages. For slightly homomorphic ciphertexts, only a limited number of ciphertexts can be added until the noise will be so large that the ciphertext cannot be decrypted anymore. Multiplicative homomorphic decryption is similar, with multiplication instead of adding.

In Section 4.3 we will define BGV ciphertexts and see that they are only slightly additive homomorphic. In Section 10 we will need the possibility to send BGV ciphertexts made in the voting protocol through a mix-net that adds encryptions of zero to the encrypted vote. This will require that a limited number of ciphertexts can be added and still be decrypted correctly. Therefore we define τ-correctness. The definition is from [Ara+22, Section 2.4].

Definition 1 (τ-Correctness). We say that $\mathcal{E}$ is τ-correct if the sum of τ honestly generated ciphertexts with overwhelming probability decrypts to the sum of the τ encrypted messages. Given keys $(\mathsf{pk}, \mathsf{sk})$ honestly generated from $\mathsf{KeyGen}(\lambda)$, and ciphertexts $\{c_i\}_{i\in[\tau]}$ honestly computed from $\mathsf{Enc}(\mathsf{pk}, \{m_i\}_{i\in[\tau]})$. Then

$$\Pr\left[\mathsf{Dec}(\sum_{i\in[\tau]} c_i, \mathsf{sk}) = \sum_{i\in[\tau]} m_i\right] \geq 1 - \epsilon(\lambda)$$

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

4.1.2 Chosen Plaintext Security

The following definition of security against chosen plaintext attacks (CPA-security) is from [Ara+22, Section 2.4].

Definition 2 (CPA-Security). We say that the public key encryption scheme is CPA-secure if an efficient adversary $\mathcal{A}$, after choosing two messages m_0 and m_1 and receiving an encryption c of either m_0 or m_1 (chosen at random), cannot distinguish which message the ciphertext c is an encryption of. Given keys $(\mathsf{pk}, \mathsf{sk})$ honestly generated from $\mathsf{KeyGen}(\lambda)$, the adversary computes the messages $(m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pk})$, receives a ciphertext $c \leftarrow \mathsf{Enc}(m_b, \mathsf{pk})$ where $b \xleftarrow{\$} \{0, 1\}$ and outputs a bit $b' \leftarrow \mathcal{A}(c, \mathsf{st})$. Then

$$|\Pr[b = b'] - \frac{1}{2}| \leq \epsilon(\lambda)$$

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

There exist stronger security notions of encryption schemes than CPA-security, for example security against chosen ciphertext attacks (CCA-security). However, CCA-secure schemes do not have

homomorphic properties. Homomorphic properties will be important when designing a voting system, and we will therefore not consider stronger security notions.

4.2 ElGamal Encryption

ElGamal [ElG85] is a public-key encryption scheme whose security is based on the DDH problem from Section 3. Let $\mathbb{G}_q$ be a cyclic group of order q with generator g, where q is a large prime. The ElGamal encryption scheme consists of the following algorithms:

- $\mathsf{KeyGen}(\lambda)$ The key generation algorithm takes as input a security parameter λ. It samples a random $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_q$ and computes $\mathsf{pk} = g^{\mathsf{sk}} \in \mathbb{G}_q$. It outputs the public/private key pair $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}(m, \mathsf{pk})$ The encryption algorithm takes as input a message $m \in \mathbb{G}_q$ and the public key $\mathsf{pk} \in \mathbb{G}_q$. It samples a random $r \xleftarrow{\$} \mathbb{Z}_q$ and computes
$$c = (u, v) = (g^r, \mathsf{pk}^r \cdot m) \in \mathbb{G}_q^2. \tag{1}$$
It outputs the ciphertext c.
- $\mathsf{Dec}(c, \mathsf{sk})$ The decryption algorithm takes as input the ciphertext $c = (u, v) \in \mathbb{G}_q^2$ and the private key $\mathsf{sk} \in \mathbb{Z}_q$. It computes $m = v \cdot u^{-\mathsf{sk}}$. It outputs the message m.

We observe that the ElGamal encryption scheme has perfect correctness. For any key pair generated by the key generation algorithm, it holds that $\mathsf{Dec}(\mathsf{Enc}(m, \mathsf{pk}), \mathsf{sk}) = m$ for all $m \in \mathbb{G}_q$.

The ciphertexts are multiplicatively homomorphic. We can multiply two or more ciphertexts that are encrypted with the same public key and then decrypt the product with the secret key, and get the same result as if we had first decrypted each ciphertext with the secret key and then taken the product of the messages. For two ElGamal ciphertexts c and c' we have the relation $c \cdot c' = \mathsf{Enc}(m, \mathsf{pk}; r) \cdot \mathsf{Enc}(m', \mathsf{pk}; r') = (g^r, \mathsf{pk}^r \cdot m) \cdot (g^{r'}, \mathsf{pk}^{r'} \cdot m') = (g^{(r+r')}, \mathsf{pk}^{(r+r')} \cdot (m \cdot m')) = \mathsf{Enc}(m'', \mathsf{sk}; r'') = c''$ for $m'' = (m \cdot m')$ and $r'' = (r + r')$.

An attacker against CPA-security of the ElGamal encryption scheme is an attacker against the DDH assumption.

Multi-Recipient ElGamal When encrypting messages for multiple recipients with different public keys, we can share the encryption randomness r. The multi-recipient ElGamal encryption scheme [BBS03] uses the same key generation algorithm, but instead of the encryption and decryption algorithms it has a multi-encrypt and a multi-decrypt algorithm.

- $\mathsf{MultiEnc}(\mathbf{m}, \mathbf{pk})$ The encryption algorithm takes as input messages $\mathbf{m} \in \mathbb{G}_q^l$ and a vector of public keys $\mathbf{pk} \in \mathbb{G}_q^k$. It samples a random $r \xleftarrow{\$} \mathbb{Z}_q$. If $l > k$, there are more messages than public keys, and the algorithm outputs an error. Otherwise, if $l = k$, there are as many messages as public keys. Then the algorithm computes
$$\mathbf{c} = (u, v_1, \ldots, v_l) = (g^r, \mathsf{pk}_1^r \cdot m_1, \ldots, \mathsf{pk}_k^r \cdot m_l) \in \mathbb{G}_q^{l+1} \tag{2}$$
Otherwise, if $l < k$, there are less messages than public keys. Then the algorithm computes
$$\mathbf{c} = (u, v_1, \ldots, v_{l-1}, v_l) = (g^r, \mathsf{pk}_1^r, \cdot m_1, \ldots, \mathsf{pk}_{l-1}^r \cdot m_{l-1}, (\mathsf{pk}_l \cdot \mathsf{pk}_{l+1} \cdot \cdots \cdot \mathsf{pk}_k)^r \cdot m_l) \in \mathbb{G}_q^{l+1} \tag{3}$$
The algorithm outputs the ciphertexts $\mathbf{c} \in \mathbb{G}_q^{l+1}$.
- $\mathsf{MultiDec}(\mathbf{c}, \mathbf{sk})$ The decryption algorithm takes as input the ciphertexts $\mathbf{c} \in \mathbb{G}_q^{l+1}$ and a vector of private keys $\mathbf{sk} \in (\mathbb{Z}_q)^k$. If $l > k$, there are more ciphertexts than private keys, and the algorithm outputs an error. Otherwise, the messages are computed so that $m_i = v_i \cdot u^{-\mathsf{sk}_i} \in \mathbb{G}_q$. If $l < k$, there are less ciphertexts than secret keys, and the last message m_l requires a compression of the secret keys such that $m_l = v_l \cdot u^{-(\mathsf{sk}_{l-1} + \mathsf{sk}_l + \cdots + \mathsf{sk}_k)} \in \mathbb{G}_q$. The algorithm outputs the messages $\mathbf{m} \in \mathbb{G}_q^l$.

4.3 BGV Encryption

We recall from Section 2: Let N be a power of 2 and $p \ll q$ primes. Let $R_q = \mathbb{Z}_q[X]/\langle X^N + 1\rangle$ and $R_p = \mathbb{Z}_p[X]/\langle X^N + 1\rangle$ be rings of polynomials modulo $X^N + 1$ with integer coefficients modulo q and p respectively.

We mean by short $e \in [\beta]$ a polynomial with integer coefficients that all have maximum absolute value β. There is a natural mapping from messages in $\mathbb{Z}_p^N$ to polynomials in R_q with coefficients modulo p.

The BGV encryption scheme [BGV14], named after Brakerski, Gentry, and Vaikuntanathan, consists of the following algorithms:

- $\mathsf{KeyGen}(\lambda)$ The key generation algorithm takes as input a security parameter λ. It samples $a \xleftarrow{\$} R_q$ and short $s \xleftarrow{\$} [\beta]$ and short noise $e \xleftarrow{\$} [\beta]$. It sets $\mathsf{sk} := (s, e)$ and computes $\mathsf{pk} = (a, b) = (a, as + pe)$. It outputs the public/private key pair $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}(m, \mathsf{pk})$ The encryption algorithm takes as input a message $m \in \mathbb{Z}_p^N$ and the public key pk. It samples short $r \xleftarrow{\$} [\beta]$ and short noise $e', e'' \xleftarrow{\$} [\beta]$ and computes
$$c = (u, v) = (ar + pe', br + pe'' + m) \tag{4}$$
It outputs the ciphertext c.
- $\mathsf{Dec}(c, \mathsf{sk})$ The decryption algorithm takes as input the ciphertext $c = (u, v)$ and the private key $\mathsf{sk} \in \mathbb{Z}_q$. It computes $m = (v - s \cdot u \mod q) \mod p$. It outputs the message m.

BGV ciphertexts are slightly additively homomorphic. That is, only a limited number of ciphertexts can be added until the noise will be so large that the ciphertext cannot be decrypted anymore. The BGV encryption scheme scheme is τ-correct for a limited number of ciphertexts τ if $p \ll q$, and β is sufficiently small so that the noise values are sufficiently small. More precisely, we must have $\tau||v - su||_\infty < \lfloor q/2 \rceil$. An attacker against CPA-security of the scheme is an attacker against the LWE problem [BGV14, Section 5.5]. As the scheme encrypts polynomials, a multi-encrypt version like we have seen for ElGamal in Section 4.2, is not necessary.

5 Commitments

Commitment schemes were first introduced by Blum [Blu83]. A commitment scheme is a protocol with two participants: The prover and the verifier. The commitment scheme consists of two phases, the commitment phase, and the opening phase. In the commitment phase, the prover commits to a value while hiding the actual value to the verifier. In the opening phase, the prover reveals which value she committed to. A commitment scheme is designed to be binding and hiding. Binding means that in the opening phase, the prover cannot reveal another value than the value committed to in the commitment phase. Hiding means that in the commitment phase, the verifier learns nothing about the hidden value.

Let λ be a security parameter. A commitment scheme consists of the following three algorithms: $\mathsf{KeyGen}, \mathsf{Com}$ and Open.

- $\mathsf{pp} \leftarrow \mathsf{KeyGen}(\lambda)$ The key generation algorithm is a probabilistic algorithm that on input the security parameter λ outputs the public parameters pp containing a definition of the message space $\mathcal{M}$.
- $[\![m]\!], r \leftarrow \mathsf{Com}(\mathsf{pp}, m)$ The commit algorithm is a probabilistic algorithm that on input the public parameters pp and a message $m \in \mathcal{M}$ outputs a commitment $[\![m]\!]$ and opening r.
- $b \leftarrow \mathsf{Open}(\mathsf{pp}, m, [\![m]\!], r)$ The open algorithm is a deterministic algorithm that on input pp, m, $[\![m]\!], r$, outputs a bit $b \in \{0, 1\}$.

Often the notation c is used for a commitment. We here use the notation $[\![m]\!]$ for a commitment, to clearly distinguish a commitment from a ciphertext. The notation we chose has the disadvantage that it seems that there is only one possible opening m of a commitment $[\![m]\!]$. As we will see in Definition 5, this is not necessarily the case.

Figure 2 shows how a prover $\mathcal{P}$ communicates with a verifier $\mathcal{V}$ in a commitment scheme. The public parameters pp can be generated by $\mathcal{P}$ or $\mathcal{V}$ before the communication begins. In some schemes it will be necessary for the party who generated the public parameters to convince the other party that pp were correctly generated with with KeyGen.

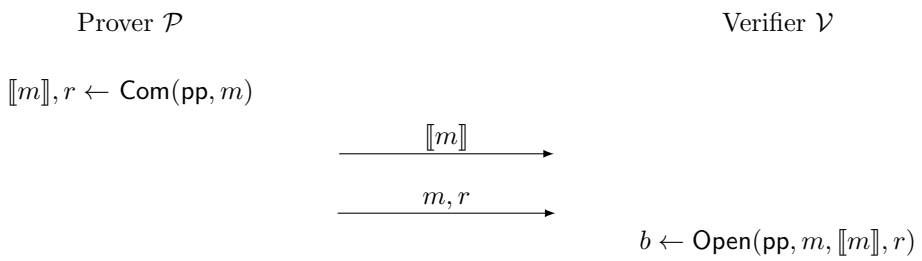


Figure 2: The public parameters pp are known to both the prover and the verifier. A message $m \in \mathcal{M}$ is input to the prover. The commitment scheme consists of two phases. In the commitment phase, the commitment $[\![m]\!]$ is sent from the prover to the verifier. In the opening phase, the message m and opening r are sent from the prover to the verifier, which verifies that m is indeed the opening of $[\![m]\!]$ given r.

5.1 Properties

The following definitions of completeness, hiding and binding are from [Ara+22, Section 2.6].

