Individual Verifiability in Electronic Voting

Sandra Guasch Castelló

Universitat Politècnica de Catalunya

Supervisor: Paz Morillo Bosch

Contents

Acknowledgements **7**

Preface **9**

1 Introduction **11**
1.1 Requirements of electronic voting . . . 12
1.2 Electronic voting basics . . . 13
1.2.1 Basic approach . . . 13
1.2.2 *Pollsterless* or code voting . . . 14
1.2.3 Two agencies model . . . 16
1.2.4 Homomorphic tally systems . . . 17
1.2.5 Mixnet-based systems . . . 18
1.3 Cryptography introduction . . . 20
1.3.1 Symmetric key encryption schemes . . . 20
1.3.2 Public key encryption schemes . . . 21
1.3.3 Security notions for encryption schemes . . . 23
1.3.4 Homomorphic public key cryptosystems . . . 25
1.3.5 Signature schemes . . . 25
1.3.6 σ-protocols and proof schemes . . . 26
1.4 Motivation, organization and contributions of this work . . . 31
1.4.1 Motivation . . . 31
1.4.2 Organization and contributions . . . 32

2 Individual verifiability **35**
2.1 Introduction . . . 35
2.2 Challenge-or-cast . . . 37
2.3 Verifiable Optical Scanning . . . 42
2.4 Verification with codes . . . 48
2.4.1 Code voting . . . 49
2.4.2 Return Codes . . . 52
2.5 Hardware-based verification . . . 55
2.6 Decryption-based verification . . . 57

3 Electronic Voting Model **59**
3.1 Introduction . . . 59
3.2 Protocol Syntax . . . 59
3.3 Security Definitions . . . 63

3.3.1 Trust model . . . 63
3.3.2 Ballot privacy . . . 64
3.3.3 Strong Consistency . . . 66
3.3.4 Strong Correctness . . . 67
3.3.5 Cast-as-Intended verifiability . . . 67
3.3.6 Coercion-resistant cast-as-intended . . . 69

4 Return Codes with Single Voting: Neuchâtel's Scheme 71
4.1 Introduction . . . 71
4.2 Improving the Norwegian solution . . . 72
4.2.1 Solution Overview . . . 74
4.3 Confirmation Phase . . . 75
4.4 Protocol description . . . 76
4.4.1 Workflow . . . 80
4.5 Security of the Protocol . . . 81
4.5.1 Ballot Privacy . . . 82
4.5.2 Strong Consistency . . . 84
4.5.3 Strong Correctness . . . 85
4.5.4 Cast-as-Intended Verifiability . . . 85
4.5.5 Coercion-resistant cast-as-intended . . . 86
4.6 Protocol implementation . . . 86
4.6.1 Performance . . . 87
4.7 Authentication, usability and correctness: implementation details . . . 89
4.7.1 Authentication and private keys provision . . . 89
4.7.2 Short Return Codes . . . 90
4.7.3 Vote correctness . . . 92
4.7.4 Ballot Box vs Bulletin Board . . . 93
4.8 Protocol extensions . . . 94
4.8.1 Supporting multiple entry points . . . 94
4.8.2 Distributed return code generation . . . 94
4.8.3 Support for multiple voting . . . 99
4.8.4 Assignation of verification cards . . . 99

5 Challenge-*and*-cast 101
5.1 Introduction . . . 101
5.2 Overview . . . 101
5.3 Related work . . . 102
5.4 Proof simulation . . . 103
5.4.1 A simulatable NIZK proof using chameleon hashes . . . 105
5.4.2 Simulatable NIZKPK scheme properties . . . 106
5.5 Core Protocol using Mixnets . . . 107
5.5.1 Security of the Protocol . . . 112
5.5.2 Concrete instantiation . . . 117
5.5.3 Performance . . . 119
5.6 Protocol for homomorphic tally-based systems . . . 119
5.6.1 Security Analysis . . . 120
5.6.2 Primitives . . . 120

5.7 Multiple Trustees 122
5.8 Voting Scheme 123
5.9 Protocol extension for multiple voting 125

6 Making Cast-as-Intended Universal 127
6.1 Introduction 127
6.2 Motivation 127
6.3 UCIV System description 128
6.3.1 Overview 128
6.3.2 Syntactical definition 130
6.3.3 Security definitions 132
6.4 Protocol based on NIZK proofs 136
6.4.1 Protocol description 136
6.4.2 Security analysis 138
6.4.3 Implementation 139
6.5 UCIV with return codes 140
6.5.1 Overview of the solution 140
6.5.2 Protocol description 140
6.5.3 Security Analysis 142
6.5.4 Implementation 146
6.5.5 Extension to multiple voting 147
6.6 Distributed generation of UCIV information 147
6.7 Setup 149

7 Conclusions 151

Acknowledgements

I would like to thank my family and friends, who have being very patient with me and supportive while I was writing this thesis. I would also like to thank my colleagues at work, who have provided a lot of useful comments, specially when opining about the drawings! My English teacher, Ryan Jones deserves a special mention. He bravely read all the document and reviewed my English without crying or falling asleep (at least this is what he said) in a week. He is the best! A special thanks to my supervisor Paz Morillo, who is the most dynamic person I know, who has been always confident that I could do this, and who has helped me a lot, specially during the last months. My boyfriend, Guillem, took care that I did not starve during the last months while I was working full time and also writing this thesis. Thanks a lot! Thanks also to my external reviewers, Rolf Haenni and Francesc Sebé, who provided great comments to this text. Finally, I would like to thank Jordi Puiggalí and Alex Escala, with whom I have make most of the research in this thesis. Working with you is great, and I hope I will continue doing it in the future.

Preface

This PhD Thesis is the fruit of the job of the author as a researcher at Scytl Secure Electronic Voting, as well as the collaboration with Paz Morillo, from the Department of Applied Mathematics at UPC and Alex Escala, PhD student.

In her job at Scytl, the author has participated in several electronic voting projects for national-level binding elections in different countries. The participation of the author covered from the protocol design phase, to the implementation phase by providing support to the development teams.

Since her participation on the Norwegian project (2009-2011), which pioneered the use of verifiable electronic voting systems on binding elections, the author has specialized in individual verifiability, which englobes the processes that can be done by the voter, in order to check that the system works correctly. Part of the author's work at Scytl has been to continue with the experience of the design and development of the electronic voting system used in Norway, and improve it. The result has been implemented and already been used in 2014 and 2015 elections in Neuchâtel, and it is planned to be used in the next years.

The analysis of existing protocols, as well as the design of new ones, fulfilling the requirements of the different projects executed at Scytl, made the author realise that there is not a gold solution for everyone, when it comes to individual verifiability. This is due to the fact that the trust assumptions that can be made are different in each project or setup, and that individual verifiability strongly defines how the voter is going to interact with the system. For the latter, customers / electoral commissions may have very specific requirements, for example given their traditional voting process.

Therefore, this thesis also contains proposals which did not emerge from the projects conducted at Scytl, but from the need of both the market and the academia of new protocols with new properties, and which work under different trust assumptions. These proposals are intended to be a contribution to the state of the art of individual verifiability, both in academic and industrial environments, as well as being the basis for new systems that may be implemented in the future.

In summary, this thesis has to be seen as the fruit of industrial research, while also contributing to the academic state of the art.

Chapter 1

Introduction

Electronic voting systems have been around for a long time. The first optical scanners were first used in 1962, and Direct Recording Electronic Voting Machines (DREs) were introduced in 1975. Complex, time-consuming and error-prone counting processes have been made faster, easier and more reliable with electronic mechanisms. New technologies also provide aids for disabled voters to independently cast a vote, and for regular voters to fill-in complex ballots without errors. Logistic costs derived, for example, from handling multiple-language ballots, long-term costs, or the time needed for setting up an election can also be reduced using electronic means.

With the rise of Internet, and later on the widespread use of intelligent portable devices (such as smart phones) in our daily life, we have become used to performing a large proportion of our transactions remotely and at any time. Although, in more or less measure, the existence of attacks inherent to the remote nature of the Internet is known by everybody (identity impersonation, interception or observation of our communications, website impersonation or *phishing*, access to personal data, etc.), and this fact does not inhibit for the provision of multiple online services, some of them as sensitive as banking, paying taxes or buying flight tickets. One could say that the advantages overcome the drawbacks. Therefore, the implementation of Internet voting naturally emerges, together with other e-government solutions which are aimed to bring the administration closer to the citizens, and collaborates to improve the welfare of the society. Internet voting cannot only enable increased participation due to its convenience, but also provides the opportunity of voting to collectives which may otherwise not be able to vote: people who are abroad during the election day, military located in conflict regions, embarked fishermen, or people with decreased mobility due to disabilities.

Usually, voters who cannot go to a polling station on election day can use the postal mail channel to cast their voters. A few weeks before the election, voters register to vote by mail in order to receive a ballot at the indicated address. Voters then send back the filled ballots to the corresponding poll sites or electoral centers in order to be counted with the regular ballots. However, this solution does not always work. The period before the election for defining the candidate lists and questions

to be printed in the ballot is quite narrow, and does not provide much time for the postal mail ballots to be sent to the remote voters, and to be further received on time for inclusion in the tally. Delays in the postal voting channel are regular news in almost every election, and by personal experience it can be said that, even if the voter is in the same country, this is not a guarantee that the postal ballot will be received at all. Besides the adjusted timings of the electoral processes, the quality of the postal channel may vary depending on the location of the voters: i.e., military in conflict regions may not regularly receive their postal mail. As an example of the criticality of this issue, it motivated the laws related with the *Military and Overseas Voter Empowerment Act* (MOVE) in the United States in 2009 [53], which forced the States to provide electronic voting mechanisms to military and citizens abroad in order to request and receive their ballot papers.

Finally, it is an opinion of the author of this thesis that a democracy cannot consist of politicians asking the citizens about their opinion every four years, and then getting *carte blanche* without having to account for their actions. A democracy should rely on tighter relations and communication between voters and their representatives, and Internet voting is a perfect tool for providing it.

1.1 Requirements of electronic voting

Although most people may compare electronic voting with other complex and sensitive systems, such as electronic banking, there are some important differences, pointed out in [9], which make more particular and complex the problem of secure electronic voting. First of all, the interests for manipulating an election may be greater than what we can imagine: people may be very well disposed to paying a lot of money for achieving a specific outcome. Second, failure detection and recovery: due to the nature of elections (for example, the secrecy of the vote), it may be possible that some failure in the system is not detected. Anyhow, in case of detection of a systemic failure, *we cannot return the money*, and therefore the elections should be repeated, which may result in a voter's loss of confidence in the system. The third main difference is the nature of requirements of electronic voting, which may often seem contradictory and therefore be hard to be entirely fulfilled.

Usually, the requirements for remote electronic voting mimic those for traditional elections:

- Vote authenticity: it has to be ensured that the votes are cast by eligible voters, and that only one vote per voter is counted.

- Voter privacy: at the same time that voters have to be identified in order to verify their eligibility, the link between a voter and her choices has to remain secret.

- Tally accuracy: the result of the election must accurately reflect the contents of the votes cast by the voters. Therefore it cannot be possible to modify or erase valid votes, or add fake votes on behalf of voters who have abstained.

- Secrecy of intermediate results or election fairness: no intermediate results can be provided before the end of the election, in order to prevent a bias on the voters who have not voted yet.

- Verifiability: the electronic voting system has to provide methods for verifying that it is working as expected. A voter has to be able to verify that her cast vote represents her intention of vote and that it has been taken into account in the tally process. An auditor has to be able to verify that all the votes cast by eligible voters, and only those, are included in the tally.

- Traceability and accountability: all the operations of the system have to leave traces which allow inspection of whether the operation was correct. In case of any malfunction, it has to be possible to identify the responsible entity.

- No coercion or vote-selling: a voter cannot prove to a third party how she voted.

As we have said before, we can see that some of these requirements seem contradictory at a first glance: we want to be able to identify the voter who cast a vote, but we do not want to be able to relate vote and voter. We want the voter to be able to verify the content of her vote, without being able to prove this content to a third party. The nature of these contradictory requirements, as well as the environment of execution of the voting system (uncontrolled voting devices, online servers, digital and unobservable processes, ...), make electronic voting an interesting field of application for cryptographic protocols.

In the next section, we provide a brief introduction of the main types of cryptographic protocols used in remote electronic voting systems, and explain how they fulfill some of the listed requirements.

1.2 Electronic voting basics

1.2.1 Basic approach

A basic approach for an electronic voting scheme is to combine encryption and digital signature schemes: Encryption schemes are used for providing secrecy of information transmitted among two parties, in front of external observers. Signature schemes are used in order to ensure the integrity of the transmitted messages, as well as providing assurance of the origin of such messages. This means that an external entity cannot modify or forge a message without being detected by the intended receiver. A more detailed explanation on encryption and signature schemes can be found in Section 1.3.

In this basic approach, voters encrypt their messages prior to casting them, in such a way that only the intended recipient - the electoral board, or the electoral commission - is able to decrypt them and see their content. After encryption and prior to casting, voters also digitally sign their votes, in order to prove later on to

the election authorities that they have been cast by eligible voters. This approach is similar to the traditional process in which a voter who casts her vote by postal mail digitally signs the outer envelope of her vote. Digital signatures allow identification of the voter who casts a vote, and therefore can also be used in order to discern whether a voter tries to cast a vote twice. Also in a similar way as in postal voting, outer envelopes are removed after verification of the signature, and prior to the recovery of the cleartext vote by decryption. Therefore, a cleartext vote cannot be connected to a voter's identity.

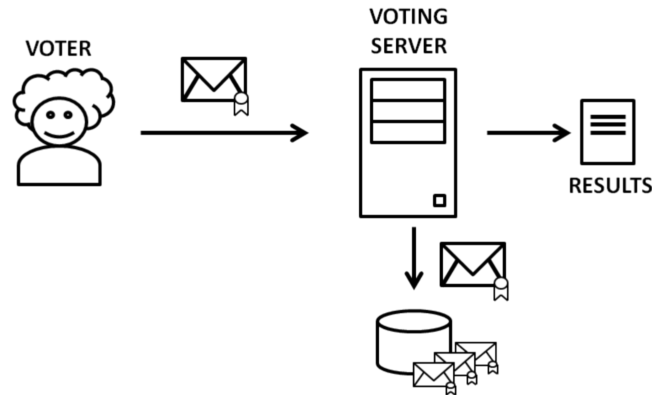

Figure 1.1: Basic approach: encrypt & sign

The security measures based on vote encryption and digital signatures seem enough to protect voters' privacy. However, these measures are only efficient during the voting process. During the election tally, decrypted votes could still be correlated with the voters who submitted them, by checking the order in which votes are decrypted: decrypted votes can be correlated to the voter identities by checking the digital signature of the encrypted votes stored in the ballot box in the same order.

Therefore, encrypting and signing is not enough, and more advanced cryptographic protocols have to be used. The most common types of electronic voting protocols are *Pollsterless* or code voting, the two-agencies model, homomorphic tally systems, and mixnet-based systems.

1.2.2 *Pollsterless* or code voting

The term *pollster* in electronic voting schemes was first noted by Malkhi et al. in the year 2002 [78], and it refers to the software or hardware artifact that participates in a voting protocol on behalf of the human voter. The pollster is necessary to perform the cryptographic operations from the electronic voting protocols, which the human voter is incapable of doing for herself.

In this kind of protocol [34], [118], [78], the methods to preserve voter privacy are mainly implemented in the configuration phase. During this phase, a code sheet is generated for each voter who participates in the election. These code sheets contain a voting code and a verification or return code assigned to each voting option in

the election. The voting and return codes assigned to a voting option vary across the code sheets, and have to be provided to the voter through a secure channel (for example a sealed envelope), in order to keep them secret.

In order to vote, the voter enters in her voting device the serial number of the code sheet and the voting code corresponding to the voting option she wants to vote for. The remote voting server, upon reception of this voting code, computes the corresponding return code, which is sent back to the voter. The voter finally checks that the return code received indeed corresponds in her voting card to the voting option she selected. The voter then knowns that the vote she cast was not modified, and that it was successfully processed by the remote voting server (which is the only one which can translate voting codes into return codes).

	Voting Code	**Verification Code**
Candidate A	4698	1682
Candidate B	2323	9467
Candidate C	6801	5138

Serial Number: 45672138672130

Figure 1.2: Code sheet

At the counting phase, voting codes are translated to voting options (by inverting the function that was used to create them during configuration), and then counted to obtain the election results.

From the point of view of privacy, these protocols allow the submission of anonymous votes, since these are not digitally signed by the voters, and the preservation of vote secrecy: the codes do not provide information about which candidate has been voted for, without having the code sheet.

A particular advantage of such systems is that they can be used to cast votes from devices without cryptographic capabilities (and this is why they receive the name *pollsterless*), since the assignation of voting codes to voting options (i.e., their *pre-encryption*) is done during the election configuration. Another property provided by such systems is that the voting device does not learn the voting options selected by the voter.

However, this method is not perfect: there is still a chance of breaking the voter privacy if the code sheets are disclosed. For example, an attacker could know the vote intention of a voter using the submitted codes and the code sheet. Another drawback of this kind of system is that traditionally the code sheet generation and tally processess cannot be verified. However, recent proposals, explained in Section 2.4 are aimed to improve such systems regarding verifiability.

1.2.3 Two agencies model

The two agencies model, first proposed in 1992, allows a voter to cast her vote anonymously, but at the same time checks that such voter is eligible to vote in the election. In order to do that, two server-side entities participate during the voting phase:

- The Validator Service: authenticates the voter, verifies her eligibility and allows her to vote in an anonymous way using an anonymous token.
- The Voting Service: receives encrypted votes with anonymous tokens from voters, and accept them after verifying if their tokens have been issued by the Validation Service.

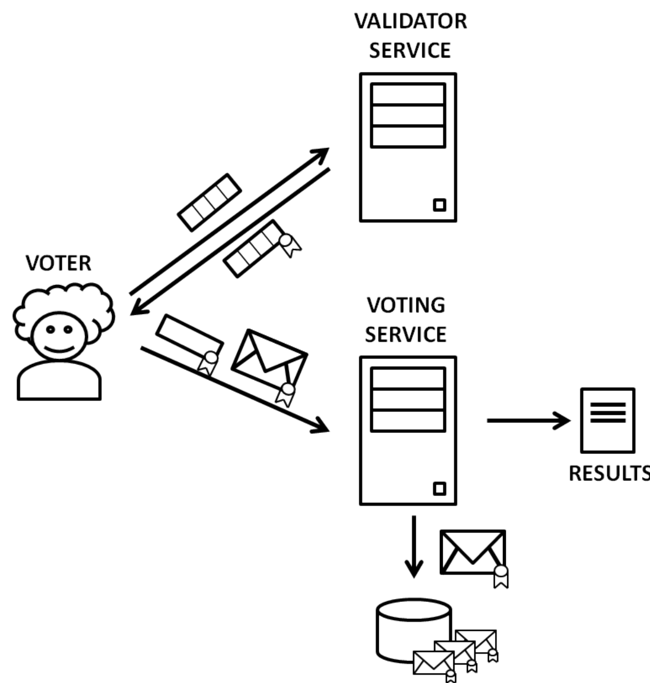

Figure 1.3: Two agencies model

This kind of scheme [56], [89] usually uses blind signatures [33]. Blind signatures allow an entity to digitally sign a message without viewing its content: the requester of the signature sends a *blind* message to the signer, who digitally signs it and returns it to the requester. The requester can then remove the *blinding factor* from the message, and obtains a digitally signed message. A very common example of blind signatures with the RSA digital signature scheme can be found in [105].

With this mechanism, the Validator Service can digitally sign the authorization token without viewing its content. The voter, after removing the blinding factor, sends the signed token to the Voting Service, which validates the token. A coalition of Validation Service and Voting Service cannot trace a token back to the voter, since the first one (who knows the identity of the voter), did not see the token in clear,

but a blind version of it. After the voting phase, votes are decrypted to perform the tally. The voters' privacy is preserved, since the votes to be decrypted are not linked to voter identities.

Even in the case of using voter signatures, voter privacy still depends on the honesty of both agencies. If they were to collaborate, they could share other information such as IP addresses, which would allow them to correlate votes with voters. Also, there is a risk of election manipulation if the Validation Service is compromised, since it could create valid tokens for non-eligible voters that would be successfully accepted by the Voting Service.

1.2.4 Homomorphic tally systems

These protocols use the homomorphic properties of some cryptosystems, by which certain operation over the encrypted votes is equivalent to the encryption of the result of an operation of the vote contents. In other words, if we have two votes v_1 and v_2, assuming that ϕ and θ are two mathematical operations, the homomorphic property of an encryption scheme can be represented by the following equivalence:

$$E(a)\ \phi\ E(b) \equiv E(a \oplus b)$$

where E denotes the encryption operation.

Depending on the definition of the two operators ϕ and $\oplus$, the homomorphism may be multiplicative (which means that the product of both encryptions results in the encryption of the product of both plaintexts) or additive (meaning that the product of both encryptions is equivalent to the encryption of the addition of both plaintexts). The additive homomorphism is generally the most commonly used in electronic voting protocols, since it generates the encrypted total sum of the votes, which is the outcome of the tally. Exponential ElGamal (explained in Section 1.3) and Paillier [90] are examples of encryption schemes which have additive homomorphic properties.

In homomorphic tally systems, such as [43], votes are encrypted in a specific format, in order to be able to obtain the tally results from the result of their operation: each voting option in the election is assigned 1 or 0, depending on whether it has been selected by the voter or not. Each value is then encrypted individualy, and therefore a vote is composed by as many ciphertexts as voting options in the election. Encrypted votes may be digitally signed prior to being cast in order to ensure their integrity during transmission and storage, and for verifying that they have been cast by eligible voters.

At the end of the election, votes stored in the ballot box are operated pair-wise, that is, the first ciphertexts of all the votes are multiplied together, the second ciphertexts are also multiplied together, and so on. The result is an aggregated ciphertext for each voting option. Then, each ciphertext is individually decrypted.

Thanks to the homomorphic properties of the encryption scheme, the result is the number of times each voting option has been selected.

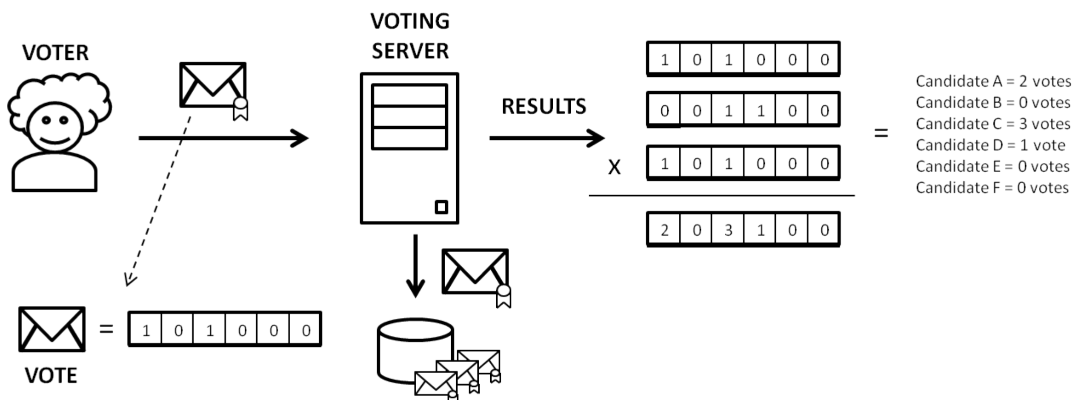

Figure 1.4: Homomorphic tally

Given that votes are not individually decrypted, the voter privacy is preserved, even in the case of using digital signatures. However, due to that, it is necessary to check that encrypted votes are well-formed. Otherwise, a voter could submit a vote where a selected voting option has been assigned 2, instead of 1, resulting in that this voting option is counted twice for the election result. Cryptographic proofs, generated by the voting device at the time of encryption, are usually used for verifying that individual votes are well-formed. Examples of such cryptographic proofs are provided in Section 1.3. The computation of such cryptographic proofs, plus the need for generating encryptions for all the possible voting options (not for those actually selected), makes these schemes practical only for elections where the number of posible choices is limited. Another limitation of this kind of scheme is that a very specific representation of the voting options has to be used, and therefore write-ins cannot be supported.

A very interesting property of these schemes is that threshold decryption techniques can be easily applied. In threshold systems, the electoral board members hold shares of the private key, from which a subset of them is needed to decrypt the votes. The key does not need to be reconstructed for decryption, but each electoral board member can make a partial decryption with his share of the key, and then all the partially decrypted values can be combined together in order to obtain the plaintext. Given that only an aggregated ciphertext for each voting option has to be decrypted, and not each individual vote, the extra workload of making multiple partial decryptions, instead of one single decryption, can be acceptable in such kind of systems. This partial decryption process is described in [43].

1.2.5 Mixnet-based systems

These proposals are based on imitating the process done in traditional elections where, at the end of the voting phase, the ballot boxes are shuffled in order to break

the storage correlation order of the votes (which could lead to the identity of the voters who cast the votes). Once the correlation between voter and vote has been broken, votes can be decrypted in order to obtain the election results.

In these protocols, voters cast encrypted and digitally signed votes which are stored in the ballot box until the end of the voting phase. Then, the votes are detached from their signatures and passed through a mix-net [32], which is composed of several nodes which shuffle the votes sequentially using a secret permutation. The purpose of the mix-net is to output votes which cannot be linked with those that were stored in the ballot box, originally signed by the voters.

The encrypted votes to be passed through a mix-net are usually encrypted using a probabilistic encryption scheme, in which some random values are used for generating the ciphertexts. The result is that each ciphertext is unique with a high probability, and therefore, votes at the output of the mix-net can be easily connected with the votes at the input by comparing their values, breaking the purpose of the mix-net. Therefore, each mix-node applies, in addition to the permutation, a transformation over its input ciphertexts, the result of which cannot be related to the original values.

There are two kinds of mix-nets:

Decryption mix-nets: Votes are encrypted in several layers (as many as nodes in the mix-net), using in each layer the key from the corresponding node. When encrypted votes are provided to the mix-net, each node permutes the input encrypted votes and uses its key to remove the outer encryption layer. This process is repeated at each node until it reaches the last one, where the last encryption layer is removed and the original vote contents are obtained.

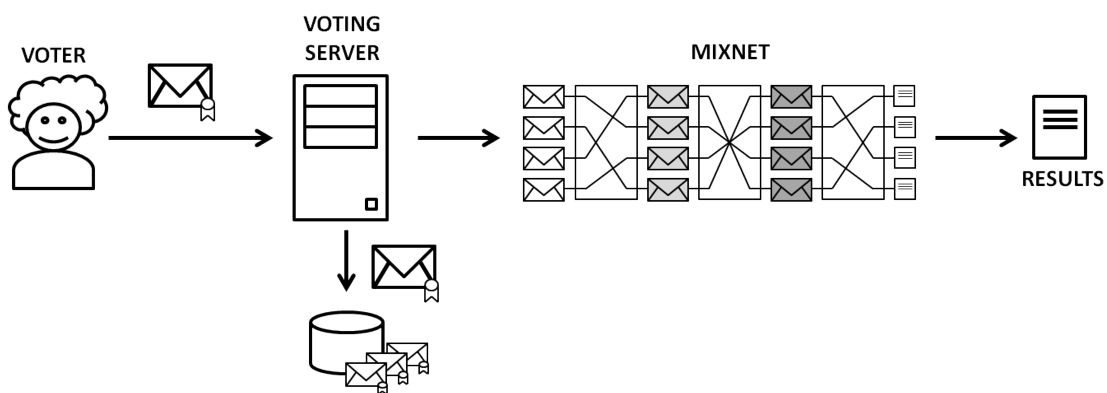

Figure 1.5: Decryption mixnet

Re-encryption mix-nets: Votes are encrypted using an encryption scheme which allows re-encryption or re-randomization of the ciphertexts multiple times, while only one decryption step is needed to recover the plaintexts. Each node, in turn,

permutes the input encrypted votes and re-encrypts / re-randomizes them in order to *make them look* totally different than in the input. Finally, a decryption step is done in the last node of the mix-net in order to recover the plaintexts.

Due to the fact that the mix-net modifies the output votes in such a way that they cannot be related to those at the input, it may easily erase and insert votes without detection. Therefore, verification methods have to be put in place in order to ensure that the mix-net behaves properly. Verifiable mix-nets are mix-nets which provide mathematical (cryptographic) proofs which demonstrate that they do not modify the processed votes during the mixing process. These proofs are designed in such a way that they do not rely on providing secret information, as the secret permutation or private keys, for proving their correct behavior. Instead, they use zero-knowledge proofs which can be verified using public information (an explanation of zero-knowledge proofs can be found in Section 1.3). Some of the most known and efficient verifiable mixnets are Randomized Partial Checking [73], Verificatum or Douglas Wikström's *Commitment-Consistent Proof of a Shuffle* [121], or the Bayer-Groth's *Efficient zero-knowledge argument for correctness of a shuffle* [13].

The main benefits of these protocols are that they can use more flexible encryption schemes than homomorphic tally protocols; they support write-ins; and they provide a better support for complex electoral processes.

In this section, we have presented the four basic types of protocols, and explained how they provide voter privacy, while ensuring that only eligible voters can cast votes. There are many variants of these four main families, which provide additional and very interesting properties, such as JCJ with coercion-resistance [75]. There are even hybrid systems which combine several of these schemes, as in the case of hybrid mix-nets [93], [94], which use a combination of homomorphic tally with mixing schemes. However, in this work we are going to focus on systems which provide voter verification methods or individual verifiability. Therefore, in Section 2 we provide a more detailed state of the art focused in these kind of systems.

1.3 Cryptography introduction

Here we provide an introduction to cryptography, in order to provide some basic notions to the reader, which will be useful for understanding the rest of this document.

1.3.1 Symmetric key encryption schemes

When Alice and Bob want to maintain a private conversation, they use an encryption scheme in order to hide their messages from a third-party. Messages, or plaintexts, are converted into ciphertexts by means of an encryption process. Without the decryption key, the original plaintext cannot be recovered.

In a symmetric key encryption scheme, Alice and Bob share a secret key. Alice uses the secret key to encrypt a message for Bob. When Bob receives the encrypted message, he uses the secret key to decrypt it and recover the plaintext. When Bob wants to send a message to Alice, they do the same process.

A symmetric key encryption scheme is defined by the following algorithms:

- The key generation algorithm $\mathsf{KGen}_\mathsf{e}^\mathsf{s}$ receives as input a security parameter 1^λ and outputs a secret key k from the key space $\mathcal{K}_{sp}$.
- The encryption algorithm Enc^s takes as input a message $m \in \{0,1\}^\lambda$ and a key $k \in \mathcal{K}_{sp}$, and produces a ciphertext $c \in \{0,1\}^\lambda$.
- The decryption algorithm Dec^s takes as input a ciphertext $c \in \{0,1\}^\lambda$ and a key $k \in \mathcal{K}_{sp}$, and produces a decrypted message $m \in \{0,1\}^\lambda$.

A symmetric key encryption scheme is said to provide correct decryption if for any key $k \in \mathcal{K}_{sp}$, given a set of messages $m_1, m_2, \ldots, m_n \in \{0,1\}^\lambda$ and a set of ciphertexts $c_1 = \mathsf{Enc}^\mathsf{s}(m_1, k)$, $c_2 = \mathsf{Enc}^\mathsf{s}(m_2, k)$, ... , $c_n = \mathsf{Enc}^\mathsf{s}(m_n, k)$, it is fulfilled that $\mathsf{Dec}^\mathsf{s}(c_i, k) = m_i$, for $i = 1, \ldots, n$.

Usually symmetric encryption schemes are combinations of a block cipher (i.e. DES [82] -deprecated-, Triple-DES [86] -legacy-, AES [83], Camellia [12],...) and a mode of operation (ECB, CBC, CTR [84], GCM [85], ...).

1.3.2 Public key encryption schemes

Symmetric key encryption schemes, or symmetric key schemes in general, have the problem of key distribution: how did Alice and Bob agree on a secret key for exchanging secret messages? Obviously, they could not use an encrypted channel because they needed a key!

Public key cryptography emerged to solve the drawback of key distribution in symmetric cryptography. It was first suggested by Diffie and Helman in 1976 [49], and first implemented in the RSA public key cryptosystem by Rivest, Shamir and Adleman [103] in 1978.

In a public key encryption scheme, Alice and Bob have, each one, a pair of keys: one public and known by everyone, and one private. When Alice wants to communicate with Bob, she uses Bob's public key for encrypting a message for him, and Bob in turn uses his private key for decrypting it and reading the content. When Bob wants to send a message to Alice, he uses Alice's public key to encrypt it, and Alice uses her private key for decrypting and recovering the original plaintext. Using this kind of scheme, there is no need for Alice and Bob to exchange a secret before establishing a private channel by means of encryption. In fact, a public key encryption scheme is usually used just at the beginning of the communication, in order to exchange the secret key to be used for encrypting the following messages.

Public key encryption is commonly limited to the communication of secret keys, due to the fact that symmetric cryptography has a better performance.

Formally, a public key encryption scheme is defined by the following algorithms:

- The key generation algorithm $\mathsf{Gen_e}$ receives as input a security parameter 1^λ and outputs a key pair composed by a public key pk_e and a private key sk_e, defines a message space $\mathcal{M}_{sp}$, a ciphertext space $\mathcal{C}_{sp}$ and a randomness space $\mathcal{R}_{sp}$ (in case of a probabilistic encryption scheme).
- The encryption algorithm Enc takes as input a message $m \in \mathcal{M}_{sp}$ and a public key pk_e, and computes a ciphertext $c \in \mathcal{C}_{sp}$. In case the algorithm is probabilistic, it uses random values $r \in \mathcal{R}_{sp}$ for computing such ciphertext.
- The decryption algorithm Dec receives as input a ciphertext $c \in \mathcal{C}_{sp}$ and a private key sk_e, and outputs a message $m \in \mathcal{M}_{sp}$ or $\perp$ in case of error.
- Some encryption schemes also have a ciphertext verification algorithm $\mathsf{EncVerify}$, which receives as input a ciphertext c and a public key pk_e, and outputs 1 if the ciphertext is correct, 0 otherwise.

A public key encryption scheme is said to be correct if for any key pair $(pk_\mathsf{e}, sk_\mathsf{e}) = \mathsf{Gen_e}(1^\lambda)$, any sequence of messages $(m_1, m_2, \ldots, m_n) \in \mathcal{M}_{sp}$ and the sequence of ciphertexts $c_1 = \mathsf{Enc}(m_1, pk_\mathsf{e})$, $c_2 = \mathsf{Enc}(m_2, pk_\mathsf{e})$, $\ldots$, $c_n = \mathsf{Enc}(m_n, pk_\mathsf{e})$, it is fulfilled that $\mathsf{Dec}(c_i, sk_\mathsf{e}) = m_i$ and $\mathsf{EncVerify}(c_i, pk_\mathsf{e}) = 1$ for $i = 1, \ldots, n$.

RSA and ElGamal are examples of public key encryption schemes:

RSA: The RSA encryption scheme [103] was the first implementation of a public key encryption scheme. It is defined as follows:

- The key generation algorithm $\mathsf{Gen_e}$ receives two primes p, q of similar bit-length $(\lambda/2)$ (which define a ring $\mathbb{Z}/n\mathbb{Z}$) and computes the public key $pk_\mathsf{e} = (n, e)$, where $n = pq$ and e is coprime with $\phi(n)$ ($\phi(n)$ denotes the *Euler totient function*, and in this case it is computed as $(p-1)(q-1)$). The private key sk_e takes the value of d, where $ed = 1 \bmod \phi(n)$.
- The Enc algorithm receives as input $m \in \mathbb{Z}_n$, such that $gcd(m, n) = 1$ and the public key pk_e, and computes $c = m^e \bmod n$.
- The Dec algorithm receives c and the private key sk_e and outputs $m = c^d \bmod n$.

The strength of the RSA cryptosystem is given by the hardness of solving the RSA problem, which is related to the problem of factoring large composite integers. Usually, raw RSA encryption is not used, but a padding is added to the message prior to being encrypted. Current standards recommend the use of RSA-OAEP [104], [17] or RSA-KEM [71] versions.

ElGamal: The ElGamal encryption scheme was defined by Taher ElGamal in 1985 [51]. In this scheme, the public key encryption algorithms ($\mathsf{Gen_e}, \mathsf{Enc}, \mathsf{Dec}$) are defined as follows:

- The key generation algorithm $\mathsf{Gen_e}$ takes as input a subgroup $\mathbb{G}$ which has a generator g of order q of elements in $\mathbb{Z}_p^*$, where p is a safe prime such that $p = 2q + 1$ and q is a prime number. It outputs an ElGamal public/secret key pair $(pk_\mathsf{e}, sk_\mathsf{e})$, where $pk_\mathsf{e} \in \mathbb{G}$ such that $pk_\mathsf{e} = g^{sk_\mathsf{e}} \mod p$ and $sk_\mathsf{e} \in \mathbb{Z}_q$.
- The encryption algorithm Enc receives as input a message $m \in \mathbb{G}$ and a public key pk_e, chooses a random $r \in \mathbb{Z}_q$ and computes $(c_1, c_2) = (g^r, pk_\mathsf{e}^r \cdot m)$.
- The decryption algorithm Dec receives (c_1, c_2) and the private key sk_e and outputs $m = c_2/c_1^{sk_\mathsf{e}}$.

The security of the ElGamal encryption scheme relies on the hardness of solving the Discrete Logarithm problem: Let $\mathbb{G}$ be a finite cyclic group of prime order q, and let $g \in \mathbb{G}$ be a generator. Given $h \in \mathbb{G}$, the discrete logarithm problem consists on computing $x \in \mathbb{Z}_q$ such that $g^x = h$.

1.3.3 Security notions for encryption schemes

Security of cryptographic algorithms is analyzed on the basis of a set of possible goals an attacker wants to achieve in respect to the algorithm, and a set of possible attack models, which define the capabilities of the atacker. The security notions defined for encryption schemes are *ciphertext indistinguishability*, by Goldwasser and Micali [63], and *non-malleability*, due to Dolev, Dwork and Naor [50].

Let m be a plaintext and c the ciphertext resulting of its encryption. **Indistinguishability** (IND) formalizes a strong notion of privacy in which an attacker is unable of learning any information about the plaintext m, given the ciphertext c. Specifically, an attacker who is given two messages and the corresponding ciphertexts cannot distinguish to which plaintext corresponds each one of the ciphertexts. **Non-malleability** (NM) refers to the resistance of ciphertexts against tampering, and formalizes the inability of an attacker to modify the ciphertext c, in such a way that the underlying plaintext m' keeps a determined relation with m.

The following attack models are considered, ordered by increasing strength: *chosen-plaintext attack* (CPA), *non-adaptive chosen-ciphertext attack* (CCA1) and *chosen-ciphertext attack* (CCA2).

CPA In the CPA model, the adversary can obtain ciphertexts from plaintexts of his choice. In public key encryption schemes, this is achieved by the adversary having access to the encryption public key.

Semantic security, defined by Goldwasser and Micali in [62], was found to be equivalent to IND-CPA. Given an encryption public key, the adversary chooses two

plaintexts and is then presented with a ciphertext corresponding to one of these plaintexts, chosen at random. In a IND-CPA secure public key encryption scheme the adversary cannot guess to which plaintext the ciphertext corresponds with more than 50% probability of success.

A NM-CPA secure scheme is known to be also IND-CPA secure.

CCA1 In the CCA1 model [81], the adversary, besides having access to the public key, also has access to a decryption oracle which he can query until just before the challenge ciphertext (the one the adversary has to attack) is provided. Queries to the decryption oracle cannot depend on the challenge ciphertext. This attack model or scenario is also called the *Lunchtime Attack*, which refers to the scenario where the adversary has access to an unattended worker's computer (the decryption oracle) only during lunchtime, and has to use the information he has learned for attacking the worker's communications at a later time.

CCA2 In the CCA2 model [102], the adversary has access (again besides of to the public key) to a decryption oracle, which can be queried before and after the adversary has received the challenge ciphertext. Therefore, the adversary can make queries related to such challenge. The only restriction is that the adversary cannot ask for the decryption of the challenge ciphertext itself. This is the strongest security notion. In this scenario, IND and NM are equivalent.

Raw RSA is known not to be IND-CPA secure given that the encryption, without using random values, is a deterministic computation. RSA-OAEP and RSA-KEM use randomness in the encryption, and have been proven to be IND-CCA2 secure in the random oracle model [57], [115].

The ElGamal encryption scheme is IND-CPA secure as long as the Decisional Diffie-Hellman assumption holds for the underlying cyclic group $\mathbb{G}$. The Decisional Diffie-Hellman assumption states that, given a cyclic group $\mathbb{G}$, a generator g of $\mathbb{G}$, the following two distributions $(\mathbb{G}, g, g^a, g^b, g^{ab})$, $(\mathbb{G}, g, g^a, g^b, g^c)$ are indistinguishable.

The Random Oracle Model (ROM) defines an algorithm that simulates a random function $D \rightarrow C$, to which all the algorithms may call as an *oracle* $\mathcal{O}$ [16]. It is usually modeled as a table $\mathcal{T}$, which at the beginning is empty. Every time an algorithm makes a call $\mathcal{O}(d)$, the oracle checks if there is already an entry (d, c) in (T). If so, it returns c. Otherwise, it picks c' at random from C, adds the entry (d, c') to the table and returns c'. In the random oracle model, hash functions are modeled as random functions (for example, as we will see later, this is used to prove the soundness of zero-knowledge proofs when the Fiat-Shamir heuristic is used for making them non-interactive). The random oracle can be programmed by simulation algorithms, which add entries (d, c) to the table $\mathcal{T}$. This is used, for example, for the simulation of NIZK proofs.

1.3.4 Homomorphic public key cryptosystems

A public key cryptosystem is said to be homomorphic if $\mathsf{Enc}(a, pk_{\mathsf{e}})\ \phi\ \mathsf{Enc}(b, pk_{\mathsf{e}}) \equiv \mathsf{Enc}((a \oplus b), pk_{\mathsf{e}})$, for some operations $\phi, \oplus$.

The ElGamal and raw RSA cryptosystems have multiplicative homomorphic properties, which means that the relation above is fulfilled when ϕ and $\oplus$ are the multiplication operation.

A variant of ElGamal, called exponential ElGamal, provides additive homomorhic properties, and therefore $\mathsf{Enc}(a, pk_{\mathsf{e}}) \cdot \mathsf{Enc}(b, pk_{\mathsf{e}}) \equiv \mathsf{Enc}((a + b), pk_{\mathsf{e}})$. This variant consists on representing a message m as g^m for encryption. It is easy to see that $\mathsf{Enc}(m_1, pk_{\mathsf{e}}) \cdot \mathsf{Enc}(m_2, pk_{\mathsf{e}}) = (g^{r_1}, pk_{\mathsf{e}}^{r_1} \cdot g^{m_1}) \cdot (g^{r_2}, pk_{\mathsf{e}}^{r_2} \cdot g^{m_2}) = (g^{r_1+r_2}, pk_{\mathsf{e}}^{r_1+r_2} \cdot g^{m_1+m_2}) \equiv \mathsf{Enc}((m_1 + m_2), pk_{\mathsf{e}})$.

1.3.5 Signature schemes

Digital signature schemes are public key cryptosystems used for preserving the integrity of messages exchanged between two parties. Besides that, digital signatures provide the properties of authentication and undeniability.

We ilustrate these properties with the following example: Alice uses her private key in order to digitally sign a message, and sends the message, together with the signature, to Bob. Bob can use Alice's public key to verify the signature. Since Alice's private key is only known by her, Bob knows that Alice was the sender of this message. Moreover, Bob knows that the message has not been modified during transmission by a malicious third-party. Finally, Alice cannot later deny having sent the message to Bob, since only Alice was capable of generating such signature. If necessary, Bob can claim to a judge that he has received the message from Alice. Since Alice's public key is known by everyone, the judge can also verify the signature and be sure that Bob's claim is true.

A signature scheme is composed by the following algorithms:

- The key generation algorithm $\mathsf{Gen}_{\mathsf{s}}$ receives as input a security parameter 1^λ, and outputs a signing key pair $(pk_{\mathsf{s}}, sk_{\mathsf{s}})$. It also defines a message space $\mathcal{M}_{sp}$ and a signature space $\mathcal{S}_{sp}$.

- The signature algorithm Sign receives a message $m \in \mathcal{M}_{sp}$ and the signing private key sk_{s}, and outputs a signature $\psi \in \mathcal{S}_{sp}$.

- The signature verification algorithm $\mathsf{SignVerify}$ receives a message $m \in \mathcal{M}_{sp}$ and a signature $\psi \in \mathcal{S}_{sp}$. It outputs 1 if the verification succeeds, 0 otherwise.