Definition 3 (Completeness). We say that the commitment scheme is complete if an honestly generated commitment is accepted by the opening algorithm. Given pp honestly generated with $\mathsf{KeyGen}(\lambda)$ and $[\![m]\!]$, r honestly generated with $\mathsf{Com}(\mathsf{pp}, m)$. Then,

$$\Pr\left[\mathsf{Open}(m, [\![m]\!], r) = 1\right] = 1$$

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

Definition 4 (Hiding). We say that a commitment scheme is hiding if an adversary $\mathcal{A}$, after choosing two messages m_0 and m_1 and receiving a commitment $[\![m_b]\!]$ to either m_0 or m_1 (chosen at random), cannot distinguish which message $[\![m_b]\!]$ is a commitment to. This is equivalent to that the adversary cannot distinguish between a commitment of a message m of her choice and a uniformly-random element in the space of commitments. Given parameters pp honestly generated from $\mathsf{KeyGen}(\lambda)$, the adversary computes the messages $(m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$, receives a commitment $[\![m_b]\!] \leftarrow \mathsf{Com}(m_b)$ where $b \xleftarrow{\$} \{0, 1\}$ and outputs a bit $b' \leftarrow \mathcal{A}([\![m_b]\!], \mathsf{st})$. Then,

$$|\Pr[b = b'] - \frac{1}{2}| \leq \epsilon(\lambda)$$

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

Definition 5 (Binding). We say that a commitment scheme is binding if an adversary $\mathcal{A}$, after creating a commitment $[\![m]\!]$, cannot find two valid openings to $[\![m]\!]$ for different messages m and m'. Given parameters pp honestly generated from $\mathsf{KeyGen}(\lambda)$, the adversary computes $([\![m]\!], m, r, m', r') \leftarrow \mathcal{A}(\mathsf{pp})$. Then,

$$\Pr\left[\begin{array}{c} m \neq m' \\ \mathsf{Open}(m, [\![m]\!], r) = 1 \\ \mathsf{Open}(m', [\![m]\!], r') = 1 \end{array}\right] \leq \epsilon(\lambda)$$

where the probability is taken over the random coins of KeyGen.

The properties hiding and binding are illustrated in Figure 3 and Figure 4. It is not possible to get both properties binding and hiding against computationally unbounded adversaries at the same time. We can get either hiding or binding against computationally unbounded adversaries. The other property will be against bounded-complexity adversaries, under assumptions on the primitives of the construction.

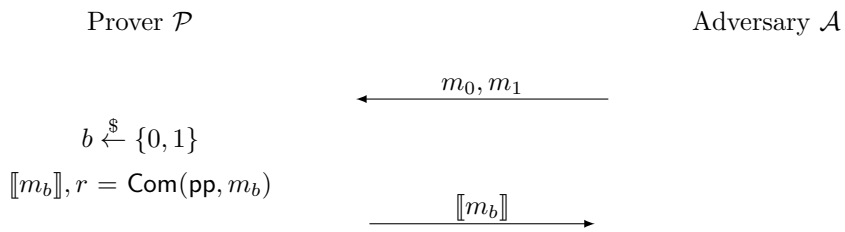


Figure 3: For an attack against the hiding property, the verifier is the adversary. The adversary sends messages m_0, m_1 to the prover. The prover flips a coin and depending on the outcome the prover commits to one of the messages. Then $[\![m_b]\!]$ is sent to the adversary. If the adversary is not able to tell if $[\![m_b]\!]$ is the commitment to message m_0 or m_1, the scheme is hiding.

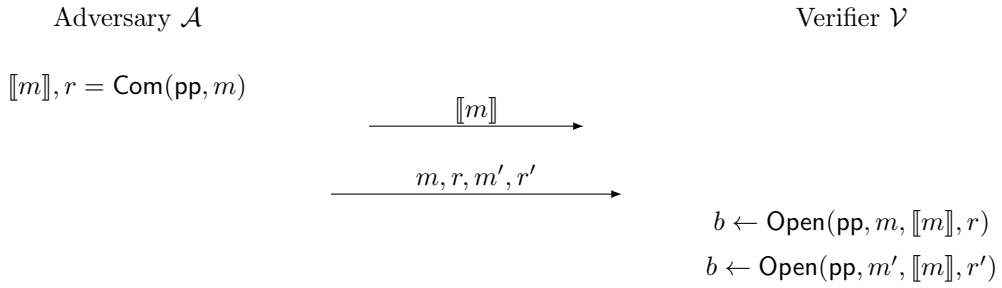


Figure 4: For an attack against the binding property, the prover is the adversary. In the commitment phase, the adversary sends a commitment $[\![m]\!]$ to the verifier. In the opening phase, the adversary sends two messages $m \neq m'$. If the verifier accepts both messages, then the binding property is broken.

Homomorphic Properties For a homomorphic commitment scheme we have for two commitments $\mathsf{Com}(m, r)$ and $\mathsf{Com}(m', r')$ the relation $\mathsf{Com}(m, r) \oplus \mathsf{Com}(m', r') = \mathsf{Com}(m'', r'')$ where there is some simple relation between the messages m, m' and m'' and some simple relation between the randomnesses r, r' and r''. If $m'' = m + m$, we say that the commitment scheme is additive homomorphic. If $m'' = m \cdot m$, we say that the commitment scheme is multiplicative homomorphic. The commitment schemes we present here are either additive homomorphic or multiplicative homomorphic. The commitment schemes we present here all have $r'' = r + r'$ and $\oplus$ is either addition or multiplication.

5.2 Dlog-Based Commitments

A discrete logarithm instance at its own is not a commitment scheme. Let $c = g^m \mod q$. This scheme would be perfectly binding, following from the fact that there is one unique solution for $m = \log_g c \mod q$. To find this discrete logarithm is hard, still the scheme would not be hiding.

An adversary could attack the hiding property like in Figure 3 by sending their message m of choice to the prover and then check if the returned commitment c satisfies $g^m \mod q = c$. A discrete logarithm instance by itself is not a commitment scheme, still we can use the discrete logarithm problem to construct commitment schemes. In the following we present two well-known commitment schemes with their security based on the Diffie-Hellman assumptions from Section 3.

ElGamal Commitments We recall the ElGamal encryption scheme of Section 4.2, with encryption algorithm $\mathsf{Enc}(m, \mathsf{pk}) = c = (u, v) = (g^r, \mathsf{pk}^r \cdot m)$ over $\mathbb{G}_q$. This scheme is also a commitment scheme. If a prover reveals randomness r and message m to the verifier, the verifier can verify the opening without learning the secret key $\mathsf{sk} = \log_g \mathsf{pk}$.

The first component u makes the ElGamal commitment unconditionally binding. Given u and g there exists a unique solution of $r = \log_g u \mod q$. This locks the prover to this specific r. Thus, m is also locked to a specific value and we have unconditionally binding of the message. The second component v hides the message m. r is chosen uniformly random, thus pk^r is uniformly random, thus $\mathsf{pk}^r m$ is also uniformly random. An adversary attacking hiding could try to break the dlog-assumption by computing the discrete logarithm $r = \log_g u$, and break hiding by computing the message $m = v(\mathsf{pk}^r)^{-1}$. There could be better attacks against DDH, and so the ElGamal scheme is computationally hiding based on the DDH assumption. The ElGamal commitment scheme is obviously not hiding against those knowing the private key sk.

An ElGamal commitment can be opened by only publishing the randomness r, and not the message m. The verifier can compute the message as $m = v \cdot (\mathsf{pk}^r)^{-1}$ and verify by checking $u = g^r$.

ElGamal commitments are multiplicatively homomorphic. Multiplying a commitment to message m and randomness r with a commitment to message m' and randomness r' results in a commitment to message $m'' = m \cdot m'$ with randomness $r'' = r + r'$. We have the relation $\mathsf{Com}(m, r) \cdot \mathsf{Com}(m', r') = (g^r, \mathsf{pk}^r \cdot m) \cdot (g^{r'}, \mathsf{pk}^{r'} \cdot m') = (g^{(r+r')}, \mathsf{pk}^{(r+r')} \cdot (m \cdot m')) = \mathsf{Com}(m'', r'')$, where $m'' = m \cdot m'$ and $r'' = r + r'$.

Pedersen Commitments Another widely used commitment scheme was proposed by Pedersen [Ped91]. Let $\mathbb{G}_q$ be a group of prime order q, with multiplicative generators $g, h \xleftarrow{\$} \mathbb{Z}_q$. A Pedersen commitment is computed as $[\![m]\!] = g^m h^r$ where $r \xleftarrow{\$} \mathbb{Z}_q$.

To break binding, an adversary must find randomness r' satisfying the relation $[\![m]\!] = g^{m'} h^{r'}$ for a message $m' \neq m$. An adversary against binding of the scheme is an adversary against the dlog-assumption. An adversary trying to find a randomness to break binding must compute the discrete logarithm instance $r' = \log_h([\![m]\!](g^{m'})^{-1})$. The Pedersen scheme is unconditionally hiding. When r is chosen uniformly random, then also h^r is uniformly random and thus also $[\![m]\!]$ is uniformly random.

Pedersen commitments are additively homomorphic under multiplication. Multiplying a commitment to message m with randomness r with a commitment to message m' with randomness r' results in a commitment to message $(m + m')$ with randomness $(r + r')$. We have the relation $\mathsf{Com}(m, r) \cdot \mathsf{Com}(m', r') = g^m h^r \cdot g^{m'} h^{r'} = g^{(m+m')} h^{(r+r')} = \mathsf{Com}(m'', r'')$, where $m'' = m + m'$ and $r'' = r + r'$.

It is important that the public parameters g and h are chosen uniformly random, to avoid that the prover constructs a backdoor to break the binding property. In particular the prover should not be allowed to choose g and h so that she learns $\log_g h$. In other words, h should not equal g^s for some s known by the prover. If so, the prover could easily break the binding by computing another message $m' = m + s(r - r')$ that corresponds to the same commitment c as message m does. If it is the prover who generates the public parameters, the prover must convince the verifier that they were correctly generated so that no such backdoor could be made.

We observe that the Pedersen commitment scheme cannot be used as an encryption scheme. Also, for opening a commitment the message m must be published together with the randomness r. Only publishing the randomness r is not enough to open the message m.

The Pedersen commitment scheme is computationally binding and unconditionally hiding. The unconditional hiding cannot be attacked, even by a future quantum computer. Assuming there exists no quantum computer that can attack the computational binding, using Pedersen commitments for a voting system could provide both integrity today and privacy in the future. However, for a voting system we would still need computationally hiding primitives in other parts of the system. We need ElGamal ciphetexts for the mix-net, and Chaum-Pedersen ZK-proofs for proving relations between commitments. Both these primitives are only computationally hiding based on dlog assumptions and so the system as a whole would still not provide long-term privacy against quantum computers.

5.3 Lattice-Based Commitments

Baum, Damgård, Lyubashevsky, Oechsner, and Peikert [Bau+18] presented in 2018 an additively homomorphic commitment scheme based on structured lattice assumptions, together with a zero-knowledge proof of opening knowledge. The scheme is at present more efficient than all other lattice-based commitment schemes.

We recall from Section 2: Let N be a power of 2 and q a prime. Let $R_q = \mathbb{Z}_q[X]/\langle X^N + 1\rangle$ be a ring of polynomials modulo $X^N + 1$ with integer coefficients modulo q. We mean by short $v \in [\beta]^k$ a polynomial vector consisting of k polynomials with integer coefficients that all have maximum absolute value β.

Let λ be a security parameter. Let the challenge space $\mathcal{C} = \{c \in [1], \|c\|_1 = \nu\}$ be the set consisting of all polynomials in R_q with exactly ν non-zero coefficients taken from the set $\{-1, 1\}$. ν is chosen so that $\mathcal{C}$ has at least 2^λ elements. Thus we choose ν such that $2^\nu \cdot \binom{N}{\nu} > 2^\lambda$. Assume N is large enough for such a ν to exist. We define the set of differences $\bar{\mathcal{C}} = \{c - c' | c \neq c' \in \mathcal{C}\}$.

Let $I_k \in R^{k\times k}$ be the identity matrix of dimension k over R. σ is a standard deviation for Gaussian sampled polynomials over R_q.

We commit to messages $m \in R_q^\ell$. The three algorithms KeyGen, Com and Open are given as follows.

- $\mathsf{KeyGen}(\lambda)$ Create public parameters that can be used to commit to messages $m \in R_q^\ell$. Create the following matrices $A_1 \in R_q^{n\times k}$ and $A_2 \in R_q^{\ell\times k}$ as

$$A_1 = \left[\begin{array}{cc} I_n & A_1' \end{array}\right], \quad \text{where } A_1' \xleftarrow{\$} R_q^{n\times(k-n)}$$

$$A_2 = \left[\begin{array}{ccc} 0^{\ell\times n} & I_\ell & A_2' \end{array}\right], \quad \text{where } A_2' \xleftarrow{\$} R_q^{\ell\times(k-n-\ell)}$$

  The output is the public matrices A_1 and A_2.

- $\mathsf{Com}(\mathsf{pp}, m)$ To commit to messages $m \in R_q^\ell$, choose randomly a short polynomial opening vector $r \xleftarrow{\$} [\beta]^k$ and output the commitment

$$[\![m]\!] = \mathsf{Com}(m, r) = (c_1, c_2) = (A_1 r, A_2 r + m) \tag{5}$$

- $\mathsf{Open}(\mathsf{pp}, m, [\![m]\!], r.f)$ A valid opening of a commitment $[\![m]\!]$ is a 3-tuple (m, r, f), where $m \in R_q^\ell, r \in [\beta]^k$, and $f \in \bar{\mathcal{C}}$. The verifier checks that for all $i, \|r_i\|_2 \leq 4\sigma\sqrt{N}$, and that

$$f \cdot \left[\begin{array}{c} c_1 \\ c_2 \end{array}\right] = \left[\begin{array}{c} A_1 \\ A_2 \end{array}\right] \cdot r + f \cdot \left[\begin{array}{c} 0^n \\ m \end{array}\right] \tag{6}$$

We observe that the commitment opening consists not only of opening randomness r and message m, but also includes a polynomial $f \in \bar{\mathcal{C}}$. This is because of the zero-knowledge proofs that will be presented in Section 6.2.2. An (honest) prover that wants to simply open a commitment can output (r, m) from (5) and $f = 1$.

An attacker against the binding property of the commitment scheme is an attacker against the Search Knapsack Problem and an attacker against the hiding property of the scheme is an attacker against the Decisional Knapsack Problem [Bau+18, Section 4.2].

The commitments are additively homomorphic. Adding a commitment to message m and randomness r with a commitment to message m' and randomness r' results in a commitment to message $(m + m')$ with randomness $(r + r')$. We have the relation $\mathsf{Com}(m, r) + \mathsf{Com}(m', r') = (A_1 r, A_2 r + m) + (A_1 r', A_2 r' + m') = (A_1(r + r'), A_2(r + r') + (m + m')) = \mathsf{Com}(m'', r'')$, where $m'' = m + m'$ and $r'' = r + r'$.

5.4 Comparison of Commitment Schemes

In the following we compare the lattice-based commitments by Baum et al. with the two dlog-based commitment schemes by ElGamal and Pedersen and comment on some of the common properties. The three commitment schemes are shown in Table 1.

	Commitment	Binding	Hiding
ElGamal	$[\![m]\!] = (c_1, c_2) = (g^r, \mathsf{pk}^r m)$	✓	DDH
Pedersen	$[\![m]\!] = g^m h^r$	Dlog	✓
Baum	$[\![m]\!] = (c_1, c_2) = (A_1 r, A_2 r + m)$	SIS	LWE

Table 1: Here, m is the message, r is the randomness and $[\![m]\!]$ is the commitment. Public values g, h, A_1 and A_2 are chosen uniformly random. pk is a public key for which only the decryptor knows the secret key $\mathsf{sk} = \log_g \mathsf{pk}$. The checkmark ✓ implies unconditional.

The commitments of all three schemes are homomorphic. Commitment schemes do not necessarily have algebraic properties, for example commitment schemes based on hash functions. We are for the purposes of this thesis only interested in homomorphic schemes and will only describe these.

Just like for ElGamal commitments, the first component (c_1) of a Baum commitment provides binding, while the second component (c_2) provides hiding. While the binding of the ElGamal commitment scheme is perfect, the binding of the Baum commitment scheme is based on the computational problem SIS. c_1 is a SIS-instance which locks the prover to a randomness r. If the randomness r is fixed to a specific value, also the message $m = c_2 - A_2 r$ is fixed to a specific value. An adversary prover must solve a SIS-instance to find another $r' \neq r$ satisfying $c_1 = A_1 r'$ in order to find another message $m' \neq m$ satisfying $m' = c_2 - A_2 r'$. While the hiding property ElGamal is based on the DDH problem, the hiding property of the Baum commitment scheme is based on the LWE problem. c_2 is an LWE-instance that hides the message m.

For dlog-based commitment schemes we must choose if we want perfect hiding and computational hiding with Pedersen or perfect binding and computational hiding with ElGamal. Because the binding of the Baum commitment scheme is based on SIS and the hiding is based on LWE, the scheme comes with different settings. By varying the parameters, we can choose statistical binding and computational hiding, computational binding and statistical hiding, or both hiding and binding computational. When we choose the third setting, the computational binding and hiding will be stronger than the computational hiding/binding of the first two settings. This is because we choose SIS and LWE to be equally difficult, which will make each of them more difficult than when we let only one of them be computational. We recall the SIS problem from Section 3. The SIS problem becomes harder when we set β to a smaller value. If we set β to a sufficiently small value, we get a statistical binding commitment scheme. With certain parameters, we could even achieve an unconditionally binding commitment scheme, when there exists only one solution of the SIS problem. Hiding is then computational. We recall the decisional LWE problem from Section 3. When setting β to a larger value, the LWE problem will become harder, and at some point, we have a statistically hiding scheme. Binding is then computational. By choosing β something in the middle, such that the hardness of the two problems SIS and LWE is equivalent, we achieve the most favourable overall security.

Just like in the Pedersen commitment scheme where the public parameters g and h must be

chosen uniformly random, for the Baum commitment scheme the public matrices A_1 and A_2 must be chosen uniformly random. If not, the matrices can be constructed so that a cheating prover has a backdoor to break the binding property or a cheating verifier has a backdoor to break hiding. In practice, some string from a well-known trusted parameter can be chosen, like π or the Euler's number e, then this string could be hashed and then used as the parameter. When choosing parameters like this, they should be uniformly random and there should then be no relation between the parameters.

Just like the Pedersen commitment scheme, the Baum commitment scheme cannot be used as an encryption scheme. Just like ElGamal scheme, a message could be opened without publishing the actual message m. If the prover only publishes the randomness r, and not the message m, the verifier could still compute the message as $m = c_2 - A_2 r$ and verify by checking $c_1 = A_1 r$.

6 Zero-Knowledge Proof System

In the following protocols we have a secret witness and a public statement about the witness. The witness can contain a secret key, some secret randomness or both, depending on the specific protocol. We have two players: the prover and the verifier. A prover wants to prove to the verifier that the public statement is true without revealing any additional information about the witness. The verifier either accepts or rejects the proof.

We distinguish between honest and dishonest players. An honest prover is a prover who actually knows the witness she tries to prove a statement about. A dishonest prover will try to prove that a statement is true without knowing the witness. An honest verifier is a verifier who follows the protocol honestly opposed to the dishonest verifier who may deviate from the protocol to learn more information about the witness.

Zero-Knowledge Proofs (ZKP) were first introduced by Goldwasser et al. [GMR89]. A Zero-Knowledge Proof is an interactive protocol where the prover wants to prove some statement to a verifier, so that the verifier does not learn anything else about the witness, than that the statement is true. A Zero-Knowledge Proof must fulfil the following three properties:

- Completeness: The honest prover is able to convince the honest verifier that a true statement is true.
- Soundness: The probability that the dishonest prover can convince the honest verifier that a false statement is true is negligible.
- Zero-Knowledge: The protocol only proves that the statement is true and leaks no additional information about the witness to the verifier.

The first two of these are properties of more general interactive proof systems. The third property is what makes the proof zero-knowledge. Sometimes we are not able to prove zero-knowledge (ZK) but only Honest-Verifier Zero-Knowledge (HVZK), which is ZK only when the verifier follows the protocol honestly.

Zero-Knowledge Proof of Knowledge (ZKPoK) We have just seen that a Zero-Knowledge protocol has the property of soundness. For a Zero-Knowledge Proof of Knowledge protocol we require the stronger soundness property Proof of Knowledge (PoK), as first defined by Bellare and Goldreich [BG92].

- Proof of Knowledge: There exists a method to extract the witness by interaction with the prover.

Σ-Protocol A Sigma Protocol is a ZKPoK in three rounds. The Greek letter Σ visualizes the flow of the protocol. Before the 3-move protocol can be run, the prover computes a public statement and a private witness. The prover publishes the public statement to the verifier, while the secret witness is kept secret. Then the prover interacts with the verifier in three moves as in the following:

1. Commitment phase: The prover commits to some randomness.
2. Challenge phase: The verifier replies with a challenge chosen at random.
3. Response phase: After receiving the challenge, the prover computes the response based on the randomness, the secret witness, and the challenge, and sends this response to the verifier.

The protocol transcript consisting of commitment, challenge and response is then accepted or rejected by the verifier.

The easiest way to prove Proof of Knowledge (PoK) of a Σ-protocol is by proving special soundness. Special soundness implies PoK which again implies soundness.

- Special Soundness: We can extract a witness from accepted proofs which have the same randomness (first message of the protocol) but distinct challenges (second messages).

If some dishonest prover would be able prove statements without knowing the witness, this prover could extract the witness from her own proofs, thus in practice, a prover who can prove statements, also knows the witness. Special soundness is proved by rewinding, which is a technique where we let the prover prove two statements with the same randomness. Then we extract the witness from these two statements. To sum up, if we have a ZKP and can prove special soundness by rewinding, we have PoK and thus the protocol is a ZKPoK.

6.1 Properties

The following formal definitions of interactive proofs, completeness, knowledge soundness and honest-verifier zero-knowledge are taken directly from Aranha et al. [Ara+22, Section 2.7]. The definitions are based on Goldwasser et al. [GMR89].

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

Definition 6 (Interactive Proofs). An interactive proof protocol Π consists of two parties: a prover P and a verifier V, and a setup algorithm Setup that on input the security parameter λ, outputs public setup parameters sp. The protocol consists of a transcript T of the communication between P and V, with respect to sp, and the conversation terminates with V outputting either 1 or 0. Let $\langle \mathsf{P}(\mathsf{sp}, x, w), \mathsf{V}(\mathsf{sp}, x) \rangle$ denote the output of V on input x after its interaction with P, who holds a witness w.

Definition 7 (Completeness). We say that a proof protocol Π is complete if V outputs 1 when P knows a witness w and both parties follow the protocol. Hence, for any efficient sampling algorithm P_0 we want that

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

where the probability is taken over the random coins of Setup, P and V.

Definition 8 (Knowledge Soundness). We say that a proof protocol Π is knowledge sound if, when a cheating prover P^* that does not know a witness w is able to convince an honest verifier V, there exists a polynomial time algorithm extractor $\mathcal{E}$ which, give black-box access to P^*, can output a witness w such that $(x, w) \in R$. Hence, we want that

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

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

Definition 9 (Honest-Verifier Zero-Knowledge). We say that a proof protocol Π is honest-verifier zero-knowledge if an honest but curious verifier $\mathsf{V}^\star$ that follows the protocol cannot learn anything beyond the fact that $x \in \mathsf{L}$. Hence, we want for real accepting transcripts $\mathsf{T}\langle \mathsf{P}(\mathsf{sp}, x, w), \mathsf{V}(\mathsf{sp}, x)\rangle$ between a prover P and a verifier V, and an accepting transcript $\mathsf{S}\langle \mathsf{P}(\mathsf{sp}, x, \cdot), \mathsf{V}(\mathsf{sp}, x)\rangle$ generated by simulator $\mathcal{S}$ that only knows x, that

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

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

An interactive honest-verifier zero-knowledge proof protocol can be made non-interactive using the Fiat-Shamir transform [FS86].

6.2 Zero-Knowledge Proofs for Commitments

Zero-Knowledge Proofs of Knowledge can be used to prove properties of commitments without revealing the openings. As we have seen in Section 5, the verifier can, when given an opening (m, r), use the algorithm Open to verify that commitment $[\![m]\!]$ actually opens to message m. The prover might not want to reveal opening (m, r) of commitment $[\![m]\!]$, but still want to convince the verifier about her knowledge of an opening. The prover also might want to prove specific properties of one or several messages, without revealing the openings. This reminds us of digital signatures, where a prover reveals the message m which has been signed but does not reveal the randomness used in the computation.

6.2.1 Zero-Knowledge Proofs for Dlog-Based Commitments

The following proof systems can be used to prove properties of the ElGamal ciphertexts from Section 4.2. All the proofs are based on the Chaum-Pedersen protocol [CP92]. The protocols are complete, an attacker against soundness is an attacker against CDH and an attacker against HVZK is an attacker against DDH.

- π_{Exp} [Swi21, Section 7.3.2] is a proof of exponentiation. Given vectors $g = (g_1, \dots g_n)$ and $y = (y_1, \dots y_n)$, it proves $y = (y_1, \dots y_n) = (g_1^x, \dots g_n^x)$, that is, vector y has been obtained by exponentiation of the elements of vector g to the same exponent x.
- π_{EqEnc} [Swi21, Section 7.3.3] is a plaintext equality proof. It proves that two ElGamal ciphertexts encrypt the same plaintext. Given two ElGamal ciphertexts $c = (u, v)$ and $c' = (u', v')$ under public keys pk and pk', it proves $(u, u', v/v') = (g^r, g^{r'}, \mathsf{pk}^r/\mathsf{pk}^{r'})$, where r is the random exponents used to encrypt c and r' is the random exponents used to encrypt c'.

- π_{Dec} [Swi21, Section 7.3.4] is a proof of correct decryption. Given the ElGamal ciphertext $c = (u, v)$ under the public key $\mathsf{pk} = g^{\mathsf{sk}}$ and plain text m resulting from decrypting c using secret key sk, it proves that $(\mathsf{pk}, v/m) = (g^{\mathsf{sk}}, u^{\mathsf{sk}})$. Similarly, given the multi-recipient ElGamal ciphertext $\mathbf{c} = (u, \mathbf{v}) = (u, v_1, \dots v_n)$ under public keys $\mathsf{pk}_1, \dots \mathsf{pk}_n$ with $\mathsf{pk}_i = g^{\mathsf{sk}_i}$ and plain texts $m_1, \dots m_n$ resulting from decrypting $\mathbf{c}$ using secret keys $\mathsf{sk}_1, \dots \mathsf{sk}_n$, it proves that $(\mathsf{pk}, \mathbf{v}/\mathbf{m}) = (g^{\mathsf{sk}_1}, \dots, g^{\mathsf{sk}_n}, u^{\mathsf{sk}_1}, \dots u^{\mathsf{sk}_n})$.

6.2.2 Zero-Knowledge Proofs for Lattice-Based Commitments

The following proofs can be used to prove properties of the commitments from Section 5.3. We remember the commitment opening of Equation (6) consists not only of opening d and message m, but also includes a polynomial $f \in \bar{\mathcal{C}}$. Existing zero-knowledge proofs proving knowledge of d and m satisfying (5) are not efficient. Therefore, the extractor for our zero-knowledge protocols does not guarantee that it will extract $f = 1$ from the prover. If the prover is honest, then the extractor will exactly recover the d, m from (5) and f will be 1.

Honest-Verifier Zero-Knowledge of the protocols is ensured by rejection sampling. Therefore, the prover might have to use several attempts in producing one proof.

- π_{NEx} [Lyu19, Section 5.2] is a proof of bounded opening. The prover in possession of a witness (m, r) satisfying Equation (5) for commitment $[\![m]\!]$ wants to demonstrate knowledge of an opening (m, r, f) satisfying Equation (6), where the m, r are from an interval somewhat larger than $[\beta]$ and $f \in \bar{\mathcal{C}}$. The protocol is complete. Special soundness can be proved by rewinding. From two accepted proofs π_{NEx} with two different challenges, a polynomial opening vector r together with $f \in \bar{\mathcal{C}}$ satisfying Equation (6) can be extracted. Therefore, an attacker against special soundness is an attacker against SIS. An attacker against HVZK is an attacker against LWE.

- π_{OPEN} [Bau+18, Section 3.1] is also a proof of bounded opening, but here the prover only proves that r is bounded, and not m.

- π_{LIN} [Ara+22, Section 3.3] is a proof of linear relation $\alpha_1 m_1 + \dots + \alpha_n m_n = \alpha_{n+1}$ with respect to commitments $[\![m_1]\!], \dots [\![m_n]\!]$ and public scalars α_i.