A common digital signature algorithm is RSA with the hash variant, also known as RSA-FDH (RSA Full Domain Hash signature scheme [16]):

- $\mathsf{Gen_s}$ does the same operations that the $\mathsf{Gen_e}$ algorithm in RSA: it receives two primes p, q of similar bit-length $(\lambda/2)$ (which define a ring $\mathbb{Z}/n\mathbb{Z}$) and computes the public key $pk_{\mathsf{s}} = (n, e)$, where $n = pq$ and e is coprime with $\phi(n)$ $(\phi(n) = (p-1)(q-1))$. The private key sk_{s} takes the value of d, where $ed = 1 \bmod \phi(n)$.

- Sign takes as input a message m, which is not restricted to a specific space, and the private key sk_{s}, and outputs $\sigma = H(m)^d \bmod n$, where H denotes a hash function which maps strings to elements in $\mathbb{Z}_n$.

- $\mathsf{SignVerify}$ takes as input the public key pk_{s}, the message m and the signature σ, and checks that $H(m) = \sigma^e \bmod n$. It outputs 1 if the verification is successful, 0 otherwise.

This signature scheme been proven to be unforgeable against chosen message attacks in the random oracle model [18]. However, using RSA-PSS [104] instead of RSA-FDH is usually recommended.

1.3.6 σ-protocols and proof schemes

σ-protocols and proof schemes are widely used in cryptographic protocols in order to prove properties of the generated information. One atractive characteristic of these schemes is the zero-knowledge property, which informally means that the verifier does not learn secret information when verifying the proof.

Let $\mathcal{R}$ be a polynomial time verifiable relation containing pairs (x, w). We will call x the statement and w the witness. We define the language $\mathcal{L}_{\mathcal{R}}$ as the set of statements x for which there exists a witness w such that $(x, w) \in \mathcal{R}$. **A zero-knowledge (ZK) proof** is a protocol between a prover P and a verifier V where the prover, who knows a witness w for which $(x, w) \in \mathcal{R}$ will convince the verifier that $x \in \mathcal{L}_{\mathcal{R}}$, without leaking any other information than the fact that x belongs to $\mathcal{L}_{\mathcal{R}}$.

Many relations can be defined. For example, $\mathcal{R}$ can be a DL relation:

$$DL = \{(x, w) | x = (p, q, g, h), ord(g) = ord(h) = q, h = g^w\}$$

where p, q are primes, $g, h \in \mathbb{Z}_p^*$ and $w \in \mathbb{Z}_q$. In this case, $\mathcal{R}$ is the set of discrete logarithm problems and their solutions. A prover of a ZK proof for a DL relation proofs that, on input x, he knows w such that $(x, w) \in \mathcal{R}_{DL}$. An example of a ZK proof for a DL relation is the Schnorr signature scheme [114].

In a proof of equality of discrete logarithms, EqDL, the prover shows that he knows w such that $x_0 = g_0^w, x_1 = g_1^w, \ldots x_t = g_t^w$, on inputs $\{x_0, x_1, \ldots, x_t, g_0, g_1, \ldots, g_t\}$. The EqDL relation is defined as:

$$\mathsf{EqDL} = \{(x_0, x_1, \ldots, x_t, w) | x_0 = g_0^w, x_1 = g_1^w, \ldots x_t = g_t^w\}$$

The Chaum-Pedersen protocol [37] is usually used to prove knowledge of such relation.

In OR-proofs, a prover shows that, given two inputs (x_0, x_1), he either knows w_0 such that $(x_0, w_0) \in \mathcal{R}_0$ or he knows w_1 such that $(x_1, w_1) \in \mathcal{R}_1$, but without revealing which one.

$$OR = \{(x_0, x_1, w_0, w_1) | (x_0, w_0) \in \mathcal{R}_0 \vee (x_1, w_1) \in \mathcal{R}_1\}$$

Finally, in AND-proofs, a prover shows that, given two inputs (x_0, x_1), he knows w_0 and w_1 such that $(x_0, w_0) \in \mathcal{R}_0$ and $(x_1, w_1) \in \mathcal{R}_1$.

$$AND = \{(x_0, x_1, w_0, w_1) | (x_0, w_0) \in \mathcal{R}_0 \wedge (x_1, w_1) \in \mathcal{R}_1)\}$$

In [42], the authors show how to construct such OR and AND proofs.

σ-protocols:

σ-protocols are three-move ZK proofs where, in order to prove that a statement x belongs to $\mathcal{L}_\mathcal{R}$, an interactive protocol is done between the prover P and the verifier V. First, P sends a commitment message a to V. V then replies with a random challenge e. Finally, P then sends an answer z to V. After the interaction, V decides to accept or reject the proof based on all the data it has seen, i.e., x, a, e and z.

We say that such protocol is a σ-protocol if it satisfies the completeness, special soundness, and special honest-verifier zero-knowledge properties defined below [46].

Completeness. A 3-move protocol of the above form is complete if when $(x, w) \in \mathcal{R}$ and P and V honestly follow the protocol then V always accepts.

Special soundness. A 3-move protocol of the above form has the special soundness property if from any x and any pair of accepting conversations on input x, (a, e, z), (a', e', z') where $e \neq e'$, one can efficiently compute w such that $(x, w) \in \mathcal{R}$.

Special honest-verifier zero-knowledge. A 3-move protocol of the above form has the special honest-verifier zero-knowledge property if there exists a probabilistic polynomial time (p.p.t.) simulator $\mathcal{S}$, which on input x and a value $e \in \mathcal{CH}$ (the challenge space) outputs an accepting conversation of the form (a, e, z) with the same probability distribution as conversations between the honest P, V, on input x.

Examples of σ-protocols:

Here we show some examples of σ-protocols, which will be used in following Chapters.

Schnorr protocol: Assume a cyclic group $\mathbb{G}$. Given a value $g^x \in \mathbb{G}$, the Schnorr protocol can be used to proof knowledge of the exponent x in the following way:

1. Prover computes $a = g^s$, where s is a random element in $\mathbb{Z}_q$, and provides them to the verifier.
2. Verifier provides a random challenge e.
3. Prover provides to the verifier $z = s + xe$.

Finally the verifier checks that $g^z = a \cdot (g^x)^e$. This σ-protocol can be simulated in the following way: the simulator samples a random $z^* \in \mathbb{G}$, a random $e^* \in \mathbb{Z}_q$ and computes $a^* = g^{z^*} \cdot (g^x)^{-e^*}$. The resulting (a^*, e^*, z^*) values have the same distribution than the original ones.

Proof of a ciphertext content: A σ-protocol can be used to prove that a specific plaintext corresponds to a given ciphertext. In the case of ElGamal encryption, where the ciphertext is of the form $(c_1, c_2) = (g^r, pk_{\mathsf{e}}^r \cdot m)$, the protocol is as follows:

1. Prover computes $(a_1, a_2) = (g^s, pk_{\mathsf{e}}^s)$, where s is a random element in $\mathbb{Z}_q$, and provides them to the verifier.
2. Verifier provides a challenge e.
3. Prover provides to the verifier $z = s + re$.

The verifier checks that $g^z = a_1 \cdot c_1^e$ and that $pk_{\mathsf{e}}^z = a_2 \cdot (c_2/m)^e$. A simulated proof can be computed in the following way: the simulator samples a random $z^* \in \mathbb{G}$, a random $e^* \in \mathbb{Z}_q$ and computes $a_1^* = g^{z^*} \cdot c_1^{-e^*}$ and $a_2^* = pk_{\mathsf{e}}^{z^*} \cdot (c_2/m)^{-e^*}$. The resulting (a^*, e^*, z^*) values have the same distribution than the original ones.

Non-interactive zero-knowledge proof of knowledge schemes:

The Fiat-Shamir [55] transformation allows the turning of interactive zero-knowledge protocols, such as σ-proofs, into non-interactive, by using a hash function to compute the random challenge e. The security of the resulting non-interactive zero-knowledge proof of knowledge (NIZKPK) is based on the assumption made in the Random Oracle Model (ROM) that a hash function behaves as a random oracle. Therefore the challenge e has a resulting distribution similar to the original and the non-interactive version of the ZKPK maintains its properties [16].

A NIZKPK scheme is composed by the algorithms (GenCRS, $\mathsf{NIZKProve}$, $\mathsf{NIZKVerify}$, $\mathsf{NIZKSimulate}$):

- The common reference string generation algorithm GenCRS generates the parameters of the NIZKPK scheme. It receives as input a security parameter 1^λ and, in some cases, a mathematical group $\mathbb{G}$, and it outputs a common reference string crs.

- $\mathsf{NIZKProve}$ is the proof generation algorithm. It receives as input a common reference string crs, a statement x and a witness w, and outputs a proof π.

- $\mathsf{NIZKVerify}$ is the verification algorithm. It receives as input the common reference string crs, the statement x and the proof π, and outputs 1 if the verification is successful, 0 otherwise.

- $\mathsf{NIZKSimulate}$ is a proof simulation algorithm. It receives as input a (false) statement x^* and outputs a simulated proof π^*.

NIZKPKs have to satisfy the properties of *completeness*, *soundness* and *zero-knowledge* [45], [113], very similar to those of the σ-protocols but adapted to their non-interactive nature:

Completeness. A triplet $(\mathsf{GenCRS}, \mathsf{NIZKProve}, \mathsf{NIZKVerify})$ is complete if given a crs generated by GenCRS and $(x, w) \in \mathcal{R}$ then $\mathsf{NIZKVerify}(\mathsf{crs}, x, \mathsf{NIZKProve}(\mathsf{crs}, x, w))$ always returns 1.

(Computational) Soundness. A triplet $(\mathsf{GenCRS}, \mathsf{NIZKProve}, \mathsf{NIZKVerify})$ is sound if, given a crs generated by GenCRS, no p.p.t. adversary can output a statement x and a proof π such that $x \notin \mathcal{L}_{\mathcal{R}}$ and $\mathsf{NIZKVerify}(\mathsf{crs}, x, \pi)$ returns 1, with non-negligible probability.

(Perfect) Zero-knowledge. A triplet $(\mathsf{GenCRS}, \mathsf{NIZKProve}, \mathsf{NIZKVerify})$ provides zero-knowledge if, given a crs generated by GenCRS, then for any pair $(x, w) \in \mathcal{R}$ the proofs π output by $\mathsf{NIZKProve}(\mathsf{crs}, x, w)$ are perfectly indistinguishable from proofs π_{sim}, output by $\mathsf{NIZKSimulate}(\mathsf{crs}, x)$.

Note that the notion of zero-knowledge defined above implies that, if it is hard to distinguish between statements $x \in \mathcal{L}_{\mathcal{R}}$ from statements $x \notin \mathcal{L}_{\mathcal{R}}$, then it is computationally hard to distinguish between pairs (x, π_{sim}), such that $x \notin \mathcal{L}_{\mathcal{R}}$ and π_{sim} was created by the simulator $\mathsf{NIZKSimulate}$, and pairs (x, π), such that $x \in \mathcal{L}_{\mathcal{R}}$ and π was created by a prover $\mathsf{NIZKProve}$ with access to a witness w, for which $(x, w) \in \mathcal{R}$.

Examples of NIZKPK schemes:

SignedElGamal: A non-interactive proof of knowledge based on the Schnorr protocol is used in the Signed ElGamal encryption scheme for proving knowledge of the encryption exponent. Specifically, being (c_1, c_2) the output of the Enc algorithm in plain ElGamal, it proves in zero-knowledge the knowledge of $\log_g c_1$ for $g, c_1 \in \mathbb{G}$. This is commonly used in voting schemes for proving *plaintext independence*.

The Signed ElGamal scheme works as follows:

- The key generation algorithm $\mathsf{Gen_e}$ takes as input a subgroup $\mathbb{G}$ which has a generator g of order q of elements in $\mathbb{Z}_p^*$, where p is a safe prime such that

$p = 2q + 1$ and q is a prime number. It outputs an ElGamal public/secret key pair $(pk_{\mathsf{e}}, sk_{\mathsf{e}})$, where $pk_{\mathsf{e}} \in \mathbb{G}$ such that $pk_{\mathsf{e}} = g^{sk_{\mathsf{e}}} \mod p$ and $sk \in \mathbb{Z}_q$.

- The encryption algorithm Enc receives as input a message $m \in \mathbb{G}$ and a public key pk_{e}, chooses a random $r \in \mathbb{Z}_q$ and computes $(c_1, c_2) = (g^r, pk_{\mathsf{e}}^r \cdot m)$. Then it computes the proof of knowledge of the encryption randomness (c_3, c_4) where: $c_3 = H_c(c_1, c_2, g^s)$, s is randomly chosen in $\mathbb{Z}_q$, $c_4 = s + rc_3$ and H_c is the hash funcion defined for the challenge in the non-interactive proof, which maps strings to $\mathbb{Z}_q$.
- The proof can be verified by executing $\mathsf{EncVerify}(c, pk_{\mathsf{e}})$, which computes $g^{s'} = g^{c_4} \cdot c_1^{-c_3}$ and checks that $c_3 = H_c(c_1, c_2, g^{s'})$. In case the validation is successful, this algorithm outputs 1, otherwise it outputs 0.

This variant of ElGamal has been shown to be NM-CPA secure in [25].

Equality of discrete logarithms: NIZK proofs are used in the protocols proposed in the following chapters, in order to prove equality of discrete logarithms. Specifically, a generalization of the Chaum-Pedersen proof system [37] is used, which we denote as EqDL:

- $\mathsf{ProveEq}(\mathsf{crs}, (a_1, a_2, a_1^x, a_2^x), x)$ takes at random s from $\mathbb{Z}_q$, computes a_1^s, a_2^s, $h = H((a_1, a_2, a_1^x, a_2^x), a_1^s, a_2^s)$ and $z = s + x \cdot h$, being H a hash function which maps strings to elements in $\mathbb{Z}_q$. The output proof π is (h, z).
- $\mathsf{VerifyEq}(\mathsf{crs}, (a_1, a_2, a_1^x, a_2^x), \pi)$ computes $a_1^{s'} = a_1^z \cdot (a_1^x)^{-h}$ and $a_2^{s'} = a_2^z \cdot (a_2^x)^{-h}$, and checks that $h = H((a_1, a_2, a_1^x, a_2^x), a_1^{s'}, a_2^{s'})$. If the validation is successful, the algorithm outputs 1. Otherwise it outputs 0.
- $\mathsf{SimEq}(\mathsf{crs}, (a_1, a_2, a_1^*, a_2^*))$ takes at random z^* and h^* from $\mathbb{Z}_q$ and forms the proof π^*. In this kind of proof, a programmed random oracle has to be used for simulation such that when the adversary asks for the value $H((a_1, a_2, a_1^*, a_2^*), a_1^{s'}, a_2^{s'})$ the oracle returns the value h^*.

Other NIZKPK proofs are used for proving the correct decryption of the ciphertexts. Although they are also based on the Chaum-Pedersen protocol, we use a different notation than in EqDL for simplicity of the protocols description. Therefore, we denote them as DecP and also add their description here:

- $\mathsf{ProveDec}(\mathsf{crs}, (c, m), sk_{\mathsf{e}})$ receives a ciphertext $c = (c_1, c_2)$, where $c_1 = g^r$ and $c_2 = pk_{\mathsf{e}}^r \cdot m$, being $pk_{\mathsf{e}} = g^{sk_{\mathsf{e}}}$. It takes at random s from $\mathbb{Z}_q$, computes $(g^r)^s$, g^s, $h = H(c, m, (g^r)^s, g^s)$ and $z = s + sk_{\mathsf{e}} \cdot h$, being H a hash function which maps strings to elements in $\mathbb{Z}_q$. The output proof π is (h, z).
- $\mathsf{VerifyDec}(\mathsf{crs}, (c, m), \pi)$ computes $(g^r)^{s'} = (c_1)^z \cdot (c_2/m)^{-h}$ and $g^{s'} = g^z \cdot pk_{\mathsf{e}}^{-h}$, and checks that $h = H(c, m, (g^r)^{s'}, g^{s'})$. If the validation is successful, the algorithm outputs 1. Otherwise it outputs 0.

- $\mathsf{SimDec}(\mathsf{crs}, (c, m^*), sk_{\mathsf{e}})$ takes at random z^* and h^* from $\mathbb{Z}_q$ and forms the proof π^*. In this kind of proof, a programmed random oracle has to be used for simulation such that when the adversary asks for the value $H(c, m^*, (g^r)^{s'}, g^{s'})$ the oracle returns the value h^*.

Now that we have explained some basic notions of cryptography and of electronic voting protocols, we are ready to focus on electronic voting protocols providing individual verifiability in the following chapter.

1.4 Motivation, organization and contributions of this work

1.4.1 Motivation

As we have seen in the previous sections, we can design electronic voting systems with intuitively contradictory security properties, thanks to the use of cryptography and cryptographic protocols. However, we, as humans, are not able to do most cryptographic operations by ourselves, and therefore we have to delegate these operations to some electronic device, such as our smartphones or computers.

Traditionally, most of the protocols abstracted the voter as the subject of the action, even when cryptography was involved (encrypting, digitally signing...). But in a real world implementation, we have to take into account that it is not the voter, but her device, which performs most of the operations required by the protocol. And the device cannot be considered a mere extension of the voter. The device is usually exposed to external entities (for example, through the Internet), which may be interested in changing its behavior to make it obey other instructions than those of the voter.

Moreover, while in traditional voting the voter personally puts her vote in the ballot box, the digital transaction of sending an electronic vote to a remote voting server is not as *tangible*. The voter has no way to tell whether her vote has reached the server or not, which affects her trust in the system. And as we know trust is a key component in democracy. If voters do not trust in the electoral process, how are they going to respect the result of the election, and the decisions of the elected government?

In this thesis, we focus on studying the mechanisms that can be provided to the voters, in order to examine and verify the processes executed in a remote electronic voting system. This work has been done as part of the tasks of the author at the electronic voting company Scytl, which implements software solutions for electronic voting which are used around the world, and since 2009 has made not only traditional security, but also verifiability, its main differentiating factor. Although this thesis does not talk about system implementations, which are interesting by themselves, it is indeed focused on protocols which have had, or may have, an application in the

real world. Therefore, this thesis may surprise the reader by the fact that it does not use state of the art cryptography such as pairings or lattices which, although they provide very interesting properties, still cannot be efficiently implemented and used in a real system. Otherwise, the protocols presented in this thesis use standard and well-known cryptographic primitives, while providing new functionalities that can be applied in current electronic voting systems.

Furthermore, during the last years, the way of analyzing electronic voting protocols has radically changed. It has to be taken into account that research in electronic voting, which begun in the 80's, is much more recent than research in cryptography, and at the beginning lacked the formality of its big brother. Protocols were designed on a whiteboard, and the security of them relied on several pairs of eyes not being able to find a hole. Now this has changed, and electronic voting protocols are analyzed by means of security proofs, which allow the relation of their security to the strength of well-known hard mathematical problems, just in the same way that it is done in cryptography. Part of the contribution of this work is to provide such kind of analysis for schemes which have been, or will be, put into practice.

1.4.2 Organization and contributions

The organization and contributions of this work are the following:

- In Chapter 2, the first contribution of the author is to present a survey on electronic voting systems which provide voter verification functionalities. The author has tried to cover a high variety of systems, giving preference to those which have had a real world implementation. Among these systems we can find the one used in the Municipal and Parliamentary Norwegian elections of 2011 and 2013, and the system used in the Australian State of New South Wales for the General State Elections in 2015, in which the author has had an active participation in the design of their electronic voting protocols. The related publications are:
  - *Internet Voting System with Cast-as-Intended Verification.* Jordi Puiggalí, Sandra Guasch. 3rd International Conference on e-Voting and Identity (VoteID 2011). Tallinn, Estonia, September 28-30 2011.
  - *Cast-as-Intended Verification in Norway.* Jordi Puiggalí, Sandra Guasch. 5th International Conference on Electronic Voting 2012 (EVOTE 2012), Co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC, July 11-14, 2012, Castle Hofen, Bregenz, Austria. Published in Proceedings of the 5th Conference on Electronic Voting 2012 (EVOTE2012) P-167, LNI GI Series, Bonn.
  - *An overview of the iVote 2015 voting system.* Ian Brightwell, Jordi Cucurull, David Galindo and Sandra Guasch. 2015. Online at: `https://www.elections.nsw.gov.au/about_us/plans_and_reports/ivote_reports`, also available at the 5th International Conference on e-Voting and Identity (VoteID 2015) program website.

  – Patent: Method for the Verification of the Correct Recording of Information. Patent family PCT/ES10/000490, granted in Spain (ES2367940), USA (US2012239932) and Philippines (PH 1-2012-501102). In process in Europe (EP2509050).

- In Chapter 3, a syntax which can be used for modeling electronic voting systems providing voter verifiability is presented. This syntax is a novel contribution of this work and is focused on systems characterized by the voter confirming the casting of her vote, after verifying some evidences provided by the protocol. Along with this syntax, definitions for the security properties required for such schemes are provided. Some of them are based on the latest definitions published this year in [23], which solve flaws from previous works. They have been adapted by the author in order to cover the particularities of the protocols presented in this work. Other definitions have been adapted from other works, or are new contributions of this thesis.

- Chapter 4 describes the electronic voting protocol and system which has been used in 2014 and 2015 elections in the Swiss Canton of Neuchâtel. Switzerland published a new regulation for electronic voting systems in 2014, by which they have to provide verifiability mechanisms in order to be used by large portions of the electorate. Part of the author's work at Scytl has been to continue with the experience of the design and development of the electronic voting system used in Norway, and improve it. The result has been implemented and has already been used in 2014 and 2015 elections in Neuchâtel, and it is planned to be used in the next years.

  A consequence of the success in the design of the system has been the initiation of a colaboration between Scytl and the Swiss postal mail company Swiss Post, in order to provide an electronic voting platform which can be used by many Swiss Cantons. The purpose of this collaboration is the evolution of the platform in order to fulfill more demanding requirements regarding verifiability, and also segregation of duties, which will in the end allow the platform to be used by up to 100% of the electorate. A contribution of this thesis is the description of the implemented protocol, as well as the formal analysis of the security properties of the scheme, by means of security proofs. This formal analysis will be used for the certification of the system in Switzerland next year. Moreover, the chapter introduces further evolutions of the protocol, designed by the author, which may be implemented throughout next year for extending the system to a larger part of the electorate.

  The contributions in this chapter, have been published at the 5th International Conference on e-Voting and Identity (VoteID 2015). Bern, Switzerland. September 2–4, 2015, under the title *2015 Neuchâtel's Cast-as-Intended Verification Mechanism*, and are part of the patent *Method for the Verification of the Correct Recording of Information*. Patent family PCT/ES10/000490, granted

in Spain (ES2367940), USA (US2012239932), Philippines (PH 1-2012-501102). In process in Europe (EP2509050).

- Chapter 5 proposes a new electronic voting protocol, which improves previous proposals from the state of the art, by generating, besides a proof of content of the vote to be cast, simulated proofs which can be used by the voter to defend against coercion. The contribution of this work is the definition of this protocol, including an analysis of its security properties. The protocol uses a special cryptographic proof, the security of which has been formally proven for the first time by a colleague of the author, Alex Escala. Patent request and conference publication are in progress.

- Finally, Chapter 6 makes a twist to individual verifiability by ensuring that all the processes executed by the voting device and the remote server are correct, without requiring a verification from the voter. We call it *universal cast-as-intended verifiability*. This chapter presents two alternative approaches for implementing this functionality. The first one is based on cryptographic proofs, and has been designed in collaboration with Alex Escala, who has done the security analysis of the scheme. The second approach is based on a protocol component used in the Norway and Neuchâtel systems: the return codes. The design of this second proposal, as well as the formal analysis of its security properties, is a novel contribution of this thesis. Both approaches have been patented under the reference EP14004456: *Method for the verification of the correct content of an encoded message.*

Chapter 2

Individual verifiability

2.1 Introduction

Electronic voting schemes which provide individual verifiability provide means to the voter to verify by herself certain aspects of the electoral process. In the literature, these aspects are traditionally divided in two phases: *cast-as-intended*, and *recorded-as-cast* verification.

Cast-as-intended verifiablity: This property provides voters with methods to ensure that their cast votes correctly represent their voting intentions. That is, that they contain their voting choices.

As explained before, in electronic voting schemes usually voters employ a voting device to select their choices. This voting device is also in charge of encrypting them after the voter has finished doing her selections. In poll-site electronic voting schemes, the voting device may be a voting machine or DRE, or a common purpose PC. In remote voting schemes, the voting device can be any device owned by the voter, such as a PC, a mobile phone, or a tablet. Depending on the scheme, after the vote casting, the voting device keeps locally the encrypted vote for further counting (in some poll-site electronic voting schemes), or send the votes to a remote voting server (in remote electronic voting schemes), where they will be stored until the end of the election.

After or during the vote casting, the voter may have access to the ciphertext generated by the voting device and examine it. However, since randomized encryption schemes are usually used to encrypt the voting choices, the voter cannot obtain any information about the content of such ciphertext by simple inspection. Therefore, a malicious voting device could change the voter's selections to be encrypted, in order to favour a specific candidate, without detection. Cast-as-intended verification consists on checking whether the encrypted vote generated by the voting device contains her selections or not, and therefore detect such modification in case of a malicious voting device. Existing cast-as-intended verification schemes are focused on providing this property while still fulfilling other requirements of electoral processes, such as vote secrecy and resistance against vote selling.

Recorded-as-cast verifiablity: This property allows voters to check that their vote has been correctly registered in the ballot box. The *Bulletin Board* [59] is a tool commonly used in verifiable voting systems providing this property: at a high level, a bulletin board is a place where relevant information for the election is published. Only authorized entities can publish in append-only mode in the bulletin board, and everybody can see the published contents. Once published, the information cannot be modified or deleted. Votes cast by the voters are published in a bulletin board upon reception at the local or remote voting server. Voters can then check that their votes have been published, and therefore that they have been correctly recorded and stored by the voting system.

The publication of the votes in the bulletin board is a sensitive issue. For example, votes made public could be decrypted at some point in the future (due to improvements on computation speed or to a breach in the encryption algorithm), and since read access to the bulletin board is open, multiple copies of it can be stored for an unlimited amount of time. Everlasting privacy, first introduced in [80], is a novel concept in electronic voting which cares about the long-term privacy provided by the information published by the electronic voting systems. By now, most schemes publish a digest function (such as a hash function) of the received vote, instead of the vote itself. Voters then have to compute the same function over their cast votes in order to look for them in the Bulletin Board. The verification properties of this solution remain unchanged thanks to the collision-resistance property of hash functions, which ensures that, in case a voter finds a match, it indeed corresponds to her ciphertext. In such case, original ciphertexts are stored in the remote/local server's ballot box for further processing at the decryption phase. Mechanisms have to be put in place in order to ensure that the contents of the ballot box match those of the Bulletin Board.

Cast-as-intended and recorded-as-cast verification allow the voter to audit all the processes that happen until the vote has been registered in the voting platform, up to the counting phase. This emulates the traditional voting process in paper, where the voter directly introduces her vote into the physical ballot box.

Universal verifiability: Once the voting phase ends, all the votes which were stored at the digital ballot box have to be taken into account for computing the election tally. Additional verification measures can be put in place so that it can be checked that the system indeed takes into account all the votes in the tally, and also that this tally is computed correctly. These verification processes fall into the categorty of what is understood as *Universal verifiability*: any entity can verify that the tally represents the content of the votes cast by the voters (**Counted-as-recorded verifiability**), and that such votes were cast by voters entitled to vote in that election (**Eligibility verifiability**). These verifications are not restricted to the voter, but are open to the public in general.

Cast-as-intended verification cannot be understood without **recorded-as-cast verification**: what is the purpose of being able to verify the content of a vote

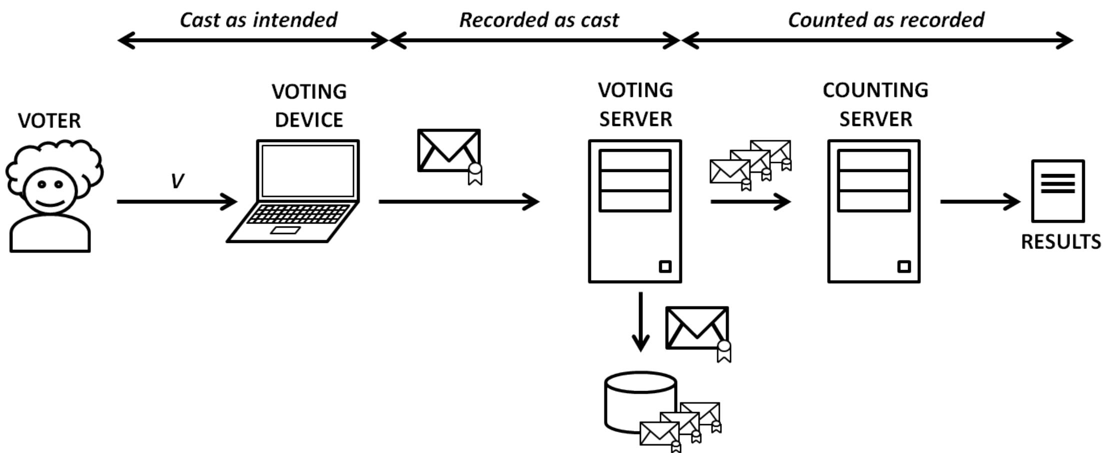

Figure 2.1: Verifiability types

if the voter does not have the assurance that such vote (and not another one) has been correctly registered at the voting platform? In this work, we will talk about systems which provide both properties, and generally refer to them as individually verifiable.

There are several proposals for individual verifiability in the literature. While recorded-as-cast verifiability is usually achieved by means of the publication of the votes (or some related value) on the bulletin board, cast-as-intended mechanisms differ considerably accross the different proposals. We classify individually verifiable systems in five categories: *Challenge-or-cast*, *Verifiable optical scanning*, *Verification with codes*, *Hardware-based verification* and *Decryption-based verification*.

2.2 Challenge-or-cast

This kind of mechanism was proposed by the first time by Josh Benaloh in the year 2006 [20]. They consist on allowing the voter to challenge the voting device after it has encrypted her selections, in order to get some information which can be used to inspect the generated ciphertext and check whether it contains her selections. The nature of the challenge mechanism is such that, in case the voting device modified the voter's selections, it has a negligible probability of answering correctly to the challenge, and therefore of cheating without being detected by the voter. A key idea of this kind of system is that a vote which is audited cannot be cast, in order to mitigate vote selling attacks: the voter could provide this same audit information to a vote buyer, who may have access to the cast vote by accessing the bulletin board, and check the contents of the ciphertext.

In Benaloh's proposal, the voting device commits to the encrypted vote prior to giving the voter the option of challenging the voting device. This commitment is done by showing the hash of the vote in the voting device's screen, although it could also be printed out. Then the voter chooses whether she challenges the voting device or not. In the negative case, the encrypted vote is cast. Later, the voter can

check that the hash shown by the voting device during the commitment phase is one of the entries of the bulletin board. In the afirmative case, the voter is given the randomness used to encrypt the vote. Using an alternative software and the election encryption public parameters (published and available to the voter), the voter should be able to reproduce an encryption of her selected voting options with the randomness given by the voting device, and check that it matches the hash of the ciphertext she was shown at the commitment phase.

In order to prevent vote selling attacks, after the audit phase the voting device generates a new encryption of the same voting options with fresh randomness. Then, the voting device commits to the new ciphertext and gives the chance to the voter of auditing again, or casting the vote. Therefore, a vote that is cast is never audited (and otherwise). The verification method is sound as far as the voting device does not know whether it will be audited or not, prior to commiting to the generated ciphertext. Specifically, a cheating voting device has a chance of 50% of being caught by the voter. Multiple audit processess are recommended in order to increase this probability.

Implementations/alternatives

This approach is followed in the Helios, Wombat, VoteBox and STAR Vote schemes.

Helios is an internet voting system which has been widely used in academic environments, both as a voting tool (mainly student organization elections, although other organizations, such as IACR, have also used it) and as a research tool. The system has evolved over time. In version 1.0 [10], it consisted on a mixing-based scheme, implementing the verifiable mixnet from Sako and Kilian [110]. Then, it was modified in version 2.0 to implement homomorphic tally with exponential El-Gamal and distributed decryption, following a scheme similar to that described in [43]. A detailed description of the homomorphic tally variant and the experience of using it in university elections can be found in [79].

Helios has been widely studied by the academic community in the last years and has a lot of variants, which are evolutions of the Helios system or academic alternatives, some of which having their own implementations. For example: Helios v3.0, Helios v4.0 [6], Helios based on mixnets (a newer version in [29]), Helios-C (distributed Helios with credentials [40] and its implementation, Belenios [1]).

Helios provides cast-as-intended and recorded-as-cast verifiability in a similar way as described in the proposal from Benaloh: after doing her selections and prior to casting her vote, the voter is presented with the commitment of the ciphertext generated by the voting device, in the form of a hash value. At that moment the voter can decide to either cast the vote, or audit it. In case the voter chooses to cast her vote, the vote is sent to the remote server, where it is posted in the bulletin board. Otherwise, the randomness, the encryption parameters and the cleartext vote are provided, so that the voter can check that the generated ciphertext is correct

according to these parameters, and that the cleartext vote matches her selections. A software application is offered by the same Helios website in order to make this audit [5]. However, it is recommended to use a third-party software (for example [4]), and preferably on a device different than the one used for voting, in order to ensure independence of the verifier and the verified entities. Because the voting client does not know, at the time of generating the ciphertext and showing the commitment to the voter, which is the option she will choose, the chance of cheating without being detected is of $\frac{1}{2}$. It is encouraged that voters perform this audit several times in order to improve this probability.

As explained for the case of the Benaloh's scheme, audited ciphertexts are not cast to prevent the voter from being able to sell her vote, but the voting options are encrypted again with new randomness after the audit.

The voter is able to check that the vote she cast was accepted by the remote voting server by checking that the hash or fingerprint, which the voting device used to commit to a generated ciphertext, matches one entry of the bulletin board. Depending on the Helios variant, ciphertexts may be published alongside the voter's identifier, an alias, or no identifier at all. Also, hashes may be published instead of the full ballots.

Wombat [8], [19] is a poll-site based dual voting system, in which votes are cast both electronically and on paper. It has been designed and developed in Israel with the cooperation of the Interdisciplinary Center in Herzliya and the Tel Aviv University, and has been used in student and on party elections.

The system works as follows: after being identified by the poll-worker, voters enter a private booth where they use a voting device equipped with a touch screen, a CPU, and a printer to cast their vote. After the voter finishes selecting her options, the vote is encrypted by the voting device using a probabilistic encryption scheme (ElGamal [51]). In order to ensure a strong encryption in front of a malicious device, the randomness for the encryption is obtained from two different vendor smartcards connected to the voting device, each one digitally signing a commitment of their part of the random. The encrypted vote is digitally signed by one of the smartcards, and then a ballot divided in two regions is printed: one region contains the cleartext options (this is the *physical vote*), and the other contains a barcode representing the signed ciphertext (and the signed commitments to the random values) (this is the *electronic vote*). The voter is then allowed to either cast or audit the vote.

In case the voter decides to cast the ballot, it is marked as *to be cast* by the printer. Then, the voter folds the ballot in such a way that the cleartext options are hidden, exits the private booth and provides the ballot to a poll-worker, who scans the barcode to read the electronic vote. The electronic vote, containing the signed ciphertext, is verified and uploaded to the electronic bulletin board. Then the two regions of the ballot are stamped by the poll-workers and stripped apart. The physical vote is put into a physical ballot box, which is used at a further stage

for making manual recounts to check the correctness of the electronic tally. The electronic vote is kept by the voter as a receipt. The voter can, later on, check that her vote is published in the electronic bulletin board, using this receipt. In case it is not found, the voter can object using the stamped receipt.

If the voter decides to audit the ballot, additional audit information is printed, namely the randomness used in the encryption, and the ballot is marked *to be audited*. The voter can then go to an audit station where she verifies that the printed randoms match the digitally signed commitments, and that the ciphertext matches the encryption of the voting options printed in clear, using the printed randomness. The voter can also use some alternative software (for example, at some point there was an Android application available for performing this audit, referentiated in Wombat's website).

In the counting phase, the votes in the digital ballot box are shuffled and re-encrypted using a verifiable mixnet [7], prior to being decrypted using a threshold decryption scheme. The mixed and decrypted votes are published in the bulletin board, together with the proofs of correct mixing and decryption, so that anyone can verify the counting process.

VoteBox [111] is also a system designed to be used in poll-sites. In this system, a set of DREs (Direct Recording Electronic machines), each one containing a local ballot box, is located in a polling station, interconnected by a local physical network, forming the Auditorium [112]. Single DREs are not trusted to store the cast votes in their local ballot box. Instead, every DRE connected to the network broadcasts every local event to the rest of DREs for storing it, and therefore the system is robust in the sense that, as far as one DRE remains fair, a record of the events and votes cast in that poll site is preserved. Moreover, all the events are recorded by the DREs in the form of hash chained logs which ensure their integrity once they are stored.

Once a voter finishes her selections in one of the DREs, the vote is encrypted and broadcast through the network, so that all the DREs store the encrypted vote. This plays the role of the commitment to the generated ciphertext by the voting device (the DRE). After that, the voter can choose to either audit her vote or not. In case of auditing it, the DRE broadcasts the randomness used to perform the encryption to the rest of interconnected DREs. A network diode is located in the local network, from which the network traffic can be read but no new traffic can be inserted. The messages broadcast by the DREs can be read from the output of this diode, to which an audit station is connected and available for the voters to check the content of their audited votes using the broadcast randomness. As in the other schemes, once a vote is audited it becomes spoiled, and the voter is required to cast a new vote.

The recorded-as-cast property is ensured by the broadcasting of any cast vote to all the DREs in the Auditorium network, as well as for the hash chain of log events

recorded by each DRE. As far as one DRE remains honest, evidences are preserved of the votes cast by the voters.

STAR Vote [14] is a new proposal for poll site voting, which has emerged from a collaboration between several academics and the Travis County (Austin, US) Texas elections office, which currently uses a DRE voting system and previously used an optical scan voting system. The design of the system is similar to VoteBox in the sense that all the DREs in a polling place are networked together. A controller station manages the DREs.

The voting process is the following:

- Registration: A poll worker identifies the voter and her ballot model and precinct. After identification, the poll worker marks the voter in a paper / electronic pollbook and prints a sticker which can be scanned by the controller station. The controller scans the ticket to determine the ballot model and precinct of the voter. Then it prints a random 5-digit code, which is unique for the whole election in that polling station, with which the voter can access a DRE to proceed to make the selections on her ballot.

- Voting: the voter marks her choices in the ballot and, after confirmation, a ballot is printed. The printed ballot has several parts: (1) a paper ballot containing a human-readable summary of the voter's selections and a random serial number, and (2) a receipt for the voter that identifies the voting terminal used, the time of the vote, and a short (16-20 character) hash code that serves as a commitment to the vote without revealing its contents. The voter reviews the printed record to verify her selections, and she is presented with two options: cast the ballot, or challenge it.

  In case the voter decides to cast the ballot, she introduces the paper ballot summary in the ballot box, which scans the serial number and communicates it to the controller. In this way, the controller keeps a record of which ballots have been stored in the ballot box and which not. An electronic ballot is not included in the tally unless the corresponding paper ballot has been entered in the ballot box.

  In case the voter decides to audit her ballot, she indicates as such to a poll worker, who scans the serial number in order to notify the controller that this ballot has been spoiled (and therefore will not be included in the tally). Instead, it is decrypted and published after the end of the election. The original printed ballot therefore corresponds to a commitment by the voting machine. Anyway, the voter can take home her receipt.

- Counting: at the end of the election, the encrypted votes are posted on the bulletin board. Then they are homomorphically aggregated and a threshold decryption scheme, in which a subset of the election authorities is required to participate, is used to decrypt the result. The decryption generates proofs of correct computation which are also posted on the bulletin board. Aditionally,

ballots which were selected for audit by the voters are individually decrypted. The cleartexts, as well as proofs of correct decryption, are also posted on the bulletin board.

At home, voters can check that the process has been executed correctly: the receipt they took home contains a hash of their vote and they can check that their votes are present on the public bulletin board. Additionally, voters can check that the decryption of the spoiled ballots outputs the expected voting options in plaintext.

Paper records are kept in the local election office so that they can be used in case of a failure on the electronic records, or for performing manual recounts.

Risk limiting audits [117] are performed by randomly selecting paper ballot summaries and decrypting the corresponding electronic ballot, in order to check that they match and provide software-independent evidence of the election result. The SOBA protocol [21] is used for its simplicity and due to the fact that it mitigates the risk of breaking the voters' privacy by decrypting individual votes.

2.3 Verifiable Optical Scanning

In several countries, and mainly in the United States, the election modernization process has been focused on the use of electronic means for counting the votes. The purpose of electronic counting is to obtain faster and more accurate voting results, while keeping the voting process as similar to traditional voting as possible, and maintaining the existence of paper records which can be easily audited by the voters and be used to make parallel recounts.

Usually, optical scanners are used to digitize the ballots marked by the voters and proceed to the electronic count. Paper ballots therefore contain bullets which the voters are required to fill in for their selections, and aligning marks which help ensuring the correctness of the scanning process. However, optical scanners are not invulnerable to attacks [123], misconfigurations, or poor detection of unclear voter selections. Several protocols have been proposed for voting schemes with optical scanning, in which the voter is provided with means to verify that the contents of their scanned ballots match their selections. Since the voters can ensure that their votes are cast-as-intended by simple inspection of the marks in their paper ballot, this verification provides voters with the assurance that their votes are recorded (by scanning them) as cast.

Prêt-à-voter was initially proposed by Peter Ryan in [106], and since then it has become a case of study with improvements [38] and adaptations to work with multiple configurations: [108] (ElGamal encryption with re-encryption mixnets), [107](Paillier encryption with homomorphic tally), [48] (Prêt-à-voter providing everlasting privacy). The system uses special ballots, which are filled by hand by the voters and passed through an optical scanner to detect the marks and compute the election result electronically. These special ballots have the particularity that they

can be used by the voters to check that their votes have been properly recorded and tallied by the system.

These special ballots are divided in two detachable halves. The left half of the ballot contains candidate names in a radomized order, and the right hand contains corresponding boxes where the voter has to put her mark, as well as encrypted information (also called *the onion*), that will enable the system to reconstruct the candidate order, so that a mark can be related to an election choice. In the voting phase, the voter removes the left half after making her selections and enters the remaining right half into the optical scanner. The right half also serves as a take-home receipt for the voter, after being stamped for authenticity. The voter's privacy is granted by the fact that the candidate ordering for that ballot is destroyed, and the remaining part contains a checkmark in a random position, and that the link between the voter and the scanned ballot is removed before decryption. Scanned votes are published and voters can check that their match their receipts. If voters detect that their votes do not appear on the public board, or have been modified, they can use their receipts to challenge the election.

At the end of the election, ballots are mixed and the *onions* are decrypted in order to recover the original candidate ordering of each ballot. Given the scanned marks, this directly translates into voter selections. Finally, the voter selections are counted and the election results are published. The correct mixing of the ballots can be verified by several audit methods, depending on the type of verifiable mixnet that is used. For example, the Randomised Partial Checking verifiable mixnet [73] can be used, in which the mix-nodes are publicly challenged by the auditors to reveal partial information about the permutation applied to the ballots. This audit method ensures that the complete information path for a ballot is not disclosed, and therefore the voter's privacy is maintained. Still, statistically the chance that a cheating mixnet is not detected is low (the probability a malicious mixnet is undetected when modifying k votes is of $1 - 2^{-k}$).

Additionally, voters have the chance to audit that the ballots have been properly generated, and that a mark in a specific position will be counted for the chosen candidate at the tally phase. In order to do that, voters can choose to audit a ballot by removing the left-hand side of the ballot, scanning the right side and asking the system to decrypt the candidate list, to see if it matches the ordering provided by the left side. Of course, the voter cannot cast an audited ballot, since her privacy would be broken straightforwardly. An alternative approach is to provide two ballot sheets: the voter chooses which one is used to cast her vote, and which one is used to audit. The voter takes home the receipts of both ballots and audits the published information for both of them.

Scratch & Vote is a poll site voting system where voters can check that their ballots are correctly scanned and counted, and that they accurately represent their selections [11]. This system is similar to Prêt-à-voter, however it offers a different audit process for the ballot construction which can be performed during the voting

phase, instead of waiting until the end of the election. Paper ballots are divided in two halves in a Prêt-à-voter fashion: one half contains the names of the candidates in a randomized order. The second half contains the blank space to put the marks for the selected candidates, as well as a barcode and a scratch surface. The barcode contains the ciphertexts corresponding to the encryption of the candidates in the same randomized order than in the first half of the ballot. The scratch surface hides the randomness of the ciphertexts.