  More formally, we assume that there are $\hat{n}$ commitments $[\![\boldsymbol{m}_i]\!] = \begin{bmatrix} \boldsymbol{c}_{i,1} \\ \boldsymbol{c}_{i,2} \end{bmatrix}$ for $1 \le i \le \hat{n}$ where $\boldsymbol{c}_{i,2} \in R_q^{\ell}$. The commitments are made like in Section 5.3. x is the statement and w is the witness. For the public scalar vector $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_{\hat{n}-1}) \in R_q^{\hat{n}-1}$ the prover wants to demonstrate that the following relation R_{LIN} holds:

$$R_{\text{LIN}} = \left\{ (x, w) \mid \begin{array}{c} x = \left(\mathsf{pk}, \{[\![\boldsymbol{m}_i]\!]\}_{i \in [\hat{n}]}, \boldsymbol{\alpha}\right) \wedge w = \left(f, \{\boldsymbol{m}_i, \boldsymbol{r}_i\}_{i \in [\hat{n}]}\right) \wedge \\ \forall i \in [\hat{n}] : \mathsf{Open}_{\mathsf{pk}}\left([\![\boldsymbol{m}_i]\!], \boldsymbol{m}_i, \boldsymbol{r}_i, f\right) = 1 \wedge \boldsymbol{m}_{\hat{n}} = \sum_{i=1}^{\hat{n}-1} \alpha_i \boldsymbol{m}_i \end{array} \right\} \tag{7}$$

  Completeness is ensured if the randomnesses r_i of the commitments $[\![m_i]\!]$ are sufficiently bounded, an attacker against special soundness is an attacker against the SIS problem for $\beta = 4\sigma\sqrt{N}$, and an attacker against HVZK is an attacker against LWE.

- π_{AEx} [Ara+22, Section 3.4] is an amortized exact proof of short openings. Let $\boldsymbol{A}$ be a $r \times v$ matrix over R_q. The prover wants to demonstrate that the following relation holds for a bounded $\boldsymbol{s}_i$:

$$\boldsymbol{t}_i = \boldsymbol{A}\boldsymbol{s}_i \tag{8}$$

  The proof is complete when the secrets have ternary coefficients. An attacker against special soundness is an attacker against the SIS problem for $\beta = 1$, and an attacker against HVZK is an attacker against LWE.

All these zero-knowledge proofs have been proved secure in the random oracle model, but not in the quantum random oracle model.

7 Desired Properties of a Voting Protocol

In this section, we briefly explain four desired properties of a voting protocol. We present the definitions of the following properties: coercion resistance, privacy, integrity, and verifiability. The definitions are based on [Ara+21, Appendix B.6].

We separate between honest, corrupt, and malicious voters. The corrupt voters are those who are (willingly or not) coerced by the coercer and must obey instructions given. Malicious voters try to attack the system themselves.

7.1 Coercion Resistance

A voting system is coercion resistant if a coercer who demands voters to vote in a particular way, is not able to decide if the voters resisted the demands. This property prevents voters being bought or blackmailed by adversaries who intend to unlawfully influence the results of the election. Coercion resistance is modelled as a game with a coercer and a set of voters. The coercer communicates with the voters before and possibly after the election, and receives all public information published during the election. The coercer is trying to make the voter cast a vote in a particular way. Obvious special cases are that the coercer specifies the vote entirely or tries to prevent someone from voting. In the game, the coercer may coerce one or more voters. A voter can obey or disobey the demands of the coercer. The coercer may also ask voters to cast ballots uncoerced. The coercers may observe the voter while the voter is casting a ballot and may assist the voter in the casting process. Eventually, there is a tally, and the coercer receives the outcome. The system is coercion resistant if the coercer cannot decide if the voters obeyed their demands or not. The coercer only has access to public information from the infrastructure players, and cannot monitor networks. However, the coercer may have access to private information about the voter. In a real-world system, we can imagine the coercer being someone in the voter's family, the boss from the working place or the leader of some organization or religious network. The coercer could try to coerce the voter by threatening, or by promising some reward if the voter does as the coercer demands.

An online voting protocol could use re-voting to resist coercion. The Norwegian Voting Protocol [Gjø11] uses this method.

7.2 Privacy

Privacy in a voting system means confidentiality of the cast ballots against an adversary who corrupts infrastructure players. Privacy is modelled as an indistinguishability game between an adversary and a set of voters, some of them honest and some of them possibly corrupt. The adversary can corrupt infrastructure players. The adversary gives pairs of ballots to the honest voters, who cast one of the two. We have privacy if the adversary is not able to decide which ballot they cast.

Election results could unavoidably leak some information about the cast votes, for example if all honest voters select the same voting option or if only one voter cast their vote. Therefore, a system might have to define vote privacy as the inability to learn information about the cast votes, beyond what is unavoidably leaked by the election results.

While a coercing adversary tries to learn what the voter votes by interacting with the voter directly, an adversary against privacy tries to learn what the voter votes by compromising the infrastructure players.

An online voting protocol can use encryption of votes and distribution of trust to ensure privacy of the system.

7.3 Integrity

Integrity of a voting system means that a third party can check that everything in the election was computed correctly so that the election result corresponds to the intention of the voters.

Integrity for a voting system is modelled as a game between an adversary and a set of voters. Some of the voters are honest and some of them may be corrupt. The adversary tells the corrupt voters what ballots to cast. After the count phase when we have an election result, the adversary wins if the election result is inconsistent with the ballots accepted as cast by the honest voters. We can define a variant notion called ε-integrity where we allow a small error and say that the adversary wins if the election result is inconsistent with any $(1-\varepsilon)$ 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.

7.4 Verifiability

A voting system can provide verifiability to let voters and third parties verify the integrity of all components of the system. We divide verifiability into individual verifiability and universal verifiability.

For individual verifiability, the voter receives a receipt that her vote was sent as intended and recorded as confirmed. Sent as intended implies that the intended vote arrives to the voting server without being altered by an adversarial voting client. Recorded as confirmed implies that an adversarial voting server cannot drop the confirmed vote of an honest voter (vote rejection), and that an adversarial voting client or voting server cannot cast a vote on behalf of an honest voter (vote injection). The voters accept or reject their receipts.

Universal verifiability means that an adversary is unable to undetectably alter the election result. Any infrastructure manipulation would be detected. For the universal verifiable voting system, an election proof must be provided together with the election outcome, and a verification algorithm accepts or rejects the election proof. This verification is comparable to the recounting of physical ballots.

For a verifiable voting system the following claim must hold: if all the receipts accepted by the honest voters verify as accepted with the election proof, then the outcome of the election is consistent with the honest voter's cast ballots. We have limited verifiability if the same claim holds when certain participants are honest during the election.

An online voting protocol can use return codes to ensure individual verifiability. For ensuring universal verifiability the infrastructure players can log all operations and make necessary ZK-proofs that auditors can verify.

8 The Swiss Post Voting System

The Swiss Post voting system [Swi21] is a return code-based electronic voting protocol. The protocol consists of a configuration phase, a voting phase, and a tally phase. We will focus on the voting phase which is a two-round protocol consisting of a SendVote protocol and a ConfirmVote protocol.

8.1 Syntax

The protocol consists of the following participants: voters (V), voting client (VC), voting server (VS), control components (CC) which are used as return codes control components (CCR) and mixing control components (CCM), setup component (SC), printing component (PC), auditors (A), verifiers (Ver), election administrators (EA) and electoral board (EB). In the following we focus

on the voting phase and assume that SC and PC from the configuration phase are trustworthy. More specific, all private and public keys of the election, all generated by SC, were honestly generated and private keys were provided only to the right owners. The voting cards generated by SC were correctly generated, printed by PC and sent to V though a trusted channel. The mapping tables generated by SC were correctly computed corresponding to the voting cards and sent to VS though a trusted channel. A trusted channel from SC to Ver is assumed for delivering the necessary information for Ver to verify the elections. The participants relevant for the voting phase are described in the following:

V: The voters participate in the election by choosing their preferred options, check the return codes, and confirm their vote. Each voter holds a personal voting card including secret keys and return codes. We assume that that at least some honest voters verify the correctness of their return codes. We assume that that at least some voters verify the correctness of their return codes. We assume that some voters might collude with the adversary. For instance, they might reveal their keys or codes or try to impersonate another voter. A trusted channel from V to VC is assumed.

VC: The voting client is in charge of casting a ballot given the voting options selected by the voter. It encodes voting options, encrypts the vote, computes the ballot, and sends it to the voting server. We assume the voting client untrustworthy for integrity. However, as the voting client sees the voting options of the voter in clear text, it must be trusted for privacy.

VS: The voting server receives, processes, and stores the ballots. It combines the return code shares from the CCR components to compute the return codes. We assume the voting server untrustworthy.

CCR: The return codes control component consists of several components. Each of them computes return code shares based on the ballots. We assume at least one of the components is trustworthy.

Further we have auditors and verifiers. The auditors detect misbehaviour of untrustworthy parties and, therefore, ensure the security goals of the system. The auditors conduct a verification after all three phases, verifying all proofs and checking the consistency of the logs. Each auditor uses a verifier which is a software capable of checking cryptographic proofs. A trusted channel from an auditor to her verifier is assumed. It is assumed that at least one honest auditor verifies the election using a trustworthy verifier.

8.2 The Voting Protocol

The voting protocol is a two-round protocol consisting of a SendVote protocol and a ConfirmVote protocol. The protocol uses ElGamal Encryption and ElGamal Multi-Encryption as described in Section 4.2, and ZK-proofs based on the Chaum-Pedersen protocol as described in Section 6.2.1.

The voter holds a voting card consisting of the start voting key k, return codes cc for each possible voting option of the election, the ballot casting key k' and a confirmation return code VCC. In the SendVote protocol, the return codes cc are used by the voter to verify that their vote is sent as intended. In the ConfirmVote protocol, the return code VCC is used by the voter to verify that their vote is recorded as confirmed. The control components hold two different user-specific keys for making the return codes: k_j used in the SendVote protocol and k'_j used in the ConfirmVote protocol. The voting server holds a mapping table linking long return codes lCC with short return codes $\mathsf{cc}^\star$. $\mathsf{EL}_{\mathsf{pk}}$ is the public election key.

8.2.1 The SendVote Protocol

The SendVote Protocol shown in Figure 5 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.

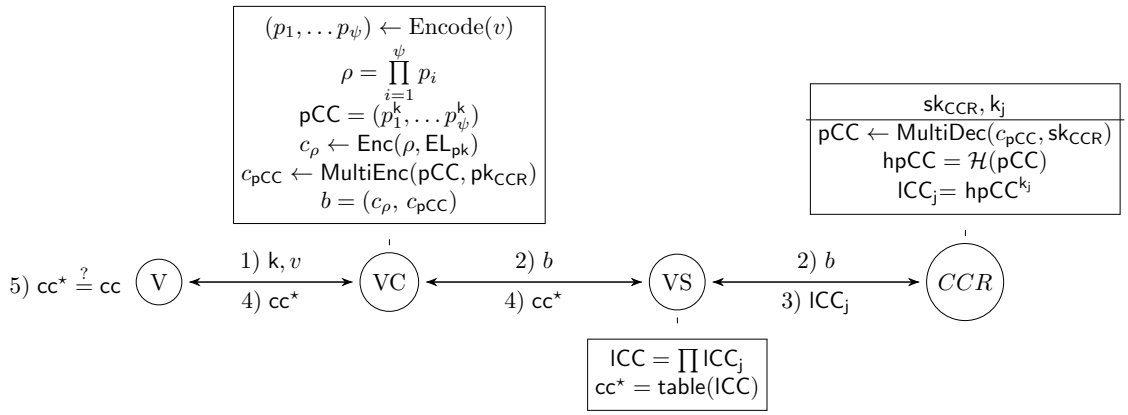


Figure 5: The SendVote protocol of the Swiss Post voting system.

The Ballot The ballot b computed by VC in step 2 includes in addition to the two ciphertexts c_ρ and c_{pCC} as shown in Figure 5, also one additional ciphertext $\tilde{\mathrm{c}}_\rho$ and zero-knowledge proofs π_{Exp} and π_{EqEnc} proving that the initial ciphertexts were computed correctly with respect to ρ. Finally, b includes the identity of the voter vcd and a signature [Swi21, Sec 12.2.1.2]. The zero-knowledge proofs ensures that the ciphertexts c_ρ and c_{pCC} were computed with the same voting options. We recall the ciphertext

$$c_\rho = \text{Enc}(\rho, \mathsf{EL_{pk}}; r) = (u, v) = (g^r, {\mathsf{EL_{pk}}}^r \rho)$$

which is an encryption of the vote ρ under public key $\mathsf{EL_{pk}}$ and with randomness r.
By exponentiating u and v of ciphertext c_ρ to the start voting key k we get

$$\tilde{\mathrm{c}}_\rho = (\tilde{u}, \tilde{v}) = (u^{\mathsf{k}}, v^{\mathsf{k}}) = (g^{r\mathsf{k}}, {\mathsf{EL_{pk}}}^{r\mathsf{k}} \rho^{\mathsf{k}}) = \text{Enc}(\rho^{\mathsf{k}}, \mathsf{EL_{pk}}; r\mathsf{k})$$

which is an encryption of ρ^{k} under public key $\mathsf{EL_{pk}}$ and randomness $r\mathsf{k}$. Further we recall the ciphertext

$$c_{\mathsf{pCC}} = \mathsf{MultiEnc}(\mathsf{pCC}, \mathsf{pk_{CCR}}; r') = (u', v'_1, \dots v'_\psi) = (g^{r'}, {\mathsf{pk_{CCR}}_1}^{r'} p_1^{\mathsf{k}}, \dots {\mathsf{pk_{CCR}}_\psi}^{r'} p_\psi^{\mathsf{k}})$$

which is a multi-recipient encryption of the partial return codes $(\mathsf{pCC}_1, \dots \mathsf{pCC}_\psi) = (p_1^{\mathsf{k}}, \dots p_\psi^{\mathsf{k}})$ under public key $\mathsf{pk_{CCR}} = (\mathsf{pk_{CCR}}_1, \dots \mathsf{pk_{CCR}}_\psi)$ and with randomness r'. By "compressing" ciphertext c_{pCC} we obtain

$$\tilde{c}_{\mathsf{pCC}} = (\tilde{u'}, \tilde{v'}) = (u', \prod_{i=1}^{\psi} v'_i) = (g^{r'}, \prod_{i=1}^{\psi} \mathsf{pk_{CCR_i}}^{r'} p_i^{\mathsf{k}}) = (g^{r'}, \prod_{i=1}^{\psi} \mathsf{pk_{CCR_i}}^{r'} \cdot \rho^{\mathsf{k}}) = \mathsf{Enc}(\rho^{\mathsf{k}}, \prod_{i=1}^{\psi} \mathsf{pk_{CCR_i}}; r')$$

which is an encryption of ρ^{k} under public key $\prod_{i=1}^{\psi} \mathsf{pk_{CCR_i}}$ and with randomness r'.