In the configuration, ballots are generated where, for each one, a random permutation of candidates is chosen and the ciphertexts are accordingly encoded in the barcode. An audit process takes half of the prepared ballots and audits that they are well-formed, by revealing the encryption randomness and using it to check that the ciphertexts represent the intended order of the candidates. The audited ballots are discarded.

During the voting phase, the voter can ask for two ballots, and randomly decide which is audited and which is used to cast her vote. For the one to be audited, the voter scratches off the scratch surface and provides it to a poll worker or election helper, who checks that the ballot is well-formed. That means that the ciphertexts in the barcode correspond to the encryptions of the permuted candidates, according to the revealed randomness. The rest is very similar to Prêt-à-voter: For the ballot to be used to cast the vote, she marks in the second half of the ballot the space next to the chosen candidate name in the first half, detaches the ballot halves and provides the second half to the poll worker, who checks that the scratch surface is intact (in order to prevent vote selling), and removes and discards it. The voter then enters the remaining piece of the ballot into the scanner, which reads the marks and the barcode, and takes it home as a receipt. Without containing the order of the candidates, it cannot be used to prove to a third party how the voter voted. After that, the voter can check that her vote has been correctly processed and stored by the system by checking in a public website that her ballot has been published.

During the counting phase, the ciphertext in the barcode corresponding to each marked position is extracted from each ballot and included in the tally. The system uses an homomorphic tally process with the Paillier cryptosystem. Election authorities perform a threshold decryption to obtain the election result.

Scantegrity is an end-to-end verification system designed to work with optical scanners already owned by the electoral commissions, in order to take profit of existing equipment [36].

The system works as follows: during configuration, random code letters are generated and printed next to each voting choice in the ballot. These random code letters are different per each ballot, which is identified with a serial number. During the voting phase, the voter fills in the bullets corresponding to her selected options, writes-down the code letters assigned to them and introduces the ballot in the ballot box for being scanned later on. The voter takes home the written code letters and

the ballot serial number.

At the end of the election, the images of the scanned ballots are used to retrieve the code letters next to the areas detected as filled bullets (that is, the voter selections), and then such code letters are published next to the serial number of each scanned ballot. The voters can check that the code letters next to their ballot's serial number correspond to their selections by checking them with the values they wrote down during voting. Without the ballot, the code letters cannot be used to prove the vote content to a third party. Therefore, voters can delegate their verification without breaking their privacy. In case a voter detects that a code letter for a non-selected option, linked to her ballot's serial number, is published, she can object and a dispute-resolution process is started to determine if there was a mistake on the scanning process or if the voter wrote down an incorrect code letter.

A switchboard with a secret permutation is used during configuration in order to assign random code letters to regions of each ballot (i.e., voting choices). A commitment of the switchboard is published, and at the end of the election it is used to verify that the result of the tally corresponds to the voting options represented by the published code letters of the cast votes. The verification of the switchboard is done in two phases: before the election, auditors randomly choose half of the ballots, for which the connection of voting options - code letters with the switchboard is publicly revealed, in order to check the correspondence with the printed ballots. After that, these ballots are destroyed. The switchboard is composed by two connected permutation circuits. At the end of the election, auditors challenge the switchboard to reveal partial information of the permutation for each ballot (as in [73]), and it can be checked that the revealed information corresponds to the published commitments. This audit, together with the audit in the configuration phase, ensures that with high probability the printed ballots have been correctly generated.

Punchscan Scantegrity is very similar to an early proposal by Chaum, called Punchscan [96]. In Punchscan, ballots are composed by two layers: the top layer contains the questions and candidates, as well as a randomized correspondence with code letters. The layer below contains the same code letters but with a different randomization. Holes in the top layer allow one to see the code letters in the layer below. When the voter wants to cast a vote, she makes a circle on the border of the hole, through which the code corresponding to her selected candidate is shown with a thick pen, in order to mark both sheets. Then, she decides which layer she keeps and which layer she destroys. The layer kept by the voter is scanned, and then taken home as a receipt: note that one layer contains candidates, code letters, and a marked position, and the other layer contains a marked code. By keeping one layer, the voter cannot prove to a third party how she voted. At the end of the election, code letters corresponding to the voter receipt are decrypted and published, so that the voter can check them and be sure that her vote was correctly recorded.

Similarly to Scantegrity, audits are performed for ensuring that the two-layered

ballots are well-formed: configuration audits allow one to check test ballots and ensure that the scanned and then decrypted codes represent the intended selections. Post-election audits from commited configuration data allow one to statistically check that the randomizations used for assigning code letters to voting options in the two-layered ballots were correct and yielded to the intended voting options. Thanks to the fact that the voter chooses which layer she keeps, a modified ballot which passes the configuration audit has a 50% chance of being detected.

Scantegrity II [35] is an evolution of Scantegrity in which confirmation codes assigned to voting options in the ballot are not visible *a priori*. This is intended to ease the dispute resolution process. Instead, voters use special pens for marking the bubbles on the ballot beside their selected candidates. Then the confirmation codes for such candidates, printed with secret ink, are revealed and the voter marks them down in a receipt to take their to home. The marked ballot is passed through the scanner, which reads the marks but not the confirmation codes. Voters are allowed to ask for audit ballots: poll workers provide them with two ballots, from which the voter selects one to vote, and one to audit. All bubbles on the ballot to audit are marked in order to reveal all the codes. The voter can take the audited ballot to home.

At the end of the election, as in previous systems, scanned marks in the ballots are transformed into confirmation codes using a switch table with a secret permutation, the commitment of which was published during configuration. The confirmation codes are published and the voter can check that they correspond to those in the receipt (corresponding to her selections). Similarly to previous systems, auditors can check that the election result is generated from the voters' selections corresponding to the published codes, and that the codes in the ballots correspond to the indicated options, by using a combination of audit of test ballots at the configuration phase, and a cut-and-choose scheme for partially revealing the permutation of codes to voting options at the end of the election. The permutation is entirely revealed for the case of audit ballots.

Confirmation codes are not published per ballot, but per contest, using the so called *contest partitioning* technique [97], in order to mitigate the chance of privacy attacks based on voting patterns (i.e., the *Italian attack* consists on giving a very particular ordering of choices, or selecting unpopular candidates, in order to be able to identify a vote, even after a mixing process has been executed).

The system was used in the Municipal Elections of the city of Takoma Park in Maryland, United States, in 2009 [31] and then in 2011 in combination with Remotegrity.

Remotegrity is a remote voting extension for Scantegrity designed for providing similar protections to absentee voters as those of voters voting at the polling place. It was used in Takoma Park's municipal election in November 2011 [122]. In this system, remote voters receive the information to vote by postal mail and use it to

vote electronically. The intention of this system is to improve in voter privacy and verifiability, provide resistance against vote selling, and eliminate the time of vote return, compared to regular voting by postal mail. The voter interacts with the system in a similar way than in pollsterless or code voting schemes, by introducing the codes for the options she chooses in the voting device.

Before the election, voters receive by mail a ballot package containing two parts: a paper ballot, which is a Scantegrity II ballot containing a serial number *VoteSerial* and a set of random short confirmation codes, one per each candidate. In Scantegrity II, the confirmation codes are printed with invisible ink and revealed when the voter marks a particular candidate with a special pen. In order to avoid sending those specific pens by mail to the voters, in Remotegrity confirmation codes are already present in the ballot (which makes no possible dispute resolution in case a wrong confirmation code is published after the voter casts the vote). The second part is the Remotegrity authorization card, containing a serial number *AuthSerial*, a set of authentication codes under a scratch-off surface, an acknowledgement code, and a lock-in code also under a scratch-off surface.

During the voting phase, the voter enters *VoteSerial* and *AuthSerial* in the voting device for authenticating to the platform, together with the confirmation codes from the Scantegrity II ballot corresponding to her selections. She also scratches the surface of one of the authentication codes and enters it into the voting device. The remote voting server checks that the authentication code has not been used before, and if so, it publishes the signed tuple *(VoteSerial, AuthSerial, confirmation codes, authentication code, acknowledgement code)*. The voter then checks the tuple values and, if she agrees, enters the lock-in code.

At the end of the election, the talliers check the tuples for which a valid lock-in and authentication serial codes have been provided, and only for those entries they convert the confirmation codes into voting options and include them in the tally. As in Scantegrity II, a switch table with a private permutation, which can be audited, is used for converting from voting options to confirmation codes (for generating the ballots) and otherwise (for obtaining the voting options to compute the tally).

Given that the values in the Scantegrity II ballot and in the Remotegrity authorization card are only known by the voter, neither a malicious device nor the remote voting server can cast votes on behalf of the voter (or modify them) without being detected.

vVote is an adaptation of the Prêt-à-voter voting system which has been used in the November 2014 State elections in the state of Victoria, Australia. The system [44] is designed to be used in poll-site voting, where voters are assisted by an *electronic ballot marker* (EBM) in order to make their selections. In this system, ballots similar to those in Prêt-à-voter are printed on demand after voters are authenticated. Specifically, a ballot looks like the left-hand half of a Prêt-à-voter ballot, containing the randomized list of candidates and a serial number. This serial number connects

the ballot with the encrypted candidate-list permutation (the *onion*), which is stored in a central online server.

A voter has the chance of auditing the printed-on-demand ballot instead of filling it. In such case, the voter has to ask for a new ballot for casting her vote, since audited ballots cannot be used to cast votes. As in Prêt-à-voter, the audit process consists on requesting the system to decrypt the *onion* containing the candidate's permutation, so that it can be verified that the ballot is well formed. Printed ballots contain their own QR code representation, which is scanned by the EBM in order to present the ballot contents (with the candidate lists unpermuted) in the electronic user interface, through which the voter navigates accross the ballot and makes her selections.

When the voter finishes, the EBM prints the equivalent of the right-hand half of the Prêt-à-voter ballot, containing the voter marks according to the candidate's randomized order, and an additional serial number. It also casts the ballot to a central server. Then the voter can check that both halves have the same serial number and align correctly, and can even check the signature made on the ballot contents by the system. In case the validations fail, the vote is cancelled (and the central server notified accordingly to mark the ballot as deleted). The voter keeps the list of printed marks as a receipt, and deletes the list of randomized candidates. Later, the voter can use the receipt to check that her vote has been correctly recorded in the platform.

Finally, the ballots are passed through a verifiable mixnet (RPC [73]) and decrypted. The election results and cryptographic proofs are published, so that they can be publicly verified.

2.4 Verification with codes

Other individual verification mechanisms are based on the comparison of codes returned by the voting server after vote casting, against a set of codes in a code sheet received prior to the voting phase.

As we have explained in Section 1.2, return codes are one of the functionalities of Pollsterless systems. The main idea of these systems is the following: the voter receives, prior to the voting phase, a voting card containing return codes, each one assigned to one of the voting options in the election. Each voter has in her voting card a different set of return codes. The voting card contents are secret and assumed to be only known by the voters. During the voting phase, the voter casts an encrypted vote using her voting device to a remote voting server. The remote voting server, in turn, uses a private key in order to compute, from the received vote, the set of return codes corresponding to the encrypted voting options present in the vote. The return codes are sent back to the voter, who checks that they match those in her voting card assigned to her selected voting options. Given that the voting device does not know the return codes in the voting card in advance, it has a

small chance of showing the expected return codes to the voter in case it encrypted different contents in the vote. Therefore, the voter detects any manipulation of her selections.

2.4.1 Code voting

Systems providing return codes are traditionally Pollsterless systems, where voters enter voting codes (also in the voting card) uniquely assigned to their preferred candidates, in the voting device. As explained in Section 1.2, these kind of mechanisms also provide privacy in front of the voting devices, since they do not know the correspondence between voting codes and candidates. Moreover, these systems are suitable to be used for casting the votes from devices without cryptographic capabilities, since encryption is not required (the voting codes are a pre-encryption of the candidates). For example, SMS text messages could be used. However, they pose severe usability issues due to the fact that voters have to enter randomized codes, specially in complex ballots.

Some examples of these kinds of systems are the following:

SureVote proposed by David Chaum [34], it is the first known proposal for Pollsterless voting. In his proposal, voting cards are generated in a verifiable and distributed way among a set of trustees: each trustee generates a set of voting codes and return codes corresponding to each voting card, and publishes commitments to them. A random shift is used to create different sets of codes for each ballot. The printer combines the set of voting and return codes from all trustees (with a modular addition), and prints the voting cards. Auditors can randomly choose voting cards to audit, for which the trustees open the commitments so that the correspondence between voting and return codes can be checked.

During the voting phase, the voter enters the voting codes assigned to her preferred candidates into the voting device, which sends them to a remote voting server. The remote voting server checks that such codes are valid (by comparing them to a reference value, such as a hash value) and computes a keyed function to generate, from them, the corresponding return codes, which are sent back to the voter. The return codes serve to provide the voter with assurance of the correct reception of her vote at the remote voting server, as well as for authenticating it. At the end of the election, the trustees jointly encrypt the voting codes and remove the random shifts (so that the values of the voting codes are *unified*), then pass the result through a decryption mixnet, and translate the decrypted voting codes to candidates in order to obtain the election results.

A drawback of this proposal is that the operations for translating codes to voting options are not verifiable, and therefore, voters cannot check that their votes were correctly taken into account for computing the election results.

Other mechanisms similar to Chaum's proposal are presented in [118] and [78].

Pretty Good Democracy is a code voting based system, which is focused on improving previous proposals by additionally providing recorded-as-cast and counted-as-recorded verifiability [109], as well as *receipt-freeness*, which means that the voter does not obtain a receipt that can be shown to a third party, to prove how she voted. The main idea is that, in case the correct return code has been provided to the voter after casting her vote, the voter knows that her vote will be accurately included in the tally. Receipt-freeness is achieved by the existence of only one return code (the ack code), which is independent from the voting option selected by the voter. The remote voting server can only recover a correct return code, to be sent back to the voter, with the cooperation of a set of trustees. Therefore, assuming that there is no collusion between the trustees, and that the voter's codes are kept secret, the fact that a voter gets the right ack code back indicates that the vote has been successfully registered on the bulletin board.

During the setup phase, the election public key is generated using a multiparty computation scheme, and the contents of the code sheets are computed on the following way: the codes are randomly generated, encrypted with randomness equal to 1, and then passed through a re-encryption mixnet. The results of the mixnet execution are published in the bulletin board and are publicly verified. Then subsets of encrypted codes are organized for composing the code sheet contents, and the encrypted codes are decrypted by a set of trustees. Then the set of codes corresponding to each ballot is permuted and the permutation is kept in an encrypted form (similar to the *onion* in Prêt-à-voter). A random subset of code sheets is audited to check that they are well-formed.

During the voting phase, the voter authenticates to the system and enters the code sheet serial number. Then she enters the vote code corresponding to the candidate she selected. The serial number and the vote code are sent to the voting server, which checks that the serial number is correct, encrypts the vote code using the election public key, and posts it on the bulletin board together with a zero-knowledge proof of plaintext knowledge. This is done to prevent ballot copying - by which a voter can cast a vote containing the ciphertext generated by another voter -, and also to prevent the voting server from randomly picking an encrypted code from those generated during the setup.. Then the trustees check the zero-knowledge proof and perform a *plaintext equivalence test* (PET [72]) against the encrypted codes mixed during the setup. When they find a match, they jointly decrypt the corresponding ack code field, which is then returned by the voting server to the voter.

In order to obtain the election results, the indexes and encrypted permutations (onions) from the shuffled table, for which a match was found with the plaintext equivalence tests, are passed through a mixnet and decrypted in order to obtain the original candidate list position to which they correspond, as in the original Prêt-à-voter scheme.

Pretty Understandable Democracy provides a security model for the server-side which is based on separation of duties, fulfilling the security requirements under the assumption that two components do not collude [28]. Besides that, the system is also focused on keeping the scheme as simple as possible. In order to do that, the scheme uses multiparty computation in several steps of the protocol, and in fact even the code sheets are computed in several parts, which are provided to the voters by postal mail: the first part consists on the permuted list of candidates, which is different for each code sheet. The second and third part consist on voting codes which are concatenated to cast a vote for a specific candidate. As in Pretty Good Democracy, only an additional code is included in the code sheet, which is the answer expected from the voting server. Having different answer codes for the different candidates is avoided in order to prevent the voter from having a receipt of how she voted.

The scheme is as follows: during the setup, the trustees generate the election key pair in a distributed way (for example, with the Pedersen scheme [92]). Exponential ElGamal is used for having an additive homomorphism. A registration authority (RA) generates the first part of the code sheets as permutations of the list of candidates. Two different voting authorities (VA1 and VA2) generate, in turn, random codes for the second and third parts of the code sheets, one for each candidate in the list plus 1 for confirmation. The three parts of the code sheets share a common index, in order to be put together in the same envelope, for sending them to the voter. As usual, methods are provided for checking that the pieces of the code sheets are correctly generated: the three entities commit to the generated code sheet parts by publishing their encryptions with the election key on the bulletin board. A random subset of the generated code sheets is audited by making the trustees jointly decrypt them, and by checking that the contents are consistent (the index in the three parts matches). The audited code sheets are discarded.

During the voting phase, a voter sends the code, resulting from concatenating the second and third parts of the code sheet, corresponding to her choice. This code is received by the registration authority RA, who divides the code in two parts and sends each one to the corresponding voting authority. The voting authorities check, for each part of the code, that it exists in the list of codes they generated. Then each voting authority obtains the corresponding encrypted candidate name, according to the position of the received code in the list of commitments, re-encrypts and digitally signs it, and sends it to the bulletin board, which publishes the information and sends back a confirmation to the voting authorities. Finally each voting authority retrieves the corresponding ack code, which is sent back to RA. The RA concatenates both parts and sends it to the voter. The voter then checks that the received code matches the ack code in her code sheet.

At the tallying phase, the trustees get the two lists of signed and re-encrypted candidates from the ballot box (one for each voting authority) and compute the homomorphic tally of each of the lists separatedly, in a distributed way. After decryption, they check that the results in both lists match and if so, the election

result is published in the bulletin board, together with zero knowledge proofs of correct decryption.

The system provides secrecy and integrity (cast-as-intended, recorded-as-cast, counted-as-recorded), under the assumption that the adversary does not corrupt more than one entity in the system. The system has been implemented within a student project as part of the lecture Electronic Voting in the winter term 2013/14 at the Technische Universität Darmstadt, Germany. Several improvements and modifications made to the original scheme, as well as a report on the implementation and the trial experiences, can be found in [88].

2.4.2 Return Codes

These systems combine the use of code sheets with a click & select interface: voters are not required to enter voting codes at the voting device, but still they receive codes from the server which they can use to verify that their votes were cast as intended. These systems emerged from the requirements of a voting system to be used in Norway.

The Norwegian protocol (2011) In 2009, Norway started a bidding process for the implementation of a remote voting system to be used in a binding pilot in 10 municipalities in the Municipal Elections of 2011. Verifiability was among the requirements of the project, and particularly the system had to provide a mechanism for cast-as-intended verification. The Ministry of Local Government and Regional Development (KRD - the Norwegian acronym), which acts as the electoral management body in Norway, started a competitive bidding process in which an open dialogue was established between the vendors and the contractor, in order to refine the scope and requirements of the solution prior to submitting the final proposal.

During this refinement process, some specific requirements arose for the individual verification process: the cast-as-intended verification did not have to rely on any trusted sofware or hardware which could be available to the voters and it had to be as intuitive as possible. Naturally, an alternative to consider was code voting. However, it was not an option for KRD due to usability issues, although it was accepted that voters could check codes returned by the platform.

Vote updating / multiple voting was also a requirement of the solution. Therefore the codes could not be provided to the voter through the same voting device used to cast the vote (the voting device could learn good codes, cast modified votes and still show the good codes). An authentication platform (MinID [2]) commonly used for online services, some of them from public agencies, was available to be used by the voting system. The authentication platform used two-factor authentication with one time passwords (OTP) sent to the users' mobile phones, whose numbers were registered on the platform. Therefore, the mobile phone could be used as a secondary channel for sending the codes to the voter.

At the end, the solutions of the two favourites (Scytl [100] and Cybernetica in collaboration with Helger Lipmaa [66]) were very similar: the system combined a click & select interface for casting the vote, with the generation of return codes at the server-side, which were sent back to the voter through the secondary channel SMS. The voters had received in advance a verification card containing return codes assigned to their voting options, which they could use to check the codes received after voting. In case wrong codes were received, the voters could vote again. Only the last vote per voter was taken into account in the tally process. The counting process was verifiable thanks to the use of a verifiable mixnet and the generation of proofs of correct decryption. No bulletin board was set up: multiple voting was required as an anti-coercion measure and the information of whether a voter had updated her vote or not, or which votes had passed to the tally process, had to be restricted to a set of auditors. Moreover, voters could not verify that their votes had been correctly registered at the voting platform. This functionality was provided in the improved system used in the 2013 Parliamentary elections.

The winning vendor was Scytl. Here we provide a brief description of their proposed solution for cast-as-intended, which can be found in [100]. The security model provided by KRD required that a single component could not subvert the election by either breaking the voter's privacy or changing the result of the election, so a segregation of duties approach was followed in order to split the sensitive operations into two or more components of the platform.

- During the **configuration phase**, the return codes to be printed in the voting cards are generated by two isolated machines, which in turn use a secret key to compute the return codes: the first machine computes partial values of return codes by applying its secret key to the voting options, and the second machine computes the final values of the return codes by applying its secret key to each partial return code. Randomized identifiers are used in each of the machines, in such a way that the second machine cannot link the final values of the return codes to the corresponding voting options, and none of the machines can link the generated contents to a specific verification card or voter.

  The return codes and the randomized identifiers are transmitted in an encrypted form to the printing service, which is in charge of linking both sets and printing the verification cards in a two-step process:

  - A first printer prints the return codes on the verification cards and seals them.
  - A second printer prints, on the outside of the sealed verification card, the verification card identifier. This identifier is used to assign a specific verification card to a voter.

  The two secret keys for computing the return codes are installed in two online servers, the vote collector server (VCS) and the return code generator (RCG), which participate in the voting phase.

- During the **voting phase**, the voter enters the verification card identifier into the voting device and proceeds to make her choices by clicking them. Once the voter finalizes the process, the voting device generates two kinds of encryptions of the voting options, using the ElGamal encryption scheme:

  - The first encryption, called the probabilistic one, uses random values to encrypt the voter's selections.
  - The second -deterministic- encryption, uses a fixed exponent for the encryption of each voting option, which is computed as a combination of the verification card identifier, the voting option value, and the secret key owned by the VCS.

  Zero-knowledge proofs are used to link both encryptions and prove that they correspond to the same voting options.

  The two ciphertexts and the zero-knowledge proofs are sent to the VCS, which performs some validations and forwards them to the RCG. The RCG in turn performs some validations and uses its secret key to compute return codes from the deterministic encryption values. The return codes are sent to the voter through the SMS channel, and the probabilistic encryption is stored at the VCS for further counting.

- At the **counting phase**, the encrypted voting options in each vote are multiplied together, in order to get a single ciphertext per vote. The ciphertexts are passed through a verifiable mixnet [99] (also designed by Scytl) prior to being decrypted. After decryption, the obtained values are factorized in order to recover the prime factors representing the individual voting options.

Kristian Gjøsteen proposed a modification of this protocol in order to increase the efficiency of the system. The drawback of the solution is that the two online servers VCS and RCG share the election private key. Therefore a collusion of both servers could systematically decrypt all the votes as cast by the voters. The modified protocol was the one that finally was implemented and used in the Norwegian elections in 2011. Details of the protocol and of the set up can be found in [60], [101], [61].

While at a high level the configuration process is very similar to the previous one, during the voting phase the voting device is only required to perform one encryption (the probabilistic one) of the voting options selected by the voter. The encrypted vote is sent to the first server, the VCS, which removes part of the encryption layer with its part of the election private key. Prior to this, it applies a secret exponent, different for each voter, to the received ciphertext. The result of this exponentiation and partial decryption is forwarded to the RCG, which uses the second part of the election private key to remove the remaining part of the encryption layer put by the voting device. The result is the set of voting options exponentiated to a voter-dependent exponent only known to the VCS. The RCG then uses a secret key to compute, from these values, the return codes to be sent back to the voter.

A more detailed explanation on how this mechanism for generating the return codes works is provided in Section 4.2. Some of the implementation details and usability improvements have also been applied to the system used in Neuchâtel, and therefore they are explained in Chapter 4.

The Norwegian protocol (2013) The Norwegian system was used for the second time in the 2013 Parliamentary elections. Besides operative and performance improvements, an interesting feature that was introduced was recorded-as-cast verifiability: at the end of the voting process, voters received a voting receipt (which consisted of a hash of their cast vote), digitally signed by the two servers VCS and RCG. A bulletin board was put in place in a publicly accessible GitHub repository, which stored the history of the bulletin board files, where the hash of the votes stored by the VCS where periodically published. In the new version, the RCG also stored the votes in a local ballot box, and therefore discrepancies between the contents of both ballot boxes and the bulletin board could be resolved in case one of the two components was malicious.

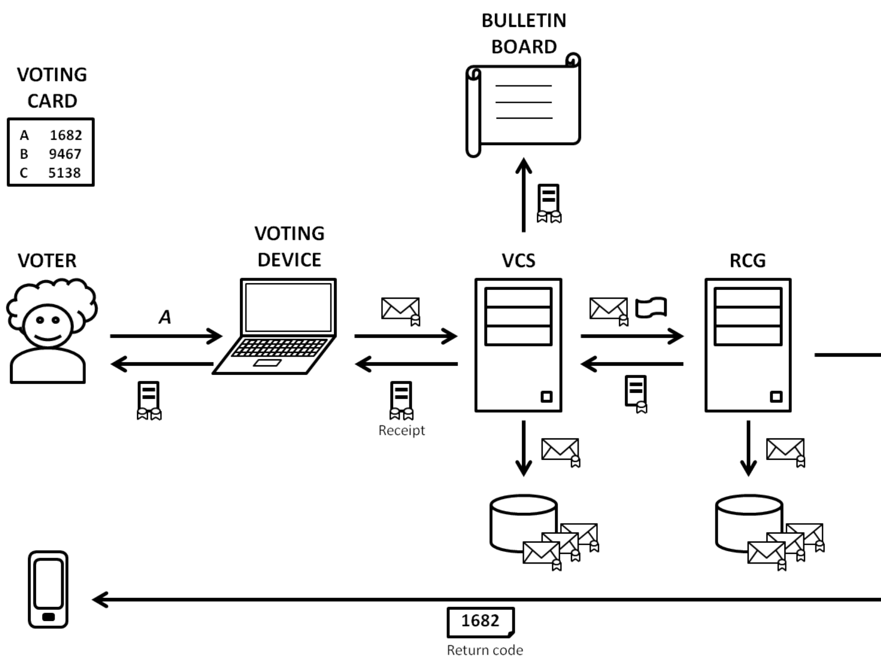

Figure 2.2: Norway 2013 scheme

2.5 Hardware-based verification

Some individual verification methods rely on the voter having access to trusted hardware. While this approach may seem impractical, there are some specific cases where this approach is very interesting: the first of such cases is Switzerland. As a country with a long tradition in direct democracy, voters participate in many voting processes during the year, and therefore they could take advantage of dedicated hardware for voting. The second case is Estonia: this country has a widely deployed

electronic administration system, and citizens are used to performing the majority of their transactions with the goverment in an electronic way. Specifically, citizens in Estonia have been able to vote electronically from their homes since 2005. Voters already have trusted devices (smart cards) for performing some transactions with the administration or with the banks.

E-Voting protocol using Smart Cards: in [77], Helger Lipmaa proposes a remote voting scheme based on smart cards that perform some basic cryptographic computations, which could be suitable for use in future evolutions of the Estonian electronic voting system. The scheme uses smart cards with embedded keyboards and LCD screens for interacting with the voter.

The described voting process is as follows: prior to connecting the smart card to the voting device, the voter enters the number assigned to her candidate using the smart card keyboard, which generates a random verification code and shows it to the voter on the LCD display. The voter writes down the code and connects the smart card in the voting device. The smart card then encrypts the candidate number using the election public key, and the verification code together with the voter identity, using the remote voting server's public key. The two ciphertexts are digitally signed and passed to the voting device, which sends them to the remote voting server. The remote voting server checks the signature and stores both signed ciphertexts. It also decrypts the verification code and the voter identity, checks that the voter identity matches the owner of the signature key pair, and sends back the decrypted verification code, digitally signed, to the voter device. The voter checks that the received verification code corresponds to that shown by the smart card in advance. If so, she knows that the vote prepared by the smart card, which contained her selections, has been successfully received at the server. The comparison with the received verification code could also be done directly by the smart card. Additional controls have to be put in place in order to ensure that the votes received by the server are properly stored and tallied.

Du-Vote [64] uses a hardware token which, in collaboration with the voter device and the voting server, creates the encryption of the voter's selections without none of them knowing which is the cleartext value.

During the registration phase, voters are assigned hardware tokens with unique embedded keys, which are registered by the voting server.

During the voting phase, the voting device prepares a code page by computing two encryptions for each candidate and showing the codes derived from them in two different columns. Additionally, a ballot identifier which is the commitment of the voting device to the set of generated ciphertexts is presented in the code page. The voter chooses one column at random and enters all the codes in that column in the hardware token. Then it enters the code in the *other* column corresponding to the selected candidate. The hardware token uses its embedded key to compute a transformation of the codes entered by the voter (concatenated), and shows it

to her, who in turn enters it into the voting device. The voting device sends the entered value, together with the generated ciphertexts, to the voting server.

The server checks that the ciphertexts provided by the voting device yielded the correct codes and posts the vote in the bulletin board, together with the ballot identifier computed as the hash of the ciphertexts. The voter can check that the ballot identifier presented in her voting device's screen is present in the bulletin board. The server then reverts the transformation done by the hardware token, in order to recover the codes column selected by the voter, and the code corresponding to her selection. For the whole column selected by the voter, the server asks the voting device for the randomness used for computing the corresponding ciphertexts, and checks that they were correctly computed. Then it re-encrypts the selection code and posts it in the bulletin board. The re-encrypted selections are processed by a mixnet or an homomorphic tally system, in order to recover the election results at the end.

2.6 Decryption-based verification

Finally, there are remote electronic voting systems in which the individual verification is based on decrypting the vote stored in the remote voting server, in order to check that the content is the expected. Two systems based on this kind of verification have been recently used in real binding elections.

The Estonian voting system has been used from 2005 for binding elections, and has recently been extended with a functionality for cast-as-intended and recorded-as-cast verification [67]. This functionality has been made available for the first time at the 2013 Estonian local municipal elections, and subsequently being used in the 2014 European Parliament elections, and 2015 Estonian Parliamentary elections.

The verification system is based on a smartphone application, which the voters can use to check the content of the votes stored in the remote voting server: after the vote is cast, the voting device shows on the screen a QR code containing the identifier of the vote and the randomness used for the encryption. The voter can use an application on her smartphone to read these values. The smartphone then requests the encrypted vote from the remote voting server, using the read identifier. Finally, the smartphone brute-forces the encryption of the vote downloaded from the server (using the randomness read from the voting device), and shows the voter the underlying voting option. The voter can re-vote in case the shown voting option is not the one she selected. This verification is time-limited in order to mitigate vote selling threats.

iVote 2015 the iVote 2015 system has been recently used in the State General Election of the Australian state of New South Wales (NSW) [68]. The iVote system was originally introduced in 2011 for facilitating the voting process for vision-impaired voters, those who could not visit a polling place due to a disability, and

those who lived more than 20 km away from the polling place. In 2015 the system was improved with a new voting protocol, which provided cast-as-intended and recorded-as-cast verification properties. The system is based on challenging the server to decrypt a voter's vote upon request.

Although the architecture of the solution is rather complex, three main components play a role in the cast-as-intended and recorded-as-cast verification: the voting device, the Vote Encoder (VE) and the Verification Server (VS). While the software being executed at the voting device and the VE was developed by Scytl (as well as the ballot box processing and vote decryption software), the software executed by the VS was developed by an independent entity contracted by the NSW Electoral Commission.

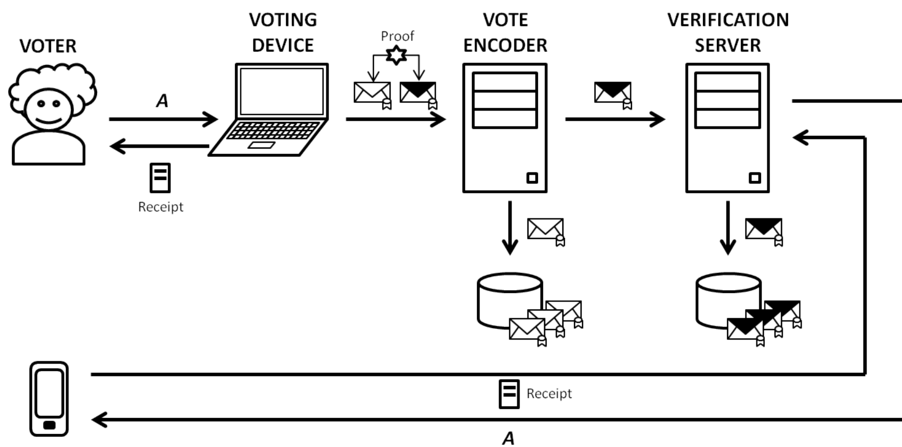

Figure 2.3: iVote 2015 scheme

During the voting phase, the voting device encrypts the voter choices in two ciphertexts, related by zero-knowledge proofs. One ciphertext is encrypted using the election public key and stored at the VE. At the end of the election, this ciphertext is decrypted by the electoral board members and used to obtain the election results. The other ciphertext is encrypted with a random voting receipt which is locally generated at the voting device, and shown to the voter after the two ciphertexts and the proofs are sent to the VE. After verifying the NIZK proofs, the VE forwards this second ciphertext to the VS, which processes and stores it. Any time after casting her vote, the voter can call the VS using a dual-tone multi-frequency (DMTF) phone and, after authenticating, enter her voting receipt. The VS uses the voting receipt to decrypt her vote and provides the decrypted voting options to the voter, who checks if they match those she selected. If not, the voter can re-register to obtain a new voting credential and cast a new vote. Coercion or vote selling was not considered a risk in that election.

The NIZK proofs generated by the voting device prove that both ciphertexts (the one counted and the one audited) have the same content inside. Besides that, at the end of the election the decrypted votes are compared with the values stored in the Verification Service during the voting phase, in order to detect any mismatch.

Chapter 3

Electronic Voting Model

3.1 Introduction

In this chapter, we present a generic model for an electronic voting protocol, which will be used to describe the protocols in Chapters 4 and 5. Both protocols are intended to be used in electronic voting systems, either remote or not (poll-site), and are focused on providing a method for cast-as-intended verification. Although the approaches followed in both protocols for providing such verifiability are quite different, both have in common the following characteristics: after the ballot is created by the voting device, the voter receives a proof of its content, which she uses to verify that it indeed corresponds to her selections. Once she has verified such proof, the voter proceeds to confirm that she agrees with the created ballot. Only confirmed ballots will be taken into account during the tally phase.

3.2 Protocol Syntax

In this section we define a general syntax for the voting protocols proposed in the following chapters, which is intended to reflect the functionalities of systems in which the voter confirms a ballot after she has verified its content.

We use as a basis the syntax defined in [116] and [41] for analyzing the properties of the Helios voting protocol [10], in which a verification step for the tally process and a registration step for providing voters with credentials to cast their votes (for example for signing them, like in the Helos-C variant [41]) are added, with respect to prior works [24] and [25]. [41] also adds a specific algorithm for the voter verifying that her vote is on the Bulletin Board. However, none of them take into account the cast-as-intended verification functionality provided in Helios, so we add it to our definition in the form of proof generation and audit algorithms. We also add extra algorithms in order to include a ballot confirmation phase.

The following are the participants of the voting protocol:

- *Election Authorities*: they are in charge of the configuration of the election and of tallying the votes to produce the election result.

- *Registrars*: they register the voters and provide them with information for participating in the election.

- *Voter*: they participate in the election by choosing their preferred options.

- *Voting Devices*: they generate and cast ballots given the voting options selected by the voters.

- *Bulletin Board Manager*: it receives, processes and publishes the votes cast by the voters in the bulletin board BB.

- *Auditors*: they are responsible for verifying the integrity of the procedures run in the counting phase.

The voting protocols to be modeled with this syntax provide proofs of the content of the generated ballots for cast-as-intended verification. Depending on the specific scheme, these proofs of content may be generated by different entities. For example, in the protocol presented in Chapter 4 the proof of content is generated by the bulletin board manager (the server-side). However, in the protocol presented in Chapter 5 the proof of content is generated by the voting device prior to submitting the ballot. In fact, a model for protocols which offer cast-as-intended verifiability should not discard any of the two variants, and even consider protocols in which proofs are generated by both entities. Therefore, two different steps for proof generation and verification have been included in this syntactical definition.

Prior to the definition of the election algorithms, we assume that the list of voting options $V = \{v_1, \ldots, v_k\}$ in the election, and the counting function $\rho : (V \cup \{\perp\})^* \rightarrow R$, where $\perp$ denotes an invalid vote and R is the set of results, have been previously defined by the electoral authorities.

Voters may use credentials in order to be able to cast their ballots. However, how the voters obtain and use such credentials is out of the scope of this definition.

The voting protocol consists on the following algorithms:

$\mathsf{Setup}(1^\lambda)$ receives as input a security parameter 1^λ, it generates and outputs an election public/private key pair (pk, sk), a global audit key pair $(pk_{\mathtt{a}}, sk_{\mathtt{a}})$ and global confirmation key pair (pk_c, sk_c). The public outputs $(pk, pk_{\mathtt{a}}, pk_c)$ are input to the next algorithms although not specified.

$\mathsf{Register}(1^\lambda, \mathtt{id}, sk_{\mathtt{a}}, sk_c)$ takes as input a security parameter 1^λ, a voter identity $\mathtt{id}$, and the private keys $sk_{\mathtt{a}}, sk_c$. It outputs the pairs of public/private voter voting data $(P_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{v}}^{\mathtt{id}})$, audit data $(P_{\mathtt{a}}^{\mathtt{id}}, S_{\mathtt{a}}^{\mathtt{id}})$, and confirmation data $(P_{\mathtt{c}}^{\mathtt{id}}, S_{\mathtt{c}}^{\mathtt{id}})$.

$\mathsf{CreateVote}(\mathtt{id}, \{v_{j_1}, \ldots, v_{j_t}\}, P_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{v}}^{\mathtt{id}})$ takes as input the voter identity $\mathtt{id}$, a set of voting options $\{v_{j_1}, \ldots, v_{j_t}\}$ and the voter's public and private voting data $(P_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{v}}^{\mathtt{id}})$. It outputs a ballot b and the encryption data $\tilde{r}$.

$\mathsf{AuditVote}(b, \tilde{r}, \{v_{j_1}, \ldots, v_{j_t}\}, P_{\mathtt{v}}^{\mathtt{id}})$ takes as input the ballot b, the encryption data $\tilde{r}$, the set of voting options $\{v_{j_1}, \ldots, v_{j_t}\}$ and the voter public voting data $P_{\mathtt{v}}^{\mathtt{id}}$. It outputs 1 if the verificaton is positive, or 0 otherwise.

$\mathsf{ProcessBallot}(\mathsf{BB}, \mathtt{id}, b)$ receives as input a bulletin board BB, a voter identity $\mathtt{id}$ and a ballot b. It outputs 1 in case all the operations succeed, 0 otherwise.

$\mathsf{CreateBallotProof}(b, sk_{\mathtt{a}}, P_{\mathtt{a}}^{\mathtt{id}})$ takes as input the ballot b, the global audit private key $sk_{\mathtt{a}}$ and the public voter audit data $P_{\mathtt{a}}^{\mathtt{id}}$, and outputs a proof of content σ. In case of any error, it returns $\perp$.

$\mathsf{AuditBallotProof}(\{v_{j_1}, \ldots, v_{j_t}\}, \sigma, S_{\mathtt{a}}^{\mathtt{id}})$ takes as input the set of voting options $\{v_{j_1}, \ldots, v_{j_t}\}$, the proof of content σ and the voter private audit data $S_{\mathtt{a}}^{\mathtt{id}}$. It outputs 1 if the verificaton of σ is positive, or 0 otherwise.

$\mathsf{Confirm}(\mathtt{id}, b, S_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{c}}^{\mathtt{id}}, P_{\mathtt{c}}^{\mathtt{id}}, sk_{\mathtt{a}})$ receives as input a voter identity $\mathtt{id}$, a ballot b, the voter's private voting data $S_{\mathtt{v}}^{\mathtt{id}}$, the voter's private confirmation data $S_{\mathtt{c}}^{\mathtt{id}}$, the voter public confirmation key $P_{\mathtt{c}}^{\mathtt{id}}$ and the global audit private key $sk_{\mathtt{a}}$ and . It outputs a ballot confirmation C_b and the auxiliary data σ' in case of success, or $\perp$ in case of error.

$\mathsf{ProcessConfirm}(\mathsf{BB}, \mathtt{id}, C_b)$ receives as input a bulletin board BB, a voter identity $\mathtt{id}$ and a ballot confirmation C_b. It outputs 1 in case all the operations succeed, 0 otherwise.

$\mathsf{VerifyVote}(\mathsf{BB}, \mathtt{id}, b)$ takes as input a bulletin board BB, a voter identity $\mathtt{id}$ and a ballot b. It outputs 1 in case all the operations succeed, 0 otherwise.

$\mathsf{Tally}(\mathsf{BB}, sk)$ takes as input the bulletin board BB and the election private key sk. It outputs a result $r \in R$ and a proof Π of the tally correctness, or $\perp$.

$\mathsf{VerifyTally}(\mathsf{BB}, r, \Pi)$ takes as input the bulletin board BB, the tally result r and the proof Π of correct tally. The output is 1 if the verification succeeds, 0 otherwise.

The algorithms execution is organized in the following phases:

Configuration phase: in this phase, the election authorities set up the public parameters of the election such as the list of voting options $\{v_i\}_{i=1}^{k} \in V$ and the result function ρ. They also run the Setup algorithm and publish the resulting election public key pk, the global audit key $pk_{\mathtt{a}}$, the global confirmation key pk_c and an empty voter list ID in the bulletin board. The election private key sk is kept in secret by the electoral authorities, the global audit and confirmation private keys $(sk_{\mathtt{a}}, sk_c)$ are provided to the registrars, and the private key $sk_{\mathtt{a}}$ is also provided to the bulletin board manager.

Registration phase: in this phase the registrars register the voters to vote in the election. For each voter with identity $\mathtt{id}$, the registrars use the global audit and confirmation private keys $(sk_{\mathtt{a}}, sk_c)$ to run the $\mathsf{Register}$ algorithm. The list ID is

updated with $\mathtt{id}$ and the tuple $(\mathtt{id}, P_{\mathtt{v}}^{\mathtt{id}}, P_{\mathtt{a}}^{\mathtt{id}}, P_{\mathtt{c}}^{\mathtt{id}})$ is published in the bulletin board. The public/private data pairs $(P_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{v}}^{\mathtt{id}})$, $(P_{\mathtt{a}}^{\mathtt{id}}, S_{\mathtt{a}}^{\mathtt{id}})$, $(P_{\mathtt{c}}^{\mathtt{id}}, S_{\mathtt{c}}^{\mathtt{id}})$ are provided to the voter.

Voting phase: this phase consists of several steps:

1. The voter provides $(\mathtt{id}, P_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{v}}^{\mathtt{id}})$ to the voting device, as well as a set of selected voting options $\{v_{j_1}, \ldots, v_{j_t}\} \in V$. The voting device then runs the CreateVote algorithm and produces a ballot b and the encryption data $\tilde{r}$.

2. The voter runs the AuditVote algorithm to check that the content of the created ballot is the expected, using the generated values $(b, \tilde{r})$, as well as the voter public voting data $P_{\mathtt{v}}^{\mathtt{id}}$ and the set voter selections $\{v_{j_1}, \ldots, v_{j_t}\}$.

3. The ballot b is sent to the bulletin board manager together with the voter identity $\mathtt{id}$.

4. Upon reception of $(\mathtt{id}, b)$, the bulletin board manager runs the ProcessBallot algorithm. In case the result is 1, the process continues and the bulletin board manager updates the bulletin board with the pair $(\mathtt{id}, b)$. Otherwise, the process stops and the voting device receives an error message.

5. The bulletin board manager runs the CreateBallotProof algorithm, using the voter public audit key $P_{\mathtt{a}}^{\mathtt{id}}$ corresponding to the voter identity $\mathtt{id}$, which is published in the bulletin board, and the global audit private key $sk_{\mathtt{a}}$. In case the execution is successful, it sends back to the voting device the proof of content σ, which is shown to the voter. Otherwise, the process stops and the voter receives an error message.