The proofs included in the ballot prove the following: π_{Exp} proves that $\tilde{\mathsf{c}}_\rho$ was computed by exponentiating u and v of ciphertext c_ρ to the start voting key k. π_{EqEnc} proves that $\tilde{\mathsf{c}}_\rho$ and $\tilde{c}_{\mathsf{pCC}}$ encrypt the same plaintext under two different public keys.

The Return Code Shares The return code shares $\mathsf{lCC_j}$ computed by the CCR components in step 3 are computed as $\mathsf{lCC_j} = \mathsf{hpCC}^{\mathsf{k_j}}$ where $\mathsf{hpCC} = \mathcal{H}(\mathsf{pCC})$. Here, $\mathcal{H}$ is a hash function and $\mathsf{k_j}$ is a secret user-specific key. Each CCR component must in addition to the return code share also provide a zero-knowledge proof π_{Exp} of correct exponentiation. This proof proves that $\mathsf{lCC_j}$ was computed by exponentiating hpCC to the key $\mathsf{k_j}$ [Swi21, Sec 12.2.1.6].

Verifying the Return Codes In step 5, V verifies the return codes. If the return codes $\mathsf{cc}^\star$ shown on the screen are not equal to the return codes cc shown in the voting card of the user, V must report this to the auditors. If the return codes are equal, V must proceed with the ConfirmVote protocol.

8.2.2 The ConfirmVote Protocol

The ConfirmVote protocol [Swi21, Sec 12.2.2] is only initiated by V if the verification from step 5 is successful. The protocol is similar to the SendVote protocol.

The ConfirmVote Protocol shown in Figure 6 consists of the following steps:

1. V enters to VC the ballot casting key $\mathsf{k'}$ from the voting card.
2. VC computes the confirmation key CK and sends it to VS which forwards to CCR.
3. CCR generates return code shares $\mathsf{lVCC_j}$ and sends them to VS.
4. VS combines the shares from CCR. With a mapping table it extracts a confirmation return code $\mathsf{VCC}^\star$ that is sent to VC and shown to V.
5. V verifies $\mathsf{VCC}^\star$ shown on the screen by checking that it is equal to VCC from the voting card. Only after successfully verifying $\mathsf{VCC}^\star$, V has completed the voting process.

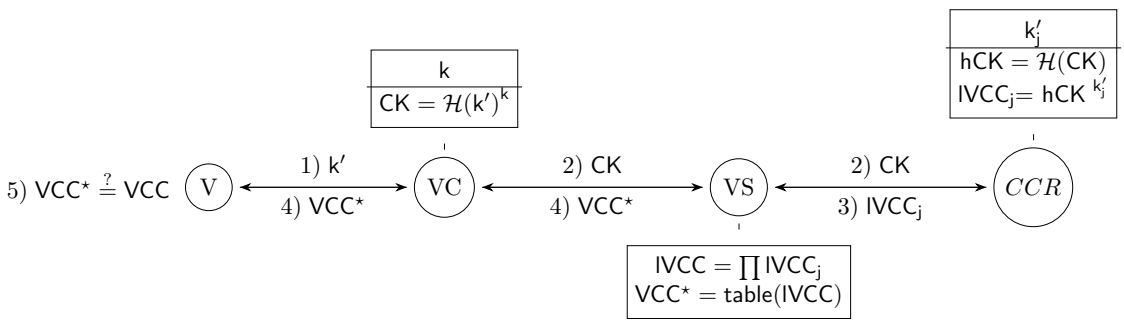


Figure 6: The ConfirmVote protocol of the Swiss Post voting system.

The Confirmation Return Code Shares $\mathsf{lVCC_j}$ computed by the CCR components in step 3 are computed as $\mathsf{lVCC_j} = \mathsf{hCK}^{\mathsf{k'_j}}$ where $\mathsf{hCK} = \mathcal{H}(\mathsf{CK})$. Here, $\mathcal{H}$ is a hash function and $\mathsf{k'_j}$ is a secret user-specific key. Each CCR component must in addition to the return code share also provide a zero-knowledge proof π_{Exp} of correct exponentiation. This proof proves that $\mathsf{lVCC_j}$ was computed by exponentiating hCK to the key $\mathsf{k'_j}$ [Swi21, Sec 12.2.2.2].

8.2.3 Verifying the Voting Phase

During the voting phase, the players must log all communication. At the end of the voting phase, and before starting the tally phase, a verification of the voting phase is done by the auditors using their verifiers, ensuring that all logs are consistent and verifying all ZK-proofs. In particular, the auditors ensure that the CCR components have the same set of ballots, that they performed the decryption of step 2 of the SendVote protocol on the same set of partial return codes and for the same set of voters, and that they performed the exponentiation of step 3 of the SendVote protocol for the same set of voters. Further they ensure that the valid ballots stored by VS and CCR are identical, and that these ballots were cast by the same set of voters. By checking that the number of confirmation attempts stored by VS and CCR are identical they ensure that a successful confirmation attempt was not overrode by VS after V received her confirmation code. Further the auditors verify the proofs π_{Exp} generated by the CCR components in the SendVote and the ConfirmVote protocol, and that all codes lCC and lVCC have matching entries in the mapping table.

8.3 Discussion

8.3.1 Simplifications

For simplicity, the following aspects of [Swi21] have been excluded from our description.

The return code shares $\mathsf{lCC_j}$, the long return codes lCC and the return codes $\mathsf{cc}^\star$ are all vectors, where each element corresponds to one element of the vector pCC. Each return code of the vector $\mathsf{cc}^\star$ corresponds to one chosen voting option of the voter. All operations on the vectors, like hashing and exponentiation, are done element wise.

We have not elaborated on the distributed decryption of pCC [Swi21, Section 12.2.1.3]. We have excluded further explanation on this topic for simplicity, and because the security reductions for privacy [Swi21, Section 19.4] omit the encryption of pCC.

In our figures, the key k is sent directly from V to VC. In the actual protocol, a start voting key called SVK is sent from V to VC, and VC used this key to open a key store and find the key k [Swi21, Section 12.2.1.1].

We have kept the description simplified by only including operations which are related to the dlog-based cryptography. The following operations could be kept as they are for a lattice-based system and are therefore not further discussed. The contents of the table held by VS are held secret with a symmetric encryption scheme [Swi21, Section 12.1.1.7]. Hash operations with a value called correctnessID are used to ensure vote compliance. The votes that get registered and counted must be of the correct format, for example the vote ρ must be a combination of valid voting options [Swi21, Section 15.2].

8.3.2 Observations

The encodings p_i of the voting options are in fact prime numbers, as explained in [Swi21, Section 10.3]. The encodings are multiplied and encrypted together in the ciphertext c_ρ. After being mixed and decrypted, the original primes can be computed by factorizing. When working over a group, using this construction with primes is straight forward, opposed to if the protocol had worked over elliptic curves.

The hash operation done on pCC in the SendVote protocol and on CK in the ConfirmVote protocol result in group elements. This is important for the following exponentiation that requires the base to be a group element.

It might seem impractical that we need one return code for each voting option of the election. Using one single return code representing the total combination of the voter's chosen voting options might seem tempting. A voting event has ψ questions each with n options. When using one return code for each voting option, the voter must verify all ψ received return codes. The voting card must include $\psi \cdot n$ different return codes. If one single return code represents the total combination of the voter's chosen voting options, the voter must only verify 1 return code. However, the voting card must include ψ^n return codes. This quickly scales up, resulting in an exceptionally large voting card.

In step 2, VC sends a confirmation key $\mathsf{CK}=\mathcal{H}(\mathsf{k}')^{\mathsf{k}}$. We observe that no exponentiation proof is given for this computation [Swi21, Section 12.2.2.1]. An incorrect exponentiation can only result in a not successful confirmation attempt and cannot change the vote. Blocking communication from the voter can always be done by VC, thus an exponentiation proof would not change the security analysis.

We remember that bugs (that do not have anything to do with the cryptography of the system) could result in the voter being shown the wrong return codes. The voter could in this case notify the auditor, who could find the bug and tell the voter to try again. Although this does not affect the security of the system, it could damage the voter's trust in the system.

8.4 Security Analysis

We informally discuss the security of the protocol, mainly focusing on the voting phase. We will see that the protocol provides no coercion resistance. The voting phase provides individual verifiability assuming at least one trustworthy control component and at least one honest auditor verifying the voting phase with a trustworthy verifier. The voting phase provides privacy assuming VC is honest. For the voting phase, an attacker against individual verifiability is an attacker against the CDH problem and an attacker against privacy is an attacker against the SGSP problem.

8.4.1 Coercion Resistance

The protocol is not coercion resistant. Switzerland evaluates the risk of coercion as low in their elections, and do not strive to achieve a coercion resistant voting system. Through their extensive use of postal voting, the possibility of vote selling is already present. All cantons accept votes by postal mail and around 90% of Swiss people use this voting channel.[4]

8.4.2 Individual Verifiability

The protocol offers individual verifiability if at least one component of CCR is honest. Individual verifiability is something that must be instant, a voter cannot wait for an auditor to check all ZK-proofs before showing the return codes. We divide individual verifiability into sent-as-intended, which is achieved in the SendVote protocol, and recorded-as-intended, which is achieved in the ConfirmVote protocol.

A malicious VC could try to modify the intended voting options without detection with two different approaches. The first approach is to attack sent-as-intended by modifying the contents of the ballot, while showing to the voter return codes corresponding to the voters chosen voting options (instead of return codes corresponding to the modified ballot). The second attack approach is to attack recorded-as-confirmed by vote injection, by confirming a vote without the participation

[4] https://www.nzz.ch/meinung/kommentare/digitale-demokratie-verlangt-pioniergeist-ld.2175?reduced=true Accessed the 29th of May 2022

of the voter. A malicious VS could attack recorded-as-confirmed by vote rejection, by rejecting already confirmed votes.

Sent-as-Intended For the attack against sent-as-intended, VC achieves voting options v from V corresponding to voting options $(p_1, \dots p_\psi)$ and vote ρ, but computes a ballot with a ciphertext $c_{\rho'}$ of another vote $\rho' \neq \rho$ so that $c_{\rho'}$ does not include the same voting options as the correctly computed partial return codes $\mathsf{pCC} = (p_1^{\mathsf{k}}, \dots p_\psi^{\mathsf{k}})$. VC includes in the ballot none or invalid ZK-proofs connecting $c_{\rho'}$ and pCC. If one CCR component is honest, it will discover the missing or invalid ZK-proofs and not send any return codes back. If VC is able to make such valid ballot, is also an attacker against the soundness of the ZK-proofs connecting $c_{\rho'}$ and pCC and thus also an attacker against the CDH problem.

Another approach, if VC cooperates with some of the CCR components, it could make a valid ballot based on a wrong vote $\rho' \neq \rho$ with ciphertext $c_{\rho'}$ and partial return codes $\mathsf{pCC}' = \rho' + \mathsf{k}$ and the valid proofs connecting $c_{\rho'}$ and pCC'. Then VC communicates to the dishonest CCR components that they should make return code shares rather corresponding to $\mathsf{pCC} = \rho + \mathsf{k}$. However, if at least one CCR component is honest, it will make its return code share based on pCC', and the dishonest CCR components will not be able to make shares cancelling the share of the honest CCR component because they do not know the key of the honest CCR.

Thus, the only strategy left for the dishonest VC is to guess the return codes the voter expects. A brute force attack cannot be done in this case since the voter will detect consecutive attempts of displaying wrong return codes. [Swi21, Section 19.3] proves sent-as-intended of the protocol.

Recorded-as-Confirmed (Vote Injection) For the attack against recorded-as-confirmed by vote injection the VC could generate a fake confirmation message CK that will result in fake confirmation return code shares $\mathsf{lVCC_j}$. When VS combines these shares to lVCC it will not find an entry for this fake lVCC in the table but could if dishonest still output a fake confirmation return code $\mathsf{VCC}^\star$ (which will not be shown to the voter). However, the auditors will in the verification phase observe that there is no entry for lVCC in the table and can discard this vote before the tally phase begins. The alternative is that the voting client guesses a valid confirmation message. In order to limit the possibility of a brute force attack, the voting server allows a limited number of retries. [Swi21, Section 19.5] proves recorded-as-confirmed (vote injection) of the protocol.

Recorded-as-Confirmed (Vote Rejection) A malicious VS could try to attack recorded-as-confirmed by rejecting confirmed votes. VS would here override the successful confirmation attempt where V received her confirmation code. As described in Section 8.2.3, the auditors check at the end of the voting phase that the number of confirmation attempts stored by VS and CCR are identical. In this way, if at least one CCR component is honest and has the correct confirmation attempts stored, the auditors would detect this attack. [Swi21, Section 19.4] proves recorded-as-confirmed (vote rejection) of the protocol.

The Importance of the Trust Assumptions We have the trust assumption that at least one of the control component is trustworthy. Without this trust assumption, it is easy to imagine attacks against individual verifiability. Imagine an attacker controlling VS, all components of CCR, and VC of some voter V. This attacker could change or discard the vote of V, but still let V think she has successfully verified her vote. The attack against sent-as-intended to change the vote is executed by VC as already described. Now, since they are all under the attacker's control, VS and all CCR components verify the invalid ZK-proofs as valid. Then all CCR components make return code shares and corresponding (correct) proofs based on pCC as usual. VS combines the shares as usual and sends the expected return codes $\mathsf{cc}^\star$ to V. The voter receives the correct return code although the ciphertext $c_{\rho'}$ contained in the ballot encrypts the fake vote $\rho' \neq \rho$. In the verification phase, only the exponentiation proofs from CCR will be checked by the auditors and these are indeed correct. The system could also attack recorded-as-confirmed to discard a vote. The attack

is run by VS as already described. Now that none CCR components are trustworthy, they all store the same number of confirmation attempts as VS, so that the auditors cannot detect the attack.

8.4.3 Universal Verifiability

The voting phase protocols offer universal verifiability. No infrastructure player must be assumed trustworthy for this security property, but it is assumed that at least one honest auditor verifies the results with a trustworthy verifier [Swi21, Section 2.1.1: Auditors]. This auditor verifying ZK-proofs and checking that logs are consistent would detect any manipulation of the infrastructure. [Swi21, Section 17 and Section 19.6] prove universal verifiability of the protocol.