6. The voter executes AuditBallotProof using the voter private audit key $S_{\mathtt{a}}^{\mathtt{id}}$, her selections $\{v_{j_1}, \ldots, v_{j_t}\}$ and the received proof of content σ. In case the result of AuditVote and of AuditBallotProof were 1 (which means both verifications are satisfactory) the voter provides her secret confirmation key $S_{\mathtt{c}}^{\mathtt{id}}$ to the voting device, which interacts with the bulletin board manager to generate the ballot confirmation C_b and the auxiliary data σ' by running Confirm. The ballot confirmation is sent to the bulletin board manager together with the voter identity $\mathtt{id}$, while the auxiliary data σ' is kept by the voting device.

7. The bulletin board manager then runs ProcessConfirm using as input the received ballot confirmation C_b. In case the operation is successful (the output is 1), it updates the corresponding entry in the bulletin board to add C_b and the success of the operation is notified to the voter. Otherwise, an error message is sent.

From this point, the voter can check that her vote has been successfully posted on the bulletin board by running the VerifyVote algorithm.

Counting phase: in this phase, the election authorities use the election private key sk to run the Tally algorithm on the contents of the bulletin board. The obtained

result r and the proof Π are posted in the bulletin board. The auditors then run the VerifyTally algorithm. In case the verification is satisfactory, the election result is considered to be correct. Otherwise, an investigation is opened in order to detect any manipulation that could lead to a corrupted result.

A voting protocol as defined above is *correct* if, when the four phases are run and the participants are honest, the result r output by the Tally algorithm is equal to the evaluation of the counting function ρ over the voting options corresponding to the ballots cast by the voters.

3.3 Security Definitions

In this section we define the notions of ballot privacy, strong consistency, strong correctness, cast-as-intended verifiability and coercion-resistant cast-as-intended for an electronic voting scheme following the described model. We take as a basis the definitions of ballot privacy, strong consistency and strong correctness from [23] and then adapt them to the particularities of our model. In the same way, the definition of cast-as-intended verifiability is based on the one provided in [52]. The definition of coercion-resistant cast-as-intended is a new apportation.

3.3.1 Trust model

First of all, we introduce the trust assumptions we make on the scheme regarding privacy and integrity:

A first necessary assumption is to suppose that the voter behaves properly when voting for her preferred candidates. We assume the voter makes the selections according to her intention and follows the protocol in the correct way. We also assume the voter follows the audit processes indicated and objects in case of any irregularity.

In order to simplify the analysis, we consider that the election authorities, and the registrars as well, behave properly in the sense that they generate correct and valid information, and that they do not divulge secret information to unintended recipients. In the protocols presented in next chapters, measures that can be applied to weaken this assumption are explained.

From the point of view of privacy, the voting device is trusted not to leak the randomness used for the encryption of the voter's choices. While this assumption may seem too strong, it is in fact needed in any voting scheme where the voting options are encrypted at the voting device (no pre-encrypted ballots are used) and the vote is not cast in an anonymous way. However, from the point of view of integrity, we consider that a malicious voting device may ignore the selections made by the voter and put other content in the ballot to be cast.

Audit devices, when used, are trusted both from the point of view of privacy (they are assumed not to divulge private information) and from the point of view of integrity (they are assumed to honestly transmit the verification result to the voter).

The bulletin board manager is trusted to accept and post on the bulletin board all the correct votes and confirmations. No assumptions are done in the case of privacy.

Finally, auditors are assumed to honestly transmit the result of their verification. However, they are assumed to be curious and try to find out the content of voter's votes from the information they get.

3.3.2 Ballot privacy

Informally, ballot privacy captures the idea that a secure voting protocol does not reveal any information about the votes cast by the voters, besides what is leaked from the result of the tally. We base our definition on the one presented in [23].

The definition is given by means of two experiments which an adversary has to be able to distinguish. In each experiment the adversary has indirect access to a ballot box which receives the ballots created by honest voters, as well as ballots cast by the adversary itself on behalf of corrupted voters. In the case of honest voters, we let the adversary choose two possible votes which they will use to create their ballots. Which vote is used to cast a voter's ballot that goes to a specific ballot box depends on the experiment that is taking place.

At the end of the experiments, the adversary is presented with the result of tallying the ballot box. As noted in [23], revealing the true tally in each experiment would easily allow the adversary to distinguish between both of them, in the case the adversary chose votes for honest voters that lead to different tally results. Therefore, the same tally result is presented to the adversary, regardless of the experiment. When necessary, a simulation algorithm is used in order to provide proofs of tally correctness.

The notion of ballot privacy is formalized via the following experiment $\mathsf{Exp}^{\mathsf{priv},\beta}_{\mathcal{A},V}$, parameterized by the set of voting options V and the random coin β that determines the experiment that takes place, and therefore which bulletin board $\mathcal{A}$ is presented with:

1. **Setup phase:** The challenger $\mathcal{C}$ sets up two empty bulletin boards BB_0 and BB_1 and runs the $\mathsf{Setup}(1^\lambda)$ algorithm to obtain the election key pair (pk, sk), the global audit key pair $(pk_{\mathsf{a}}, sk_{\mathsf{a}})$ and the global confirmation key pair (pk_c, sk_c). $\mathcal{C}$ sets up the empty list of voters ID, provides (sk_{a}, sk_c) to $\mathcal{A}$ and publishes $(pk, pk_{\mathsf{a}}, pk_c)$, as well as the lists of identities $(\mathsf{ID}, \mathsf{ID}_c)$ on both bulletin boards $(\mathsf{BB}_0, \mathsf{BB}_1)$. $\mathcal{A}$ is given access to BB_0 when $\beta = 0$ and to BB_1 when $\beta = 1$.

2. **Registration phase:** The adversary may make the following query:

   - $\mathcal{O}$**Register**($\mathtt{id}$): $\mathcal{A}$ provides an identity $\mathtt{id} \notin \mathsf{ID}$. $\mathcal{C}$ runs $\mathsf{Register}(1^\lambda, \mathtt{id}, sk_{\mathtt{a}}, sk_c)$, keeps the voter private voting and audit data $(S_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{a}}^{\mathtt{id}})$ and provides the private confirmation data $S_{\mathtt{c}}^{\mathtt{id}}$ to $\mathcal{A}$. Then it adds $\mathtt{id}$ to ID and posts the tuple $(\mathtt{id}, P_{\mathtt{v}}^{\mathtt{id}}, P_{\mathtt{a}}^{\mathtt{id}}, P_{\mathtt{c}}^{\mathtt{id}})$ on the bulletin boards $(\mathsf{BB}_0, \mathsf{BB}_1)$.

3. **Voting phase:** The adversary may make the following types of queries:

   - $\mathcal{O}$**VoteLR**($\mathtt{id}, v_0, v_1$): this query models the votes cast by honest voters. $\mathcal{A}$ provides an identity $\mathtt{id} \in \mathsf{ID}$, $\mathtt{id} \notin \mathsf{ID}_c$, and two possible votes v_0, v_1 $\in V$. The challenger $\mathcal{C}$ does the following:
     - It executes $\mathsf{CreateVote}(\mathtt{id}, v_0, P_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{v}}^{\mathtt{id}})$ and $\mathsf{CreateVote}(\mathtt{id}, v_1, P_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{v}}^{\mathtt{id}})$ which produce the ballots b^0 and b^1, and the encryption data $\tilde{r}^0$, $\tilde{r}^1$ respectively.
     - Then it executes $\mathsf{ProcessBallot}(\mathsf{BB}_0, \mathtt{id}, b^0)$ and $\mathsf{ProcessBallot}(\mathsf{BB}_1, \mathtt{id}, b^1)$. If both algorithms return 1, the bulletin boards BB_0 and BB_1 are updated with $(\mathtt{id}, b^0)$ and $(\mathtt{id}, b^1)$ respectively. Otherwise, $\mathcal{C}$ stops and returns $\perp$. Note that $\mathcal{A}$ can execute $\mathsf{CreateBallotProof}(b, sk_{\mathtt{a}}, P_{\mathtt{a}}^{\mathtt{id}})$ to get the proof of content σ.
     - $\mathcal{C}$ executes $\mathsf{Confirm}(\mathtt{id}, b^0, S_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{c}}^{\mathtt{id}}, P_{\mathtt{c}}^{\mathtt{id}}, sk_{\mathtt{a}})$, $\mathsf{Confirm}(\mathtt{id}, b^1, S_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{c}}^{\mathtt{id}}, P_{\mathtt{c}}^{\mathtt{id}}, sk_{\mathtt{a}})$, and outputs the ballot confirmations C_b^0 and C_b^1 and the auxiliary data σ_0' and σ_1'. Then it executes $\mathsf{ProcessConfirm}(\mathsf{BB}_0, \mathtt{id}, C_b^0)$ and $\mathsf{ProcessConfirm}(\mathsf{BB}_1, \mathtt{id}, C_b^1)$. In case both outputs are 1, the entry in each bulletin board $(\mathsf{BB}_0, \mathsf{BB}_1)$ for the identity $\mathtt{id}$ is updated with C_b^0 and C_b^1 respectively. Otherwise, $\mathcal{C}$ stops and returns $\perp$.
   - $\mathcal{O}$**getVotingData**($\mathtt{id}$): $\mathcal{A}$ provides an identity $\mathtt{id} \in \mathsf{ID}$ and $\mathcal{C}$ provides the voter private voting and audit data $(S_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{a}}^{\mathtt{id}})$, and adds $\mathtt{id}$ to ID_c.
   - $\mathcal{O}$**Cast**($\mathtt{id}, b, C_b$): this query models the votes cast by corrupted voters. $\mathcal{A}$ provides an identity $\mathtt{id} \in \mathsf{ID}_c$, a ballot b and a ballot confirmation C_b. $\mathcal{C}$ executes $\mathsf{ProcessBallot}(\mathsf{BB}_0, \mathtt{id}, b)$, $\mathsf{ProcessBallot}(\mathsf{BB}_1, \mathtt{id}, b)$, if both algorithms return 1 it runs $\mathsf{ProcessConfirm}(\mathsf{BB}_0, \mathtt{id}, C_b)$ and if the result is 1 it also runs $\mathsf{ProcessConfirm}(\mathsf{BB}_1, \mathtt{id}, C_b)$. If the output is 1, both ballot boxes $(\mathsf{BB}_0, \mathsf{BB}_1)$ are updated with the entry $(\mathtt{id}, b, C_b)$. Otherwise, $\mathcal{C}$ halts and none of the ballot boxes are updated.

4. **Counting phase:** The challenger runs $\mathsf{Tally}(\mathsf{BB}_0, sk)$ and obtains the result r and the tally proof Π, which are provided to $\mathcal{A}$ in case $\beta = 0$. In case $\beta = 1$, $\mathcal{C}$ runs $\mathsf{SimProof}(\mathsf{BB}_1, r)$ to obtain Π^*, and provides (r, Π^*) to $\mathcal{A}$.

5. **Output:** The output of the experiment is the guess of the adversary for the bit β.

We say that a voting protocol has ballot privacy if there exists an algorithm $\mathsf{SimProof}$ such that for any probabilistic polynomial time (p.p.t.) adversary $\mathcal{A}$, the following advantage is negligible in the security parameter λ:

$$\mathsf{Adv}^{\mathsf{priv}}_{\mathcal{A}} = |\ \Pr[\mathsf{Exp}^{\mathsf{priv},0}_{\mathcal{A},V} = 1]\ \text{-}\ \Pr[\mathsf{Exp}^{\mathsf{priv},1}_{\mathcal{A},V} = 1]\ |$$

One of the main differences with the definition provided in [23] is the existence of the encryption data, which is not shown to the adversary. However, we do give the adversary the necessary keys to generate proofs of ballot content from the votes cast by both corrupt and honest voters. Even in this case, the adversary can learn nothing from the generated proofs. Another difference is that a confirmation has to be additionally posted in the bulletin board so that a vote from a specific voter is taken into account in the tally. For understandability of the scheme, the confirmation is always added to the ballots in the bulletin board.

3.3.3 Strong Consistency

Strong consistency is a property that guarantees that the result of tallying a bulletin board corresponds to the result of applying the counting function ρ over the underlying plaintexts of the votes cast by honest voters. An extraction algorithm $\mathsf{Extract}$, is defined, which on input of a ballot and a private key, outputs a cleartext vote or $\perp$ in case of an invalid vote. Strong consistency is required in order to ensure that the tally algorithm is correct and that the output result directly reflects the content of the ballots posted on the bulletin board. It is also required for preventing leaky tally algorithms: As stated in [23], it prevents an adversary from encoding instructions in her own ballots, such that the tallying algorithm may leak information on the honest votes, or prevent them from being validated, since the extraction algorithm works locally on each ballot. The extractor algorithm will later be used in the definition of cast-as-intended verifiability.

The notion of strong consistency is then formalized via the following experiment $\mathsf{Exp}^{\mathsf{cons}}_{\mathcal{A},V}$, parameterized by the set of voting options V:

1. **Setup phase:** The challenger runs the $\mathsf{Setup}(1^\lambda)$ algorithm to obtain the election key pair (pk, sk), the global audit key pair $(pk_{\mathtt{a}}, sk_{\mathtt{a}})$ and the global confirmation key pair (pk_c, sk_c). It gives $(pk, pk_{\mathtt{a}}, pk_c)$ and $(sk, sk_{\mathtt{a}}, sk_c)$ to $\mathcal{A}$.

2. **BulletinBoard:** The adversary submits a bulletin board BB.

3. **Counting phase:** The challenger runs $\mathsf{Tally}(\mathsf{BB}, sk)$ and obtains the result r and the tally proof Π.

4. **Output:** The output of the experiment is 1 if $r \neq \rho(\mathsf{Extract}(\mathsf{BB}, sk))$, where $\mathsf{Extract}$ is applied individually to each confirmed ballot in BB. Otherwise, 0 is output.

A voting protocol has strong consistency if the following premises are satisfied:

- For any (pk, sk), $(pk_{\mathtt{a}}, sk_{\mathtt{a}})$, (pk_c, sk_c) produced by Setup, for any $v \in V$, for any voter identity $\mathtt{id}$ and for any correctly created pairs of voter data $(P^{\mathtt{id}}_{\mathtt{v}}, S^{\mathtt{id}}_{\mathtt{v}}), (P^{\mathtt{id}}_{\mathtt{a}}, S^{\mathtt{id}}_{\mathtt{a}}), (P^{\mathtt{id}}_{\mathtt{c}}, S^{\mathtt{id}}_{\mathtt{c}})$, it is satisfied that $\mathsf{Extract}(\mathsf{CreateVote}(\mathtt{id}, v, P^{\mathtt{id}}_{\mathtt{v}}, S^{\mathtt{id}}_{\mathtt{v}}); sk) = v$.

- There is an algorithm $\mathsf{ValidInd}$ such that $\mathsf{ProcessBallot}(\mathsf{BB}, \mathtt{id}, b) = 1$, $\mathsf{Confirm}(\mathtt{id}, b, S_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{c}}^{\mathtt{id}}, P_{\mathtt{c}}^{\mathtt{id}}, sk_{\mathtt{a}}) \neq \perp$, $\mathsf{ProcessConfirm}(\mathsf{BB}, \mathtt{id}, C_b) = 1$ implies that $\mathsf{ValidInd}(\mathtt{id}, b, C_b) = 1$.
- Given any set of voting options V, the following advantage is negligible in the parameter λ:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{cons}} = \Pr[\mathsf{Exp}_{\mathcal{A},V}^{\mathsf{cons}} = 1],$$

  for a p.p.t. adversary $\mathcal{A}$ that returns a ballot box BB composed of ballots b, such that $\mathsf{ValidInd}(\mathtt{id}, b, C_b) = 1$.

Note that strong consistency holds only for confirmed ballots.

3.3.4 Strong Correctness

The strong correctness property requires that the votes of honest voters are considered valid, even with respect to a ballot box created by the adversary. This property is used to prevent a dishonest voter from preventing honest voters to vote.

According to the definition in [23], a voting protocol has strong correctness if the following probability, given $\mathsf{Setup}(1^\lambda) = (pk, sk), (pk_{\mathtt{a}}, sk_{\mathtt{a}}), (pk_c, sk_c)$

$$\begin{gathered}\Pr\ [(\mathtt{id}, v, \mathsf{BB}) \leftarrow \mathcal{A}(pk);\ \mathsf{Register}(1^\lambda, \mathtt{id}, sk_{\mathtt{a}}, sk_c) = (P_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{v}}^{\mathtt{id}}, P_{\mathtt{a}}^{\mathtt{id}}, S_{\mathtt{a}}^{\mathtt{id}}, P_{\mathtt{c}}^{\mathtt{id}}, S_{\mathtt{c}}^{\mathtt{id}}); \\ \mathsf{CreateVote}(\mathtt{id}, v, P_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{v}}^{\mathtt{id}}) = b;\ \mathsf{Confirm}(\mathtt{id}, b, S_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{c}}^{\mathtt{id}}, P_{\mathtt{c}}^{\mathtt{id}}, sk_{\mathtt{a}}) = C_b : \\ \mathsf{ProcessBallot}(\mathsf{BB}, \mathtt{id}, b) = 0 \vee \mathsf{ProcessConfirm}(\mathsf{BB}, \mathtt{id}, C_b) = 0)],\end{gathered}$$

is negligible.

We have added several conditions to the original definition, specifically that the voter has been correctly registered and that $\mathsf{ProcessConfirm}$ succeeds given a valid ballot confirmation. In the original definition, the adversary is restricted to provide an $\mathtt{id}$ such that there is no entry in BB. We also keep this restriction.

3.3.5 Cast-as-Intended verifiability

A voting system is defined to be cast-as-intended verifiable if a corrupt voting device is unable to cast a vote on behalf of a voter with a voting option different than the one chosen by the voter, without being detected. Traditionally this verification can only be performed by the voter, and therefore this property should hold as far as the voter is honest and follows the protocol rules.

In our definition, we consider an adversary who posts ballots in the bulletin board on behalf of honest and corrupt voters. In case of honest voters, they follow the protocol and perform some validations before approving the ballot to be cast. Corrupt voters provide their approval without doing any prior verification.

Our definition does not consider the tally phase. Instead, we focus on the verification of the individual votes which are posted on the bulletin board and use an extraction algorithm for which the system is strongly consistent. A scheme with strong consistency ensures that the contents of such individual votes are directly reflected in the tally result.

The notion of cast-as-intended is formalized via the following experiment $\mathsf{Exp}^{\mathsf{CaI}}_{\mathcal{A},V}$, parameterized by the set of voting options V and a p.p.t adversary $\mathcal{A}$:

1. **Setup phase:** The challenger $\mathcal{C}$ runs the $\mathsf{Setup}(1^\lambda)$ algorithm and provides the election key pair (pk, sk) to $\mathcal{A}$, publishes $(pk_{\mathtt{a}}, pk_c)$ in BB and keeps the private keys $(sk_{\mathtt{a}}, sk_c)$. Then it publishes the empty lists of voter identities ID, ID_h, ID_c. Finally $\mathcal{A}$ is given read access to BB.

2. **Registration phase:** The adversary may make the following query:

   - $\mathcal{O}$**Register**$(\mathtt{id})$: $\mathcal{A}$ provides an identity such that $\mathtt{id} \notin \mathsf{ID}$. The challenger $\mathcal{C}$ runs $\mathsf{Register}(1^\lambda, \mathtt{id}, sk_{\mathtt{a}}, sk_c)$, keeps $(S^{\mathtt{id}}_{\mathtt{a}}, S^{\mathtt{id}}_{\mathtt{c}})$ and provides $S^{\mathtt{id}}_{\mathtt{v}}$ to $\mathcal{A}$. Then it adds $\mathtt{id}$ to ID_h and publishes $(\mathtt{id}, P^{\mathtt{id}}_{\mathtt{v}}, P^{\mathtt{id}}_{\mathtt{a}}, P^{\mathtt{id}}_{\mathtt{c}})$.

3. **Voting phase:** The adversary may make the following types of queries:

   - $\mathcal{O}$**ProofVote**$(\mathtt{id}, b, \tilde{r}, v_x)$: $\mathcal{A}$ provides an identity such that $\mathtt{id} \in \mathsf{ID}$ and $\mathtt{id} \notin \mathsf{ID}_c$, the ballot b, the encryption data $\tilde{r}$ and the voting option v_x. The challenger adds $\mathtt{id}$ to ID_h and runs the following algorithms while the output is 1:
     - $\mathsf{AuditVote}(b, \tilde{r}, v_x, P^{\mathtt{id}}_{\mathtt{v}})$
     - $\mathsf{ProcessBallot}(\mathsf{BB}, \mathtt{id}, b)$
     - $\mathsf{AuditBallotProof}(v_x, \mathsf{CreateBallotProof}(b, sk_{\mathtt{a}}, P^{\mathtt{id}}_{\mathtt{a}}), S^{\mathtt{id}}_{\mathtt{a}})$

     If the output of any of the algorithms is not 1, the process stops and $\mathcal{C}$ returns $\perp$. Otherwise, the challenger provides $S^{\mathtt{id}}_{\mathtt{c}}$ and the result σ of $\mathsf{CreateBallotProof}$ to $\mathcal{A}$.
   - $\mathcal{O}$**VoteCorrupt**$(b, \mathtt{id})$: $\mathcal{A}$ provides a ballot b and an identity $\mathtt{id} \in \mathsf{ID}$. $\mathcal{C}$ answers with $S^{\mathtt{id}}_{\mathtt{c}}$ and $S^{\mathtt{id}}_{\mathtt{a}}$ and adds $\mathtt{id}$ to ID_c.
   - $\mathcal{O}$**Cast**$(\mathtt{id}, b, S^{\mathtt{id}}_{\mathtt{v}}, S^{\mathtt{id}}_{\mathtt{c}})$: $\mathcal{A}$ provides an identity $\mathtt{id} \in \mathsf{ID}$, a ballot b, a voter secret voting data $S^{\mathtt{id}}_{\mathtt{v}}$ and a voter secret confirmation data $S^{\mathtt{id}}_{\mathtt{c}}$. $\mathcal{C}$ runs $\mathsf{ProcessBallot}(\mathsf{BB}, \mathtt{id}, b)$ and $\mathsf{ProcessConfirm}(\mathsf{BB},$ $\mathtt{id}, \mathsf{Confirm}(\mathtt{id}, b, S^{\mathtt{id}}_{\mathtt{v}}, S^{\mathtt{id}}_{\mathtt{c}}, P^{\mathtt{id}}_{\mathtt{c}}, sk_{\mathtt{a}}))$, and if both algorithms return 1 posts $(\mathtt{id}, b, C_b)$ in the bulletin board.

4. **Output:** The output of the experiment is a bit δ^V which is defined as 1 in any of the following cases:

   Case A. The adversary provides the tuple $(\mathtt{id}, b, v_x, \tilde{r}, \sigma)$ such that:

   (i) $\mathtt{id} \in \mathsf{ID}_h$

   (ii) $\mathsf{AuditVote}(b, \tilde{r}, v_x, P^{\mathtt{id}}_{\mathtt{v}}) = 1$

(iii) $\mathsf{AuditBallotProof}(v_x, \sigma, S_{\mathtt{a}}^{\mathtt{id}}) = 1$
(iv) $\mathsf{Extract}(b, sk) = v_j$, $v_j \neq v_x$
(v) $\mathsf{VerifyVote}(\mathsf{BB}, \mathtt{id}, b) = 1$

Case B. The adversary can confirm an arbitrary ballot without having to prove its contents: $\mathcal{A}$ submits a triplet $(\mathtt{id}, b, C_b)$ such that:

(i) $\mathtt{id} \notin \mathsf{ID}_h$, $\mathtt{id} \notin \mathsf{ID}_c$
(ii) $\mathsf{ProcessBallot}(\mathsf{BB}, \mathtt{id}, b) = 1$
(iii) $\mathsf{ProcessConfirm}(\mathsf{BB}, \mathtt{id}, C_b) = 1$

Otherwise, δ^V is defined to be 0.

We say that a voting protocol has cast-as-intended verifiability if, given an $\mathsf{Extract}$ algorithm for which the protocol is strongly consistent with respect to ρ, the following advantage is negligible in the security parameter λ for any probabilistic polynomial time (p.p.t.) adversary $\mathcal{A}$:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{CaI}} = |\ \Pr[\mathsf{Exp}_{\mathcal{A},V}^{\mathsf{CaI}} = 1]\ |$$

3.3.6 Coercion-resistant cast-as-intended

Here we define a novel concept which has not been formally discussed in previous works. We refer to it as coercion-resistant cast-as-intended verifiability, and we define a voting system to have such property when the cast-as-intended verification method does not provide the voter with a proof that can be shown to third parties, to prove how she voted. The concept is similar to that of designated-verifier proofs (which we will see in Chapter 5), where the verifier can verify a proof from the prover, but it cannot convince a third party of the validity of such proof.

In order to prove this property, we use an experiment where the adversary is presented with a ballot and the proofs of content for the cast-as-intended verification, and has to decide to which voting option the ballot corresponds. An algorithm $\mathsf{FakeProof}$ is used by the challenger to generate some simulated information. We formalize the notion of coercion-resistant cast-as-intended via the experiment $\mathsf{Exp}_{\mathcal{A},V}^{\mathsf{CR\text{-}CaI},\beta}$, parameterized by the set of voting options V and the random coin β which determines which ballot the adversary is presented with:

1. **Setup phase:** The challenger $\mathcal{C}$ runs the $\mathsf{Setup}(1^\lambda)$ algorithm to obtain the election key pair (pk, sk), the global audit keypair $(pk_{\mathtt{a}}, sk_{\mathtt{a}})$ and the global confirmation keypair (pk_c, sk_c), and sets the empty list of identities ID. $\mathcal{C}$ keeps $(sk, sk_{\mathtt{a}})$ and provides $(pk, pk_{\mathtt{a}}, pk_c, sk_c)$ to $\mathcal{A}$.

2. **Registration phase:** The adversary may make the following query:

   - $\mathcal{O}$**Register**$(\mathtt{id})$: $\mathcal{A}$ provides an identity $\mathtt{id} \notin \mathsf{ID}$. $\mathcal{C}$ runs $\mathsf{Register}(1^\lambda, \mathtt{id}, sk_{\mathtt{a}}, sk_c)$, keeps the voter private voting and audit data $(S_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{a}}^{\mathtt{id}})$ and provides the private confirmation data $S_{\mathtt{c}}^{\mathtt{id}}$ to $\mathcal{A}$. Then it adds $\mathtt{id}$ to ID and provides the tuple $(\mathtt{id}, P_{\mathtt{v}}^{\mathtt{id}}, P_{\mathtt{a}}^{\mathtt{id}}, P_{\mathtt{c}}^{\mathtt{id}})$ to $\mathcal{A}$.

3. **Voting phase:** The adversary may make the following query:

   - $\mathcal{O}$**VoteLR**$(\texttt{id}, v_0, v_1)$: $\mathcal{A}$ provides an identity $\texttt{id} \in \mathsf{ID}$, and two possible votes v_0, $v_1 \in V$. The challenger $\mathcal{C}$ executes $\mathsf{CreateVote}(\texttt{id}, v_\beta, P_{\texttt{v}}^{\texttt{id}}, S_{\texttt{v}}^{\texttt{id}})$, which produces the ballot b^β and the encryption data $\tilde{r}^\beta$. Then it executes $\mathsf{FakeProof}$ to obtain the simulated encryption data $\tilde{r}'^\beta$. Finally, $\mathcal{C}$ executes $\mathsf{CreateBallotProof}(b^\beta, sk_{\texttt{a}}, P_{\texttt{a}}^{\texttt{id}})$ and outputs the content proof σ^β. $\mathcal{C}$ provides $(b^\beta, \tilde{r}^\beta, \tilde{r}'^\beta, \sigma^\beta)$ to $\mathcal{A}$.

4. **Output:** The output of the experiment is the guess of the adversary for the bit β.

We say that a voting protocol provides coercion-resistant cast-as-intended verifiability if it provides cast-as-intended verifiability and there exists an algorithm $\mathsf{FakeProof}$ such that for any probabilistic polynomial time (p.p.t.) adversary $\mathcal{A}$, the following advantage is negligible in the security parameter λ:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{CR\text{-}CaI}} = |\ \Pr[\mathsf{Exp}_{\mathcal{A},V}^{\mathsf{CR\text{-}CaI},0} = 1] \text{ - } \Pr[\mathsf{Exp}_{\mathcal{A},V}^{\mathsf{CR\text{-}CaI},1} = 1]\ |$$

Chapter 4

Return Codes with Single Voting: Neuchâtel's Scheme

4.1 Introduction

Switzerland has a long history on direct participation of its citizens in decision making processes. Besides traditional elections where voters choose their representatives in the Federal Assembly, citizens can participate in several other voting events. Citizens can propose popular voting initiatives on their own (after having obtained enough popular support by collecting signatures), and then parties and governments themselves (at the communal, cantonal or federal level) can organize referendums in order to ask the citizens for their opinion on a new law or a modification of the Constitution, among others. Ultimately, Swiss citizens have the chance to participate in 3-4 voting processes a year in average.

Remote electronic voting was first introduced in Switzerland in three cantons: Geneva, Zurich and Neuchâtel [58]. The first binding trials were held in 2004. Nowadays 14 cantons offer the electronic voting channel to their electors, which until recently has been restricted use by up to 10% of the eligible voters.

In 2011, the Federal Council of Switzerland started a task force to study the security issues of electronic voting. As a result, the Federal Council published in 2013 a report with the requirements for extending the use of electronic voting systems to a larger part of the electorate. This framework [119], which became binding in January 2014, provides requirements for functionality, security, verifiability, and testing/certification which could allow the electronic voting systems to be extended to 30%, 50% or up to 100% of the electorate. More specifically, while current electronic voting systems may be allowed to be used for up to 30% of the electorate provided that they fulfill a certain set of functional and security requirements, systems to be used for up to 100% of the electorate are required to additionally provide verifiability features. Although the modality of electronic voting (DRE, remote, ...) is not specified in the report, it refers to electronic voting systems where the vote is cast electronically. In this chapter, we will talk specifically of remote electronic voting systems.

According to the report by the Federal Council, systems to be used for up to 50% of electors are required to provide methods for cast-as-intended verification, and systems for up to 100% of the electorate are required to additionally provide methods for recorded-as-cast and counted-as-recorded verification, while also enforcing the separation of duties on operations impacting the privacy, integrity and verifiability of the system.

In this chapter we present an electronic voting protocol for providing cast-as-intended verification, which is an evolution of the protocol implemented for the Norwegian elections in 2011 and 2013 [60, 61, 101] (called the Norwegian protocol/scheme from now on), which has been presented in Section 2.4.2 . The presented protocol improves the Norwegian scheme by not needing to rely on the assumption that two independent server-side entities do not collude to preserve voter privacy.

Although the early proposal for the Norwegian scheme, [100] (not the one finally implemented) also did not require that assumption for privacy, this new protocol provides a great performance improvement on the client-side number of operations comparatively, therefore taking the best of both schemes.

Then we add the particularity of only allowing voters to cast one vote through the electronic voting channel, and therefore we give provisions for ensuring that such vote is considered to be cast only in the case that it represents the voter intention, by means of a confirmation phase executed by the voter. This feature is one of the requirements of the Federal Council for systems to be used by up to 50% of the electorate in Switzerland, and therefore the protocol is suitable to be used in such scenarios.

In fact, this protocol has been implemented by Scytl and used for the first time in a binding election in the canton of Neuchâtel, in the federal referendum conducted on March 8th 2015. From 111,080 eligible voters, 23,927 were registered at the citizen electronic portal Guichet Unique [87] (from which the electronic voting application could be accessed) and 5,132 chose to cast their vote electronically using this protocol, which represents 21,45% of the voters who had the chance to use the electronic voting channel. Considering all voting channels, general participation in the referendum was 41,24%. The contribution of the author of this thesis to the project consisted of the design of the protocol, the analysis of the security of the scheme, and the provision of support to the development team during the implementation.

4.2 Improving the Norwegian solution

As we have explained in 2.4.2, in proposals based on return codes [60, 100, 101], after the voter has selected her choices, the voting device sends an encrypted vote to the remote voting server, where return codes are computed from the encrypted vote and sent back to the voter for verification. Voters have received, prior to the voting

phase, a *verification card* where a return code (pre-computed during the election configuration phase) associated to each voting option is shown. If the return codes received by the voter match the options she selected, her vote was cast as intended. Otherwise, the voting device is corrupted and the voter may cast a new vote with a different device or using another channel.

The votes cast by the voters are traditionally encrypted using a probabilistic encryption scheme. This kind of scheme guarantees the privacy of the voter even in the case of multiple voting (thanks to the randomization, there is no way to tell if a voter votes for the same options twice), and also are suitable to be used in homomorphic tally or mixnet-based voting systems.

On the other hand, the return codes have to be computed with a deterministic function, since the values computed by the server-side during the voting phase have to match the values in the voter's verification card (for the same voting options), and such values have been computed in advance during the configuration of the election.

A naive approach would to remove the randomness of the encrypted voting options received at the server-side during the voting phase. However, this would pose a risk on the voter privacy. The first proposal for the Norwegian voting system [100] addressed this issue by generating two sets of encryptions of the voting options: one based on a probabilistic encryption scheme, and another one based on a deterministic encryption scheme. The first encryption was the one to be processed by the mixnet and decrypted at the end of the election, while the second one was used by the voting server to generate the return codes during the voting phase. The voting device generated zero-knowledge proofs of knowledge in order to prove that both sets of encryptions corresponded to the same voting options.

This solution posed serious performance issues at the voting device side, and therefore a variation of it was proposed by Kristian Gjoosteen and used in the Norwegian elections [60, 61, 101]. In that variation, which reduced approximately 2,5 times the number of operations at the client-side, the generation of the return codes during the voting phase was split in two independent server-side entities: a ballot box server and a code generation server. Each of these two servers had one half of the election private key. Therefore, when a vote encrypted with a probabilistic encryption scheme (and with the election public key) was cast, each one in turn could perform a partial decryption with its part of the election private key, so that at the code generation server the randomness of the encryption was finally removed. Before the partial decryption, the ballot box server applied a deterministic function over the encrypted vote, so that what was recovered by the code generation server was not the vote in clear, but some deterministic value from which the return code to be sent back to the voter could be computed.

The ballot box server and the code generation server were assumed not to collude. Otherwise, they could systematically decrypt all the votes cast by the voters, since

together they could recompose the private key of the election. In order to ensure their independence, both components were located in independent locations and managed by different companies.

Two independent server-side entities is not a requirement easy to solve, due to the high economic and organizational costs of setting up two different and independent environments. Moreover, global connectivity and the restricted number of vendors and providers for datacenters (for example) reduce the feasibility of such independence: the ballot box and the code generator servers may be at two datacenters which receive database support from the same company, for example.

Therefore, the solution we present in this chapter goes back to the one-server approach of [100], and focuses on the improvement on the client-side operations. Because client-side technology has undergone big changes in the last years, these operations may have even more impact today than when the initial scheme was presented in 2011, even if the voting devices are more powerful. For example, in 2011 the majority of web applications relied in the use of Java Applets for some non-standard web functionalities or time-consuming operations. However with the evolution of the World Wide Web and the advent of HTML5 [120] a whole set of new functionalities and better performance has been made available for just browser and JavaScript-based applications. Moreover, mobile devices usually lack support for Java Applets and the compatibility issues with the newest browsers make them unusable, therefore nowadays most of the web applications have moved to HTML + JavaScript. In fact, Scytl implemented its first voting device entirely in JavaScript in 2013 and now all their web-based applications are exclusively based in this technology at the client-side.

However, while Java Applets are compiled into bytecode that can be interpreted by the JVM, JavaScript is directly interpreted by browsers without any previous compilation, and therefore does not offer the same performance as Java Applets (although it is getting closer). Naturally, vote encryption and casting needs to be executed in an acceptable time-frame to prevent voter disenfranchisement.

In the proposal presented here, the number of operations to be performed at the client-side is considerably reduced compared to the proposal in [100]. Specifically, the cost of encrypting the vote and generating zero-knowledge proofs does not depend on the number of voting options selected by the voter anymore.

4.2.1 Solution Overview

The mechanisms for generating the return codes in [100, 60, 61, 101] and in our proposal have the same principle: (at least during the voting phase) a sole entity should not be able to generate return codes on its own. That would imply several things: (1) if the server is the one who generates the return codes, this means that it can decrypt the votes cast by the voters (since it has to entirely remove the randomness used for the encryption), or that it can systematically compute return

codes for all the voting options and then compare them with the ones it obtains from the cast votes. In both cases, it breaks the voters' privacy. (2) if the voting device is the one who generates the return codes, it can generate return codes for the options chosen by the voter and then cast a vote with any other content encrypted. Therefore, it defeats the purpose of the return codes, which is the verification that the vote is cast as intended.

As in prevous proposals, the generation of return codes in our protocol is divided in two steps: partial return code and final return code generation. While in [60, 61, 101], partial return codes were computed by the ballot box server and final return codes were computed by the code generation server, in our solution, as well as in [100], partial return codes are computed by the voting device.

The following is an overview of the solution:

Configuration phase: besides the election public and private keys, a set of return codes corresponding to the voting options is computed for each voter. Creating a different set of return codes per each voter enforces their secrecy, which is a key property for verifiability and privacy reasons, as we will see in next sections. The set of return codes for each voter is printed in a verification card, which is sealed and sent to the voter by postal mail (during the explanation of the protocol we do not take into account the possibility of a distributed generation of the verification cards for keeping the scheme simple. However, we will talk about that in Section 4.8.2).

Voting phase: the voter uses a voting device to select her voting options. Once she has finished, the voting device encrypts them using a probabilistic encryption scheme and the election public key. Besides that, the voting device also computes the partial return codes corresponding to the same voting options, and some zero-knowledge proofs to prove that in fact, the partial return codes and the encrypted vote correspond to the same voting options. The encrypted vote, the partial return codes and the proofs are sent to the voting server. The voting server first verifies the proofs, stores the encrypted vote and then uses the partial return codes to compute the final return codes, which are sent back to the voter.

Counting phase: the encrypted votes stored at the server are retrieved, passed through a mix-net and then decrypted to obtain the election results.

4.3 Confirmation Phase

In return code-based schemes, voters may re-vote, invalidating the previous vote, in case the return codes received do not match the selected voting options. Typically, this would happen if the voting device is malicious and encrypts voting options independently of what the voter decided to select, so voters may re-vote using a different voting device, or a different voting channel. However, some countries do not allow voters to cast multiple votes (such as France or Switzerland [95, 119]).

The solution we propose for such cases is to let the sending of the vote to the server and return code reception be a *preliminary vote casting* which has no validity (that vote will not be taken into account in the tally). Then we add a *confirmation phase* where the voter, after having inspected the received return codes, provides a confirmation that is sent to the voting server if she agrees with the vote she had preliminarily cast. The server stores this confirmation together with the cast vote as proof that the vote has been confirmed by the voter. Only confirmed votes will be taken into account at the tally. In case the voter does not confirm her vote, she can cast the vote through a different channel.

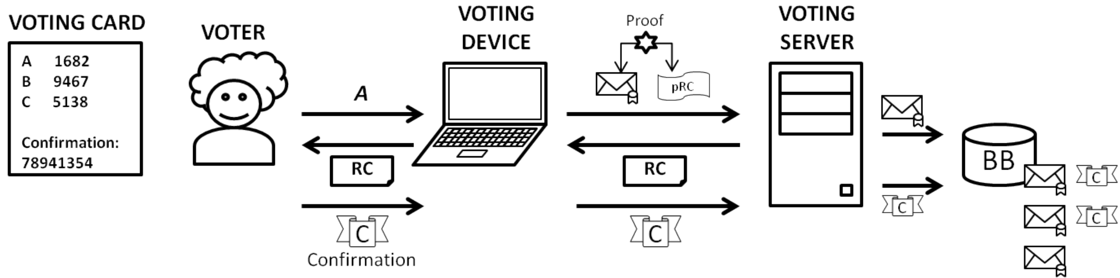

Figure 4.1: Neuchâtel voting phase overview

Single voting is a requirement of the Swiss Federal Council [119]. Therefore, our return code-based scheme, together with the confirmation phase, is suitable to be used (and has been used) in Swiss elections.

4.4 Protocol description

We start by describing the protocol used for the first time in March 2015's elections in Neuchâtel, using as a basis the general syntax presented in Chapter 3.

As stated in Chapter 3, we assume that non-cryptographic election specifications such as the set of voting options $V = \{v_1, \ldots, v_k\}$ and the counting function $\rho : (V \cup \{\bot\})^* \to R$ are already defined. A mixnet-based system is used, and therefore the set of possible results R is given by the multiset function ρ, which provides the cleartext votes cast by the voters in a random order [23]. The use of voting credentials to cast votes is not included in this description. However, it is later explained in Section 4.7.1.

The voting protocol uses an encryption scheme $(\mathsf{Gen_e}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EncVerify})$, a signature scheme $(\mathsf{Gen_s}, \mathsf{Sign}, \mathsf{SignVerify})$, and two NIZKPK schemes $(\mathsf{ProveEq}, \mathsf{VerifyEq}, \mathsf{SimEq})$ and $(\mathsf{ProveDec}, \mathsf{VerifyDec}, \mathsf{SimDec})$, which have already been defined in Section 1.3. Besides that, the protocol uses the following additional primitives/components:

Pseudo-random function family. A function family is a map $F : T \times D \to R$, where T is the set of keys, D is the domain and R is the range. A pseudoran-

dom function family (PRF) is a family of efficiently computable functions, with the following property: a random instance of the family is computationally indistinguishable from a random function, as long as the key remains secret. The function $f_K(x) = y$ denotes the function f from the family F, parameterized by the key K. Two keyed pseudo-random functions $f_k()$, $w_k()$ are used in the protocol, the latter with homomorphic properties.

Verifiable Mixnet. A verifiable mixnet is composed by two algorithms: the algorithm Mix receives a set of ciphertexts as input, and outputs a set of ciphertexts and a proof π_{mix} of correct mixing. The algorithm $\mathsf{MixVerify}$ receives as input two sets of ciphertexts and the proof of correct mixing π_{mix}, and outputs 1 or 0 depending on the result of the verification.

Additionally, an aggregation function ϕ which takes advantage of the homomorphic properties of $w_k()$ and a hash function H are also used in the protocol, which is characterized by the following protocols/algorithms:

> $\mathsf{Setup}(1^\lambda)$ runs $\mathsf{Gen_e}$ from the encryption scheme to generate an encryption key pair $(pk_\mathsf{e}, sk_\mathsf{e})$. The election public key is $pk = pk_\mathsf{e}$ and the election private key is $sk = sk_\mathsf{e}$. Alternatively, sk may consist of the shares of sk_e if there are several trustees. The algorithm then generates a random key K for the pseudo-random function f. The global audit key pair $(pk_\mathsf{a}, sk_\mathsf{a})$ is set to be $(\perp, K)$, and the global confirmation key pair (pk_c, sk_c) is set to be the signing key pair $(pk_\mathsf{s}, sk_\mathsf{s})$, which is produced by executing the algorithm $\mathsf{Gen_s}$ from the signature scheme.
>
> $\mathsf{Register}(1^\lambda, \mathtt{id}, sk_\mathsf{a}, sk_c)$ runs $\mathsf{Gen_e}$ of the encryption scheme, and the resulting key pair $(pk_\mathtt{id}, sk_\mathtt{id})$ is set to be the voter voting public and private data $(P_\mathsf{v}^\mathtt{id}, S_\mathsf{v}^\mathtt{id})$ (although this is formally an encryption key pair, it will be used differently in the next steps of the protocol). Then it chooses at random a voter confirmation code $\mathtt{CC}^\mathtt{id}$ from the code space $\mathcal{CS}$. For each voting option $v_i \in V$ it computes the corresponding return code $\mathtt{RC}_i^\mathtt{id} = f_K(\mathsf{w}_{sk_\mathtt{id}}(v_i))$, and computes the finalization code $\mathtt{FC}^\mathtt{id} = f_K(\mathsf{w}_{sk_\mathtt{id}}(\mathtt{CC}^\mathtt{id})$. The confirmation reference value $\mathtt{CRF}^\mathtt{id}$ is computed by running $\mathsf{Sign}(\mathtt{FC}^\mathtt{id}, sk_\mathsf{s})$ from the signature scheme. Finally, the set of reference values $\{\mathtt{RF}_i^\mathtt{id}\}_{i=1}^k$ is generated by computing $\mathtt{RF}_i^\mathtt{id} = H(\mathtt{RC}_i^\mathtt{id})$ for each generated return code. The voter public and private audit data are set as $P_\mathsf{a}^\mathtt{id} = \{\mathtt{RF}_i^\mathtt{id}\}_{i=1}^k$, $S_\mathsf{a}^\mathtt{id} = (\{v_i - \mathtt{RC}_i^\mathtt{id}\}_{i=1}^k, \mathtt{FC}^\mathtt{id})$, and the voter public and private confirmation data are set as $P_\mathsf{c}^\mathtt{id} = \mathtt{CRF}^\mathtt{id}$, $S_\mathsf{c}^\mathtt{id} = \mathtt{CC}^\mathtt{id}$.
>
> $\mathsf{CreateVote}(\mathtt{id}, \{v_{j_1}, \ldots, v_{j_t}\}, P_\mathsf{v}^\mathtt{id}, S_\mathsf{v}^\mathtt{id})$ receives the voting options selected by the voter as input, sets $v = \phi(\{v_{j_1}, \ldots, v_{j_t}\})$ and runs Enc from the encryption scheme, using the election public key pk and v to get the ciphertext c. Then it parses $S_\mathsf{v}^\mathtt{id}$ as $sk_\mathtt{id}$ and computes the partial return codes from the input voting options as $\{\mathtt{pRC}_{j_l}^\mathtt{id}\}_{l=1}^t = (\mathsf{w}_{sk_\mathtt{id}}(v_{j_1}), \ldots, \mathsf{w}_{sk_\mathtt{id}}(v_{j_t})$. Finally, it also computes $\mathsf{w}_{sk_\mathtt{id}}(c)$.