8.4.4 Privacy

The voting phase protocols offer privacy with the given trust assumptions. We assume that VC is honest. The voter types their voting options directly to VC, thus if VC is compromised, it could leak this information. For privacy we also need that the voting card contents must only be known to the voter. This means the setup component SC and the printing component PC must be trustworthy. We assume that SC and PC do not give away the voting card contents to anyone, and that they are destroyed or turned off before the voting phase begins. We must also assume some trusted channel to send the voting cards to the voters. In practice the voting cards will be sent by postal service, which means this channel must be trusted.

An attacker knowing the voting card contents of some voter and knowing which device the voter uses as her voting client, can attack privacy by tracking the outgoing and incoming traffic to VC. They could get information about the vote ρ using pCC from the outgoing communication from VC and k from the voting card, or they could match the return codes $\mathsf{cc}^\star$ from the incoming communication to VC with the return codes cc from the voting card content.

Assuming the mentioned trust assumptions, an adversary against vote privacy is also an adversary against either the CPA security of the encrypted vote c_ρ, the hiding of the partial return codes pCC or the HVZK of the ZK-proofs π_{Exp} or π_{EqEnc}. The CPA security of c_ρ relies on DDH, the hiding of pCC relies on ESGSP, and the HVZK of the ZK-proofs relies on DDH. The strongest assumption of these is the ESGSP assumption, thus the partial return codes are for privacy the weakest part of the system.

Finally, even if it is possible for a malicious voter to copy the vote of another voter and cast it as it was theirs, this voter will not have any individual verifiability of their vote. Malicious voters copying a vote of another voter and casting it as it was theirs, receive return codes which they cannot understand. They do not receive return codes matching those in their voting card, because the return codes belong to another voting card.

8.4.5 Availability

A malicious VC, VS or CCR component could prevent the voter from receiving return codes and in this way attack availability. If VC shows no return codes or the wrong ones, the voter could change to another VC and continue the process. If VS or CCR prevent the generation of return codes, an investigation can be started. The logs will show which component is attacking the system and this component can be replaced.

9 The Voting Protocol by Aranha et al.

Published in 2021, 'Lattice-Based proof of Shuffle and Applications to Electronic Voting' [Ara+21] by Aranha, Baum, Gjøsteen, Silde, and Tunge, presents the first practical verifiable shuffle of known values for lattice-based commitments. The shuffle can be used to prove that a collection of

commitments opens to a given collection of known messages. The scheme is the first construction from candidate post-quantum assumptions to defend against compromise of the voter's device using return codes. For privacy of the ballots, the protocol assumes that the voting server does not cooperate with the mixing server. For sent-as-intended integrity, the protocol assumes that the return code server does not cooperate with the voting client.

All ciphertexts in this section are computed like in Section 4.3 and all commitments are computed like in Section 5.3. The ZK-proofs used were presented in Section 6.2.2. In the following we focus on the voting phase and assume that all private and public keys and all voting cards of the election were honestly generated, private keys are only known by the right owners and the voting card is only known by the voter.

V: The voters participate in the election by choosing their preferred options and check the return codes. Each voter holds a personal voting card including secret keys and return codes. We assume that that at least some voters verify the correctness of their return codes. We assume that some voters might collude with the adversary. For instance, they might reveal their keys or codes or try to impersonate another voter. A trusted channel from V to VC is assumed.

VC: The voting client is in charge of casting a ballot given the voting options selected by the voter. It encodes voting options, encrypts the vote, computes the ballot, and sends it to the voting server. For integrity, we assume that the voting client does not cooperate with the return code server. The voting client can see the voting options of the voter in clear text, thus it must be trusted for privacy.

VS: The voting server receives, processes, and stores the ballots. For privacy, we assume the voting server does not cooperate with the mixing server.

CCR: The return code server computes return code based on the ballots. For integrity, we assume that the return code server does not cooperate with the voting client.

We assume auditors that check proofs and consistency of the logs. It is assumed that at least one honest auditor verifies the election.

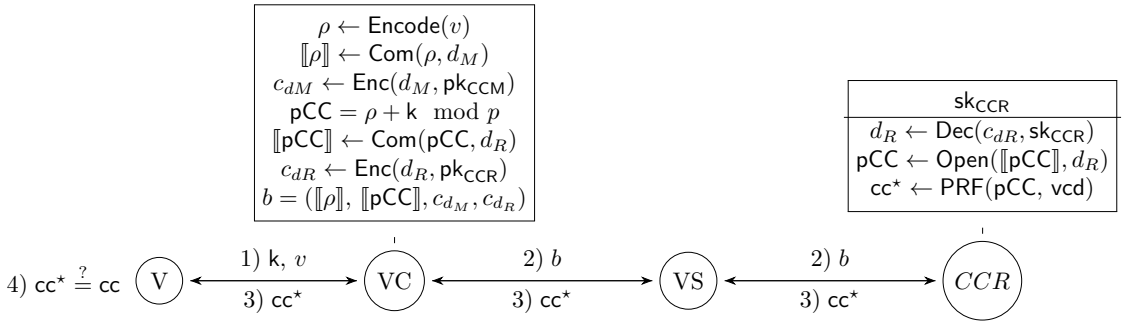


Figure 7: The voting phase of [Ara+21].

Figure 7 shows a simplified version of the lattice-based send vote-phase of [Ara+21, Figure 6]. A commitment $[\![\mathsf{k}]\!]$ is public information. V has a voting card including the start voting key k and a list of all possible voting options, each voting option linked to a return code cc. The protocol has the following steps:

1. V enters to VC the start voting key k and her selected voting options v.
2. VC creates the ballot b by using the voters selected voting options v and the start voting key k. VC sends the ballot to VS which forwards it to CCR.
3. CCR validate the proofs. Then, CCR decrypts opening d_R and uses this opening to find the partial computed return code pCC. Then it computes a return code $\mathsf{cc}^\star$ that is sent back to the voter V.

4. The voter verifies the short choice return codes shown on the screen of the voting client $\mathsf{cc}^\star$ by checking that they are equal to the corresponding short choice return codes from the voting card cc.

The Ballot The ballot is computed by VC on input a vote ρ and a start voting key k. In addition to the commitments to ρ and pCC and the encrypted openings of these commitments, the ballot includes the voter's id vcd, and several proofs: Two proofs of opening π_{OPEN} prove that d_M is a valid opening of $[\![\rho]\!]$ and d_R is a valid opening of $[\![\mathsf{pCC}]\!]$. A proof π_{LIN} proves that the linear relation $\mathsf{pCC} = \rho + \mathsf{k}$ holds. Another proof π_{LIN} proves that the message pCC behind commitment $[\![\mathsf{pCC}]\!]$ with opening d_R is equal to the message behind $[\![\rho]\!] + [\![\mathsf{k}]\!]$ with opening $d_M + d$. Finally, a shortness proof of ρ ensures correctness of the ciphertext.

The Return Codes The return codes are computed by CCR on input $[\![\mathsf{pCC}]\!]$ and c_{d_R}. CCR decrypts the opening d_R with its secret key $\mathsf{sk}_{\mathsf{CCR}}$ and then computes the partial return code pCC by opening the commitment $[\![\mathsf{pCC}]\!]$ with d_R. Then the return codes $\mathsf{cc}^\star$ are computed with a pseudo-random function with input the partial return code pCC and the identity vcd of the voter.

The Tally Phase After the voting phase is completed, the tally phase can begin. VS makes a sorted list of the commitments $[\![\rho]\!]$, a sorted list of the identities vcd and a sorted list of the encrypted openings c_{d_M} under the public key pk_{CCM} of the mixing server. All these sorted lists are sent to the mixing server. The mixing server opens the commitments and finds the plaintexts ρ, then mixes these plaintexts with a secret permutation. The mixing server publishes the permuted list of votes and a proof π of correctly shuffled votes.

The Openings of Commitments We recall that a commitment $[\![\mathsf{k}]\!]$ is public information. Let the opening of $[\![\mathsf{k}]\!]$ be d. For the ballot computations we have $\mathsf{pCC} = \rho + \mathsf{k}$, so the message behind commitment $[\![\mathsf{pCC}]\!]$ with opening d_R is equal to the message behind $[\![\rho]\!] + [\![\mathsf{k}]\!]$ with opening $d_M + d$, but $[\![\mathsf{pCC}]\!] \neq [\![\rho]\!] + [\![\mathsf{k}]\!]$. The commitment to pCC is not done with the opening $d_M + d$. Because both openings d_M and d are short, committing to pCC with opening $d_M + d$ could have revealed information about pCC.

Discussion The presented voting phase from [Ara+21] could not be used for the protocol of [Swi21]. The protocol of [Swi21] assumes an untrustworthy voting server and several components for return code computation and mixing, where only one control component is assumed trustworthy. For the voting phase of [Ara+21], the voting server must be trusted to not cooperate with the mixing server, and to actually sort the list of votes in the beginning of the tally phase, because if not, the mixing server learns what the voters voted when opening the commitments. As the shuffle is a shuffle of known content and opens the commitments before mixing them, it does not make any sense to have several components of the mixer. The protocol also only includes one return code component. The protocol could be changed to include several components of CCR. This would require a a distributed decryption of the opening of commitment $[\![\mathsf{pCC}]\!]$. There is no ConfirmVote protocol, rather it is assumed that voters will complain to the auditors if they receive the wrong return codes. A ConfirmVote protocol could be implemented similar to the SendVote protocol.

10 Our Voting Protocol

Cryptographic primitives based on discrete log-type assumptions are used in the Swiss Post voting system described in Section 8. This applies for steps 2 and 3 of the SendVote protocol shown in Figure 5, and for steps 2 and 3 of the ConfirmVote protocol shown in Figure 6. We focus on these parts in this section. The hash-functions used are considered post-quantum secure.

10.1 The Voting Protocol

We describe protocols SendVote and a ConfirmVote that use cryptographic primitives based on lattice assumptions. Our SendVote protocol is inspired by the voting protocol by Aranha et al. described in Section 9. Unlike the voting protocol by Aranha et al., the ballots computed in our protocol contain ciphertexts c_ρ under the election public key $\mathsf{EL_{pk}}$, and these ciphertexts can be used as input to the shuffle described in [Ara+22]. This makes our voting protocol compatible with the trust assumptions of [Swi21].

All ciphertexts in this section are computed like in Section 4.3 and all commitments are computed like in Section 5.3. The ZK-proofs used were presented in Section 6.2.2. The syntax is equivalent to the syntax of the Swiss Post voting system, described in Section 8.1. The workflow of protocols SendVote and ConfirmVote is equivalent to the workflow described in Section 8.2.1 and Section 8.2.2.

10.1.1 The SendVote Protocol

Figure 8 presents a SendVote protocol using primitives based on lattice assumptions. In our protocol, VC does not encrypt the partial return code pCC as the protocol security reductions for privacy [Swi21, Sec 19.4] omit this encryption (but it could, if required). $\mathsf{EL_{pk}}$ is the public election key. Commitments to the polynomials k and $\mathsf{k_j}$ 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 $p = 2$. The rounding function $\lfloor \cdot \rceil$ rounds a decimal value down to the next integer value.

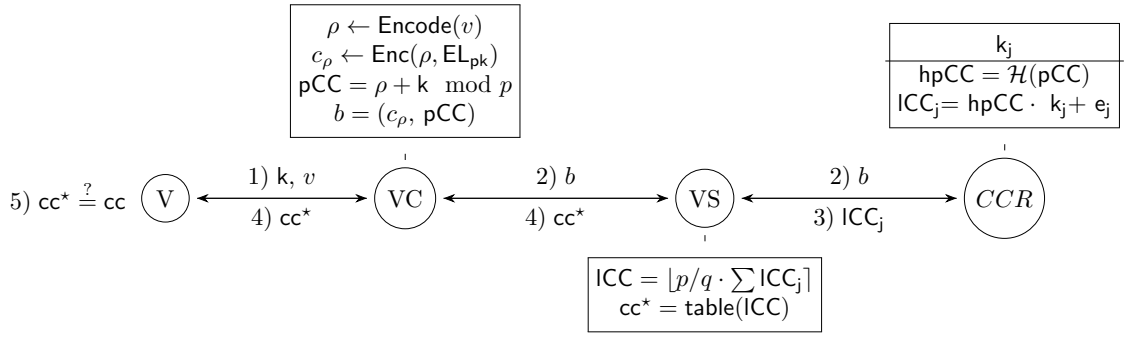


Figure 8: Our SendVote protocol for lattice-based electronic voting.

In step 2, when VC computes $\mathsf{pCC} \mod p$, this might produce some computational overflow which must be 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 $\mathsf{pCC} + 2z = \rho + \mathsf{k} \mod q$. Proofs π_{LIN} prove correct computation of c_ρ as in Equation (4). An amortized shortness-proof π_{AEx} ensures that z and the randomness used in c_ρ is binary.

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.

10.1.2 The ConfirmVote Protocol

Figure 9 presents a ConfirmVote protocol using primitives based on lattice assumptions. Commitments to the polynomials k' and k'_j are public information.

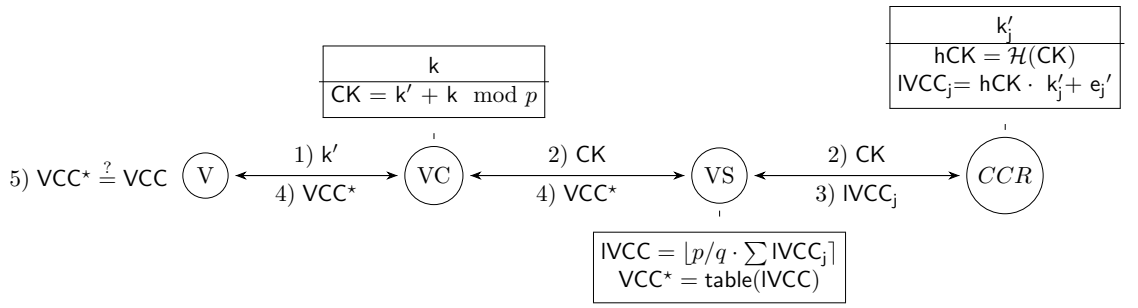


Figure 9: Our ConfirmVote protocol for lattice-based electronic voting.

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

10.1.3 Verifying the Voting Phase

Just like in [Swi21], the players log all communication during the voting phase. At the end of the voting phase, and before starting the tally phase, a verification of the voting phase is done by the auditors using their verifiers, ensuring that all logs are consistent and verifying the ZK-proofs π_{LIN} and π_{NEx} computed by the CCR components in step 3 of the SendVote protocol and in step 3 of the ConfirmVote protocol.