The following NIZKPK proofs are computed, to prove that the voting options in the ciphertext c and the voting options used for computing the partial return codes are the same:

- A proof $\pi_1 = \mathsf{ProveEq}(g, c, pk_{\mathtt{id}}, \mathsf{w}_{sk_{\mathtt{id}}}(c), sk_{\mathtt{id}})$ which proves that $\mathsf{w}_{sk_{\mathtt{id}}}(c)$ is computed with the private key $sk_{\mathtt{id}}$ corresponding to the public key $pk_{\mathtt{id}}$.
- A proof $\pi_2 = \mathsf{ProveEq}(g, pk, \mathsf{w}_{sk_{\mathtt{id}}}(c)/\phi(\mathtt{pRC}^{\mathtt{id}}_{j_1}, \ldots, \mathtt{pRC}^{\mathtt{id}}_{j_t}), r \cdot sk_{\mathtt{id}})$ which proves that the value $\mathsf{w}_{sk_{\mathtt{id}}}(c)$ is equivalent to the encryption of the aggregation ϕ of partial return codes $\{\mathtt{pRC}^{\mathtt{id}}_{j_l}\}^t_{l=1}$ under the election public key pk (note that r denotes the encryption randomness used to compute c).

The result of the above computations is a ballot b consisting of

$$b = \left(c, \{\mathtt{pRC}^{\mathtt{id}}_{j_l}\}^t_{l=1}, \mathsf{w}_{sk_{\mathtt{id}}}(c), P^{\mathtt{id}}_{\mathtt{v}}, \pi_1, \pi_2\right) .$$

The encryption data $\tilde{r}$ is set to $\perp$.

$\mathsf{ProcessBallot}(\mathsf{BB}, \mathtt{id}, b)$ checks that there is not already a ballot in BB for the voter identity $\mathtt{id}$, that $\mathtt{id} \in \mathsf{ID}$ and that there is not another ballot in BB with the same ciphertext value c. If any of these validations fails, it stops and outputs 0. Otherwise, the algorithm continues by validating the NIZKPK proofs π_1, π_2 from the ballot b running $\mathsf{VerifyEq}$, and runs $\mathsf{EncVerify}$ to verify that the ciphertext c is correctly formed. In case all the validations are successful, 1 is returned. Otherwise, the algorithm returns 0.

$\mathsf{CreateBallotProof}(b, sk_{\mathtt{a}}, P^{\mathtt{id}}_{\mathtt{a}})$ parses b as $(c, \{\mathtt{pRC}^{\mathtt{id}}_{j_l}\}^t_{l=1}, \mathsf{w}_{sk_{\mathtt{id}}}(c), pk_{\mathtt{id}}, \pi_1, \pi_2)$, parses $sk_{\mathtt{a}}$ as K, and $P^{\mathtt{id}}_{\mathtt{a}}$ as $\{\mathtt{RF}^{\mathtt{id}}_i\}^k_{i=1}$. Then it computes the set of return codes corresponding to the voting options encrypted in the ballot b as follows:

- Computes the final return code value $\overline{\mathtt{RC}^{\mathtt{id}}_{j_l}} = f_K(\mathtt{pRC}^{\mathtt{id}}_{j_l})$ for each of the partial return codes $\{\mathtt{pRC}^{\mathtt{id}}_{j_l}\}^t_{l=1}$ in b.
- Checks that for each return code $\overline{\mathtt{RC}^{\mathtt{id}}_{j_l}}$, where $l = 1, \ldots, t$, $H(\overline{\mathtt{RC}^{\mathtt{id}}_{j_l}}) \in \{\mathtt{RF}^{\mathtt{id}}_i\}^k_{i=1}$. In a positive case, the proof of content σ takes the value of the set of return codes $\{\overline{\mathtt{RC}^{\mathtt{id}}_{j_l}}\}^t_{l=1} = (\overline{\mathtt{RC}^{\mathtt{id}}_{j_1}}, \ldots, \overline{\mathtt{RC}^{\mathtt{id}}_{j_t}})$. Otherwise, $\sigma = \perp$.

$\mathsf{AuditBallotProof}(\{v_{j_1}, \ldots, v_{j_t}\}, \sigma, S^{\mathtt{id}}_{\mathtt{a}})$ parses σ as $\{\overline{\mathtt{RC}^{\mathtt{id}}_{j_l}}\}^t_{l=1}$ and $S^{\mathtt{id}}_{\mathtt{a}}$ as $(\{v_i - \mathtt{RC}^{\mathtt{id}}_i\}^k_{i=1}, \mathtt{FC}^{\mathtt{id}})$. Then it checks that for each $v_x \in \{v_{j_1}, \ldots, v_{j_t}\}$ the corresponding $\mathtt{RC}^{\mathtt{id}}_x$ matches one of the values in $\{\overline{\mathtt{RC}^{\mathtt{id}}_{j_l}}\}^t_{l=1}$. If the verification is positive, it outputs 1. Otherwise it outputs 0.

$\mathsf{Confirm}(\mathtt{id}, b, S^{\mathtt{id}}_{\mathtt{v}}, S^{\mathtt{id}}_{\mathtt{c}}, P^{\mathtt{id}}_{\mathtt{c}}, sk_{\mathtt{a}})$ is an interactive protocol between two parties, A and B:

- Party A parses $S^{\mathtt{id}}_{\mathtt{v}}$ as $sk_{\mathtt{id}}$ and $S^{\mathtt{id}}_{\mathtt{c}}$ as $\mathtt{CC}^{\mathtt{id}}$, and computes $\mathtt{CM}^{\mathtt{id}} = \mathsf{w}_{sk_{\mathtt{id}}}(\mathtt{CC}^{\mathtt{id}})$.

– Party B parses $sk_{\mathtt{a}}$ as K and $P_{\mathtt{c}}^{\mathtt{id}}$ as $\mathtt{CRF}^{\mathtt{id}}$. Then it computes $\overline{\mathtt{FC}^{\mathtt{id}}} = f_K(\mathtt{CM}^{\mathtt{id}})$ and runs $\mathsf{SignVerify}(pk_{\mathtt{s}}, \overline{\mathtt{FC}^{\mathtt{id}}}, \mathtt{CRF}^{\mathtt{id}})$ from the signature scheme (where $pk_{\mathtt{s}}$ is parsed from the global public confirmation key pk_c). In case the signature verification is successful, it sets the ballot confirmation $C_b = (\overline{\mathtt{FC}^{\mathtt{id}}}, \mathtt{CRF}^{\mathtt{id}})$ and the auxiliary data $\sigma' = \perp$. Otherwise, it returns $\perp$.

$\mathsf{ProcessConfirm}(\mathsf{BB}, \mathtt{id}, C_b)$ checks that there is a ballot entry in BB for the voter identity id, and that it has already not been confirmed. Then it runs $\mathsf{SignVerify}(pk_{\mathtt{s}}, \overline{\mathtt{FC}^{\mathtt{id}}}, \mathtt{CRF}^{\mathtt{id}})$ from the signature scheme (where $pk_{\mathtt{s}}$ is parsed from the global public confirmation key pk_c). In case all verifications succeed, 1 is returned. Otherwise it returns 0.

$\mathsf{Tally}(\mathsf{BB}, sk)$ runs ProcessBallot for all the ballots in the bulletin board which have a confirmation C_b stored. Then for those which passed the verification it parses C_b as $(\overline{\mathtt{FC}^{\mathtt{id}}}, \mathtt{CRF}^{\mathtt{id}})$ and runs $\mathsf{SignVerify}(pk_{\mathtt{s}}, \overline{\mathtt{FC}^{\mathtt{id}}}, \mathtt{CRF}^{\mathtt{id}})$, discarding those for which the validation result is 0. Ciphertexts c are extracted from the remaining ballots, shuffled and then decryted running $\mathsf{Dec}(c, sk)$ for each one, to obtain the cleartext v (in case sk was divided in shares, a secret reconstruction algorithm is used to recover the private key previous to decryption). Then $\phi^{-1}(v)$ outputs the factors v_i composing v for which it is tested that $v_i \in V$. Otherwise, the whole factorized vote is discarded. The result r composed of the values v_i recovered from each vote is provided as the output.

In next developments of the eVoting project conducted in Neuchâtel, a verifiable mixnet and a verifiable decryption will be used in order to also allow to verify the counting process. Therefore, Tally and VerifyTally will be defined as follows:

$\mathsf{Tally}(\mathsf{BB}, sk)$ runs ProcessBallot for all the ballots in the bulletin board which have a confirmation C_b stored. Then for those which passed the verification it parses C_b as $(\overline{\mathtt{FC}^{\mathtt{id}}}, \mathtt{CRF}^{\mathtt{id}})$ and runs $\mathsf{SignVerify}(pk_{\mathtt{s}}, \overline{\mathtt{FC}^{\mathtt{id}}}, \mathtt{CRF}^{\mathtt{id}})$, discarding those for which this validation failed. Ciphertexts c are extracted from the remaining ballots and passed as input to the mixnet, which runs the Mix algorithm. The resulting list of mixed ciphertexts $\{C_m\}$ is decrypted: for each ciphertext $c_z \in \{C_m\}$, $\mathsf{Dec}(c_z, sk)$ is run to obtain v_z, which is tested to belong to V. The ProveDec algorithm is run with the statement (c, v_z) and the witness sk as input. The outputs are the list of decrypted votes $r = \{v_z\}$ and the proofs of correct mixing and decryption, $\Pi = (\pi_{mix}, \{C_m\}, \pi_{dec})$. Finally $\phi^{-1}(v)$ outputs the factors v_i composing v for which it is tested that $v_i \in V$. Otherwise, the whole factorized vote is discarded. The result r composed of the values v_i recovered from each vote is provided as the output.

$\mathsf{VerifyTally}(\mathsf{BB}, r, \Pi)$ performs the same validations as Tally: runs ProcessBallot for all the ballots in the bulletin board which have a confirmation C_b stored. Then for those which passed the verification it parses C_b as $(\overline{\mathtt{FC}^{\mathtt{id}}}, \mathtt{CRF}^{\mathtt{id}})$ and runs $\mathsf{SignVerify}(pk_{\mathtt{s}}, \overline{\mathtt{FC}^{\mathtt{id}}}, \mathtt{CRF}^{\mathtt{id}})$, discarding those for which this validation is not successful. It extracts the ciphertexts c from the ballots which have passed the previous validations and composes the list $\{C\}$. Then it parses

Π as $(\pi_{mix}, \{C_m\}, \pi_{dec})$ and verifies that the mixing was correct by running $\mathsf{MixVerify}(C, C_m, \pi_{mix})$. Finally it checks that the decryption of each ciphertext was correct by running VerifyDec from the NIZKPK scheme, using as input the statement $(c_z, \phi(\{v_i\}))$, for all the ciphertexts $c_z \in \{C_m\}$ and the proof $\pi_l \in \pi_{dec}$, where $\phi(\{v_i\})$ denotes the aggregation of all the voting options v_i in the z-th entry of r (belonging to the same ballot). The output is the result of these validations.

4.4.1 Workflow

The described algorithms are organized in the following phases:

Configuration phase: the election authorities run the Setup algorithm. They publish the election public key pk, the global audit and confirmation public keys $(pk_{\mathtt{a}}, pk_c)$ and an empty list ID of voter identities, as well as the set of voting options V. They provide the global audit private key $sk_{\mathtt{a}}$ to both the registrars and the bulletin board manager. Finally, the global confirmation private key sk_c is provided to the registrars.

Registration phase: Voters register to participate in the election. To register, a voter first provides her identity id to the registrars, who run the Register algorithm. The outputs $(S_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{a}}^{\mathtt{id}}, S_{\mathtt{c}}^{\mathtt{id}})$ are provided to the voter in the verification card, while the tuple $(\mathtt{id}, P_{\mathtt{v}}^{\mathtt{id}}, P_{\mathtt{a}}^{\mathtt{id}}, P_{\mathtt{c}}^{\mathtt{id}})$ is published in the bulletin board BB and id is added to ID.

Voting phase: This phase consists of several steps:

1. The voter provides $(\mathtt{id}, S_{\mathtt{v}}^{\mathtt{id}})$ to the voting device, as well as a set of selected voting options $\{v_{j_1}, \ldots, v_{j_t}\} \in V$. The voting device may get $P_{\mathtt{v}}^{\mathtt{id}}$ from the bulletin board, or receive it additionally from the voter. Then it runs the Vote protocol and produces a ballot b. The ballot b and the voter identity id are sent to the bulletin board manager.

2. Upon reception of $(\mathtt{id}, b)$, the bulletin board manager runs the ProcessBallot algorithm to verify the incoming ballot. In case the result of the execution is 1, the bulletin board BB is updated with the ballot b and the voter identity id, together with the status *ballot received* and the process continues. Otherwise the process stops and the voting device is notified of the error.

3. The bulletin board manager executes CreateBallotProof with the newly arrived ballot, parsing the voter's public audit data $P_{\mathtt{a}}^{\mathtt{id}}$ from the bulletin board. In case the operation is successful, it outputs the proof of content σ consisting of the set of return codes $\{\overline{\mathtt{RC}_{j_l}^{\mathtt{id}}}\}_{l=1}^{t}$. The bulletin board manager updates the status of the ballot in the BB to *return codes generated*, and forwards the return codes to the voting device. In case the operation is not successful the process stops and the voting device is notified accordingly.

4. The voting device shows the voter the set of received return codes $\{\overline{\mathtt{RC}^{\mathtt{id}}_{j_l}}\}_{l=1}^{t}$ from the proof of vote content σ. She is then asked to confirm the ballot cast by providing the confirmation value $\mathtt{CC}^{\mathtt{id}}$ from her confirmation private data $S^{\mathtt{id}}_{\mathsf{c}}$ to the voting device, which she will do **only** in case the AuditBallotProof algorithm accepts.

5. The voting device then runs the first phase of the Confirm algorithm and sends the resulting confirmation message $\mathtt{CM}^{\mathtt{id}}$, together with the voter identity $\mathtt{id}$, to the bulletin board manager. The bulletin board manager then runs the second phase of the Confirm algorithm.

6. If the operation in the bulletin board manager is successful, the resulting ballot confirmation C_b is stored together with the ballot b for the given identity $\mathtt{id}$, in the bulletin board and the ballot status is updated to *confirmed*. The ballot confirmation C_b is also sent back to the voting device. In case the operation is not successful, the process stops and the voter is notified accordingly.

7. Finally, the voter checks whether the displayed ballot confirmation C_b matches the value $\mathtt{FC}^{\mathtt{id}}$ in her private audit data $S^{\mathtt{id}}_{\mathtt{a}}$, which is printed in the verification card. In case of a successful verification, the received finalization code serves the voter as a confirmation of the correct submission and confirmation of her vote. Otherwise, she can raise a complaint to the election administrators, who will initiate an investigation. The voter may need to cast her vote using a different channel (i.e. at a polling station).

Counting phase: The election authorities run the interactive protocol Tally on BB using the election private key sk, obtaining and publishing in the bulletin board the result r and the proof Π, or set $r = \perp$ in case of error. The auditors run the VerifyTally protocol using as input the contents in the bulletin board. In case their output is 1, the result r is announced to be fair. Otherwise, an investigation is opened to detect the reason of failure.

4.5 Security of the Protocol

In this section we show that the protocol presented in Section 4.4 (considering the complete version of Tally and VerifyTally) satisfies the properties of ballot privacy, strong consistency, strong correctness, cast-as-intended verifiability and coercion-resistant cast-as-intended verifiability according to the definitions in Section 3.3.

First of all, we include the definition of the Enc2Vote scheme, as defined in [25], which we use for proving some of the properties of our scheme:

- $\mathsf{Setup}(1^\lambda)$: Runs $\mathsf{Gen}_{\mathsf{e}}(1^\lambda)$ to produce a key pair $(pk_{\mathsf{e}}, sk_{\mathsf{e}})$; the public output is pk and the secret one, sk.

- $\mathsf{Vote}(v, y)$: Encrypts v with $y = pk_{\mathsf{e}} : c \leftarrow \mathsf{Enc}_r(v)$. Returns c.

- $\mathsf{ProcessBallot}(\mathsf{BB}, s)$: If the new submission s already appears on the board BB, rejects it; otherwise accepts it and adds it to BB.

- Tally: Decrypts all ballots $\mathbf{b}$ on the board using sk_{e} to get the underlying votes $\mathbf{v}$ and evaluates $r \leftarrow \rho(\mathbf{v})$; Returns r. (Here $\mathbf{b}$ and $\mathbf{v}$ are used to represent all of the ballots and underlying votes, respectively).

4.5.1 Ballot Privacy

Theorem 4.1. *Let* $(\mathsf{Gen_e}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EncVerify})$ *be an NM-CPA secure encryption scheme and* $(\mathsf{ProveDec}, \mathsf{VerifyDec}, \mathsf{SimDec})$*,* $(\mathsf{ProveEq}, \mathsf{VerifyEq}, \mathsf{SimEq})$ *be NIZKPK schemes with the zero-knowledge property. Then the protocol presented in Section 4.4 satisfies the ballot privacy property.*

The ballot privacy definition from Section 3.3.2 is based in the indistinguishability of two experiments, which depend on a bit β:

- $\mathsf{Exp}^{\mathsf{priv},0}_{\mathcal{A},V}$ is the experiment when $\beta = 0$ and the adversary is presented with the bulletin board BB_0.

- $\mathsf{Exp}^{\mathsf{priv},1}_{\mathcal{A},V}$ is the experiment when $\beta = 1$ and the adversary is presented with the bulletin board BB_1.

We perform the following steps to prove that the protocol provides ballot privacy: in a first step, we prove in Lemma 4.1 that the original experiment $\mathsf{Exp}^{\mathsf{priv},\beta}_{\mathcal{A},V}$ is indistinguishable from the point of view of $\mathcal{A}$ from an experiment $\mathsf{Exp}^{\mathsf{priv},\beta'}_{\mathcal{A},V}$, where the tallier provides a simulated proof of the tally result (needed due to the ballot privacy definition, as indicated in [23]). In a second step we prove in Lemma 4.2 that the experiments $\mathsf{Exp}^{\mathsf{priv},\beta'}_{\mathcal{A},V}$ and $\mathsf{Exp}^{\mathsf{priv},\beta''}_{\mathcal{A},V}$, where in $\mathsf{Exp}^{\mathsf{priv},\beta''}_{\mathcal{A},V}$ the NIZK proofs inside the ballot b are simulated instead of honestly generated, are indistinguishable by $\mathcal{A}$. In a third step we prove that this scenario is indistinguishable from one where the partial return codes are generated at random, through Lemma 4.3. Finally, in Lemma 4.4 we make a reduction of the last experiment to the ballot privacy of the $\mathsf{Enc2Vote}$ scheme.

Lets consider $\mathsf{SimProof}$ to be a simulator of the mixing and decryption proofs which produces proofs with the same distribution as the honest ones. Then consider the experiment $\mathsf{Exp}^{\mathsf{priv},\beta'}_{\mathcal{A},V}$ in which the challenger, when executing $\mathsf{Tally}(\mathsf{BB}_0, sk)$, provides the result r and the proof Π^* which is the output of $\mathsf{SimProof}$. The following lemma is straightforward to prove:

Lemma 4.1. The experiments $\mathsf{Exp}^{\mathsf{priv},\beta}_{\mathcal{A},V}$ and $\mathsf{Exp}^{\mathsf{priv},\beta'}_{\mathcal{A},V}$ are computationally indistinguishable for $\beta \in \{0, 1\}$.

Then let's consider $\mathsf{SimVote}(\mathtt{id}, v_j, P^{\mathtt{id}}_{\mathtt{v}}, S^{\mathtt{id}}_{\mathtt{v}})$ to be a modification of $\mathsf{CreateVote}(\mathtt{id}, v_j, P^{\mathtt{id}}_{\mathtt{v}}, S^{\mathtt{id}}_{\mathtt{v}})$ from Section 4.4 where instead of running $\mathsf{ProveEq}$ for the generation of the proof π_2, the algorithm SimEq is run to obtain the simulated proof π'_2 with the

same distribution as the non-simulated one. Then define $\mathsf{Exp}^{\mathsf{priv},\beta''}_{\mathcal{A},V}$ as the experiment in which the challenger runs SimVote instead of CreateVote. The following lemma is also straightforward to prove:

Lemma 4.2. The experiments $\mathsf{Exp}^{\mathsf{priv},\beta'}_{\mathcal{A},V}$ and $\mathsf{Exp}^{\mathsf{priv},\beta''}_{\mathcal{A},V}$ are computationally indistinguishable for $\beta \in \{0,1\}$, given a (ProveEq, VerifyEq, SimEq) zero-knowledge NIZKPK scheme.

In the following transformation, we define $\mathsf{SimVote}_2(\mathtt{id}, v_j, P^{\mathtt{id}}_{\mathtt{v}}, S^{\mathtt{id}}_{\mathtt{v}})$ to be a modification of SimVote where, instead of computing the partial return code value as $\mathsf{w}_{sk_{\mathtt{id}}}(v_j)$, it is selected at random from the same value space. We also define an oracle which, when $\mathcal{A}$ makes a query to compute the hash value of the generated return codes, it returns one of the values in the public voter audit data $P^{\mathtt{id}}_{\mathtt{a}} = \{\mathtt{RF}^{\mathtt{id}}_i\}^k_{i=1}$.

Then we consider the experiment $\mathsf{Exp}^{\mathsf{priv},\beta'''}_{\mathcal{A},V}$ where, when the adversary submits the query $\mathcal{O}$**VoteLR**$(\mathtt{id}, v_0, v_1)$ the challenger executes $\mathsf{SimVote}_2$ instead of SimVote, and when the adversary asks for the hash value of a return code, one of the values from $\{\mathtt{RF}^{\mathtt{id}}_i\}^k_{i=1}$ is returned by the oracle. The following lemma is easy to prove:

Lemma 4.3. The experiments $\mathsf{Exp}^{\mathsf{priv},\beta''}_{\mathcal{A},V}$ and $\mathsf{Exp}^{\mathsf{priv},\beta'''}_{\mathcal{A},V}$ are computationally indistinguishable for $\beta \in \{0,1\}$, given the pseudorandom function $\mathsf{w}()$ and a hash function which acts as a random oracle.

Now we consider the Enc2Vote scheme defined in [25]. In their work, the authors have proven the following theorem:

Theorem 4.2. *Let* $(\mathsf{Gen_e}, \mathsf{Enc}, \mathsf{Dec})$ *be an NM-CPA secure encryption scheme, then* Enc2Vote *has ballot privacy.*

Next, we proceed to reduce the ballot privacy property of our scheme to the ballot privacy property of the Enc2Vote scheme.

Lemma 4.4. Let $\mathcal{A}$' be a p.p.t. adversary that interacts with a challenger $\mathcal{C}$, such that $|$ Pr[$\mathsf{Exp}^{\mathsf{priv},0'''}_{\mathcal{A},V} = 1$] - Pr[$\mathsf{Exp}^{\mathsf{priv},1'''}_{\mathcal{A},V} = 1$] $|$ is non-negligible. Then, there exists an adversary $\mathcal{A}$" that breaks the ballot privacy property of the Enc2Vote scheme.

Proof. In the reduction, we use $\mathcal{A}$" as the challenger for $\mathcal{A}$', and $\mathcal{A}$" interacts with $\mathcal{C}$ in the same way as in the experiment defined in [25]. The reduction is as follows:

In the **Setup phase**, $\mathcal{C}$ sets up two empty bulletin boards BB_0 and BB_1, runs the $\mathsf{Gen_e}$ algorithm and keeps the sk^{e2v} key for itself, while it publishes the pk^{e2v} key on the bulletin board. In turn, $\mathcal{A}$" generates a random key K for the pseudo-random function $f_K()$, runs $\mathsf{Gen_s}$ to generate a key pair $(pk_{\mathsf{s}}, sk_{\mathsf{s}})$ and publishes $pk = pk^{e2v}$ and the keys $sk_{\mathsf{a}} = K$, $pk_c = pk_{\mathsf{s}}$, $sk_c = sk_{\mathsf{s}}$ on the bulletin board visible by $\mathcal{A}$'.

In the **Registration phase**, when $\mathcal{A}$' makes the $\mathcal{O}$Register query, $\mathcal{A}$" runs the $\mathsf{Register}(1^\lambda, \mathtt{id}, sk_{\mathsf{a}}, sk_c)$ algorithm from our protocol, keeps the voter's private voting

and audit data $S^{\mathtt{id}}_{\mathtt{v}} = sk_{\mathtt{id}}$, $S^{\mathtt{id}}_{\mathtt{a}} = (\{v_i - \mathtt{RC}^{\mathtt{id}}_i\}_{i=1}^k, \mathtt{FC}^{\mathtt{id}})$, provides the voter's confirmation private data $S^{\mathtt{id}}_{\mathtt{c}} = \mathtt{CC}^{\mathtt{id}}$ to $\mathcal{A}$', and finally publishes $(\mathtt{id}, P^{\mathtt{id}}_{\mathtt{v}} = pk_{\mathtt{id}}, P^{\mathtt{id}}_{\mathtt{a}} = \{\mathtt{RF}^{\mathtt{id}}_i\}_{i=1}^k, P^{\mathtt{id}}_{\mathtt{c}} = \mathtt{CRF}^{\mathtt{id}})$ on the bulletin board visible by $\mathcal{A}$'.

During the **Voting phase**, when $\mathcal{A}$' submits the $\mathcal{O}$VoteLR query, $\mathcal{A}$" submits the **Vote** query to $\mathcal{C}$, who responds by publishing a ballot b_{e2v} to the bulletin board visible by $\mathcal{A}$". $\mathcal{A}$" parses b_{e2v} as c, computes $\mathsf{w}_{sk_{\mathtt{id}}}(c)$, $\pi_1 = \mathsf{ProveEq}(g, c, pk_{\mathtt{id}}, \mathsf{w}_{sk_{\mathtt{id}}}(c), sk_{\mathtt{id}})$, picks v'_i at random from the $\mathsf{w}()$ output message space and computes $\pi'_2 = \mathsf{SimEq}(g, pk, \mathsf{w}_{sk_{\mathtt{id}}}(c)/v'_i)$. The resulting ballot $b = (b_{e2v}, v'_i, \mathsf{w}_{sk_{\mathtt{id}}}(c), pk_{\mathtt{id}}, \pi_1, \pi'_2)$ is published in the bulletin board visible by $\mathcal{A}$', together with the voter identity $\mathtt{id}$. Finally $\mathcal{A}$" executes the $\mathsf{Confirm}$ algorithm from our protocol and posts the generated ballot confirmation C_b next to the corresponding ballot b, in the bulletin board visible by $\mathcal{A}$'.

When $\mathcal{A}$' submits the $\mathcal{O}$getVotingData($\mathtt{id}$) query, $\mathcal{A}$" just returns the values $(S^{\mathtt{id}}_{\mathtt{v}}, S^{\mathtt{id}}_{\mathtt{a}})$ to $\mathcal{A}$'. When $\mathcal{A}$' submits the $\mathcal{O}$Cast($\mathtt{id}, b, C_b$) query, $\mathcal{A}$" parses C_b as $(\overline{\mathtt{FC}^{\mathtt{id}}}, \mathtt{CRF}^{\mathtt{id}})$ and runs the $\mathsf{SignVerify}(pk_{\mathtt{s}}, \overline{\mathtt{FC}^{\mathtt{id}}}, \mathtt{CRF}^{\mathtt{id}})$ algorithm. If the result is 1, then it parses b as $\left(c, \mathtt{pRC}^{\mathtt{id}}_j, \mathsf{w}_{sk_{\mathtt{id}}}(c), P^{\mathtt{id}}_{\mathtt{v}}, \pi_1, \pi_2\right)$ and submits a **Ballot**(c) query to $\mathcal{C}$. Then $\mathcal{A}$" puts $\mathtt{id}, b$ and C_b in the bulletin board visible by $\mathcal{A}$'.

In the **Counting phase**, $\mathcal{C}$ posts the result of evaluating $\mathsf{Tally}(\mathsf{BB}_0, sk^{e2v})$ on the bulletin board visible to $\mathcal{A}$". $\mathcal{A}$" in turn runs $\mathsf{SimProof}(\mathsf{BB}_\beta, r)$, where BB_β is the bulletin board shown by $\mathcal{C}$ to $\mathcal{A}$", and publishes (r, Π^*) on the bulletin board visible by $\mathcal{A}$'.

At the end of the experiment, $\mathcal{A}$' outputs a bit and $\mathcal{A}$" outputs the same bit. As we can see, the outputs of $\mathcal{A}$"as a result of the interaction with $\mathcal{A}$' have the same distribution as in the ballot privacy experiment in [25]. Therefore, the reduction is sound. □

4.5.2 Strong Consistency

We define the following algorithms for which the presented protocol provides strong consistency, according to the definition in Section 3.3.3:

- Let the extraction algorithm $\mathsf{Extract}((b), sk)$ defined in Section 3.3.3 parse b as $\left(c, \{\mathtt{pRC}^{\mathtt{id}}_{jl}\}_{l=1}^t, \mathsf{w}_{sk_{\mathtt{id}}}(c), P^{\mathtt{id}}_{\mathtt{v}}, \pi_1, \pi_2\right)$, run $\mathsf{EncVerify}(c, pk_{\mathsf{e}})$ and in case it returns 1, get $m = \mathsf{Dec}(c, sk_{\mathsf{e}})$ and compute $\phi^{-1}(m)$ to get the values $\{m_i\}_{i=1}^t$. Then it tests whether each $m_i \in V$ or not. In a positive case, it returns $\{v_i\}_{i=1}^t = \{m_i\}_{i=1}^t$, otherwise it returns $\perp$.

- Let the validation algorithm $\mathsf{ValidInd}(\mathtt{id}, b, C_b) = 1$ run $\mathsf{EncVerify}(c, pk_{\mathsf{e}})$, validate the NIZK proofs π_1, π_2, parse C_b as $(\overline{\mathtt{FC}^{\mathtt{id}}}, \mathtt{CRF}^{\mathtt{id}})$ and run $\mathsf{SignVerify}(pk_{\mathtt{s}}, \overline{\mathtt{FC}^{\mathtt{id}}}, \mathtt{CRF}^{\mathtt{id}})$. The output is 1 if all the validations succeed, 0 otherwise.

- Let ρ be the counting function that provides its inputs as outputs in a shuffled order, removing any input set $\{v_i\}_{i=1}^t$ for which some $v_i \notin V$.

Theorem 4.3. *Let* Tally *produce a sound proof* Π *of correct mixing and decryption, and* VerifyTally *output* 1*. Then the protocol defined in Section 4.4 has the property of strong consistency with respect to the above definitions of* Extract, ValidInd, ρ.

Given the definition we have provided of the ValidInd algorithm, all the ballots which have been accepted in the ballot box will pass the validations done by the Tally algorithm prior to decryption, described in Section 4.4. Therefore, we can see that clearly Tally and the combination of the extraction algorithm and ρ remove the same ballots, given that the mixing and decryption algorithms are correct (which is ensured by the soundness of the proof Π and the fact that VerifyTally outputs 1).

4.5.3 Strong Correctness

Theorem 4.4. *Let* $(\mathsf{Gen_e}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EncVerify})$ *be a randomized encryption scheme and* $(\mathsf{Gen_s}, \mathsf{Sign}, \mathsf{SignVerify})$ *be a signature scheme with the correctness property. Then the protocol defined in Section 4.4 has the property of strong correctness.*

In the protocol defined in Section 4.4, the only condition for which ProcessBallot $(\mathsf{BB}, \mathtt{id}, b)$ may output 0, given a ballot b produced by a honest registered voter that was not already registered by the adversary, is that a previous entry with the same ciphertext c is already present in the bulletin board. Given that $(\mathsf{Gen_e}, \mathsf{Enc}, \mathsf{Dec},$ $\mathsf{EncVerify})$ is a probabilistic encryption scheme, this probability is negligible in the security parameter λ. Given the correctness of the signature scheme, ProcessConfirm will always succeed for honestly created ballot confirmations.

4.5.4 Cast-as-Intended Verifiability

Theorem 4.5. *Let* $(\mathsf{ProveEq}, \mathsf{VerifyEq}, \mathsf{SimEq})$ *be a sound NIZKPK scheme,* $f_k()$ *be a (collision-resistant) pseudorandom function,* $(\mathsf{Gen_s}, \mathsf{Sign}, \mathsf{SignVerify})$ *be an unforgeable signature scheme,* $w_k()$ *be a random permutation on a large space according to* λ*, and the code space* $\mathcal{CS}$ *be large according to the security parameter* λ*. Then the protocol presented in Section 4.4 satisfies the cast-as-intended verifiability property.*

According to the definition presented in 3.3.5, the adversary succeeds if it is able to generate the expected return code for a voting option which is not in the ballot, or if it is able to confirm a vote without collaboration.

First we examine **Case A**, where the adversary has to provide the return code corresponding to a voting option which is not in the ballot. Lets consider a ballot b for the voting option v_j which is used by $\mathcal{A}$ to do the $\mathcal{O}$ProofVote query, and parse it as $b = \big(c, \mathtt{pRC}_j^{\mathtt{id}}, \mathsf{w}_{sk_{\mathtt{id}}}(c), P_{\mathtt{v}}^{\mathtt{id}}, \pi_1, \pi_2\big)$. Recall that the challenger computes the return code as $\overline{\mathtt{RC}_j^{\mathtt{id}}} = f_{sk_c}(\mathtt{pRC}_j^{\mathtt{id}})$. Then for fulfilling that at the output of the experiment $\mathsf{AuditBallotProof}(v_x, \sigma, S_{\mathsf{a}}^{\mathtt{id}}) = 1$ it is necessary that $f_{sk_c}(\mathtt{pRC}_j^{\mathtt{id}}) = f_{sk_c}(\mathtt{pRC}_x^{\mathtt{id}})$ when $v_j \neq v_x$. One possibility is that $\mathtt{pRC}_j^{\mathtt{id}} = \mathtt{pRC}_x^{\mathtt{id}}$. However, $w_k()$ is a permutation on large space according to λ and therefore it is infeasible that $w_k(v_x) = w_k(v_j)$ when $v_x \neq v_j$. Another possibility is that $f_{sk_c}(\mathtt{pRC}_j^{\mathtt{id}}) = f_{sk_c}(\mathtt{pRC}_x^{\mathtt{id}})$ when $\mathtt{pRC}_j^{\mathtt{id}} \neq \mathtt{pRC}_x^{\mathtt{id}}$, which is also negligible since $f_k()$ is a collission-resistant function.

Another strategy for $\mathcal{A}$ is to provide, in the $\mathcal{O}$ProofVote query a ballot b where $\mathsf{Extract}(b, sk) = v_j$, but the partial return code corresponds to v_x. Therefore the return code generated by the challenger when running CreateBallotProof would correspond to the pair $(v_x - \mathtt{RC}_x^{\mathtt{id}})$ and the validation when executing AuditBallotProof would succeed. However, the challenger executes ProcessBallot prior to generating the return code. Recall that in the ProcessBallot algorithm the NIZKPK proofs (π_1, π_2) are verified. The probability that such proofs are verified successfully when the voting option v_j encrypted in c does not match the one used to compute the partial return code is negligible given the soundness property of the NIZKPK scheme.

Secondly we take **Case B**, where the adversary can confirm and cast a vote on behalf of a non-corrupt voter without a previous collaboration of the challenger (which means, without having to succeed in the $\mathcal{O}$ProofVote query). Recall that the algorithm ProcessConfirm verifies if a signature published during the configuration phase matches the generated value by running SignVerify. Given the unforgeability property of the signature scheme, the best strategy for $\mathcal{A}$ is to guess the voter confirmation private data $S_{\mathsf{c}}^{\mathtt{id}}$ such that when it makes a $\mathcal{O}$getConfirmation query $\mathcal{C}$ answers with the ballot confirmation C_b. The probability of guessing this value is of $\frac{1}{\mathcal{CS}}$, where $\mathcal{CS}$ is the message space size for the voter confirmation private data $S_{\mathsf{c}}^{\mathtt{id}}$. According to the security parameter λ, this probability is negligible.

4.5.5 Coercion-resistant cast-as-intended

According to the definition in Section 3.3.6, a voting protocol provides the coercion-resistant cast-as-intended verifiability property in case an adversary cannot distinguish between two ballots given the proofs of content, which in the case of this specific protocol consists of σ. Note that in the definition of the ballot privacy property it was already considered the fact that $\mathcal{A}$ cannot distinguish between ballots posted in different bulletin boards, even when it can execute $\mathsf{CreateBallotProof}(b, sk_{\mathsf{a}}, P_{\mathsf{a}}^{\mathtt{id}})$ from the posted ballot, and get the proof of content σ. Therefore, this indistinguishability property has already been considered during the previous analysis on ballot privacy.

4.6 Protocol implementation

In this section we provide an implementation for the described protocol. This implementation is the one used in the electronic voting system implemented for elections in Neuchâtel in 2015.

Some of the primitives used for this concrete instantiation have already been defined in Section 1.3 :

- For encryption, the Signed ElGamal encryption scheme is used. This scheme generates randomized ciphertexts, and has been proven to be NM-CPA secure in [25].

- The signature scheme is RSA with the hash variant (RSA-FDH). This signature scheme has been proven to be unforgeable against chosen message attacks in the random oracle model [18].

- The NIZKPK schemes EqDL and DecP are used for proving equality of discrete logarithms and for proving correctness of the decryption process. These NIZKPK schemes satisfy the properties of completeness, knowledge soundness and zero-knowledge [45], [113].

Additionally, the following building blocks are used:

Voting options. The voting options $V = \{v_1, \ldots, v_k\}$ are chosen as small bit-length primes belonging to the group $\mathbb{G}$ defined for the ElGamal encryption scheme. The aggregation function ϕ is then described as the product operation, and a vote is encoded as the product of voting options chosen by the voter prior to the encryption. The function ϕ^{-1} is the factorisation operation: after the votes are decrypted, the individual voting options are recovered by factorizing the resulting value. Therefore, it has to be ensured that the product of t of such primes, where t is the number of selections a voter can make, is not larger than p.

Pseudo-random function family. We use two different pseudo-random functions in the protocol: The function denoted by $f_k()$ is an HMAC function composed by a SHA-256 hash function, parameterized by the symmetric key k. As detailed in [15], HMAC is a PRF whose resistance against collision is the one of the underlying hash scheme. Up to date, the collision of the SHA-256 hash function is considered to be negligible. The function $w_k()$ is the exponentiation function $w_k(g) = g^k$. This function defined for the group $\mathbb{G}$ and computed over the voting options, which are small primes, is pseudo-random: according to what is discussed in [60], the hardness of distinguishing values v_i^k where v_i are small primes $\in \mathbb{G}$ and the set size $|V|$ is small, from random values uniformly taken from $\mathbb{G}$, is, under the DDH assumption, equivalent to solve a discrete logarithm. It has to be pointed out that the function $w_k(c)$, only takes into account the elements (c_1, c_2) for the exponentiation.

Mixnet. We use the verifiable mixnet proposed by Stephanie Bayer and Jens Groth [13]. This mixnet has been proven by its authors to be sound, meaning that MixVerify will only output 1 given a correct execution of Mix, and zero-knowledge in the standard model, or in the random oracle model in case of using the Fiat-Shamir heuristic for making the proofs non-interactive.

4.6.1 Performance

This instantiation is efficient. For a *t-out-of-k* voting scheme where voters can select up to t values from k available in the election, considering that t prime numbers can be encrypted into a single ciphertext, the cost of a ballot generation at the voting device is the following: the encryption of the t prime numbers using the Signed ElGamal encryption scheme costs 3 exponentiations. The generation of the partial

return codes for t options requires t exponentiations and the proofs and intermediate values require 7 exponentiations.

In case that t prime numbers cannot be fit into one ciphertext, the cost of the encryption, proofs and intermediate values has to be multiplied by the number of ciphertexts required, although in case of using multiple public key encryption two exponentiations can be avoided. We denote multiple public key encryption to a solution for encrypting many plaintexts which consists on using a different public key for computing each ciphertext, but the same randomness for all of them. The security of the ElGamal encryption scheme is not affected by this variation (still relies on the ability of solving discrete logarithms), and a considerable proportion of the exponentiations can be avoided by having a common component c_1 and Schnorr signature (c_3, c_4) for all the individually encrypted plaintexts.

In the real implementation, an extra layer of protection has been added for the partial return codes, which are encrypted using ElGamal with a key owned by the bulletin board manager. This adds a total of $t+1$ exponentiations if multiple public key encryption is used.

Finally, it has to be taken into account that some of these exponentiations can be computed by the voting device while the voter navigates through the application, prior to receiving the voter's choices. In fact, from $t+10$ exponentiations (considering a single ciphertext), 8 can be pre-computed. If we take into account the extra encryption layer of partial return codes, $t+9$ exponentiations can be pre-computed from a total of $2t+11$.

This protocol is much more efficient than the early proposal for the Nowegian elections [100], described in Section 2.4.2. In that proposal, 3 exponentiations were required for individually encrypting each voting option using the Signed ElGamal encryption scheme. Each partial return code required 2 exponentiations, and then a NIZK proof relating each pair [encrypted voting option - partial return code] costs 2 more exponentiations. Moreover, partial return codes were also encrypted with the ElGamal scheme. In total, this means that the voting device had to perform up to $(3+2+2+2) \cdot t$ exponentiations for t voter selections. In case multiple public key encryption was used, the number of exponentiations could be reduced to $(2+2+1+1) \cdot t + 3$ exponentiations.

The system implemented in Norway [60, 61] needed less exponentiations, even compared with the protocol we present in this chapter, due to the fact that the computation of the partial return codes, and the NIZK proofs which linked them to the encrypted voting options, are done at the server-side. Therefore, the voting device is only required to compute the Signed ElGamal encryption of the voter's choices. However, each voting option is required to be individually encrypted, since individual return codes have to be computed from them. This results in an operational cost at the voting device of $3t$ exponentiations. In case of using multiple public key encryption the cost is of $t+2$ exponentiations.

4.7 Authentication, usability and correctness: implementation details

The protocol described in Section 4.4 may pose significant usability problems to the voters. In order to cast a vote, the voter is asked to type in the voting device a series of secret values from her verification card, such as the private key $sk_{\mathtt{id}}$ or the confirmation value $\mathtt{CC}^{\mathtt{id}}$. Moreover, for the cast-as-intended verification the voter has to compare the return codes she receives at the voting device, $\mathtt{RC}^{\mathtt{id}}$, with those in her verification card corresponding to her selections. The same applies to the finalization code $\overline{\mathtt{FC}^{\mathtt{id}}}$.

The problem is that, according to current cryptographic key length recommendations [3], the aforementioned values have a length of 2048 or 256 bits, depending on whether they are used in an asymmetric or a symmetric key cryptographic operation ($sk_{\mathtt{id}}$ and $\mathtt{CC}^{\mathtt{id}}$ are of 2048 bits, while $\{\overline{\mathtt{RC}^{\mathtt{id}}}\}$ and $\overline{\mathtt{FC}^{\mathtt{id}}}$ are of 256 bits). To be more concrete, in case a Base32 encoding is used to represent such values, this implies 410 and 52 characters respectively. It is clearly not realistic to ask a voter to perform such tasks.

Therefore, an additional layer for improving usability is required on top of the protocol. This layer allows the reduction of the length of the values in the verification card, and provides the voter's code generation private key to the voting device in a way that is transparent (unnoticed) for the voter. Moreover, we mix this layer with authentication features, and also use it for checking the correctness of the votes cast by the voters. This usability layer was used in the referendum conducted in Neuchâtel, and it is similar to that in the Norwegian system used in 2011 and 2013.

4.7.1 Authentication and private keys provision

Although it has not been detailed in the protocol description for the sake of clarity, some authentication mechanism is assumed to be put in place so that only authorized voters are allowed to cast a vote.

In the system implemented in Neuchâtel there are two layers of authentication: the first one is handled by the citizen portal *Guichet Unique* [87], from which the voters access the voting application, and the second one is managed by the electronic voting system. This second layer consists on a username/PIN-based authentication. The PIN is randomly generated during the registration phase and printed onto the voter's verification card, while the username is provided by the first layer of authentication, i.e the *Guichet Unique* [87]. However, the protocol can also work with its own authentication system. In this case, voters receive their username and PIN, preferably by two different channels (for example, the PIN could be provided in the verification card and the username by mail or SMS). The username is then also a random value generated during registration.

The authentication layer managed by the electronic voting system is used not

only to qualify a user as an authorized voter in the election, but also to provide her some cryptographic secrets, such as the voter's key pair $(pk_{\mathtt{id}}, sk_{\mathtt{id}})$, or even a voter's signing key pair $(pk^s_{\mathtt{id}}, sk^s_{\mathtt{id}})$ if the voters do not have one, for digitally signing their votes prior to casting them. Digital signatures are commonly used for protecting the integrity of the ballots during transmission and storage, as well as for identifying the origin of a ballot, in order to ensure that only one vote per voter is counted.

The voter's key pairs are generated during registration and uploaded to the voting server, using a key container (for example, a PKCS#12 [69]) sealed with a password which is derived from a combination of the username and the PIN, for example using a password-based key derivation function (PBKDF [70], bcrypt [98]). During authentication, the password-protected secrets are provided by the voting server to the voting device, where the voter enters the username and PIN in order to open the key container and recover the secrets. Therefore, only the voter in possession of her username and PIN is able to recover her secrets.

4.7.2 Short Return Codes

As we explained before, the return codes and the confirmation and finalization codes are too long for being usable. Therefore, the tasks of entering them into the voting device or comparing them may suppose a real challenge for the majority of the voters. In order to solve that, shorter values which can be univocally mapped to the long ones in the protocol are used.

In our protocol, the Register algorithm executed during voter registration is extended to generate such short values, which we denote by $\{\mathtt{sRC}^{\mathtt{id}}\}_{i=1}^{k}, \mathtt{sCC}^{\mathtt{id}}, \mathtt{sFC}^{\mathtt{id}}$, that are printed in the verification card. One key factor of this approach is the length of such values, which actually represents a trade-off between usability and security: the longer they are, the harder it is to guess them by a corrupted voting device, but the harder is to use them by the voter. Specifically, in the Neuchâtel setup, short return codes are of 4 digits, and the confirmation and finalization codes are of 7 digits.

Remember that the protocol described in sections 4.4, 4.6 generates long values for the return and finalization codes, according to the security parameter λ. Therefore, how are these codes related to the shorter values to be shown to the voter? A trivial approach would be to obtain the short values by truncation. However, this solution may not be compatible with some of the requirements of the scheme. For example, we have seen in Section 4.5 that one of the requirements for cast-as-intended verifiability is that the probability that two different voting options have the same return code is negligible, which is satisfied according to the return code generation explained in sections 4.4, 4.6: consider the output of the SHA-256 hash function as uniformly random, and that we are generating return codes for 20 voting options. According to the birthday paradox, the probability that two of them have the same value is of $1,72 \cdot 10^{-75}$. However, in case of using truncation the probability that two options have the same 4-digit short return code is about $0,019$. In case of

generating return codes for 100 voting options the probabilities are of $4,31 \cdot 10^{-74}$ and $0,39$ respectively.

Since return codes and their short versions are generated during registration, it is possible to put in place some control which ensures that all the short return codes are unique. For example, in the case of using truncation, a suitable function f for the Register and RCGen algorithms could be tested and found, for which the generated return codes corresponding to a voting card, when truncated, are unique.

The alternative that was implemented in the Norwegian system and also in Neuchâtel is based in another procedure: the short return, confirmation and finalization codes are generated at random and of the desired length, checking that unique values are used in each verification card. Then the registrars compute a *mapping table* in which each code $\mathtt{sRC}_i^{\mathtt{id}}$ or $\mathtt{sFC}^{\mathtt{id}}$ is related to the corresponding long code $\mathtt{RC}_i^{\mathtt{id}}$ or $\mathtt{FC}^{\mathtt{id}}$. During the voting phase, the code generator uses this table to obtain the corresponding short codes, given the long codes generated by the protocol. The mapping table is designed to be an injective function from codes $\{\mathtt{RC}^{\mathtt{id}}\}_{i=1}^{k}, \mathtt{FC}^{\mathtt{id}}$ to short codes $\{\mathtt{sRC}^{\mathtt{id}}\}_{i=1}^{k}, \mathtt{sFC}^{\mathtt{id}}$, and its design takes care of the secrecy of such short codes after the registration phase.

Our implementation of the mapping table contains one entry for each (long) return code $\mathtt{RC}_i^{\mathtt{id}}$ of the form:

$$[H(\mathtt{RC}_i^{\mathtt{id}}), \mathsf{Enc}^{\mathsf{s}}(\mathtt{sRC}_i^{\mathtt{id}}; \mathtt{RC}_i^{\mathtt{id}})],$$

where H denotes a hash function, and $\mathsf{Enc}^{\mathsf{s}}(m;k)$ denotes the encryption of the message m with a symmetric encryption algorithm and a secret key k. The SHA-256 hash function and the AES-128 symmetric encryption algorithm are used in the system implemented in Neuchâtel.

An additional entry is computed in the same way with the (long) finalization code $\mathtt{FC}^{\mathtt{id}}$ and the short finalization code $\mathtt{sFC}^{\mathtt{id}}$.

Security: As we mentioned, using these short values has an impact on the security of the scheme, specifically in the cast-as-intended verifiability property. In the case of the short return codes, assuming that they are randomly chosen and that the voting device has only one chance to show the values to the voter, the probability of the voting device cheating in the content of the vote, but guessing the right values to show to the voter is of 10^{-4t}, being t the number of return codes that the voting device has to guess (which in fact corresponds to the number of selections the voter made). We assume that the voter will notice that the return codes do not match the first time they are presented on the screen.

Regarding the confirmation code, one of the assumptions from the security analysis in Section 4.5 for the cast-as-intended verifiability property is that such code was uniformly distributed in a large code space and therefore it was hard to guess. Shortening the value to, for example, 7 digits, reduces this difficulty and makes this

attack feasible by brute force, although an average of $10^7/2$ calls to the server-side would have to be made. In order to mitigate this attack, the number of calls the voting client can do to confirm a vote is limited by the voting server (this can be done by IP, by voter identifier, ... etc.).

The case of the short finalization code is similar to the one of the short return codes. Assuming that it is randomly generated, and that the voter will notice any discrepancy the first time she sees it, the chance of the voting client of successfully guessing this value is of 10^{-7}.

The injectivity of the map from long to short codes is granted given the non-collision property of the hash function, as far as the map entries are of limited size: for example, given a SHA-256 hash function, the map would need to have a size of $4,8 \cdot 10^{35}$ to have a probability of 10^{-6} of finding a map entry which corresponds to two different long values. The limitation can be ensured by having a different map per voter / verification card.

The secrecy of the short values to be returned to the voter is granted due to the fact that they are stored in an encrypted form, being the key for decryption the corresponding long value, which is known by the server only when a ballot or confirmation message is received and the $\mathsf{CreateBallotProof}$ or $\mathsf{Confirm}$ algorithms are executed. The map entries are uploaded to the voting server in randomized order (ordering them alphabetically, for example), in order to remove any link between the map entries and the voting options that could be inferred by the order of the computation of the map entries during configuration.

4.7.3 Vote correctness

Additionally, we use the mapping table to ensure that a ballot that is accepted by the voting server contains valid choices according to the election rules. For example, in case a ballot contains some $v'_j \notin V$, the (long) return code computed by the code generator as $f_{sk_c}(v_j'^{sk_{\mathtt{id}}})$ will not find an entry on the mapping table described above (and the same happens for the reference values $\{\mathtt{RF}_i^{\mathtt{id}}\}_{i=1}^k$ in the underlying protocol).

Other information can also be checked: imagine the scenario where a voter can select one party list, and then can give some weight to individual candidates. An approach for implementing this validation consists of checking that the first (long) return code corresponds to an entry in the table which has been labelled to contain a party, and that the rest correspond to candidate-type mappings.

The alternative that has been used both in the Norwegian protocol and in the solution implemented in Neuchâtel is to add such metadata (party or candidate labels) to the return code generation mechanism itself, in such a way that the mapping table entries are not labelled. This alternative has been chosen for preventing targeted attacks against table entries of a specific type.

In the Norwegian protocol [60, 61, 101], type identifiers are added to the encrypted short return codes in the mapping table. Therefore a party voting option and a candidate voting option may have respective entries in the mapping table which look like the following:

$$H(\mathtt{RC}_i^{\mathtt{id}}), \mathsf{Enc}^{\mathsf{s}}((\mathtt{sRC}_i^{\mathtt{id}}, \text{"party"}); \mathtt{RC}_i^{\mathtt{id}}),$$
$$H(\mathtt{RC}_j^{\mathtt{id}}), \mathsf{Enc}^{\mathsf{s}}((\mathtt{sRC}_j^{\mathtt{id}}, \text{"candidate"}); \mathtt{RC}_j^{\mathtt{id}}).$$

The code generator retrieves these type identifiers when decrypting the short return codes from the mapping table, and therefore can check that the structure of the vote is correct according to the election rules.

In the protocol used in Neuchâtel, these type identifiers are included in the ballot cast by the voter, and then used to compute the return codes at the server. The ballot cast by the voter, then, has the following contents:

$$b = (c, \mathtt{pRC}_1^{\mathtt{id}} \text{ - "party"}, \ldots, \mathtt{pRC}_t^{\mathtt{id}} \text{ - "candidate"}), \mathsf{w}_{sk_{\mathtt{id}}}(c), P_{\mathtt{v}}^{\mathtt{id}}, \pi_1, \pi_2)$$

In a first step the type identifiers are checked (in the example, that there is only one type identifier for party list and that the rest are for candidates). In the second step, these type identifiers are added to the computation of the return codes:

$$\overline{\mathtt{RC}_{j1}^{\mathtt{id}}} = f_{sk_c}(v_{j1}^{sk_{\mathtt{id}}}, \text{"party"})$$
$$\overline{\mathtt{RC}_{jt}^{\mathtt{id}}} = f_{sk_c}(v_{jt}^{sk_{\mathtt{id}}}, \text{"candidate"})$$

where ',' denotes a concatenation.

The same is done in the registration phase when generating the *mapping table*. Therefore, an invalid combination of a pair (partial return code , type identifier) will result in an entry of the table with negligible probability, given the collission-resistance properties of cryptographic hash functions.

4.7.4 Ballot Box vs Bulletin Board

There was no public bulletin board in the system deployed in Neuchâtel, although its future use has not been ruled out. Instead, ballots and confirmation tags were stored in a private ballot box managed by the voting server, and all the public information of the protocol was provided to the system entities which needed it. However, voters did have the chance to check that their ballots were indeed stored in the ballot box: at the end of the voting process, together with the finalization code $\overline{\mathtt{FC}^{\mathtt{id}}}$, voters received a voting receipt R_v which was computed by the voting server as a hash of the ballot b. This voting receipt was digitally signed, with a private key owned by the voting server.

In the counting phase, the electoral authorities published the hash of all the confirmed ballots in the ballot box on a web-site (the Guichet Unique). That is, the ones that the Tally algorithm selected to be processed by the mix-net. The voters could check that their receipts were present in the published list, and therefore that their ballots were present at the counting phase. The voting receipt signature could be used to detect any false claim of a voter who complained for her receipt not being published.

4.8 Protocol extensions

In this section we describe some functionalities with which the protocol can be extended in order to provide some additional properties. Some of them are related to practical issues, while others are more related to improving the trust model.

4.8.1 Supporting multiple entry points

The presented protocol is designed to have two phases: the ballot submission phase, and the ballot confirmation phase. For usability purposes, the system could allow the voter to initiate two different sessions in order to perform each phase. For example, the voter could decide, after receiving the return codes, to check them and confirm her vote at a latter time (maybe because she didn't have time to check them at that moment). Bad connectivity could also cause an accidental session break after the voter has submitted her vote at the first step, and it would not be convenient that this break prevents the voter from checking her return codes and confirming her vote once the connectivity is back.

The protocol used in Neuchâtel supports vote sending and confirmation in different sessions in the following way: if a voter who authenticates to the system already has a ballot b stored in the bulletin board, the bulletin board manager executes CreateBallotProof and provides the generated return codes, which are forwarded to the voting device. From this point, the return code verification and confirmation phases are developed as usual, as described in previous sections. The time frame in which this may happen can be restricted in order to reduce vote selling attacks.

In a similar way, the finalization code may be sent back to the voter during a specific period of time when she logs-in after having confirmed her vote. This allows the voter to check whether the vote was correctly confirmed although the voting device fails to show it after confirmation, accidentally due to connectivity issues, or on purpose in order to diminish the voter's trust.

4.8.2 Distributed return code generation

In the introduction of this chapter, we explained that according to the Swiss Federal Council report on electronic voting, systems to be used by up to 100% of the electorate had to provide some sort of separation of duties on operations impacting the verifiability of the system. Specifically, in further evolutions of this protocol, the

generation of verification cards will be distributed into a set of trustees which preserve the secrecy of their contents, together with the printing service. The purpose is that the verification cards are only known by the printing service and the voter, at the end of the registration phase. The return code generation during the voting phase will be also distributed among the trustees, in such a way that without the collaboration of all of them valid return codes cannot be provided to the voter (understanding by valid, that for those values the $\mathsf{AuditBallotProof}$ algorithm succeeds with overwhelming probability).

Here we describe a proposal for a new protocol which uses a set of z code generators computing the codes in a distributed manner. We also include the short codes in the description for taking into account the usability layer.

An overview of the system is the following: during the registration phase, each code generator generates a part of the long return (and finalization) codes and voters' private keys $sk_{\mathtt{id}}$. These pieces of information are sent to the printing service, which recomposes them and relates them to a random set of short return codes, previously unrelated to voting options, in order to construct the mapping table that will be used, during the voting phase, to translate the long return codes generated by the protocol to the short return codes to be sent to the voters. All the information is processed in an encrypted form, in such a way that the printing service only decrypts the contents to be printed in the verification card, and the code generators only recover the mapping table which does not reveal neither the long nor the short return codes.

During the voting phase, upon reception of a ballot at the bulletin board manager, it is forwarded to all the code generators, each of which compute their part of the long return codes. The bulletin board manager is able to recompose the complete long return codes in order to extract, from the mapping table, the short codes to be delivered to the voter.

The following is a more detailed description of the modifications to the protocol:

Setup: during the setup phase the code generators jointly generate an encryption key pair $(pk_{\mathtt{CG}}, sk_{\mathtt{CG}})$ using a distributed key generation algorithm, where the m-th code generator has a private share $sk_{\mathtt{CG}}(m)$. The printing service also generates a key pair $(pk_{\mathtt{print}}, sk_{\mathtt{print}})$ by running $\mathsf{Gen_e}$, and publishes $(pk_{\mathtt{print}})$.

Registration: during this phase, a series of steps are generated by the code generators and the printing service.

1. Each code generator m runs $\mathsf{Gen_e}$ from the encryption scheme to generate two key pairs $(pk_{\mathtt{cga}}, sk_{\mathtt{cga}})$, $(pk_{\mathtt{cgb}}, sk_{\mathtt{cgb}})$, calculates $\beta = sk_{\mathtt{cga}} \cdot sk_{\mathtt{cgb}}$ and computes the m-th long return code for each voting option v_i in V: $\mathtt{RC}_i^{\mathtt{id}}(m) = (v_i)^{\beta}$. It also computes the m-th long finalization code as $\mathtt{FC}^{\mathtt{id}}(m) = (\mathtt{CC}^{\mathtt{id}})^{\beta}$, where $\mathtt{CC}^{\mathtt{id}}$

is taken at random from the chosen code space. Then it encrypts them by running $E(\texttt{RC}_i^{\texttt{id}}(m)) = \mathsf{Enc}(\texttt{RC}_i^{\texttt{id}}(m), pk_{\texttt{CG}})$ and $E(\texttt{FC}^{\texttt{id}}(m)) = \mathsf{Enc}(\texttt{FC}^{\texttt{id}}(m), pk_{\texttt{CG}})$, $E(\texttt{CC}^{\texttt{id}}(m)) = \mathsf{Enc}(\texttt{CC}^{\texttt{id}}(m), pk_{\texttt{print}})$.

2. The code generators publicly take all the possible values of short return codes (according to the defined short code space for return codes, we denote its size as N), and all the possible values of short finalization codes (also according to the defined short code space for finalization codes, we denote its size as P) and mix them using a re-encryption mixnet, where they are the mix-nodes, using the printing service public key $pk_{\texttt{print}}$.

3. Each code generator m provides the set of encrypted long return codes $\{E(\texttt{RC}_i^{\texttt{id}}(m))\}_{i=1}^{k}$, corresponding to the k voting options in the election, the encrypted finalization code $E(\texttt{FC}^{\texttt{id}}(m))$, the encrypted confirmation code $E(\texttt{CC}^{\texttt{id}}(m))$ and the private key $sk_{\texttt{cga}}(m)$ to the printing service.

4. The shuffled and reencrypted short return codes $\{E(\texttt{sRC}_x^{\texttt{id}})\}_{x=1}^{N}$ and short finalization codes $\{E(\texttt{sFC}_y^{\texttt{id}})\}_{y=1}^{P}$ are also provided to the printing service.

5. The printing service receives the sets $\{E(\texttt{RC}_i^{\texttt{id}}(m))\}_{i=1}^{k}$, $E(\texttt{FC}^{\texttt{id}}(m))$, $E(\texttt{CC}^{\texttt{id}}(m))$ and $sk_{\texttt{cga}}(m)$ from the z code generators and puts them together by computing: $E(\texttt{RC}_i^{\texttt{id}}) = \prod_{m=1}^{z} E(\texttt{RC}_i^{\texttt{id}}(m))$ for each i such that $v_i \in V$, $E(\texttt{FC}^{\texttt{id}}) = \prod_{m=1}^{z} E(\texttt{FC}^{\texttt{id}}(m))$, $E(\texttt{CC}^{\texttt{id}}) = \prod_{m=1}^{z} E(\texttt{CC}^{\texttt{id}}(m))$ and $sk_{\texttt{id}} = \prod_{m=1}^{z} sk_{\texttt{cga}}(m)$.

6. Then it selects at random a subset of k encrypted short codes from $\{E(\texttt{sRC}_x^{\texttt{id}})\}_{x=1}^{N}$, and one encrypted short finalization code from $\{E(\texttt{sFC}_y^{\texttt{id}})\}_{y=1}^{P}$.

7. For each selected encrypted short code, the printing service computes $\mathsf{Dec}((E(\texttt{sRC}_x^{\texttt{id}}) \cdot E(\texttt{RC}_i^{\texttt{id}})), sk_{\texttt{print}})$ and $\mathsf{Dec}((E(\texttt{sFC}_y^{\texttt{id}}) \cdot E(\texttt{FC}^{\texttt{id}})), sk_{\texttt{print}})$. The result is the combination of pairs short-long code encrypted with the code generators' public key, which is sent back to the code generators.

8. The printing service then uses its private key to decrypt the selected short return codes, preserving the link with the voting option index i corresponding to the assigned long code, the confirmation code and the short finalization code. All the information to be printed in the verification card is ready, including the voter's private key $sk_{\texttt{id}}$.

9. Finally, the code generators distributedly decrypt the information received from the printing service, and provide the combinations $(\texttt{sRC}_x^{\texttt{id}} \cdot \texttt{RC}_i^{\texttt{id}})$ for i such that $v_i \in V$, and $(\texttt{sFC}_y^{\texttt{id}} \cdot \texttt{FC}^{\texttt{id}})$ to the bulletin board manager as the mapping table. A one-way homomorphic function may additionally be computed by the printing service over the recovered values $E(\texttt{RC}_i^{\texttt{id}})$, for example $(E(\texttt{RC}_i^{\texttt{id}}))^{\alpha}$ where α is a known value. This way, the mapping entries can be indexed by the corresponding $(\texttt{RC}_i^{\texttt{id}})^{\alpha}$ key.

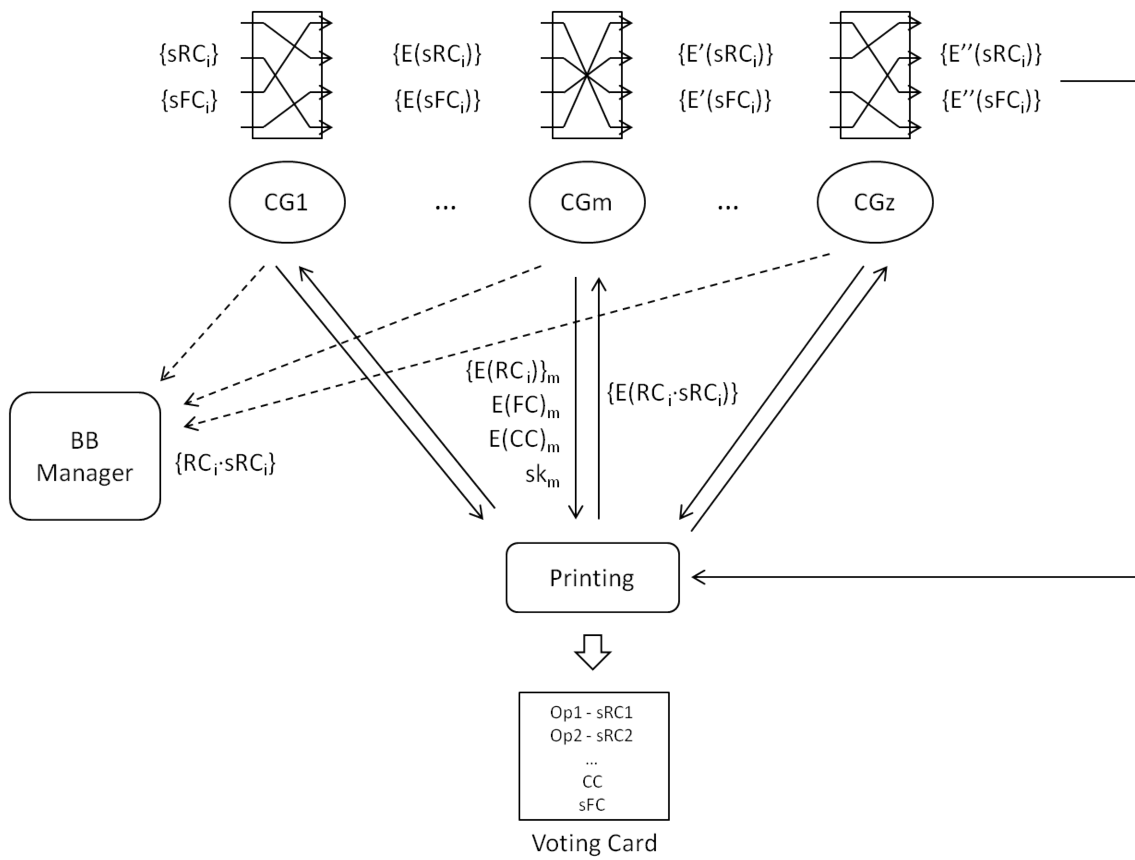

Figure 4.2: Distributed registration process

Voting: during the voting phase, when the bulletin board manager receives an incoming pair $(\mathtt{id}, b)$, it runs $\mathsf{ProcessBallot}$ and if the result is 1 it submits $(\mathtt{id}, b)$ to the code generators. Each code generator parses b as $(c, \{\mathtt{pRC}^{\mathtt{id}}_{j_l}\}^{t}_{l=1}, \mathsf{w}_{sk_{\mathtt{id}}}(c), P^{\mathtt{id}}_{\mathtt{v}}, \pi_1,$ $\pi_2)$ and computes each m-th long return code as $\mathtt{RC}^{\mathtt{id}}_{j_l}(m) = (\mathtt{pRC}^{\mathtt{id}}_{j_l})^{sk_{\mathtt{cgb}}}$. The m-th return codes from the z code generators are sent back to the bulletin board manager, who puts the values together by computing for each one $\prod^{z}_{m=1} \mathtt{RC}^{\mathtt{id}}_{j_l}(m)$ and using the resulting value to recover the short code $\mathtt{sRC}^{\mathtt{id}}_{j_l}$, which can be sent back to the voter.

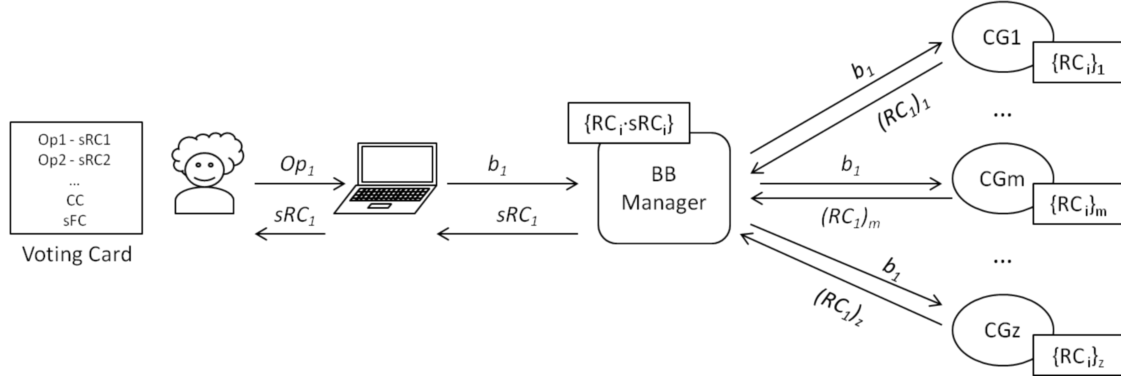

Figure 4.3: Voting process with distributed code generators

Similar steps are done for performing the confirmation of a vote.

This modification of the protocol from Section 4.4 can be implemented with the primitives described in Section 4.6. All the operations can generate proofs of correct computation, for example by using a verifiable mixnet and generating NIZK proofs of correct decryption. Thus, it can be ensured that the contents of the verification cards and the mapping table are correct, and that the submission of a ballot for a specific candidate will result in the generation of the expected return code.

One of the trust assumptions we do in the protocol described in Section 4.4 is that the bulletin board manager is honest, in the sense that it performs the described validations and does not continue with the process when they fail. Otherwise, a collusion between a malicious voting device and a malicious bulletin board manager could lead to the generation of return codes which do not correspond to the ballot contents, defeating the purpose of the cast-as-intended verification mechanism. Recall that the bulletin board manager verifies the NIZK proofs of knowledge in the ballot b during the execution of the $\mathsf{ProcessBallot}$ algorithm. In case of a collusion, the voting device may send a ballot b containing partial return codes, which do not correspond to the encrypted voting options, without being detected. This results in the fact that the return codes the voter checks do not correspond to the voting options that are taken into account for computing the election result.

In a distributed computation environment with multiple trustees, as described in this section, we can remove this assumption by repeating the NIZKPK validation

in each code generator, which only contributes to the generation of return codes if it agrees with the result of the verification. This way, a collusion of the voting device, the bulletin board manager and all the code generators is needed for defeating the cast-as-intended verifiability property.

4.8.3 Support for multiple voting

Although the description of the protocol has been focused on a single-voting scheme in which a confirmation phase is added after vote verification, the improvements on the return code generation related to previous schemes such as [60, 100, 101] can still be used in multiple voting scenarios. In such case, the restriction of one vote per voter which is executed in the ProcessBallot algorithm is removed and the Tally algorithm will select the last ballot cast by each voter. Moreover, the confirmation phase is not needed since, in case the voters don't agree with the received return codes, they can choose to submit a new ballot through a different voting device.

Of course, it is important to take into account that returning the return codes through the same voting device when multiple ballots are cast is a threat to the cast-as-intended verifiability property, when the same verification card is used: the voting device may learn the return codes corresponding to a vote according to the voter's selections and then cast a second one containing different content. The voter would not detect this attack because the first set of return codes would be shown on the screen.

A mitigation to this attack, which was used in the Norwegian project, was to return the return codes through a second channel, in that case the SMS channel, which was assumed to be independent to the voting device.

Another alternative is that voters have access to several verification cards, which they can use to cast multiple votes. Since the sets of return codes are different in each case, the voting device cannot use previously received return codes to cheat the voter. In case of using this approach, some system has to be put in place for ensuring that only one vote per voter is counted. For example, the voter may use signing keys to digitally sign her ballot, which are independent of the verification cards. Therefore all the ballots cast by the same voter can be identified to only take one into account at the counting phase. The verification cards may also be pre-assigned to the voter for the same purpose (identifying several ballots from the same voter). Another alternative is to have a dynamic system which allows multiple registration proccesses, in which when the voter requests to re-register, all her previous ballots are revoked.

4.8.4 Assignation of verification cards

Finally, although in the description of the protocol we have always linked a verification card to a voter for a more comprehensive explanation, it has to be clarified that the verification cards may not be related to a voter when they are generated.

Instead, they are related to a serial number to which the public registration information ($\mathtt{id}, P_{\mathtt{v}}^{\mathtt{id}}, P_{\mathtt{a}}^{\mathtt{id}}, P_{\mathtt{c}}^{\mathtt{id}}$) is linked. This serial number would be the id in the protocol. This can be used for improving the logistics of the scheme: instead of printing and sending the verification cards to the voters in advance, spare sets can be distributed to different locations so that voters can request one of them when they want to vote.

As said before, the authentication mechanism can be in charge of ensuring that, even if she has access to multiple verification cards, the voter is authorized to cast only one vote. In case the authorization to vote is given by the secret information provided in the verification card (the PIN, or the voter's private key $sk_{\mathtt{id}}$), some mechanism has to be put in place in order to ensure that the voter is only provided with one verification card (for example, a distributed electoral roll in which the voter is marked).

Chapter 5

Challenge-*and*-cast

5.1 Introduction

As we have seen in Section 2.2, systems based on the *challenge-or-cast* mechanism do not audit the same ballot that is cast. Instead, they play with the fact that the voting device does not know whether the voter is going to audit the ballot to which it has commited or not. In case such systems have allowed to verify the same ballot to be cast, they would fail on fulfilling other security requirements for electronic voting systems such as protection against voter coercion and vote selling, given that their verification involves providing the randomness of the encrypted vote.

The proposal presented here is a modification of such cast-as-intended verification systems in which the same ballot to be cast is audited. We think that this method provides an improvement with respect to the soundness of the verification, as well as representing a more straightforward process for average voters, who will better understand the process. Still, measures are applied in order to ensure that this verification does not provide the voter with a receipt that can be used to sell her vote. We call this variant *challenge-and-cast*.

5.2 Overview

The solution is the following: the voting device encrypts the vote and shows the resulting ballot to the voter, together with a zero-knowledge proof of knowledge (ZKPK) of the encryption randomness instead of revealing the plain value, as in the challenge-or-cast mechanisms. After the voter agrees on the proof, the ballot is cast and published on the bulletin board, so that the voter can check that her ballot has been correctly received at the voting platform. The voter agreement of the proof is represented with an authentication of the ballot, and only authenticated ballots are accepted in the system (posted on the bulletin board).

The cast-as-intended verification is still sound compared to prior systems, thanks to the properties of the proofs of knowledge: the verification of the proof will succeed only in case the voter's device is honest (i.e., the device is encrypting the voting options selected by the voter). In case of a dishonest device, the probability of the

proof being successfully verified (and thus, the voter being cheated without notice) is negligible.

The scheme provides protection in front of vote selling/voter coercion scenarios thanks to the fact that it generates a ZKPK instead of providing the value itself. With the proof itself, the voter can be easily coerced or she can sell her vote: the coercer or vote buyer can check that the proof verifies for specific content. However, we take advantage of the fact that ZKPKs can be simulated to give a chance to the voter to cheat the coercers/vote buyers: In our scheme, the voter is allowed to generate *fake* proofs that will look like good proofs to anyone else.

5.3 Related work

Early works have intended to provide the voter with information of the vote content without revealing this information to third parties: in [22] Benaloh and Tuinstra describe the first receipt-free voting system. An election authority prepares and proves the correctness of a set of ciphertexts to the voter, who is the only one who knows their correspondence with the original voting options. The voter is able to fake the proof information, so that a third party can be convinced of an arbitrary correspondence between ciphertexts and voting options, however she knows which ciphertext she has to cast to vote for a specific option.

While this idea is very similar to ours regarding the cast-as-intended verification, the scheme presented by Benaloh and Tuinstra needs a high degree of communication between the voter, the election authorities and a public bulletin board: for choosing between 0 and 1, the voter receives up to N bits, the election authorities have to produce and publish N pairs of ciphertexts, and then make the same amount of decryptions, where N is presumably large (the probability of the election authorities producing corrupted ciphertexts without being detected is $\frac{1}{2^N}$).

In a similar proposal by Sako and Kilian [110], the election authorities provide the voter with a large set of encryptions of 0 and 1, and then show the voter in private their correspondence with cleartexts. Chameleon commitments are used to commit to the ciphertexts generated, in such a way that the voter can later open them to arbitrary values. Again, the voter knows the content of the ciphertexts from which she picks the vote to be cast, and can provide fake information to a possible coercer.

As we will see in the following sections, our proposal only needs one encryption and one ZKPK to be generated and transmitted to the voter for schemes where the voter has to choose between 2 values. In opposition to the two previous proposals, the number of encryptions to be generated and provided to the voter is not proportional to the number of available voting options, but proportional to the number of options she can select for mixnet-based schemes.

Other systems such as [39] use trapdoor commitments in order to provide receipt-freeness in blind signature voting schemes, however they struggle in the way of providing the voter with the trapdoor key. Although we also use trapdoor commitments in our scheme, we have naturally associated the trapdoor key to a voting credential needed to cast a ballot, in order to improve the usability and understandability of the scheme.

We would like to remark that we do not aim to solve the problem of receipt-freeness with this proposal. Instead, our motivation is to provide a method for cast-as-intended verification which does not involve providing a receipt to the voter (or that at least allows her to fake it for a possible coercer). For this purpose, we have defined in Section 3.3.6 the concept of coercion-resistant cast-as-intended. We show in Section 5.5.1 that this scheme fulfills this property.

At the same time, we want this cast-as-intended verification system to be compatible with some of the features that can be provided with Helios or its variants, for example the use of digital signatures for casting the votes (no blind signature schemes), which may be required in some binding elections, and compatibility with both mixnet-type and homomorphic-tally based voting systems.

5.4 Proof simulation

The scheme uses Designated Verifier Proofs [74], which allow a proof verifier to be convinced of a statement, while she is able to simulate proofs for different statements to other verifiers.

In their paper, Jakobsson, Sako and Impagliazzo construct designated verifier proofs which can be both interactive and non-interactive. The motivation of their proposal is to provide a mechanism by which a prover can choose who can be convinced by a proof - the designated verifier - . In our scheme, the prover is the voting device, who proves knowledge of the encryption randomness, and the designated verifier is the voter. Other verifiers such as possible coercers or vote buyers cannot be convinced by the proof.

The authors give a very intuitive description of how the proof works in the following paragraph: *Instead of proving σ, Alice proves the statement "Either σ is true, or I am Bob". Bob trusts σ is true upon seeing the proof generated by Alice. However, if Bob provides the same proof to Cindy, Cindy will have no reason to believe that σ is true, since Bob is himself capable of proving to be Bob.*

Trapdoor commitments, known also as chameleon commitments [27], are used for the proof generation. The trapdoor information is only available to the designated verifier of the proof, who can use it to generate simulated proofs for other verifiers. Chameleon hashes [76] are used in non-interactive settings, such as in non-interactive zero-knowledge proofs of knowledge (NIZKPKs), rather than chameleon commitments. As the authors in [76] explain, the main difference is their intended

use. While commitments are intended to be generated in a first step, and be opened later, chameleon hashes are used for computing a one-way function of the value, with no later opening.

Chameleon Commitments A chameleon commitment is a trapdoor commitment. Without knowledge of the trapdoor, the commitment is binding and therefore can only be opened to the original commited value. However, possession of the trapdoor allows to overcome this binding property and open the commitment to any arbitrary value.

A chameleon commitment scheme is composed by four p.p.t. algorithms: $\mathsf{Gen}_{\mathsf{ch}}$, Commit_{ch}, Open_{ch}, Sim_{ch}.

> $\mathsf{Gen}_{\mathsf{ch}}$ takes as input a security parameter 1^k, outputs an evaluation key $\mathrm{ek}_{\mathrm{ch}}$ and a trapdoor key $\mathrm{tk}_{\mathrm{ch}}$, and defines a message space $\mathcal{M}_{ch}$, a randomness space $\mathcal{R}_{ch}$ and a commitment space $\mathcal{Y}_{ch}$.
>
> Commit_{ch} takes as input an evaluation key $\mathrm{ek}_{\mathrm{ch}}$, a message $m \in \mathcal{M}_{ch}$ and a random value $\mathrm{r}_{\mathrm{ch}} \in \mathcal{R}_{ch}$ and outputs a commitment $\mathrm{c}_{\mathrm{ch}} \in \mathcal{Y}_{ch}$.
>
> Open_{ch} receives a commitment $\mathrm{c}_{\mathrm{ch}} \in \mathcal{Y}_{ch}$ and outputs a message $m \in \mathcal{M}_{ch}$ and a random value $\mathrm{r}_{\mathrm{ch}} \in \mathcal{R}_{ch}$.
>
> Sim_{ch} takes as input the trapdoor $\mathrm{tk}_{\mathrm{ch}}$, two messages $m, m' \in \mathcal{M}_{ch}$ and a random $\mathrm{r}_{\mathrm{ch}} \in \mathcal{R}_{ch}$, and returns a value $\mathrm{r}_{\mathrm{ch}}' \in \mathcal{R}_{ch}$ such that $\mathsf{Commit}_{ch}(\mathrm{ek}_{\mathrm{ch}}, m, \mathrm{r}_{\mathrm{ch}}) = \mathsf{Commit}_{ch}(\mathrm{ek}_{\mathrm{ch}}, m', \mathrm{r}_{\mathrm{ch}}')$.

Chameleon commitments have the following properties:

COLLISION RESISTANCE. Provides that, given only the evaluation key $\mathrm{ek}_{\mathrm{ch}}$, the probability of finding $(m, \mathrm{r}_{\mathrm{ch}}) \neq (m', \mathrm{r}_{\mathrm{ch}}')$ such that $\mathsf{Commit}_{ch}(\mathrm{ek}_{\mathrm{ch}}, m, \mathrm{r}_{\mathrm{ch}}) = \mathsf{Commit}_{ch}(\mathrm{ek}_{\mathrm{ch}}, m', \mathrm{r}_{\mathrm{ch}}')$ is negligible in polynomial time.

TRAPDOOR COLLISION. Provides that there is an efficient algorithm Sim_{ch} which finds two openings for the same commitment value, using the trapdoor key $\mathrm{tk}_{\mathrm{ch}}$.

UNIFORMITY. For any message $m \in \mathcal{M}_{ch}$, and any r_{ch} uniformly distributed in $\mathcal{R}_{ch}$, the commitment c_{ch} is uniformly distributed in $\mathcal{Y}_{ch}$. Therefore the probability of an adversary of distinguishing between the commitment to m and m', both in $\mathcal{M}_{ch}$ is negligible in polynomial time.

Chameleon Hashes A chameleon hash function is a trapdoor collision-resistant hash function. Without knowledge of the trapdoor, the chameleon hash behaves as an ordinary collision-resistant hash function. However, using the trapdoor, collisions can be found efficiently.

The three p.p.t. algorithms $\mathsf{Gen}_{\mathsf{ch}}$, $\mathcal{H}_{ch}$ and $\mathcal{H}_{ch}^{-1}$, which define a chameleon hash scheme, can be directly identified with those defined for the chameleon commitments

above ($\mathsf{Gen}_{\mathsf{ch}}$ keeps the same name, while $\mathcal{H}_{ch}$ behaves as Commit_{ch} and $\mathcal{H}_{ch}^{-1}$ behaves as Sim_{ch}). The exception is the opening algorithm Open_{ch}, which does not have a correspondence in chameleon hash schemes.

The three properties of *collision resistance, trapdoor collision, uniformity* defined for the chameleon commitments can be also attributed to the chameleon hashes.

5.4.1 A simulatable NIZK proof using chameleon hashes

Although examples of simulatable NIZKPK proofs are given by the authors in [74], here we provide a formal description of the algorithms that will be used in further sections, in order to prove their properties and those of the scheme where they are used.

Recall that in a σ-protocol, in order to prove that a statement x belongs to $\mathcal{L}_{\mathcal{R}}$, a prover P and a verifier V engage in an interactive protocol where first, P sends a commitment message a to V; then V replies with a random challenge e; finally, P sends an answer z to V. Interactive zero-knowledge protocols such as σ proofs can be turned into non-interactive using the Fiat-Shamir [55] transformation, where a hash function is used to compute the random challenge e.

The transformation into a (trapdoor) simulatable NIZKPK works by substituting the challenge e with the result of a chameleon hash: P chooses a random value r_{ch} and evaluates the chameleon hash function $\mathcal{H}_{ch}$ on the message $m = H(x, a)$ using the randomness r_{ch}, where H is a regular collision-resistant hash function. The challenge of the σ-protocol is then defined as $e = \mathcal{H}_{ch}(H(x, a); \mathrm{r}_{\mathrm{ch}})$. In addition, P also sends the randomness r_{ch} which it used in the computation of the chameleon hash.

This non-interactive protocol allows the simulation of valid proofs by means of the trapdoor key of the chameleon hash scheme: indeed, given a trapdoor $\mathrm{tk}_{\mathrm{ch}}$ for the chameleon hash, the simulator can compute the triplet (a^*, e^*, z^*) as the simulator of the σ-protocol would do. Then, by using the trapdoor of the chameleon hash, the simulator will be able to find a random value ${\mathrm{r}_{\mathrm{ch}}}^*$ such that $e^* = \mathcal{H}_{ch}(H(x^*, a^*); {\mathrm{r}_{\mathrm{ch}}}^*)$. The uniformity property of the chameleon hash scheme guarantees that simulated proofs have the same distribution as honest proofs.

Concretely, the trapdoor-simulatable NIZKPK scheme to be used in our protocol uses a σ-protocol, a chameleon hash scheme ($\mathsf{Gen}_{\mathsf{ch}}$, $\mathcal{H}_{ch}$, $\mathcal{H}_{ch}^{-1}$) and two hash functions $H_1 : \{0,1\}^* \to \mathcal{M}_{ch}$ and $H_2 : \{0,1\}^* \to \mathcal{CH}$ (the challenge space). Then, the NIZK proof is given by the following algorithms:

> GenCRS: receives as input a security parameter, it runs $\mathsf{Gen}_{\mathsf{ch}}$ and outputs $\mathsf{crs} = \mathrm{ek}_{\mathrm{ch}}$ and $\mathsf{tk} = \mathrm{tk}_{\mathrm{ch}}$.
>
> $\mathsf{NIZKProve}$: receives as input the common reference string crs, a statement x and a witness w. Then it follows the next steps:

1. Run the first phase of the prover P of the σ-protocol, which outputs a commitment a.
2. Sample a random $\mathrm{r_{ch}} \in \mathcal{R}_{ch}$ and compute $e = H_2(\mathcal{H}_{ch}(H_1(x,a), \mathrm{r_{ch}}))$.
3. Run the second phase of the prover P of the σ-protocol, obtaining an answer z.
4. Define the proof $\pi = (a, e, \mathrm{r_{ch}}, z)$.

NIZKVerify: receives as input a proof π and a statement x. Then it returns 1 if $e = H_2(\mathcal{H}_{ch}(H_1(x,a),$
$\mathrm{r_{ch}})$ and if the verification of the σ-protocol passes for the values (a, e, z), 0 otherwise.

NIZKSimulate: on input a statement x and a trapdoor tk, the simulator runs the following steps:

1. Run the simulator $\mathcal{S}$ of the σ-protocol to obtain a triplet (a^*, e^*, z^*).
2. Use the trapdoor $\mathrm{tk_{ch}}$ to obtain a value $\mathrm{r_{ch}}^*$ s.t. $e^* = H_2(\mathcal{H}_{ch}(H_1(x,a^*), \mathrm{r_{ch}}^*))$
3. Output a simulated proof $\pi^* = (a^*, e^*, \mathrm{r_{ch}}^*, z^*)$

5.4.2 Simulatable NIZKPK scheme properties

In this section we prove that the trapdoor-simulatable NIZKPK scheme presented above is complete, sound and zero-knowledge.