10.2 Comparison with Swiss Post and Aranha et al.

Our voting protocol is inspired by the voting protocol by Aranha et al. [Ara+21]. While that protocol assumes a trusted VS, our protocol allows for an untrustworthy VS and only assumes that at least one of the CCR components is trustworthy. These are the same trust assumptions as the trust assumptions by [Swi21].

Table 2 compares our two-round lattice-based protocol (as described in Section 10.1) with the two-round d-log based voting protocol by Swiss Post (as described in Section 8) and the one-round lattice-based voting protocol by Aranha et al. (as described in Section 9).

		[Swi21]	[Ara+21]	Our Voting Protocol
1	VC	$p_1, \dots, p_\psi \leftarrow$Encode(v)	$m_1, \dots, m_\psi \leftarrow$Encode(v)	$m_1, \dots, m_\psi \leftarrow$Encode(v)
2	VC	$\rho = p_1 \cdots p_\psi$	$\rho = [m_1 \vert \dots \vert m_\psi]$	$\rho = [m_1 \vert \dots \vert m_\psi]$
3	VC	$c_\rho \leftarrow \mathsf{Enc}(\rho, \mathsf{EL_{pk}})$	$[\![\rho]\!] \leftarrow \text{Com}(\rho, d_M)$	$c_\rho \leftarrow \mathsf{Enc}(\rho, \mathsf{EL_{pk}})$
4	VC	$\mathsf{pCC} = [p_1^{\mathsf{k}}, \dots, p_\psi^{\mathsf{k}}]$	$\mathsf{pCC} = \rho + \mathsf{k} \mod p$	$\mathsf{pCC} = \rho + \mathsf{k} \mod p$
5	VC	$c_{\mathsf{pCC}} \leftarrow$MultiEnc(pCC,$\mathsf{pk_{CCR}}$)	$[\![\mathsf{pCC}]\!] = \mathsf{Com}(\mathsf{pCC}, d_R)$	pCC
6	VC	π_{Exp}, π_{EqEnc}	π_{OPEN}, π_{LIN}, π_{ANEx}	π_{LIN}, π_{ANEx}
7	CCR	$\mathsf{lCC_j} = \mathcal{H}(\mathsf{pCC})^{\mathsf{k_j}}$	$\mathsf{cc}^\star = \text{PRF}(\mathsf{pCC}, \mathsf{vcd})$	$\mathsf{lCC_j} = \mathcal{H}(\mathsf{pCC}) \cdot \mathsf{k_j} + \mathsf{e_j}$
8	CCR	π_{Exp}	–	π_{LIN}, π_{NEx}
9	VS	$\mathsf{cc}^\star = \mathsf{table}(\Pi\ \mathsf{lCC_j})$	–	$\mathsf{cc}^\star = \mathsf{table}(\lfloor p/q \cdot \sum \mathsf{lCC_j} \rceil)$
10	VC	$\mathsf{CK} = \mathcal{H}(\mathsf{k}')^{\mathsf{k}}$	–	$\mathsf{CK} = \mathsf{k} + \mathsf{k}' \mod p$
11	CCR	$\mathsf{lVCC_j} = \mathcal{H}(\mathsf{CK})^{\mathsf{k}_j'}$	–	$\mathsf{lVCC_j} = \mathcal{H}(\mathsf{CK}) \cdot \mathsf{k}_j' + \mathsf{e_j}$
12	CCR	π_{Exp}	–	π_{LIN}, π_{NEx}
13	VS	$\mathsf{VCC}^\star = \mathsf{table}(\Pi\ \mathsf{lVCC_j})$	–	$\mathsf{VCC}^\star = \mathsf{table}(\lfloor p/q \cdot \sum \mathsf{lVCC_j} \rceil)$

Table 2: Column 2 shows which component computes the values shown in columns 3-5. VC is the voting client, VS is the voting server, and CCR is the control component(s). CCR is only one component in [Ara+21], while CCR consists of several components in [Swi21] and in our protocol. Rows 1-9 describe the SendVote protocol and rows 10-13 describe the ConfirmVote protocol. Rows 8-13 are not relevant for the one-round protocol by [Ara+21].

- Row 1 shows how VC encodes the voting options. While in [Swi21] the voting options are encoded to prime numbers, in [Ara+21] and in our protocol the voting options are encoded to bit messages.
- Row 2 shows how VC computes the total vote. While in [Swi21] the prime numbers are multiplied, in [Ara+21] and in our protocol a concatenation of the bit messages to a bit string is done.
- Row 3 shows what VC computes that is sent to the mixing component(s) of the system. In our voting protocol, just like in [Swi21], a ciphertext is sent. In [Ara+21], a commitment is sent. Therefore also the opening d_M, encrypted under the public key $\mathsf{pk}_{\mathsf{CCM}}$, must be sent together with the commitment.
- Row 4 shows how VC computes the partial return codes. For SPVS, the hiding of the partial return codes $\mathsf{pCC} = [p_1^{\mathsf{k}}, \ldots, p_{\psi}^{\mathsf{k}}]$ is based on ESGSP. In the protocol of [Ara+21] and in our protocol, the partial return code $\mathsf{pCC} = \rho + \mathsf{k}$ is truly uniformly random.
- Row 5 shows what VC sends to CCR. In our protocol, the partial return code pCC is sent as it is. In [Swi21], an encryption of pCC is sent. In [Ara+21], a commitment to pCC is sent. Therefore also the opening d_R, encrypted under the public key $\mathsf{pk}_{\mathsf{CCR}}$, must be sent together with the commitment.

  As the security reductions for privacy [Swi21, Section 19.4] omit the encryption of pCC, we have decided not to encrypt pCC in our protocol. If desired, pCC could be encrypted with BGV encryption just like the vote ρ. This would however result in additional proofs and a distributed decryption for the CCR components, which would make the protocol less efficient.

  In [Swi21], pCC consists of as many components as selected voting options, and a multi-encryption scheme is needed for the encryption. BGV encryption can encrypt several coefficients of a polynomial simultaneously, thus, no multi-encryption scheme would have been needed if we were to encrypt pCC in our protocol.
- Row 6 shows which proofs VC must provide as part of the ballot.
- Row 7 shows for [Swi21] and our protocol how each CCR component compute a return code share $\mathsf{lCC}_{\mathsf{j}}$. For [Ara+21], row 7 shows how the one single CCR component already computes the return codes $\mathsf{cc}^{\star}$.
- Row 8 shows which proofs each CCR component must provide together with the return code share.
- Row 9 shows how VS computes the return codes $\mathsf{cc}^{\star}$ by combining the shares from all the CCR components.
- Rows 10-13 describe the ConfirmVote protocol. Row 10 shows how VC computes the confirmation key. Row 11 shows how the CCR components compute the confirmation return code shares. Row 12 shows which proofs each CCR component must provide together with the return code shares. Row 13 shows how VS computes the confirmation return code by combining the shares from all CCR components.

As we have seen in Section 5, ElGamal can be used both as an encryption scheme and a commitment scheme. It is efficient to make ZK-proofs about ElGamal commitments. In the lattice setting, BGV encryption could also be used both as an encryption scheme and a commitment scheme and by making ZK-proofs about the BGV ciphertexts we would get verifiable encryption. However, this produces large ZK-proofs and results in an inefficient scheme. To get an efficient scheme, the BGV encryption scheme is combined with lattice commitments.

10.3 Security Analysis

VC computes proofs π_{LIN} and π_{ANEx} as part of the ballot in the SendVote protocol. These proofs leave no options for an untrustworthy VC to compute c_ρ and pCC with different values of ρ or to add too much noise in the ciphertext c_ρ. If too much noise would be added in the ciphertext, the vote could become unreadable when decrypted. Here, exact proofs of shortness are required in order to keep the overall parameters of the system low.

The CCR components must for both the SendVote and ConfirmVote protocol provide proofs π_{LIN} and π_{NEx} together with the return code shares. If VS is not able to compute a return code based on the shares from the CCR components, the proofs allow an auditor to find out which CCR component is not doing its job. Here, bounded proofs are sufficient because when $q >> p$, it is improbable that the added noise in the return code shares change the value computed by VS when rounding the sum.

Attacks against recorded-as-confirmed (vote injection and vote rejection) will either be detected by the auditors or includes guessing of correct codes and will be limited by having a large enough code space. (See choice of parameters [Swi21, Section 20.2].)

The partial return code pCC is truly uniformly random and thus perfectly hiding.

Theorem 1 (Cast-As-Intended Integrity) *Let $\mathcal{A}_0$ be an adversary against integrity of the protocol with advantage ϵ_0. Then there exist adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ against soundness for Π_{LIN} and Π_{ANEx}, respectively, with advantages ϵ_1 and ϵ_2, such that $\epsilon_0 \leq \epsilon_1 + \epsilon_2$. The runtime of $\mathcal{A}_1$ and $\mathcal{A}_2$ are essentially the same as the runtime of $\mathcal{A}_0$.*

We sketch the argument. An adversary $\mathcal{A}_0$ controlling VC attacks sent-as-intended verifiability. An adversary that includes different voting options in c_ρ and pCC is an adversary against soundness of Π_{LIN}. An adversary that adds too much noise in the ciphertext c_ρ is an adversary against soundness of Π_{ANEx}.

A successful attack against soundness of Π_{LIN} or Π_{ANEx} is also a successful attack against binding of the commitments, therefore no attacker against binding of commitments is included.

Theorem 2 (Privacy) *Let $\mathcal{A}_0$ be an adversary against privacy of the protocol with advantage ϵ_0. Suppose Π_{LIN} and Π_{ANEx} are honest-verifier zero-knowledge. Then there exists a simulator for the protocol such that for any distinguisher $\mathcal{A}_0$ for this simulator with advantage ϵ_0, there exists a distinguisher $\mathcal{A}_1$ with advantage ϵ_1 for the simulator of Π_{LIN}, a distinguisher $\mathcal{A}_2$ with advantage ϵ_2 for the simulator of Π_{ANEx}, an adversary $\mathcal{A}_3$ with advantage ϵ_3 against hiding of the commitment scheme, and an adversary $\mathcal{A}_4$ with advantage ϵ_4 against CPA-security of the encryption scheme such that $\epsilon_0 \leq \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4$. The runtime of $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ and $\mathcal{A}_4$ are essentially the same as the runtime of $\mathcal{A}_0$.*

We sketch the argument. The simulator simulates the arguments of protocols Π_{LIN} and Π_{ANEx}, replaces all commitments by random commitments and replaces the ciphertext with a random ciphertext. We replace the protocols first, and then the ciphertext at last. (If we were to replace the ciphertexts first then we would not have anything that could be proven in the remaining protocols.) First, we replace the Π_{LIN} arguments by simulated arguments, which gives us a distinguisher $\mathcal{A}_1$ for the Π_{LIN} honest verifier simulator. Second, we replace the Π_{ANEx} arguments by simulated arguments, which gives us a distinguisher $\mathcal{A}_2$ for the Π_{ANEx} honest verifier simulator. Third, we replace the commitments to the secrets and the noises by random commitments, which gives us an adversary $\mathcal{A}_3$ against hiding for the commitment scheme. Fourth, we replace the ciphertext c_ρ with a random ciphertext, which gives us an adversary $\mathcal{A}_4$ against CPA-security of the encryption scheme. After the four changes, we are left with a transcript where no secrets were used in the computations. Obviously, an adversary cannot get any information out of this transcript.

11 Performance of Our Protocol

We use equations, parameters, and computed values from [Ara+22] to compute communication size and timings of our voting protocol.

11.1 Communication Size

Sizes of ciphertexts, commitments, and the proof π_{LIN} are found in [Table 3].

Size of π_{AEx} From [Equation 2] we can compute the size of an exact amortized zero-knowledge proof of knowledge of ternary openings in terms of prover-to-verifier communication. In our protocol we prove knowledge of binary openings. By interchanging the 3τ of [Equation 2] with 2τ we achieve the following equation for the proof size of π_{AEx} of knowledge of binary openings

$$|c_d| + |r_d| + |\mathcal{M}| + (2vN + (2\tau + 4)\eta)\log_2 q + \lambda\eta(1 + \log_2 l) \text{ bits} \tag{9}$$

We use the following parameters from [Section 7.4]: $|c_d| = 256$, $|r_d| = 256$, $|\mathcal{M}| = 256$, $v = k + \ell = 3 + 1 = 4$, $N = 4096$, $\eta = 325$, $q = 2^{78}$, $\lambda = 128$, $l = 2^{20.3}$. The width v of the commitment matrix is the length of the secret vector consisting of randomness of dimension k and ℓ messages. N is the dimension of the polynomials (number of coefficients). With Equation (9) we compute the size of π_{AEx} for binary secrets and τ commitments to $(443 + 6.3\tau)$ KB.

Size of π_{NEx} We compute the proof size of π_{NEx} for only one commitment. While the π_{NEx} from [Lyu19, Section 5.2] uses values chosen from a uniform distribution we use values chosen from a Gaussian distribution like in [Ara+22].

From [Table 1] we have the standard deviation for one-time commitments $\sigma_C = 0.954 \cdot \nu \cdot \beta \cdot \sqrt{kN}$. The factor 0.954 is used to control the rejection rate of the proofs. ν is the number of non-zero coefficients of the challenge polynomial, as defined in Section 5.3. $\beta\sqrt{kN}$ is the maximal 2-norm of the randomness vector of k polynomials each with N coefficients of maximal absolute value β.

The standard deviation σ_{NEx} of the commitment of the one-time proof π_{NEx} must be chosen to hide both the commitment randomness and the message. We need not only the $k = 3$ polynomials of the commitment randomness r to be short, but the also the message polynomial m must be short. Therefore, we use $v = k + 1$ in our equation:

$$\sigma_{\text{NEx}} = 0.954 \cdot \nu \cdot \beta \cdot \sqrt{vN} \tag{10}$$

From [Table 2] we have $\nu = 36$ and $N = 4096$. We use $\beta = 1$. With Equation (10) we compute $\sigma_{\text{NEx}} \approx 2^{12}$.

To hide both the randomness and the message we sample a vector with $v = 4$ polynomials each with $N = 4096$ coefficients from a Gaussian distribution with the standard deviation σ_{NEx}. This gives us the following proof size for π_{NEx}:

$$v \cdot N \cdot \log_2(6 \cdot \sigma_{\text{NEx}}) \text{ bits} \tag{11}$$

We use the factor 6 because Gaussian values with very high probability are within +/- 6 times the standard deviation. With Equation (11) we compute the proof size of π_{NEx} to 240631 bits $\approx$ 30 KB.