Theorem 5.1. *The NIZKPK scheme is complete if the underlying σ-protocol is complete and the chameleon hash function fulfills the property of correctness.*

The completeness of the protocol follows easily from inspection.

Before proving that the protocol is sound, we will prove the following useful lemma:

Lemma 5.1. Let H be modeled as a random oracle. Let $\mathcal{A}$ be an adversary which has access to such random oracle and can produce a proof $\pi = (a, e, \mathrm{r_{ch}}, z)$ for a statement x of its choice. Then, there exists a p.p.t. simulator $\mathcal{S}$ which accesses $\mathcal{A}$ and outputs two proofs $\pi_1 = (a, e_1, \mathrm{r_{ch1}}, z_1)$ and $\pi_2 = (a, e_2, \mathrm{r_{ch2}}, z_2)$ such that $e_1 = \mathcal{H}_{ch}(\tilde{e}_1, \mathrm{r_{ch1}})$, $e_2 = \mathcal{H}_{ch}(\tilde{e}_2, \mathrm{r_{ch2}})$ and $\tilde{e}_1 \neq \tilde{e}_2$.

Proof. The simulator $\mathcal{S}$ uses $\mathcal{A}$ as a black-box and also acts as its random oracle. In particular, whenever $\mathcal{A}$ makes a query to the random oracle, $\mathcal{S}$ saves the state of $\mathcal{A}$ and answers the random oracle query by returning a random value. Then, $\mathcal{S}$ runs many copies of $\mathcal{A}$ (from the saved state) and returns different answers to the random oracle.

For each of the copies of $\mathcal{A}$, it then answers to subsequent oracle queries with random values, whereas for the *main* execution of $\mathcal{A}$ it repeats the explained process for every oracle query.

The adversary $\mathcal{A}$ must use one of its random oracle queries for producing a proof $\pi = (a, e, \mathrm{r_{ch}}, z)$, where $e = \mathcal{H}_{ch}(\tilde{e}, \mathrm{r_{ch}})$ and $\tilde{e}$ is the answer of the random oracle on input (x, a). As explained in [65], by making a polynomial number of copies of $\mathcal{A}$ for each oracle query, there will be at least one which will use the same oracle query for producing the proof $\pi' = (a, e', \mathrm{r_{ch}}', z')$ where $e' = \mathcal{H}_{ch}(\tilde{e}', \mathrm{r_{ch}}')$ and $\tilde{e}' \neq \tilde{e}$. □

Proposition 5.1. The scheme defined above is sound in the random oracle model if the underlying σ-protocol has the special soundness property and the chameleon hash is collision-resistant.

Proof. Assume that the hash H is modeled as a random oracle and assume that there exists an adversary $\mathcal{A}$ which is able to produce proofs for statements of its choice (x, π). Then we will show that there exists a p.p.t. adversary $\mathcal{B}$ and a p.p.t. simulator $\mathcal{S}$ such that either $\mathcal{B}$ breaks the collision-resistance of the chameleon hash or $\mathcal{S}$ extracts a witness w such that $(x, w) \in \mathcal{R}$.

By using the Lemma 5.1, we can extract two proofs $\pi_1 = (a, e_1, \mathrm{r_{ch1}}, z_1)$ and $\pi_2 = (a, e_2, \mathrm{r_{ch2}}, z_2)$ such that $e_1 = \mathcal{H}_{ch}(\tilde{e}_1, \mathrm{r_{ch1}})$, $e_2 = \mathcal{H}_{ch}(\tilde{e}_2, \mathrm{r_{ch2}})$ and $\tilde{e}_1 \neq \tilde{e}_2$. Now we distinguish between two cases: (i) $e_1 \neq e_2$ and (ii) $e_1 = e_2$. In the first case, we can use the special soundness property of the σ-protocol to extract a witness w for the statement x belonging to $\mathcal{L}_{\mathcal{R}}$. In the second case, we can use e_1 and e_2 as a collision for the chameleon hash, as we have that $\tilde{e}_1 \neq \tilde{e}_2$ and $\mathcal{H}_{ch}(\tilde{e}_1, ch_1) = \mathcal{H}_{ch}(\tilde{e}_2, ch_2)$ for some ch_1, ch_2. □

Proposition 5.2. The scheme defined above is zero-knowledge (in the common reference string model) if the σ-protocol is honest-verifier zero-knowledge.

Proof. We have already given the simulator, we just need to show that the probability distributions of the proofs generated by the simulator and of the proofs generated by a honest prover are computationally indistinguishable.

To see that, consider a honestly generated proof $\pi_1 = (a_1, e_1, \mathrm{r_{ch1}}, z_1)$ and a simulated proof $\pi_2 = (a_2, e_2, \mathrm{r_{ch2}}, z_2)$. From the zero-knowledge property of the σ-protocol, we have that the sub-tuples (a_1, e_1, z_1) and (a_2, e_2, z_2) are indistinguishable, and by using the uniformity property of the chameleon hash we get that π_1 and π_2 are also indistinguishable. □

5.5 Core Protocol using Mixnets

In this section we define the core protocol with a mixnet-based approach, using as basis the syntax for an electronic protocol that we defined in Chapter 3. We add the *audit device* participant with respect to the definition in Section 3.2. The audit device is used by the voter to verify cryptographic evidences generated by the voting device.

As defined in 3.2, the list of voting options $V = \{v_1, \ldots, v_k\}$ in the election, and the counting function $\rho : (V \cup \{\perp\})^* \to R$ have been previously defined by the electoral authorities. The voting protocol uses an encryption scheme with algorithms $(\mathsf{Gen_e}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EncVerify})$, a signature scheme $(\mathsf{Gen_s}, \mathsf{Sign}, \mathsf{SignVerify})$ and a mixnet with algorithms Mix and $\mathsf{MixVerify}$. It additionally uses two NIZKPK schemes, one which is trapdoor-simulatable, denoted by the four algorithms $(\mathsf{GenCRS}, \mathsf{NIZKProve}, \mathsf{NIZKVerify}, \mathsf{NIZKSimulate})$, and one for proving decryption correctness, which is denoted by the algorithms $(\mathsf{ProveDec}, \mathsf{VerifyDec}, \mathsf{SimDec})$. It consists on the following algorithms:

$\mathsf{Setup}(1^\lambda)$ runs $\mathsf{Gen_e}$ from the encryption scheme to generate the key pair $(pk_\mathsf{e}, sk_\mathsf{e})$. Then it sets the election public key to $pk = pk_\mathsf{e}$ and the election private key to $sk = (sk_\mathsf{e}, pk_\mathsf{e})$. The global confirmation and audit key pairs $(pk_\mathsf{a}, sk_\mathsf{a})$, (pk_c, sk_c) are set to $\perp$.

$\mathsf{Register}(1^\lambda, \mathtt{id}, sk_\mathsf{a}, sk_c)$ runs GenCRS from the NIZKPK scheme and $\mathsf{Gen_s}$ from the signature scheme, and sets the voter audit key pair $(P_\mathsf{a}^{\mathtt{id}}, S_\mathsf{a}^{\mathtt{id}}) = \perp$, the voter confirmation key pair $P_\mathsf{c}^{\mathtt{id}} = pk_\mathsf{s}$ and $S_\mathsf{c}^{\mathtt{id}} = (sk_\mathsf{s}, \mathsf{tk})$, and the voter voting key pair $P_\mathsf{v}^{\mathtt{id}} = \mathsf{crs}$, $S_\mathsf{v}^{\mathtt{id}} = \perp$.

$\mathsf{CreateVote}(\mathtt{id}, \{v_{j_1}, \ldots, v_{j_t}\}, P_\mathsf{v}^{\mathtt{id}}, S_\mathsf{v}^{\mathtt{id}})$ runs Enc from the encryption scheme with inputs pk and each voting option $\{v_{j_l}\}_{l=1}^t$ and obtains the set of ciphertexts $\{c_l\}_{l=1}^t$. Then it parses $P_\mathsf{v}^{\mathtt{id}}$ as crs and runs $\mathsf{NIZKProve}$ from the NIZKPK scheme for each ciphertext c_l, using as input crs, the statement (c_l/v_{j_l}) and the encryption randomness r_l. The ballot b is set to be the set of ciphertexts $\{c_l\}_{l=1}^t$, and the encryption data $\tilde{r}$ is $\{\pi_l\}_{l=1}^t$.

$\mathsf{AuditVote}(b, \tilde{r}, \{v_{j_1}, \ldots, v_{j_t}, P_\mathsf{v}^{\mathtt{id}})$ parses b as $\{c_l\}_{l=1}^t$, $\tilde{r}$ as $\{\pi_l\}_{l=1}^t$ and $P_\mathsf{v}^{\mathtt{id}}$ as crs. Then it runs $\mathsf{NIZKVerify}$ from the NIZKPK scheme for each π_l with the statement (c_l/v_{j_l}) and the common reference string crs. It outputs 1 if all the proof verifications return 1, 0 otherwise.

$\mathsf{Confirm}(\mathtt{id}, b, S_\mathsf{v}^{\mathtt{id}}, S_\mathsf{c}^{\mathtt{id}}, P_\mathsf{c}^{\mathtt{id}}, sk_\mathsf{a})$ parses $S_\mathsf{c}^{\mathtt{id}}$ as $(sk_\mathsf{s}, \mathsf{tk})$ and runs Sign with inputs the voter's signing private key sk_s and the ballot b to be signed, together with the voter identity $\mathtt{id}$. The resulting ballot signature ψ is set to be the ballot confirmation C_b. Then it parses b as $\{c_l\}_{l=1}^t$ and runs $\mathsf{NIZKSimulate}$ from the NIZKPK scheme for each statement $(c_l/v_{j_l'})$, where the values $\{v_{j_l'}\}_{l=1}^t$ are additionally provided or randomly selected from V. The simulated proofs $\{\pi_l^*\}$ are set to be the auxiliary data σ'.

$\mathsf{ProcessBallot}(\mathsf{BB}, \mathtt{id}, b)$ performs some validations: it checks that there is not already an entry in the bulletin board for the same $\mathtt{id}$ and that this $\mathtt{id}$ is present in the list ID. Then it checks that there is not another ballot b' in BB for which $b' = b$. Finally it parses b as $\{c_l\}_{l=1}^t$ and for each ciphertext c_l it runs $\mathsf{EncVerify}$. If any of these validations returns 0, the process stops and outputs 0. Otherwise it outputs 1.

$\mathsf{ProcessConfirm}(\mathsf{BB}, \mathtt{id}, C_b)$ locates the ballot b in the bulletin board corresponding to the received identifier id and runs SignVerify, using the corresponding voter public key $P_{\mathtt{c}}^{\mathtt{id}}$ (which can be parsed as $pk_{\mathtt{s}}$). If the validation returns 1 it outputs 1. Otherwise, it outputs 0.

$\mathsf{VerifyVote}(\mathsf{BB}, \mathtt{id}, b)$ checks that there is an entry in the bulletin board corresponding to the identity id. In the affirmative case, it compares the ballot b' in such entry with b, checking that all the fields are equal. If the validation is successful, it outputs 1. Otherwise it outputs 0.

$\mathsf{Tally}(\mathsf{BB}, sk)$ in the first place runs the ProcessBallot and ProcessConfirm algorithms, and discards the ballots for which any of the verifications fail. Finally it extracts the ciphertexts c from the ballots which have passed the verifications and run Mix with them as input. The resulting list of mixed ciphertexts $\{C_m\}$ is decrypted: for each ciphertext $c_z \in C_m$, $\mathsf{Dec}(c_z, sk)$ is run to obtain v_z, which is tested to belong to V. Finally the ProveDec algorithm from the NIZKPK scheme is run with inputs each statement (c_z, v_z) and the private key sk as the witness. The outputs are the list of decrypted voting options $r = \{v_z\}$ and the proofs of correct mixing and decryption, $\Pi = (\pi_{mix}, \{C_m\}, \pi_{dec})$.

$\mathsf{VerifyTally}(\mathsf{BB}, r, \Pi)$ runs the ProcessBallot and ProcessConfirm algorithms over the ballots in BB. It extracts the ciphertexts c_l from the ballots which have passed the previous validations and composes the list $\{C\}$. Then it parses Π as $(\pi_{mix}, \{C_m\}, \pi_{dec})$ and verifies that the mixing was correct by running $\mathsf{MixVerify}(C, C_m, \pi_{mix})$. Finally it checks that the decryption of each ciphertext was correct by running VerifyDec from the NIZKPK scheme, using as input the statement (c_z, v_z), for all the ciphertexts $c_z \in \{C_m\}$ and all the plaintexts $v_z \in r$, and the proof $\pi_l \in \pi_{dec}$. The output is the result of these validations.

The voting protocol algorithms are organised in the following phases:

Configuration phase: in this phase, the election authorities set up the public parameters of the election such as the list of voting options $\{v_i\} \in V$ and the result function ρ. They also run the Setup algorithm and publish the resulting election public key pk and the empty credential list ID in the bulletin board. The private key sk is kept in secret by the electoral authorities.

Registration phase: in this phase the registrars register the voters to vote in the election. For each voter with identity id, the registrars run Register and update the credential list ID in the bulletin board with id. They provide the voter voting key $P_{\mathtt{v}}^{\mathtt{id}}$ and the voter confirmation key pair $(P_{\mathtt{c}}^{\mathtt{id}}, S_{\mathtt{c}}^{\mathtt{id}})$ to the voter, and publish the tuple $(\mathtt{id}, P_{\mathtt{v}}^{\mathtt{id}}, P_{\mathtt{c}}^{\mathtt{id}})$ on the bulletin board.

Voting phase: in this phase the voter chooses the voting options $\{v_{j_1}, \ldots, v_{j_t}\} \in V$ and interacts in the following way with the voting device, in order to cast a vote:

1. The voter provides her identity id and the chosen voting options $\{v_{j_1}, \ldots, v_{j_t}\}$ to the voting device, which gathers the corresponding public key $P_{\mathtt{v}}^{\mathtt{id}}$ from the

bulletin board and runs the CreateVote algorithm. The outputs b and $\tilde{r}$ are provided to the voter.

2. The voter uses an audit device to run AuditVote using b and $\tilde{r}$ provided by the voting device, as well as her selections $\{v_{j_1}, \ldots, v_{j_t}\}$. The voter may enter her voting public key $P_{\mathtt{v}}^{\mathtt{id}}$ herself, or her identity id so that the audit device picks the corresponding public key $P_{\mathtt{v}}^{\mathtt{id}}$ from the bulletin board. A positive result means that b is encrypting the voter's selections $\{v_{j_1}, \ldots, v_{j_t}\}$ and the voter can continue the process. Otherwise, the voter is instructed to abort the process and choose another voting device to cast her vote, since the one she is using is corrupted and did not encrypt what she selected.

3. As a sign of approval of the generated ballot, the voter provides her confirmation private key $S_{\mathtt{c}}^{\mathtt{id}}$ to the voting device, which proceeds to run Confirm. The resulting ballot confirmation C_b is sent, together with the ballot and the voter identity, to the bulletin board manager. The simulated encryption data $\tilde{r}'$ is provided to the voter.

4. Upon reception of $(\mathtt{id}, b, C_b)$, the bulletin board manager runs the ProcessBallot and ProcessConfirm algorithms. In case both results are 1, the confirmed ballot is posted on the bulletin board. Otherwise, the voting device receives an error message. From that point, the voter can run VerifyVote to check that her vote has been posted in the bulletin board.

The voter can provide the ballot b and the simulated encryption data $\tilde{r}'$ to a coercer, who might want to check that a ballot for the requested voting options $\{v_{j'_1}, \ldots, v_{j'_t}\}$ is present in the bulletin board by running the AuditVote and VerifyVote algorithms.

Counting phase: in this phase, the election authorities provide the election private key sk and run the Tally algorithm on the contents of the bulletin board. The obtained result r and the proof Π are posted in the bulletin board. The auditors then run the VerifyTally algorithm. In case the verification is satisfactory, the election result is considered to be correct. Otherwise, an investigation is opened in order to detect any manipulation that could lead to a corrupted result.

Note that we have presented this scheme with a small modification of the voting process compared to the description in Section 3.2, since the ballot is sent together with the ballot confirmation to the bulletin board manager. This has been done for convenience, since the original steps in Section 3.2 where CreateBallotProof and AuditBallotProof were executed, between the submission of the ballot and its confirmation, are not executed in this protocol.

The result of this different ordering is that a ballot may not be accepted by the bulletin board manager after the voter has confirmed. The voter may only accept if the AuditVote algorithm returns 1, and this algorithm verifies that the ballot is correct up to some extent. However, some validations which depend on the bulletin board contents may be out of its scope (for example, if the audit device is off-line).

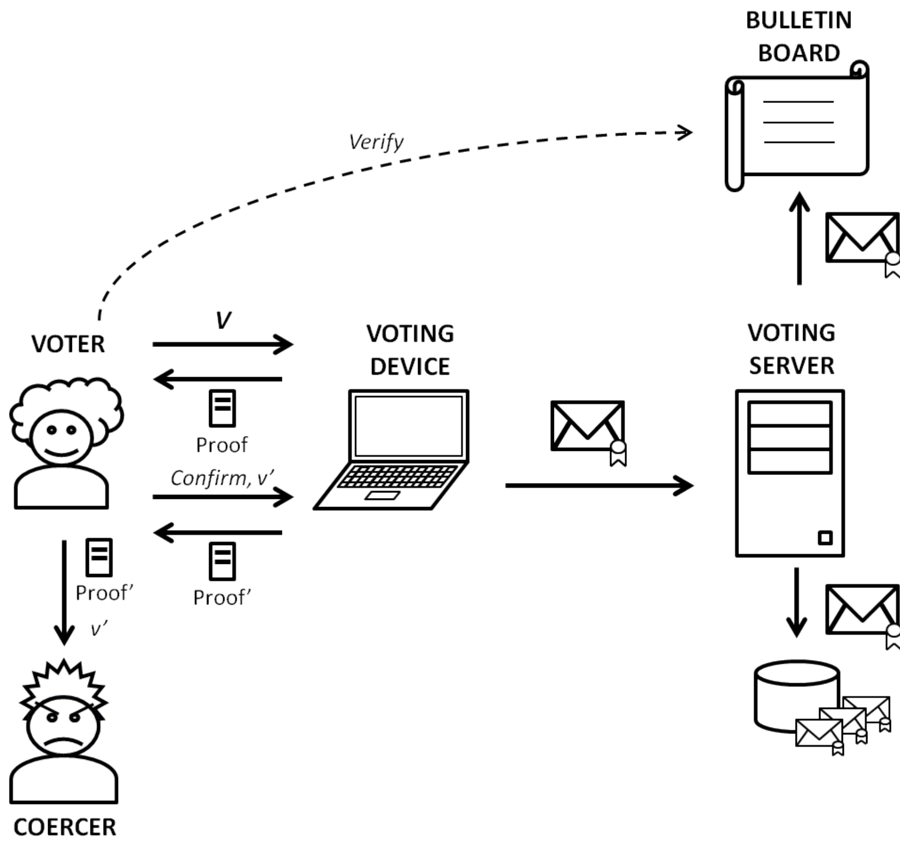

Figure 5.1: Challenge-and-cast: Voting process overview

We consider that even keeping the original ordering the same could happen with the ballot confirmation, and in both cases the result is that the voter has given her confirmation but her vote is not accepted. Still, the voter would detect this when running the algorithm VerifyVote. A discussion on how the voter may re-vote in case such problems occur can be found in Section 5.9.

The scheme can be seen as a modification of *Helios with Credentials* (Helios-C) [40], the Helios variant in which there is a voter registration process where voters receive a private key to digitally sign and cast their ballots. The main difference is in the interactive protocol executed between the voter and the voting device during the voting phase. For ilustrative purposes, we provide the description of the voting phase Helios-C here:

Helios-C Voting phase:

1. The voter provides her chosen voting options $\{v_{j_1}, \dots, v_{j_t}\}$ to the voting device, which runs the CreateVote algorithm with $P_{\mathsf{v}}^{\mathtt{id}} = 0$. The generated ballot b is provided to the voter *and the encryption data is not shown.*
2. The voter then can decide either to audit the ballot or to proceed to cast it. In case she decides to audit the ballot, the voting device shows her the encryption data $\tilde{r}$ (which is the randomness used for encryption).
3. The voter uses an audit device to run AuditVote, using b and $\tilde{r}$ provided by the voting device, as well as her selections $\{v_{j_1}, \dots, v_{j_t}\}$. A positive result means that b encrypts the voter's selection and that the voter can continue the process. Otherwise, the voter aborts and starts the process again with another voting device, since the one she was using did not encrypt what she selected.
4. The voting device runs CreateVote again in order to generate a ballot containing a fresh encryption of the voter's selection.

   Again, the voter can choose either to audit the ballot or to cast it. In case she decides to audit the ballot, the previous two steps are executed again. Otherwise, the process continues.
5. The voter provides her confirmation private key $S_{\mathsf{c}}^{\mathtt{id}}$ to the voting device, which proceeds to run Confirm. The resulting ballot and confirmation are sent to the bulletin board manager, where ProcessBallot and ProcessConfirm are executed.
6. From that point, the voter can run VerifyVote to check that her vote has been posted in the bulletin board.

5.5.1 Security of the Protocol

In this section we show that the protocol presented in Section 5.5 satisfies the properties of ballot privacy, strong consistency, strong correctness, cast-as-intended verifiability and coercion-resistant cast-as-intended verifiability defined in Section 3.3.

Ballot Privacy

Theorem 5.2. *Let* $(\mathsf{Gen_e}, \mathsf{Enc}, \mathsf{Dec})$ *be an NM-CPA secure encryption scheme and* $(\mathsf{GenCRS}, \mathsf{NIZKProve}, \mathsf{NIZKVerify}, \mathsf{NIZKSimulate})$ *a NIZKPK with the zero-knowledge property. Then the protocol presented in Section 5.5 satisfies the ballot privacy property.*

The ballot privacy definition provided in Section 3.3.2 is based in the indistinguishability of two experiments which depend on a bit β. We will refer to $\mathsf{Exp}^{\mathsf{priv},0}_{\mathcal{A},V}$ for the experiment when $\beta = 0$, and $\mathsf{Exp}^{\mathsf{priv},1}_{\mathcal{A},V}$ when $\beta = 1$.

The next steps will be followed for proving that the ballot privacy property is fulfilled by the protocol: first, we will prove that the original experiment $\mathsf{Exp}^{\mathsf{priv},\beta}_{\mathcal{A},V}$ is indistinguishable by $\mathcal{A}$ from an experiment $\mathsf{Exp}^{\mathsf{priv},\beta'}_{\mathcal{A},V}$ where the tallier provides a simulated proof of the tally result (needed due to the ballot privacy definition, as indicated in [23]), through the Lemma 5.2. Then, we will provide a security reduction of $\mathsf{Exp}^{\mathsf{priv},\beta'}_{\mathcal{A},V}$ to the ballot privacy property of the $\mathsf{Enc2Vote}$ scheme with the demonstration of Lemma 5.3.

Lets consider $\mathsf{SimProof}$ be a simulator of the mixing and decryption proofs, which produces simulated proofs with the same distribution that the honest ones. Then consider the experiment $\mathsf{Exp}^{\mathsf{priv},\beta'}_{\mathcal{A},V}$ in which the challenger, when executing $\mathsf{Tally}(\mathsf{BB}_0, sk)$, provides the result r and the proof Π^* which is the output of $\mathsf{SimProof}$. The following lemma is straightforward to prove:

Lemma 5.2. The experiments $\mathsf{Exp}^{\mathsf{priv},\beta}_{\mathcal{A},V}$ and $\mathsf{Exp}^{\mathsf{priv},\beta'}_{\mathcal{A},V}$ are computationally indistinguishable for $\beta \in \{0, 1\}$.

Now we consider the $\mathsf{Enc2Vote}$ scheme defined in [25]. In their work, the authors have proven the following theorem:

Theorem 5.3. *Let* $(\mathsf{Gen_e}, \mathsf{Enc}, \mathsf{Dec})$ *be an NM-CPA secure encryption scheme, then* $\mathsf{Enc2Vote}$ *has ballot privacy.*

Next, we proceed to reduce the ballot privacy property of our scheme to the ballot privacy property of the $\mathsf{Enc2Vote}$ scheme.

Lemma 5.3. Let $\mathcal{A}$' be a p.p.t. adversary that interacts with a challenger $\mathcal{C}$, such that | Pr[$\mathsf{Exp}^{\mathsf{priv},0'}_{\mathcal{A},V} = 1$] - Pr[$\mathsf{Exp}^{\mathsf{priv},1'}_{\mathcal{A},V} = 1$] | is non-negligible. Then, there exists an adversary $\mathcal{A}$" that breaks the ballot privacy property of the $\mathsf{Enc2Vote}$ scheme.

Proof. In the reduction, we use $\mathcal{A}$" as the challenger for $\mathcal{A}$', and $\mathcal{A}$" interacts with $\mathcal{C}$ in the same way as in the experiment defined in [25]. The reduction is as follows:

In the **Setup phase**, $\mathcal{C}$ sets up two empty bulletin boards BB_0 and BB_1, runs the $\mathsf{Gen_e}$ algorithm and keeps the sk^{e2v}_{e} key for itself, while it publishes the pk^{e2v}_{e} key on the bulletin board. In turn, $\mathcal{A}$" publishes $pk_{\mathsf{e}} = pk^{e2v}_{\mathsf{e}}$ on the bulletin board visible by $\mathcal{A}$'.

In the **Registration phase**, when $\mathcal{A}$' makes the $\mathcal{O}$Register query, $\mathcal{A}$" runs the $\mathsf{Register}(1^\lambda, \mathtt{id}, \bot, \bot)$ algorithm of our protocol and provides the voter confirmation private key $S_{\mathtt{c}}^{\mathtt{id}} = (sk_{\mathtt{s}}, \mathsf{tk})$ to $\mathcal{A}$', while also keeping it internally. It also publishes the tuple $(\mathtt{id}, P_{\mathtt{v}}^{\mathtt{id}}, P_{\mathtt{a}}^{\mathtt{id}}, P_{\mathtt{c}}^{\mathtt{id}})$ on the bulletin board visible by $\mathcal{A}$'.

During the **Voting phase**, when $\mathcal{A}$' submits the $\mathcal{O}$VoteLR query, $\mathcal{A}$" submits the **Vote** query to $\mathcal{C}$, which responds by publishing a ballot b_{e2v} to the bulletin board visible by $\mathcal{A}$". Then $\mathcal{A}$" runs Sign with inputs the voter's signing key $sk_{\mathtt{s}}$ and the ballot b_{e2v} to be signed together with the voter identity $\mathtt{id}$. Finally, $\mathcal{A}$" posts the resulting signature ψ to the bulletin board visible by $\mathcal{A}$', together with the voter identity $\mathtt{id}$ and the ballot b_{e2v}. When $\mathcal{A}$' submits the $\mathcal{O}\text{Cast}(b, C_b, \mathtt{id})$ query $\mathcal{A}$" submits a **Ballot(b)** query to $\mathcal{C}$, and posts $(\mathtt{id}, b, C_b)$ in the bulletin board visible by $\mathcal{A}$'.

In the **Counting phase**, $\mathcal{C}$ posts the result of evaluating $\mathsf{Tally}(\mathsf{BB}_0, sk_{\mathtt{e}}^{e2v})$ on the bulletin board visible to $\mathcal{A}$". $\mathcal{A}$" in turn runs $\mathsf{SimProof}(\mathsf{BB}_\beta, r)$, where BB_β is the bulletin board shown by $\mathcal{C}$ to $\mathcal{A}$", and publishes (r, Π^*) on the bulletin board visible by $\mathcal{A}$'.

At the end of the experiment, $\mathcal{A}$' outputs a bit and $\mathcal{A}$" outputs the same bit. As we can see, the outputs of $\mathcal{A}$"as a result of the interaction with $\mathcal{A}$' have the same distribution as in the ballot privacy experiment in [25]. Therefore, the reduction is sound. □

Strong Consistency

We define the following algorithms for proving that our protocol provides the strong consistency property, according to the definition in Section 3.3.3:

- Let the extraction algorithm $\mathsf{Extract}((b, sk)$, defined in Section 3.3.3 parse b as the ciphertext c, run $\mathsf{EncVerify}(c, pk_{\mathtt{e}})$ and in case it returns 1, output $m = \mathsf{Dec}(c, sk_{\mathtt{e}})$. Then it tests whether $m \in V$ or not. In a positive case, it returns $v = m$, otherwise it returns $\bot$.
- Let the validation algorithm $\mathsf{ValidInd}(\mathtt{id}, b, C_b)$ parse b as the ciphertext c and C_b as the signature ψ, run $\mathsf{EncVerify}(c, pk_{\mathtt{e}})$, and run $\mathsf{SignVerify}$ to check that ψ is a valid signature of b and $\mathtt{id}$, using the voter public confirmation key $P_{\mathtt{c}}^{\mathtt{id}}$. The output of the algorithm is 1 if all the validations return 1, 0 otherwise.
- Let ρ be the counting function that provides its inputs as outputs in a shuffled order, removing any input v for which $v \notin V$.

Theorem 5.4. *Let* Tally *produce a sound proof* Π *of correct mixing and decryption, and* $\mathsf{VerifyTally}$ *output* 1. *Then the protocol defined in Section 5.5 has the property of strong consistency with respect to the above definitions of* $\mathsf{Extract}, \mathsf{ValidInd}, \rho$.

Clearly Tally and the combination of the extraction algorithm and ρ remove the same ballots, provided that the ciphertexts and the ballot confirmations are correct

(what is enforced by the definition of the $\mathsf{ValidInd}$ algorithm). Mixing and decryption do not remove/add/modify any ballot, since otherwise $\mathsf{VerifyTally}$ would fail, given the soundness of the proofs generated by Tally.

Strong Correctness

Theorem 5.5. *Let* $(\mathsf{Gen_e}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EncVerify})$ *be a randomized encryption scheme and* $(\mathsf{Gen_s}, \mathsf{Sign}, \mathsf{VerifyTally})$ *be a signature scheme with the property of correctness. Then the protocol defined in Section 5.5 has the property of strong correctness.*

In the protocol defined in Section 5.5, the only condition for which $\mathsf{ProcessBallot}$ $(\mathsf{BB}, \mathtt{id}, b)$ may output 0, given a ballot b produced by a honest registered voter that was not registered by the adversary is that a previous ballot with the same ciphertext c is already present in the bulletin board. Given that $(\mathsf{Gen_e}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EncVerify})$ is a probabilistic encryption scheme, this probability is negligible in the security parameter λ. The only condition for which $\mathsf{ProcessConfirm}$ outputs 0 is in case the signature in the ballot confirmation does not correctly verify. By the definition of correctness, a signature produced with the Sign algorithm is always accepted by $\mathsf{SignVerify}$ for the corresponding public key.

Cast-as-Intended Verifiability

Theorem 5.6. *Let* $(\mathsf{GenCRS}, \mathsf{NIZKProve}, \mathsf{NIZKVerify}, \mathsf{NIZKSimulate})$ *be a NIZKPK scheme which is sound,* $(\mathsf{Gen_s}, \mathsf{Sign}, \mathsf{SignVerify})$ *an unforgeable signature scheme and* $(\mathsf{Gen_e}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EncVerify})$ *be a randomized encryption scheme. Then the protocol presented in Section 5.5 satisfies the cast-as-intended verifiability property.*

According to the definition presented in 3.3.5, the attacker may follow different strategies to succeed. Recall that the definition included to cases in which the adversary may try to win: In **Case A** the adversary presents as output a ballot, encryption data and a voting option which are consistent (the corresponding verification algorithm outputs 1), the ballot has been successfully cast, but for which the voting option does not correspond to the encrypted content in the ballot. In **Case B** the adversary presents a valid ballot and confirmation (which means they are accepted by the bulletin board manager) without having to provide any proof of the content of the ballot that had to be accepted by the challenger.

There are two strategies for winning in **Case A**: The first strategy is that the adversary provides at the output a tuple $(\mathtt{id}, b, \tilde{r}, v_x)$ for which the $\mathcal{O}$ProofVote query returned $S_{\mathsf{c}}^{\mathtt{id}}$ and $\mathsf{Extract}(b, sk) \neq v_x$. Recall that in the $\mathcal{O}$ProofVote query the $\mathsf{AuditVote}$ algorithm parses b as c, $\tilde{r}$ as π and $P_{\mathsf{v}}^{\mathtt{id}}$ as crs, and then runs $\mathsf{NIZKVerify}(\mathsf{crs}, (c/v_x), \pi)$. Only if this verification is successful, the adversary will obtain $S_{\mathsf{c}}^{\mathtt{id}}$ necessary for computing C_b and making the $\mathcal{O}$Cast query, which is necessary in order to make $\mathsf{VerifyVote}$ succeed. We can see that an adversary who follows this strategy is an adversary against the soundness of the NIZKPK scheme.

The second strategy is that the adversary does the $\mathcal{O}$ProofVote query with inputs $(\mathtt{id}, b, \tilde{r}, v_x)$, where $\mathsf{Extract}(b, sk) = v_x$. Therefore, $\mathsf{AuditVote}$ is successful and the

adversary has the voter confirmation private key $S_{\mathsf{c}}^{\mathtt{id}}$. However, after receiving it, the adversary submits the query $\mathcal{O}\text{Cast}(\mathtt{id}, b^*, C_b^*)$ such that $\mathsf{Extract}(b', sk) \neq v_x$ and then presents the original tuple $(\mathtt{id}, b, \tilde{r}, v_x)$ at the output. Recall that $\mathsf{VerifyVote}$ checks that, for the entry $(\mathtt{id}', b', C_b')$ in BB for which $\mathtt{id}' = \mathtt{id}$, all the fields in b' are equal to all the fields in b. Given that $\mathsf{AuditVote}$ and $\mathsf{ProcessBallot}$ succeed, we know that both are two valid ciphertexts. Given a probabilistic encryption algorithm, the probability of finding two ciphertexts with the same value, which decrypt to different plaintexts, is negligible according to the security parameter λ.

Finally, the strategy the attacker may follow for winning in **Case B** is to forge the ballot confirmation C_b in order to provide a tuple $(\mathtt{id}, b, C_b)$ at the output which is successfully processed by $\mathsf{ProcessBallot}(\mathsf{BB}, \mathtt{id}, b)$ and $\mathsf{ProcessConfirm}(\mathsf{BB}, \mathtt{id}, C_b)$, without the collaboration of the challenger. Recall that $\mathsf{ProcessConfirm}$ performs the validation of the ballot's signature running $\mathsf{SignVerify}(pk_{\mathsf{s}}, b, \psi)$. Since $\mathcal{A}$ does not have the voter's private signing key sk_{s} (because it did not do the $\mathcal{O}$ProofVote or $\mathcal{O}$VoteCorrupt queries), it has to forge the signature in order to succeed. Therefore an adversary which succeeds with this strategy is an adversary against the unforgeability of the signature scheme.

Coercion-resistant cast-as-intended

Theorem 5.7. *Let* $(\mathsf{GenCRS}, \mathsf{NIZKProve}, \mathsf{NIZKVerify}, \mathsf{NIZKSimulate})$ *be a NIZKPK scheme with the zero-knowledge property, and let the protocol satisfy the cast-as.intended verifiability property. Then the protocol presented in Section 5.5 also satisfies the coercion-resistant cast-as-intended property.*

In the experiment defined in Section 3.3.6, an adversary $\mathcal{A}$ has to distinguish between two experiments denoted $\mathsf{Exp}_{\mathcal{A},V}^{\mathsf{CR\text{-}CaI},\beta}$ for $\beta = \{0, 1\}$. Specifically, the adversary can make calls to an oracle $\mathcal{O}$VoteLR to which it provides two different voting options, v_0 and v_1. Depending on the value of the bit β, $\mathcal{A}$ obtains either $(b^0, \tilde{r}^0, \tilde{r}'^0)$ or $(b^1, \tilde{r}^1, \tilde{r}'^1)$, where $\tilde{r}'^\beta$ denotes simulated encryption data computed with the algorithm $\mathsf{FakeProof}$.

We define this algorithm in the following way: $\mathsf{FakeProof}(c, \mathsf{tk}, \mathsf{crs}, v_j)$ uses the trapdoor tk to run $\mathsf{NIZKSimulate}$, from the simulatable NIZKPK scheme, for the statement $(c_1, c_2/v_j)$. Then the simulated encryption proof data σ' is the simulated proof π^*.

Given that $\mathcal{A}$ does not have the election private key sk, the only chance for distinguishing between both experiments is by distinguishing between the proofs $\tilde{r}^\beta$, $\tilde{r}'^\beta$, which is equivalent to defeating the zero-knowledge property of the NIZKPK scheme.

Formally, lets consider $\mathsf{SimVote}(\mathtt{id}, v_i, P_{\mathsf{c}}^{\mathtt{id}}, S_{\mathsf{c}}^{\mathtt{id}}, P_{\mathsf{v}}^{\mathtt{id}}, S_{\mathsf{v}}^{\mathtt{id}})$ be a modification of $\mathsf{CreateVote}$ from Section 5.5 where, instead of $\mathsf{NIZKProve}(\mathsf{crs}, (c/v_i), r)$, the algorithm $\mathsf{NIZKSimulate}(\mathsf{crs}, (c/v_i), \mathsf{tk})$ algorithm is executed to generate the encryption data $\tilde{r}$. Note that $\mathsf{SimVote}$ receives the voter's confirmation private data $S_{\mathsf{c}}^{\mathtt{id}}$

which can be parsed as $(sk_{\mathtt{s}}, \mathsf{tk})$. Then, consider the experiment $\mathsf{Exp}_{\mathcal{A},V}^{\mathsf{CR\text{-}CaI},\beta'}$ where, when the adversary submits the query $\mathcal{O}\mathbf{VoteLR}(\mathtt{id}, v_0, v_1)$ the challenger executes $\mathsf{SimVote}(\mathtt{id}, v_0, P_{\mathtt{c}}^{\mathtt{id}}, S_{\mathtt{c}}^{\mathtt{id}}, P_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{v}}^{\mathtt{id}})$ and $\mathsf{SimVote}(\mathtt{id}, v_1, P_{\mathtt{c}}^{\mathtt{id}}, S_{\mathtt{c}}^{\mathtt{id}}, P_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{v}}^{\mathtt{id}})$ instead of $\mathsf{CreateVote}(\mathtt{id}, v_0, P_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{v}}^{\mathtt{id}})$ and $\mathsf{CreateVote}(\mathtt{id}, v_1, P_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{v}}^{\mathtt{id}})$ respectively. Then it is straightforward to see that the experiments $\mathsf{Exp}_{\mathcal{A},V}^{\mathsf{CR\text{-}CaI},\beta}$ and $\mathsf{Exp}_{\mathcal{A},V}^{\mathsf{CR\text{-}CaI},\beta'}$ are computationally indistinguishable for $\beta \in \{0, 1\}$ if the NIZKPK scheme has the zero-knowledge property.

5.5.2 Concrete instantiation

In the previous sections we have defined a *challenge-and-cast* electronic voting protocol, and shown that it fulfills the properties of ballot privacy, strong consistency, strong correctness, cast-as-intended, and coercion-resistant cast-as-intended provided in Section 3.3. The proposed scheme can work both with mixnet-based and homomorphic tally-based voting systems. In this Section, we provide the definition of the protocol primitives for a mixnet-based system, based on ElGamal over a finite field. In Section 5.6 we detail the implementation for an homomorphic tally-based electronic voting system and how the properties of the scheme are fulfilled.

Primitives

Some of the primitives used for this concrete instantiation are also used by the Neuchâtel protocol, and have been defined in Section 1.3:

- For encryption, the Signed ElGamal encryption scheme is used. This scheme generates randomized ciphertexts, and has been proven to be NM-CPA secure in [25].

- The signature scheme is RSA with the hash variant (RSA-FDH). This signature scheme has been proven to be unforgeable against chosen message attacks in the random oracle model [18].

- The non trapdoor-simulatable NIZKPK scheme DecP is used for proving correctness of the decryption process. A NIZKPK satisfies the properties of completeness, knowledge soundness and zero-knowledge [45], [113].

Other specific primitives used in this instantiation are:

Chameleon hash The following instantiation of a chameleon hash $(\mathsf{Gen}_{\mathsf{ch}}, \mathcal{H}_{ch}, \mathcal{H}_{ch}^{-1})$ based on the discrete logarithm problem [76] is used: $\mathsf{Gen}_{\mathsf{ch}}$ receives a group $\mathbb{G}$ of prime order q of elements in $\mathbb{Z}_p^*$ with generator g. An element x is sampled uniformly from $\mathbb{Z}_q$ and $h = g^x$ is computed. Then, the evaluation key $\mathrm{ek}_{\mathrm{ch}}$ is defined as $\mathrm{ek}_{\mathrm{ch}} = (\mathbb{G}, g, h)$ and the trapdoor key $\mathrm{tk}_{\mathrm{ch}}$ is defined as $\mathrm{tk}_{\mathrm{ch}} = (\mathrm{ek}_{\mathrm{ch}}, x)$. The message space and the randomness space are $\mathbb{Z}_q$ and the hash space is $\mathbb{G}$. The algorithm $\mathcal{H}_{ch}$ is defined for $(m, \mathrm{r}_{\mathrm{ch}}) \in \mathbb{Z}_q \times \mathbb{Z}_q$ to output $\mathrm{c}_{\mathrm{ch}} = g^m \cdot h^{\mathrm{r}_{\mathrm{ch}}}$. Finally, $\mathcal{H}_{ch}^{-1}(m, \mathrm{r}_{\mathrm{ch}}, m')$ outputs $\mathrm{r}_{\mathrm{ch}}{}' = (m - m') \cdot x^{-1} + r$.

Simulatable NIZKPK We use a simulatable NIZKPK based on a σ-protocol which proves that a specific plaintext corresponds to a given ciphertext. The σ-protocol computed over an ElGamal ciphertext of the form $(c_1, c_2) = (g^r, pk_{\mathsf{e}}^r \cdot m)$ is as follows:

1. Prover computes $(a_1, a_2) = (g^s, pk_{\mathsf{e}}^s)$, where s is a random element $\in \mathbb{Z}_q$, and provides them to the verifier.
2. Verifier provides a challenge e.
3. Prover provides to the verifier $z = s + re$.

Finally the verifier checks that $g^z = a_1 \cdot c_1^e$ and that $h^z = a_2 \cdot (c_2/m)^e$. This σ-protocol can be simulated in the following way: the simulator samples a random $z^* \in \mathbb{G}$, a random $e^* \in \mathbb{Z}_q$ and computes $a_1^* = g^{z^*} \cdot c_1^{-e^*}$ and $a_2^* = h^{z^*} \cdot (c_2/m)^{-e^*}$. The resulting (a^*, e^*, z^*) values have the same distribution as the original ones.

Note that the σ-protocol is computed on the ElGamal ciphertext, not on the signed ElGamal one. With this protocol the voter is assured that the ElGamal ciphertext contains the selected voting option. As the Schnorr proof of the signed ElGamal encryption is publicly verifiable, this assures the voter that the signed ElGamal ciphertext contains her selected voting option too.

The NIZKPK algorithms ($\mathsf{GenCRS}, \mathsf{NIZKProve}, \mathsf{NIZKVerify}, \mathsf{NIZKSimulate}$) are then defined by using the discrete log-based chameleon hash scheme and the σ-protocol defined above, as well as two hash functions H_1, H_2 mapping inputs to $\mathbb{Z}_q$, as follows:

GenCRS runs $\mathsf{Gen}_{\mathsf{ch}}$ and outputs $\mathsf{crs} = (\mathbb{G}, g, h)$ and $\mathsf{tk} = (\mathsf{crs}, x)$;

$\mathsf{NIZKProve}$ receives crs, the statement $x = (c_1, c_2/m)$ and the witness r, and computes: (1) the commitment $(a_1, a_2) = (g^s, pk_{\mathsf{e}}^s)$, (2) the non-interactive challenge $e = H_2(g^{(H_1(x,a))} \cdot h^{\mathrm{r_{ch}}})$, where $\mathrm{r_{ch}}$ is picked at random from $\mathbb{Z}_q$, (3) the answer $z = s + re$, and (4) provides the proof $\pi = (a, e, \mathrm{r_{ch}}, z)$;

$\mathsf{NIZKVerify}$ checks that $g^z = a_1 \cdot c_1^e$, $h^z = a_2 \cdot (c_2/m)^e$, and that $e = H_2(g^{(H_1(x,a))} \cdot h^{\mathrm{r_{ch}}})$;

$\mathsf{NIZKSimulate}$ receives as input a statement $x^* = (c_1, c_2/m^*)$ and the trapdoor tk, and does the following: takes at random $z^* \in \mathbb{G}$ and random pair $(\alpha, \beta) \in \mathbb{Z}_q$, and sets $e^* = H_2(g^\alpha \cdot h^\beta)$. Then it computes $a_1^* = g^{z^*} \cdot c_1^{-e^*}$ and $a_2^* = h^{z^*} \cdot (c_2/m^*)^{-e^*}$, and finally it obtains $\mathrm{r_{ch}}^* = (\alpha - H_1(x^*, a^*)) \cdot x^{-1} + \beta$. The simulated proof is then $\pi^* = (a^*, e^*, \mathrm{r_{ch}}^*, z^*)$.