	Ciphertext	Commitment	π_{LIN}	π_{AEx}	π_{NEx}
Size	80 KB	80 KB	17.5τ KB	$(443 + 6.3\tau)$ KB	30 KB

Table 3: Size of the ciphertexts, commitments and proofs. τ is the number of commitments.

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

Total Communication Size For the SendVote protocol we achieve a communication of 1095 KB from VC, and 145 KB from each CCR component. For the ConfirmVote protocol we achieve a communication of 145 KB from each CCR component. As a concrete example having four CCR components the communication size of SendVote is 1.7 MB, the communication size of ConfirmVote is 0.6 MB, and the total communication size of the two-round voting phase is 2.3 MB.

11.2 Communication Timings

We use timings of cryptographic operations to encrypt and commit from [Table 4]. The timings of protocols Π_{LIN}, Π_{ANEx}, Π_{AEx} from [Table 5] are average protocol timings per commitment when the protocols are run with an input of 1000 commitments. For our protocol, we need protocol timings per commitment when the input is only 1-5 commitments. We can expect the average timing per commitment to be smaller when the input is smaller. For the protocol Π_{LIN} we use the given timings directly as a pessimistic guess. We assume the timings of Π_{NEx} are at most the given timings of Π_{ANEx} although not amortized. The timings of Π_{AEx} from [Table 5] are by far the most expensive with 1009τ ms for Π_{AEx} and 20τ ms for Π_{AExV}. By contacting the authors of [Ara+22] we received the following timings for an input of 10 commitments: 90τ ms for Π_{AEx} and 60τ ms for Π_{AExV}.

	Encrypt	Commit	$\Pi_{\text{LIN}} + \Pi_{\text{LINV}}$	$\Pi_{\text{NEx}} + \Pi_{\text{NExV}}$	$\Pi_{\text{AEx}} + \Pi_{\text{AExV}}$
Time	2.5 ms	0.45 ms	$(10.7 + 15.7)\tau$ ms	$(30 + 25)$ ms	$(90 + 60)\tau$ ms

Table 4: Timings of cryptographic operations and protocols. τ is the number of commitments.

Protocol Operations For the SendVote protocol, VC computes 1 ciphertext, 5 commitments, π_{LIN} for 8 commitments, and π_{AEx} for 5 commitments. Each CCR component verifies the proofs π_{LIN} for 8 commitments, and π_{AEx} for 5 commitments, then 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.

Total Timings For the SendVote protocol we achieve a timing of 498 ms for VC and 404 ms for each CCR component, including verifying the proofs from VC. The components of CCR compute this in parallel. This results in a total timing of 902 ms. For the ConfirmVote protocol we achieve a timing of 65 ms for each CCR component, which they compute in parallel. In total, the communication timing for the two-round protocol is less than 1 s.

11.3 Discussion

We have neglected sizes of simple elements like the partial return code pCC and the return codes cc*. We have neglected timings of standard operations like hashing and checking mapping tables. The ring-elements and the proofs are much more size- and time-consuming.

The communication size quickly scales up for a real-world election including many voters. With 100 000 voters the total proof size from the voting phase is about 230 GB. Auditors would need to download this to their verifiers.

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 timing. Because exact proofs keep the overall parameters of the system low, they are to prefer 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.

12 Concluding Remarks

We have presented an approach for creating return codes for lattice-based electronic voting, extending the framework by Aranha et al. [Ara+21; Ara+22]. For a voting system with four control components and two-round communications our scheme results in a total of 2.3 MB of communication per voter, taking less than 1 s of computation. The presented voting phase uses the same trust assumptions of [Swi21], allowing for an untrustworthy voting server while assuming one out of several control components to be trustworthy.

The Swiss Post voting protocol [Swi21], and other electronic voting protocols in use or planned for use in elections are all based on discrete log-type assumptions. Constructing a system based on lattice-assumptions is motivated by the potential future threat by quantum computers, against future integrity of voting systems, and privacy of votes cast today.

As we have seen in Section 8.4.4, the partial return codes used in [Swi21] are for privacy the weakest part of the cryptographic protocol, with or without quantum computers. These partial return codes could be made uniformly random, like in our voting protocol presented in Section 10, and therefore not be 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 also be post-quantum secure to achieve long-time privacy of the voting system. Therefore, when constructing a post-quantum secure voting system, not only the tally phase as described by [Ara+22] must be considered, but also the voting phase, which we have described in this thesis.

Together, the shuffle and the decryption protocols by Aranha et al. [Ara+21; Ara+22] and the return codes presented here can be used to build a post-quantum secure cryptographic voting scheme offering privacy, individual verifiability, and universal verifiability.

Bibliography

[ALS20] Thomas Attema, Vadim Lyubashevsky and Gregor Seiler. 'Practical product proofs for lattice commitments'. In: *Annual International Cryptology Conference*. Springer. 2020, pp. 470–499.

[Ara+21] Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, Tjerand Silde and Thor Tunge. 'Lattice-Based Proof of Shuffle and Applications to Electronic Voting'. In: *CT-RSA*. San Francisco, CA, USA, 2021. ISBN: 978-3-030-75538-6.

[Ara+22] Diego F. Aranha, Carsten Baum, Kristian Gjøsteen and Tjerand Silde. *Verifiable Mix-Nets and Distributed Decryption for Voting from Lattice-Based Assumptions*. Cryptology ePrint Archive, Paper 2022/422. 2022. URL: https://eprint.iacr.org/2022/422.

[Bau+18] Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner and Chris Peikert. 'More efficient commitments from structured lattice assumptions'. In: *International Conference on Security and Cryptography for Networks*. Springer. 2018, pp. 368–385.

[BBS03] Mihir Bellare, Alexandra Boldyreva and Jessica Staddon. 'Randomness re-use in multi-recipient encryption schemeas'. In: *International Workshop on Public Key Cryptography*. Springer. 2003, pp. 85–99.

[BG92] Mihir Bellare and Oded Goldreich. 'On defining proofs of knowledge'. In: *Annual International Cryptology Conference*. Springer. 1992, pp. 390–420.

[BGV14] Zvika Brakerski, Craig Gentry and Vinod Vaikuntanathan. '(Leveled) fully homomorphic encryption without bootstrapping'. In: *ACM Transactions on Computation Theory (TOCT)* 6.3 (2014), pp. 1–36.

[BHM20] Xavier Boyen, Thomas Haines and Johannes Müller. 'A verifiable and practical lattice-based decryption mix net with external auditing'. In: *European Symposium on Research in Computer Security*. Springer. 2020, pp. 336–356.

[Blu83] Manuel Blum. 'How to exchange (secret) keys'. In: *AcM Transactions on computer systems (Tocs)* 1.2 (1983), pp. 175–193.

[CMM19] Núria Costa, Ramiro Martınez Pinilla and M Paz Morillo Bosch. 'Lattice-based proof of a shuffle'. In: *FC 2019 International Workshops, VOTING and WTSC, St. Kitts, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers*. 2019, pp. 330–346.

[CP92] David Chaum and Torben Pryds Pedersen. 'Wallet databases with observers'. In: *Annual international cryptology conference*. Springer. 1992, pp. 89–105.

[ElG85] Taher ElGamal. 'A public key cryptosystem and a signature scheme based on discrete logarithms'. In: *IEEE transactions on information theory* 31.4 (1985), pp. 469–472.

[FS86] Amos Fiat and Adi Shamir. 'How to prove yourself: Practical solutions to identification and signature problems'. In: *Conference on the theory and application of cryptographic techniques*. Springer. 1986, pp. 186–194.

[FWK21] Valeh Farzaliyev, Jan Willemson and Jaan Kristjan Kaasik. *Improved Lattice-Based Mix-Nets for Electronic Voting*. Cryptology ePrint Archive, Report 2021/1499. https://ia.cr/2021/1499. 2021.

[Gjø11] Kristian Gjøsteen. 'The Norwegian internet voting protocol'. In: *International Conference on E-Voting and Identity*. Springer. 2011, pp. 1–18.

[GMR89] Shafi Goldwasser, Silvio Micali and Charles Rackoff. 'The knowledge complexity of interactive proof systems'. In: *SIAM Journal on computing* 18.1 (1989), pp. 186–208.

[Hai+20] Thomas Haines, Sarah Jamie Lewis, Olivier Pereira and Vanessa Teague. 'How not to prove your election outcome'. In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 644–660.

[HMS21] Javier Herranz, Ramiro Martınez and Manuel Sánchez. 'Shorter Lattice-Based Zero-Knowledge Proofs for the Correctness of a Shuffle'. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2021, pp. 315–329.

[HT15] J Alex Halderman and Vanessa Teague. 'The new south wales ivote system: Security failures and verification flaws in a live online election'. In: *International conference on e-voting and identity*. Springer. 2015, pp. 35–53.

[Lyu19] Vadim Lyubashevsky. *Basic Lattice Cryptography: Encryption and Fiat-Shamir Signatures*. 2019. URL: https://drive.google.com/file/d/1JTdW5ryznp-dUBBjN12QbvWz9R41NDGU/view (visited on 28th June 2022).

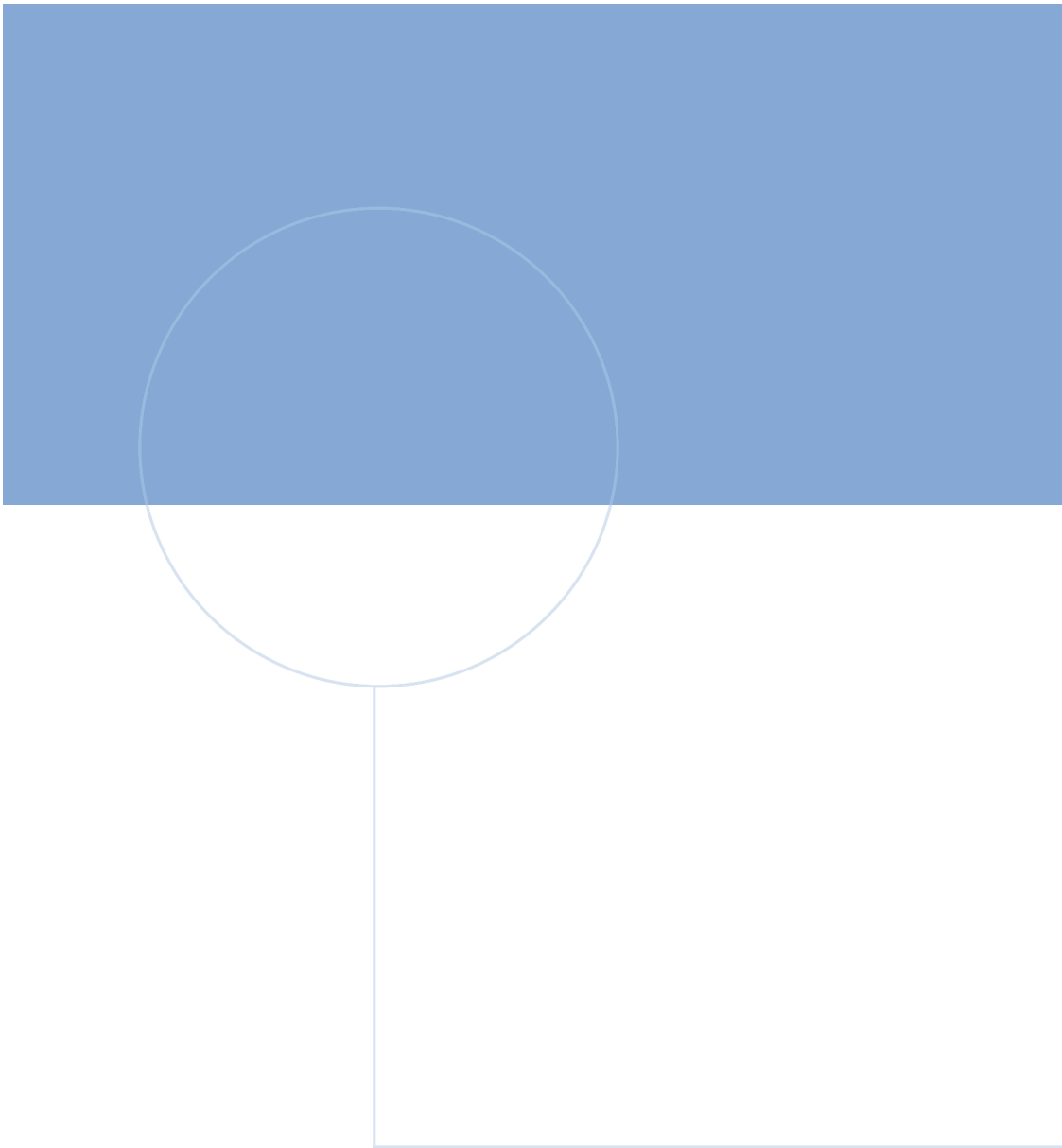
[Ped91] Torben Pryds Pedersen. 'Non-interactive and information-theoretic secure verifiable secret sharing'. In: *Annual international cryptology conference*. Springer. 1991, pp. 129–140.

[Per21] Olivier Pereira. *Individual Verifiability and Revoting in the Estonian Internet Voting System*. Cryptology ePrint Archive, Paper 2021/1098. https://eprint.iacr.org/2021/1098. 2021. URL: https://eprint.iacr.org/2021/1098.

[SKW20] Michael A Specter, James Koppel and Daniel Weitzner. 'The ballot is busted before the blockchain: A security analysis of voatz, the first internet voting application used in us federal elections'. In: *29th {USENIX} Security Symposium*. 2020, pp. 1535–1553.

[Spr+14] Drew Springall, Travis Finkenauer, Zakir Durumeric, Jason Kitcat, Harri Hursti, Margaret MacAlpine and J Alex Halderman. 'Security analysis of the Estonian internet voting system'. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 2014, pp. 703–715.

[Swi21] SwissPost. *Protocol of the Swiss Post Voting System – Computational Proof of Complete Verifiability and Privacy – Version 0.9.11*. 2021. URL: https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/97c83a77c9ebda4c3a47fca022c60cbcb006d452/Protocol/Swiss_Post_Voting_Protocol_Computational_proof.pdf (visited on 24th Jan. 2022).



NTNU

Norwegian University of
Science and Technology