The properties of completeness, knowledge soundness and zero-knowledge of this scheme have been proven in Section 5.4.2.

Mixnet. We use a verifiable mixnet such as those proposed by Stephanie Bayer and Jens Groth [13], or Douglas Wikstrom in [121]. These mixnets have been proven by their authors to be sound, meaning that $\mathsf{MixVerify}$ will only output 1 given a correct execution of Mix, and zero-knowledge in the random oracle model when non-interactive proofs are used.

5.5.3 Performance

This instantiation is simple and efficient. For a *t-out-of-k* voting scheme, where voters can select up to t voting options from the k available, the encryption of the voter selections using the Signed ElGamal encryption scheme requires 3 exponentiations (assuming t can be encrypted into one ElGamal ciphertext). The computation of the NIZKPK requires 6 additional exponentiations (2 of them for the computation of the chameleon hash), and 6 more for verification. Each proof simulation costs 6 exponentiations.

An important detail is that, for efficiency purposes, the prime group and the generator of such group used in all these primitives must be the same. Obviously, the public key of the encryption scheme should be different from the evaluation key of the chameleon hash, as we will give the trapdoor key of the latter to the voter.

5.6 Protocol for homomorphic tally-based systems

Besides the different method of anonymization at the counting phase (recall that mixnet-based voting systems shuffle the anonymous ciphertexts prior to decryption, while homomorphic tally systems operate the ciphertexts together and decrypt the result), the main difference between mixnet-based and homomorphic tally systems is in the representation of the voting options for encryption: in mixnet-based voting systems, the value v_i to encrypt represents the voting option selected by the voter. In homomorphic tally-based voting systems, such value denotes the voting option for which *a one* is going to be encrypted, while *a zero* will be contained in the ciphertexts corresponding to the rest of the options.

For an homomorphic tally-based system, the encryption scheme ($\mathsf{Gen}_\mathsf{e}^\mathsf{h}, \mathsf{Enc}^\mathsf{h}, \mathsf{Dec}^\mathsf{h}$, $\mathsf{EncVerify}^\mathsf{h}$) is used instead of ($\mathsf{Gen}_\mathsf{e}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EncVerify}$), and the NIZKPK scheme ($\mathsf{GenCRS}^\mathsf{h}, \mathsf{NIZKProve}^\mathsf{h}, \mathsf{NIZKVerify}^\mathsf{h}, \mathsf{NIZKSimulate}^\mathsf{h}$), is used instead of ($\mathsf{GenCRS}$, $\mathsf{NIZKProve}, \mathsf{NIZKVerify}, \mathsf{NIZKSimulate}$). The following algorithms change with respect to those described in Section 5.5:

> $\mathsf{CreateVote}(\mathtt{id}, \{v_{j_1}, \ldots, v_{j_t}\}, P_\mathtt{v}^\mathtt{id}, S_\mathtt{v}^\mathtt{id})$ takes the whole set of voting options $V = \{v_1, \ldots, v_n\}$ from the bulletin board, sets $m_i = 0$ for i such that $v_i \notin \{v_{j_1}, \ldots, v_{j_t}\}$ and $m_i = 1$ for the rest, and runs Enc^h from the encryption scheme with inputs pk and the list of messages $\{m_i\}_{i=1}^k$, obtaining the ciphertext c. Then it parses $P_\mathtt{v}^\mathtt{id}$ as crs and runs $\mathsf{NIZKProve}^\mathsf{h}$ from the NIZKPK scheme, using as input crs, the statement $(c, \{m_i\}_{i=1}^k)$ and the encryption randomness r. The ballot b is set to be the ciphertext c, and the encryption data $\tilde{r}$ is π.
>
> $\mathsf{AuditVote}(b, \tilde{r}, \{v_{j_1}, \ldots, v_{j_t}, P_\mathtt{v}^\mathtt{id})$ takes the whole set of voting options $V = \{v_1, \ldots, v_n\}$ from the bulletin board, sets $m_i = 0$ for i such that $v_i \notin \{v_{j_1}, \ldots, v_{j_t}\}$ and $m_i = 1$ for the rest. Then it parses b as c, $\tilde{r}$ as π and $P_\mathtt{v}^\mathtt{id}$ as crs, and runs $\mathsf{NIZKVerify}^\mathsf{h}$ from the NIZKPK scheme for the statement $(c, \{m_i\}_{i=1}^k)$ and the common reference string crs.

$\mathsf{Confirm}(\mathtt{id}, b, S_{\mathtt{v}}^{\mathtt{id}}, S_{\mathtt{c}}^{\mathtt{id}}, P_{\mathtt{c}}^{\mathtt{id}}, sk_{\mathtt{a}})$ parses $S_{\mathtt{c}}^{\mathtt{id}}$ as $(sk_{\mathtt{s}}, \mathsf{tk})$ and runs Sign with inputs the voter's signing private key $sk_{\mathtt{s}}$ and the ballot b to be signed, together with the voter identity id. The resulting ballot signature ψ is set to be the ballot confirmation C_b. Then it parses b as c and runs $\mathsf{NIZKSimulate}^{\mathsf{h}}$ from the NIZKPK scheme the statement $(c, \{m_i^*\}_{i=1}^k)$, where the values $\{m_i^*\}_{i=1}^k$ are additionally provided or randomly selected. The simulated proof π^* is set to be the auxiliary data σ'.

$\mathsf{ProcessBallot}(\mathsf{BB}, b_a)$ runs $\mathsf{EncVerify}^{\mathsf{h}}$ instead of EncVerify.

$\mathsf{Tally}(\mathsf{BB}, sk)$ in the first place runs the ProcessBallot and ProcessConfirm algorithms, and discards the ballots for which any of the verifications fail. Finally it extracts the ciphertexts c from the ballots which have passed the verifications and operates component-wise all the ciphertexts. Then it runs the $\mathsf{Dec}^{\mathsf{h}}$ algorithm with the resulting ciphertext c_H, obtaining v_H, from which the number of votes received for each voting option $v_i \in V$ is obtained. Finally the ProveDec algorithm from the NIZKPK scheme is run with the statement (c_H, v_H) and the private key sk as the witness. The outputs are the number of votes received for each option, r, the decrypted value v_H and the proof of correct decryption, Π.

$\mathsf{VerifyTally}(\mathsf{BB}, r, \Pi)$ in the first place runs the ProcessBallot and ProcessConfirm algorithms over the votes in the bulletin board, and discards the ballots for which any of the verifications fail. Finally it extracts the ciphertexts c from the ballots which have passed the verifications and operates component-wise all the ciphertexts, obtaining the aggregated ciphertext c_H. Then it runs the VerifyDec algorithm from the NIZKPK scheme with inputs the aggregated ciphertext c_H and the decrypted value v_H. Finally it checks that the result r corresponds to counting the number of times each candidate has been selected using the value v_H.

The change in the defined algorithms consists basically of changing the encryption and NIZKPK schemes, and operating the ciphertexts component-wise prior to decryption, instead of running Mix, in the Tally algorithm.

5.6.1 Security Analysis

The security analysis remains the same as in Section 5.5.1, considering this change in the algorithms, provided that the properties of the schemes $(\mathsf{Gen}_{\mathsf{e}}^{\mathsf{h}}, \mathsf{Enc}^{\mathsf{h}}, \mathsf{Dec}^{\mathsf{h}}, \mathsf{EncVerify}^{\mathsf{h}})$, $(\mathsf{GenCRS}^{\mathsf{h}}, \mathsf{NIZKProve}^{\mathsf{h}}, \mathsf{NIZKVerify}^{\mathsf{h}}, \mathsf{NIZKSimulate}^{\mathsf{h}})$, are the same as those of $(\mathsf{Gen}_{\mathsf{e}}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EncVerify})$, $(\mathsf{GenCRS}, \mathsf{NIZKProve}, \mathsf{NIZKVerify}, \mathsf{NIZKSimulate})$.

5.6.2 Primitives

Here we define the specific primitives which are used in the case of an homomorphic tally-based system:

Encryption scheme the encryption scheme $(\mathsf{Gen}^{\mathsf{h}}_{\mathsf{e}}, \mathsf{Enc}^{\mathsf{h}}, \mathsf{Dec}^{\mathsf{h}}, \mathsf{EncVerify}^{\mathsf{h}})$ works as follows:

$\mathsf{Gen}^{\mathsf{h}}_{\mathsf{e}}$ calls the ElGamal key generation algorithm to obtain the key pair $(pk_{\mathsf{e}}, sk_{\mathsf{e}})$.

$\mathsf{Enc}^{\mathsf{h}}$ receives a set of messages $\{m_i\}_{i=1}^k$, computes for each one $m'_i = g^{m_i}$ and $\mathsf{Enc}(m'_i, pk_{\mathsf{e}}) = c'_i$. It also generates the set of proofs π_{enc}, which ensure that the vote is well-formed: these proofs include a proof that each ciphertext encrypts g^0 or g^1, and a proof that at most t ciphertexts encrypt a g^1 value (see [37], [43]). The set of ciphertexts c'_i and the proof π_{enc} are set to be the output ciphertext c.

$\mathsf{Dec}^{\mathsf{h}}$ receives as input a ciphertext $c = \{c'_i\}_{i=1}^k$, and for each $i = 1, \ldots, k$ it calls to the ElGamal decryption algorithm Dec with the ciphertext c'_i and the private key sk_{e} as input, to obtain the decrypted message $m'_i = g^{m_i}$. Usually, a table of pre-computed values containing pairs (m_i, g^{m_i}) is used to retrieve the original messages m_i.

$\mathsf{EncVerify}^{\mathsf{h}}$ verifies the proofs π_{enc} of the ciphertext c, according to the description in [37], [43], and returns 1 if the verifications are successful, 0 otherwise.

According to [25], this exponential ElGamal variant with proofs of well-formed ciphertexts is NM-CPA secure.

Trapdoor-simulatable NIZKPK scheme the trapdoor-simulatable NIZKPK scheme $(\mathsf{GenCRS}^{\mathsf{h}}, \mathsf{NIZKProve}^{\mathsf{h}}, \mathsf{NIZKVerify}^{\mathsf{h}}, \mathsf{NIZKSimulate}^{\mathsf{h}})$ uses the algorithms $(\mathsf{GenCRS}, \mathsf{NIZKProve}, \mathsf{NIZKVerify}, \mathsf{NIZKSimulate})$ defined in Section 5.5.2 as follows:

$\mathsf{GenCRS}^{\mathsf{h}}$ calls to the GenCRS algorithm to obtain the common reference string crs and the trapdoor key tk.

$\mathsf{NIZKProve}^{\mathsf{h}}$ receives the input statement $(c, \{m_i\}_{i=1}^k)$ and witness r. It parses c as $\{c'_i\}_{i=1}^k$ and r as $\{r'_i\}_{i=1}^k$. Then for each m_i in the statement it computes $m'_i = g^{m_i}$, and runs $\mathsf{NIZKProve}$ with inputs crs, the statement (c'_i/m'_i), and the randomness r'_i. The set of all generated proofs $\{\pi'_i\}_{i=1}^k$ is set to be the provided output π.

$\mathsf{NIZKVerify}^{\mathsf{h}}$ receives the input statement $(c, \{m_i\}_{i=1}^k)$ and the proof π. It parses c as $\{c'_i\}_{i=1}^k$ and π as $\{\pi'_i\}_{i=1}^k$. For each m_i in the statement, it computes $m'_i = g^{m_i}$ and runs $\mathsf{NIZKVerify}(\pi'_i, \mathsf{crs}, (c'_i/m'_i))$. The output is 1 if all the executions of $\mathsf{NIZKVerify}$ output 1, 0 otherwise.

$\mathsf{NIZKSimulate}^{\mathsf{h}}$ receives the input statement $(c, \{m^*_i\}_{i=1}^k)$ and parses c as $\{c'_i\}_{i=1}^k$. Then it computes $m'_i = g^{m^*_i}$ for each m^*_i in the statement and runs the $\mathsf{NIZKSimulate}$ algorithm with input (c'_i/m'_i) and the comon reference string crs. The resulting set of simulated proofs $\{\pi_i*\}_{i=1}^k$ is set to be the provided output π^*.

We can see that the homomorphic-variant scheme basically transforms messages m_i into g^{m_i} and executes multiple runs of the underlying NIZKPK scheme algorithms. This variant has the same properties of completeness, knowledge soundness and zero-knowledge of the underlying NIZKPK scheme.

5.7 Multiple Trustees

Multiparty computation techniques can be used in order to allow a subset of election authorities and of registrars to be corrupt while maintaining the security properties of the scheme.

Regarding the election authorities, in our privacy analysis we require them not to divulge the election private key. For strong consistency, it is necessary that the private key has been generated in a proper way. These two premises can be achieved while supporting a subset of them to be corrupt. Particularly, we define that for l election authorities, $t+1$ are honest, being $0 \leq t \leq l-1$.

At the beginning of the election, in the Setup algorithm, they generate the private and public key of the election using the Pedersen distributed key generation protocol [91]. A description of how this protocol can be used to distributedly generate an ElGamal key pair can be found in [43], and a security analysis is found in [40]. At a high level, each election authority generates a private key, provides shares of it to the rest of the participants using a verifiable secret sharing scheme, and then all together compute the resulting public key. The private key piece to be held by each participant corresponds to the additions of the shares provided by the other participants. The protocol provides proofs which allow the detection of cheaters that may disrupt the key generation process. It also ensures that the election private key is kept secret even in the presence of up to $l-(t+1)$ corrupt electoral authorities.

Depending on the setup, during the execution of the Tally algorithm the election private key may be reconstructed in order to decrypt the votes, or a threshold decryption scheme may be used in which each electoral authority partially decrypts each vote.

In the case of the registration authorities, they are required to generate the voter's public and private keys in a proper way for strong correctness, and they are assumed not to divulge the voter private keys for cast-as-intended verifiability. Multiparty computation techniques can be used, as in the case of electoral authorities, in order to ensure that if at least $t+1$ registration authorities are honest, the generated keys are correct, and are kept secret.

During registration, the registrars engage in an interactive protocol in order to generate the shares of sk_{s} and tk in a verifiable way (techniques in [26] and [47] can be used for RSA keys). The corresponding public keys are posted in the bulletin board and each registrar provides its corresponding share of the private keys to the voter. During the voting phase, the voter introduces the shares into the voting

device where the private keys are reconstructed prior to being used. Given that the public keys are already published on the bulletin board, any attempt of a registrar to provide wrong shares to the voter will be detected. Moreover, commitments to the individual shares can be also published so that the malicious registrar can be detected.

5.8 Voting Scheme

In this section we describe how a voting scheme that uses our protocol would be. The protocol can be used both in remote and in poll-site voting, as far as the vote is cast electronically and the voter has access to a trusted device with which she performs the cast-as-intended verification. In the description of our voting scheme, we provide details on how certain processes may be managed for each case, as well as other customizations that depend on the scenario or country where the electoral process is held.

Before describing the voting scheme, it is important to recall the critical nature of the voter's trapdoor key in the protocol we have proposed. A voter who doesn't have access to the trapdoor key will not be able to simulate a proof. Thus, the cast-as-intended verification mechanism will no longer protect the privacy of the voter. On the other hand, the voter device has to learn the trapdoor key only after it has already generated an honest proof for the voter. Otherwise, the device could simulate a proof the voter expects to be honest, and the scheme would no longer be cast-as-intended verifiable.

In order to present an easy and intuitive voting process for the voter, we have related the private information she uses to authenticate her vote (for example, her private signing key) with the trapdoor key which is used to generate false proofs. We think that it is meaningful that the voter provides both secrets at the same time, as a confirmation that she agrees to cast that vote (which she is expected to do only after verifying the honest proof). Before the voter provides these secrets, the voting device can neither cast a valid vote, nor cheat the voter by generating a fake proof.

On the other hand, in case the voter does not have access to the trapdoor key (and therefore she cannot simulate proofs for a coercer), she doesn't have access to the signing key either, which is necessary for successfully casting the ballot. Therefore, she cannot sell her vote.

The voting scheme we envisage is the following:

During the ***election preparation stage*** the electoral commission defines the questions to be voted for in the election, and setup parameters such as the election public and private keys. In case of a multi-authority scheme, the election private key is generated in a distributed way by the electoral commission members, who keep their shares in private, and the election public key is constructed and published, along with proofs of the correct computation of such public key.

During the ***voter registration stage*** the electoral roll, containing the identities of the voters who participate in the election, is defined. Depending on the scenario or on the country, this electoral roll is automatically generated by the electoral commission from already existing census data, or the voters actively register to participate in the election.

Voters in the electoral roll are issued a public and private voter key pair containing a signing key pair to confirm or authenticate their vote, and an evaluation/trapdoor key pair for the NIZKPK scheme. In remote electronic voting scenarios voters may receive this key pair in advance through a private channel (for example, in person when they go to register), or in a password-protected keystore that is downloaded at the voting device through the internet, for which they have received a password through a secondary private channel such as sealed envelopes transmitted by postal mail. In case of multiple registration entities, voters may receive a share of the private keys from each registrar.

In poll-site electronic voting scenarios, poll workers may issue the voter's key pair at the time the voter enters the poll-site and authenticates herself. In this case, a smart-card or any other hardware token may be used to provide the private keys to the voter.

During the ***voting stage***, the voter proceeds to access her ballot and make her selections. It may be the case that the voter needs to authenticate before accessing her ballot, for example by means of a citizen portal or a Single-Sign-On (SSO) system deployed at the country or institution level, for which the voter already has a login and password information (since these systems are often used for activities other than electronic voting, such as paying taxes). In that case, the voter's private key provided during registration - or the password to recover it - may be presented as a *vote confirmation key* such as in the system implemented in Neuchâtel (see previous Chapter), instead of as an authentication key, to make it clearer to the voter that she is expected to provide it once she agrees with her vote.

Once the voter has finished making her choices, they are encrypted by the voting device. The resulting ciphertext and the proof of content are shown to the voter, who then can use an audit device to check that the ciphertext contents match her selections. In case of a remote voting scenario, voters may use their smartphone (if they were voting from their PC) or a software application different than the one used to generate the vote. In case they do not trust any device to maintain their privacy (since the audit device learns the voter's choices), voters may opt to check the encryption of test choices, rather than the real ones. Similarly to Helios, a voter may be convinced that the voting device is honest by checking test votes, and then cast the real vote without auditing it. In a poll-site voting scenario, voters may use a dedicated hardware device available in the poll-site to check the content of the generated vote.

After a positive audit, the voter enters her private key (or the password to recover it) into the voting device, which uses the private signing key part to digitally sign the vote to be cast, and the NIZKPK trapdoor key part to generate one or several fake proofs. The number of fake proofs that can be generated can be configured up to the total of voting options and depend on the requirements and assumptions of the electoral process. In the same way, the voter may be asked to enter several voting options for which she wants to generate a fake proof. Otherwise, the voting device can randomly choose them. The fake proofs have to be presented in the same way as the honest one (for example, in a printed form), so that they cannot be distinguished by a potential coercer.

In the case of poll-site voting, if a hardware token is used for providing the private keys to the voter, such token could be in charge of performing the audit of the proofs generated by the voting device, and only in the case it is successful proceed to release or use the keys to confirm the ballot. The hardware token should have an input/output interface for communicating with the voter, in order to receive the voter selections and communicate the result of the verification.

Once the vote is cast and received by the bulletin board manager running in the remote server, it is published in the bulletin board, so that voters can check that their audited votes have been correctly received. As we have explained in Chapter 2, a hash of the vote may be published in the bulletin board, instead of the vote itself, for privacy reasons. In that case, such hash of the vote is additionally provided to the voter in the vote audit/approval phase, together with the encrypted vote and the proofs of content. The audit device will check, besides the proofs, that the hash matches the provided ciphertext.

At the ***counting stage*** the electoral commission collects the votes from the bulletin board (or from the private ballot box if only hashes were published) and pass them through a mixnet before reconstructing the election private key to proceed to the decryption and tally of the votes. The tally result is published in the bulletin board together with the proofs of correct mixing and decryption. Auditors proceed then to verify the contents of the bulletin board / ballot box, and to check the proofs of mixing and decryption in order to confirm the result.

5.9 Protocol extension for multiple voting

We have focused our description in the single-voting case. This means that in case something bad happens after the voter confirmation (the ballot is not cast, or a different ballot is posted on the bulletin board on behalf of the voter), although the voter can detect it, she cannot re-vote. This is due to the fact that the voting device already knows her private keys, and specifically the tradoor key, and could cheat the voter in further ballot generations with simulated proofs of content. Several approaches can be followed in order to allow multiple voting to prevent such situations.

The **first approach** is multiple registration: in case the voter detects some failure, she can re-register and ask for a new set of keys. The previous ballot cast on behalf of the voter is revoked. A similar approach was used in the NSW iVote system [68].

The **second approach** can be applied when voters are provided with hardware tokens storing their private keys. For example, it can be used in poll-site voting scenarios: the hardware token contains the voter's private signing and trapdoor keys. After verification of the proof of the ballot content (which can be done by the hardware token itself, or by the voter, who then indicates to the token that everything is OK), the token uses the private signing and trapdoor keys to confirm the vote, digitally sign it and generate simulated proofs, without releasing such private keys to the voting device. Given that the private keys are never known to the voting device, the voter can cast multiple ballots while fulfilling the coercion-resistant cast-as-intended verifiability property. In such case, the bulletin board manager may accept multiple confirmed ballots from the same voter and the tally may select the last one for participating in the count.

The **third approach** consists of delegating the generation of simulated proofs to the bulletin board manager: the trapdoor keys for all the voters are kept by the bulletin board manager, who generates and provides the voting devices with simulated proofs *only* when receiving confirmed ballots (which means that voters already verified their contents and agreed with them). From the security analysis it follows that the bulletin board having the trapdoor keys does not endanger the voter's privacy. However, a collusion of the bulletin board manager and the voting device defeats the property of cast-as-intended verifiability: the bulletin board manager may provide simulated proofs to the voting device, to show them to the voter as honest proofs of ballot content. Also, in case the bulletin board manager refuses to generate the simulated proofs for a confirmed ballot, the coercion-resistant cast-as-intended property is not fulfilled. The assumption that the bulletin board manager is trusted, in the sense that it performs the designated operations, was already stated in Section 3.3.1. A distributed setting, where multiple bulletin board managers hold shares of the voters' trapdoor keys generated with a threshold scheme can be used to enforce this property, even if a subset of the bulletin board managers are malicious.

Chapter 6

Making Cast-as-Intended Universal

6.1 Introduction

In the previous chapter we showed a protocol in which the voting device, after generating an encrypted vote, provided a proof of knowledge so that the voter could check that the ciphertext contained what she selected. In order to defend from possible coercers who may request the proof after the voter has voted, we gave her (or her voting device) the ability to simulate proofs using a trapdoor key, in such a way that she could make the coercer believe that any content (real or not) was inside the encrypted vote.

This scheme gave us an idea that implies a change of paradigm. The voting device could prove that the content of a generated ciphertext is any of the voting options in the election. Then, anybody could verify such proofs without posing any risk on the privacy of the voter. The proofs would tell nothing about what the voter has voted.

In order to do that, the voting device would have to use a trapdoor to simulate the proof in the case of the voting options that are not in the ciphertext, but it would not need such a trapdoor in the case of the voting option that is indeed encrypted. By relating one trapdoor key to each voting option, and letting the voter control which are the trapdoor keys the voting device has access to and which not, the voter knows which is the content of the ciphertext in case all the proofs verify successfully. Moreover, the proof verification is not restricted to the voter, but anyone can do it. *Universal cast-as-intended* was born.

6.2 Motivation

In Chapter 2 we have seen several proposals for cast-as-intended verification which have been proposed mainly during the last 10 years. All these proposals have something in common: they depend on the voter participation. In case voters do not

follow the protocol accordingly or do not verify the proofs they are provided, a malicious voting device can modify the vote that is cast on behalf of the voter without detection.

On one hand, we might think of systems which make the cast-as-intended verification mandatory in order to cast a vote. However, current cast-as-intended verification systems present important drawbacks for the voter: verification mechanisms are usually not very usable and in most cases voters have to engage in highly interactive protocols and/or be able to perform complex computations. Therefore, mandatory cast-as-intended verification would disenfranchise less skilled voters. On the other hand, we could allow the cast-as-intended verification to be an optional step which the voter can do before casting the vote. This does not solve the problem, since targeted attacks against non-skilled voters, who will probably not use the verification system, can succeed undetectably.

The voting device may even subvert the verification protocol by not showing the verification information to the voter at all. A voter who is not well informed about the process may not detect this modification, or even may think that the procedure has been updated without notification.

Another drawback of some current cast-as-intended verification mechanisms is that the verification depends on providing the voter with a proof of how she voted, which can be shown to a third party to sell her vote. Although some mitigations for this issue exist, they do not completely solve the problem.

With this proposal, we aim to reduce the effort required from the voter to perform the cast-as-intended verification. In fact, the voter does not have to do anything regarding this verification because the generated proofs can be universally verified. What the voter is expected to do, is to enter the codes related to the voting options she selects, as in the case of a code-voting scheme. In the same way as in these kinds of schemes, the main source of trust regarding cast-as-intended verification relies on how these codes are provided to the voter, and how their integrity is preserved. After voting the system does not provide a proof of how the voter voted, even if she provides her codes to the coercer.

6.3 UCIV System description

6.3.1 Overview

At a high level, the system is based on providing each voter, during the registration phase, with a set of secret values related to the voting options in the election. During the voting phase, the voter uses a voting device to make her choices, and then provides a subset of the secret values she received before. The subset she provides depends on the voting option she has selected.

The voting device encrypts the voter's choices and, using the secrets provided to the voter, it generates a proof of the content of the ballot. This proof of content is sent, together with the ballot, to the voting server, where it is verified prior to being posted on the bulletin board. The proof does not disclose any information regarding the voter selections, and therefore it can be universally verified. Auditors may collaborate with the server in order to independently verify such proof.

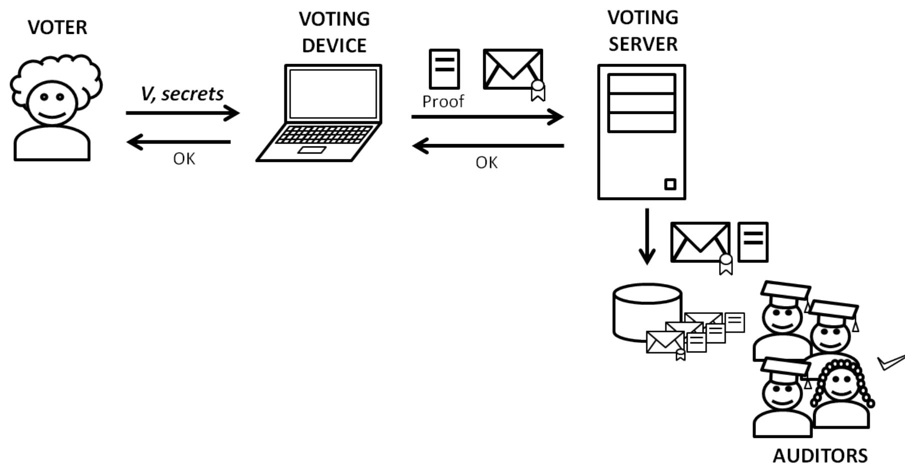

Figure 6.1: UCIV: Voting process overview

Given that the secrets entered by the voter in the voting device depend on what she selected, and that the voting device is only able to generate a valid proof for the right combination of secrets and voting options, a successful verification proves that the vote has been cast as intended.

In this Chapter, we propose two alternatives for a voting protocol providing universal cast as intended. The first one is based on non-interactive zero-knowledge proofs of knowledge: the voter enters the secrets **not** related to her selected options into the voting device, which generates a NIZK proof for each voting option saying: *"Either this voting option is in the encrypted vote, or I know its corresponding secret"*. Naturally, since the voter did only provide the secrets for the voting options she did not choose, the only way that a valid proof can be generated by the voting device is by encrypting the voter's selections.

The second alternative is based on using our "old friends", the return codes, but in a different way than what was previously described. In this case, the voter is required to enter those return codes from her verification card corresponding to her selected voting options into the voting device.

As in previously described return code based protocols, the voting device generates the (probabilistic) encryption of the voter's choices, and the corresponding partial return codes (which are generated by a deterministic computation), and sends them, together with the return codes provided by the voter, to the server.

Recall that in the Neuchâtel's protocol described in Chapter 4, we used a digital signature generated during voter registration as a reference value to know if a ballot confirmation was valid: in case the server could use a received confirmation message to compute a value, for which such signature was verified successfully, then the confirmation message was correct. In this proposal, we use a similar approach for checking at the server whether the combination of a return code (which corresponds to a specific voting option in the verification card), and a partial return code (which is directly related to the ballot's encrypted options), is correct. In an affirmative case, the ballot's encrypted voting options correspond to the voter's selections.

With this technique, we can apply the measures for multi-party computation of the verification cards described in Section 4.8.2, in order to ensure their secrecy and correctness. With the first approach, the generation of the secret values for the proof generation can also be distributed among a set of registrars, for which a voter may decide to register with a subset. Only one registrar, from those selected by the voter, must remain honest in order to ensure that the audit data is kept secret and has been correctly generated.

This chapter is organized as follows: first we provide the syntax and security definitions for UCIV schemes, which can be found in [52]. Then we describe the protocols for the two approaches, provide a security analysis and proposals for implementation. Finally, we show examples of voting schemes using such protocols.

6.3.2 Syntactical definition

In this Section we include a syntactical definition from [52] which is particular for schemes providing the UCIV property. We also present how the algorithms are organized in the different phases of the voting protocol.

As in Chapter 3 we have as participants the *Election Authorities*, the *Voters*, the *Voting Device* and the *Registrars*. In the original definition from [52], authors include the functionality of a distributed set of registrars, from which the voter may decide to register with a subset. However, for the sake of clarity, we will consider they work as a sole registration entity and discuss alternative approaches in Section 6.6.

The syntax is focused on the universal cast-as-intended verification mechanism, which involves specific procedures mainly in the registration and voting phases. Therefore, verification of the counting process is left aside and no auditors of the tally participate in this definition. The definition can be extended in the future in order to include this process.

As usual, we consider that non-cryptographic election specifications such as the set of voting options V, the set of voters and the counting function $\rho : (V \cup \{\perp\})^* \rightarrow R$ are fixed in advance by the *Election Authorities*. Also, the use of voter credentials

for authentication and vote casting is out of the scope of this description.

The voting scheme is characterized by the following algorithms:

$\mathsf{Setup}(1^\lambda)$ is a protocol executed by the Election Authorities. On input a security parameter 1^λ, it generates and outputs an election public/private key pair (pk, sk).

$\mathsf{Register}(\mathtt{id}, V, pk)$ is run by the Registrars. It takes as input a voter identity $\mathtt{id}$, the set of voting options V and the election public key pk. It outputs the secret UCIV information $s_{uciv}^{\mathtt{id}}$ and the public UCIV information $p_{uciv}^{\mathtt{id}}$.

$\mathsf{Vote}(\mathtt{id}, v, \sigma_v(s_{uciv}^{\mathtt{id}}), p_{uciv}^{\mathtt{id}}, pk)$ is a probabilistic protocol run by the voting device. It receives as input the voter identity $\mathtt{id}$, a voting option $v \in V$, the function σ_v evaluated on the secret UCIV information $s_{uciv}^{\mathtt{id}}$, the public UCIV information $p_{uciv}^{\mathtt{id}}$, the election public key pk, and outputs a ballot b.

$\mathsf{ProcessBallot}(\mathsf{BB}, b, \mathtt{id})$ is run by the bulletin board manager. It receives as input a bulletin board BB, a ballot b and a voter identity $\mathtt{id}$ and outputs either 1 if the process is successful, 0 otherwise.

$\mathsf{Tally}(\mathsf{BB}, sk)$ is run by the Election Authorities. It takes as input a bulletin board BB and the election secret key sk (or its shares if it was split during Setup) and outputs a result $r \in R$ and a correct tabulation proof Π.

Finally, the protocol is executed in the following phases:

Configuration phase: in this phase, the set of voting options V, the set of voters who can vote, and the counting function ρ are defined. Then the election authorities run the Setup algorithm, publish the election public key pk in the bulletin board BB, and keep the election private key sk.

Registration phase: in this phase, a voter with identity $\mathtt{id}$ proceeds to register to vote in the election. The registrars run the $\mathsf{Register}$ algorithm and provide the secret UCIV information $s_{uciv}^{\mathtt{id}}$ to the voter through a secure channel, so that their secrecy is preserved in front of other participants in the protocol. The public UCIV information $p_{uciv}^{\mathtt{id}}$ is published in the bulletin board BB.

Voting phase: in this phase the registered voter with identity $\mathtt{id}$ uses a voting device in order to choose her preferred voting option v. The voter also evaluates $\sigma_v(s_{uciv}^{\mathtt{id}})$ and provides the result to the voting device. The voting device takes the election public key pk and the public UCIV information $p_{uciv}^{\mathtt{id}}$ from the bulletin board and runs the Vote algorithm, which outputs a ballot b. The ballot b is then sent to the bulletin board manager, together with the voter identity $\mathtt{id}$. Upon reception of the ballot, the bulletin board manager executes the $\mathsf{ProcessBallot}$ algorithm. In case the output is 1, the ballot is published in the bulletin board, otherwise the ballot is discarded and the voter is notified accordingly.

Counting phase: the election authorities run the Tally algorithm using the election private key sk and the information in the bulletin board, including the posted ballots. The output of the Tally algorithm, containing the election result and the proofs of correct tabulation, is published on the bulletin board.

A voting system as defined above is correct if, when the four phases are run with all the participants behaving correctly, the result r output by the Tally algorithm is equal to the evaluation of the counting function ρ on the voting options corresponding to the ballots cast by the voters.

6.3.3 Security definitions

In this section we detail the security definitions, provided in [52], for an electronic voting scheme as defined in Section 6.3.2, which include ballot privacy, strong consistency and universal cast-as-intended verifiability. We additionally provide a definition for strong correctness.

Security assumptions

First of all, we informally detail which are the security assumptions made on the scheme.

Regarding voting devices, authors make a distinction between privacy and integrity. In order to guarantee privacy, it is assumed that the voting device behaves properly by correctly encrypting the voter's choice and not leaking any information. This is a common assumption in voting systems where the voting device is in charge of encrypting the voter's choices, such as Helios. On the other hand, the voting device is not trusted for integrity: it is assumed to try to change the selections made by the voters before the encryption. The verifiability of the scheme does not rely on the honesty of the voting device.

It is assumed that there is only one election authority, which is trusted for privacy. As authors comment in the paper, secret sharing and multi-party computation techniques can be used for overcoming this limitation. In such case, privacy would be guaranteed as long as a subset of the election authorities are trusted. The electoral authorities are not needed to be trusted for the UCIV property.

In the original definition in [52], authors consider a set of registrars which are trusted to produce the UCIV information correctly and not to leak the private information. We modify the definitions here in order to consider only one registrar to keep the scheme definition simpler, and then detail in further sections how this assumption may be reduced.

Ballot privacy

As in Section 3.3, authors adopt the formalization given in [23]. Ballot privacy is then defined using two experiments between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

The goal of the adversary is to disinguish between the two experiments. In both experiments, the adversary can corrupt voters and submit ballots on their behalf. In addition, for each honest voter the adversary can also specify two votes to be used for casting her ballot. The votes which will be used to cast the honest voters' ballots will depend on which experiment is taking place. The goal of the adversary is to distinguish between both experiments, which is the same as distinguishing which votes were used to cast the honest voters' ballots. As revealing the "true" tally would easily allow the adversary to distinguish between the experiments, the same tally is always shown to the adversary, regardless of which vote was used to cast honest voters' ballots.

For compactness, the two experiments are presented as a single experiment depending on a bit β. The experiments are parametrized by the set of voting options V and an algorithm $\mathsf{SimProof}(BB, r)$ such that, given a bulletin board and a result, it simulates a proof of correct tabulation.

1. **Setup phase**. The challenger sets up two empty bulletin boards L and R. It runs the $\mathsf{Setup}(1^\lambda)$ protocol to obtain the election public key pk and the election private key sk. It then posts pk on both bulletin boards. The adversary is given read access to either L if $\beta = 0$ or R if $\beta = 1$. In addition, $\mathcal{C}$ initializes an empty list ID.

2. **Registration phase**. The adversary may make one type of query.

   - **Register**($\mathtt{id}$) query. The adversary provides a voter identity such that $\mathtt{id} \notin \mathsf{ID}$. The challenger runs $\mathsf{Register}$ on inputs $(\mathtt{id}, V, pk)$ to generate the public UCIV information $p^{\mathtt{id}}_{uciv}$ and the secret UCIV information $s^{\mathtt{id}}_{uciv}$. $\mathcal{C}$ provides both $p^{\mathtt{id}}_{uciv}$ and $s^{\mathtt{id}}_{uciv}$ to $\mathcal{A}$ and $p^{\mathtt{id}}_{uciv}$ is published on both bulletin boards. The identity $\mathtt{id}$ is added to ID.

3. **Voting phase**. The adversary may make two types of queries.

   - **Vote**($\mathtt{id}, v_L, v_R$) queries. The adversary provides a voter identity $\mathtt{id}$ such that $\mathtt{id} \in \mathsf{ID}$ and two votes $v_L, v_R \in V$. The challenger runs $\mathsf{Vote}(\mathtt{id}, v_L, \sigma_{v_L}(s^{\mathtt{id}}_{uciv}), p^{\mathtt{id}}_{uciv}, pk)$ which outputs b_L and $\mathsf{Vote}(\mathtt{id}, v_R, \sigma_{v_R}(s^{\mathtt{id}}_{uciv}), p^{\mathtt{id}}_{uciv}, pk)$ which outputs b_R. $\mathcal{C}$ then obtains new versions of the boards L and R by running $\mathsf{ProcessBallot}(L, b_L, \mathtt{id})$ and $\mathsf{ProcessBallot}(R, b_R, \mathtt{id})$ and updating the boards accordingly.
   - **Ballot**($b, \mathtt{id}$) queries. These are queries made on behalf of corrupt voters. Here the adversary provides a ballot b and an identity $\mathtt{id}$ such that $\mathtt{id} \in \mathsf{ID}$. The challenger runs $\mathsf{ProcessBallot}(L, b, \mathtt{id})$ and if the process accepts it also runs $\mathsf{ProcessBallot}(R, b, \mathtt{id})$ and updates the boards accordingly.

4. **Tallying phase**. The challenger evaluates $\mathsf{Tally}(L, sk)$ obtaining the result r and the proof of correct tabulation Π. If $\beta = 0$, the challenger posts (r, Π) on the bulletin board L. If $\beta = 1$, the challenger runs $\mathsf{SimProof}(R, r)$ obtaining a simulated proof Π' and posts (r, Π') on the bulletin board R.

5. **Output**. The adversary $\mathcal{A}$ outputs a bit $\alpha^{\mathcal{A},V}$.

A voting protocol as defined in Section 6.3.2 provides ballot privacy, if there exists an algorithm $\mathsf{SimProof}$ such that for any probabilistic polynomial time (p.p.t.) adversary $\mathcal{A}$ and any set of voting options V, the following advantage is negligible in the security parameter λ.

$$\mathsf{Adv}_V^{\mathbf{priv}}(\mathcal{A}) := |\Pr[\alpha^{\mathcal{A},V} = 1|\beta = 1] - \Pr[\alpha^{\mathcal{A},V} = 1|\beta = 0]|$$

Strong consistency

In order to define Universal Cast-as-Intended Verifiability, the notion of strong consistency has to be defined first. Strong consistency states that the tally of a bulletin board must correspond to the result of applying a counting function to the *contents* of the ballots in the bulletin board. As shown in [23], this property is needed to avoid having leaky tallying algorithms. In addition, it is used to define a content extractor, which will be used in the universal cast-as-intended verifiability definition. The definition given in [23] is used.

Strong consistency is given by the following game, parametrized by a set of voting options V, a result function ρ and an algorithm $\mathsf{Extract}(b, sk)$ which takes a ballot and an election private key and outputs a vote, or $\perp$ in case of an invalid vote:

1. **Setup phase**. The challenger runs the $\mathsf{Setup}(1^\lambda)$ protocol to obtain the election public key pk and the election private key sk. It gives both pk and sk to the adversary $\mathcal{A}$.

2. **Bulletin Board**(BB). The adversary $\mathcal{A}$ submits a bulletin board BB.

3. **Tally**. The challenger runs $\mathsf{Tally}(BB, sk)$ to obtain a result r and a correct tabulation proof Π.

4. **Output**. The output of the game is a bit γ^V which is defined as 1 if $r \neq \rho(\mathsf{Extract}(BB, sk)$, where $\mathsf{Extract}$ is applied on the bulletin board by applying it to each individual vote.

A voting protocol as defined in Section 6.3.2 has strong consistency with respect to a counting function ρ and an extract algorithm $\mathsf{Extract}$ if the following conditions are satisfied:

(i) For any (pk, sk) in the image of Setup, for any voter identity $\mathtt{id}$, for any correctly generated public and secret UCIV information $p_{uciv}^{\mathtt{id}}, s_{uciv}^{\mathtt{id}}$ and for any $v \in V$ it is satisfied that $\mathsf{Extract}(\mathsf{Vote}(\mathtt{id}, v, \sigma_v(s_{uciv}^{\mathtt{id}}), p_{uciv}^{\mathtt{id}}, pk), sk) = v$.

(ii) for any probabilistic polynomial time (p.p.t.) adversary $\mathcal{A}$ and any set of voting options V, the following advantage is negligible in the security parameter λ

$$\mathsf{Adv}_V^{\mathbf{const}}(\mathcal{A}) := \Pr[\gamma^V = 1]$$

Strong Correctness

The strong correctness property requires that the votes of honest voters are considered valid, even with respect to a ballot box created by the adversary. This property is used to prevent a dishonest voter from preventing honest voters to vote.

According to the definition in [23], a voting protocol has strong correctness if, considering an adversary that receives as input pk generated by Setup, the following probability

$$\Pr\left[(\mathtt{id}, v, \mathsf{BB}) = \mathcal{A}(pk);\ (p^{\mathtt{id}}_{uciv}, s^{\mathtt{id}}_{uciv}) = \mathsf{Register}(\mathtt{id}, V, pk); \right.$$
$$\left. b = \mathsf{Vote}(\mathtt{id}, v, \sigma_v(s^{\mathtt{id}}_{uciv}), p^{\mathtt{id}}_{uciv}, pk)\ :\ \mathsf{ProcessBallot}(\mathsf{BB}, b, \mathtt{id}) = 0)\right].$$

is negligible.

Universal cast-as-intended verifiability

Intuitively, a voting system satisfies the cast-as-intended property if a corrupt voting device is not able to cast a ballot for a voting option different to the one chosen by the voter. This should hold as long as the voter is honest. If the voter is malicious no guarantees can be given besides the fact that the ballot must correspond to at most one voting option (i.e., it could also correspond to an invalid voting option which will not be counted). We define cast-as-intended on a per-ballot basis, not considering the tallying phase inside the definition.

Universal cast-as-intended verifiability is defined as an experiment between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. In this experiment, the adversary may corrupt registrars, voters or voting devices. The goal of the adversary is to cast ballots on behalf of a non-corrupt voter so that the ballot does not extract to the voting option chosen by the voter. Here, the *extract* algorithm is required to be one with which the voting scheme is strongly consistent. The experiment is parametrized by the set of voting options V and an algorithm $\mathsf{Extract}(b, sk)$ such that, given a ballot and the election private key, returns a vote or $\perp$ denoting an invalid vote.

1. **Setup phase**. The challenger sets up an empty bulletin board BB and runs the $\mathsf{Setup}(1^\lambda)$ protocol to obtain the election public key pk and the election private key sk, posts pk on the board and gives (pk, sk) to the adversary. The adversary is given read access to BB. In addition, $\mathcal{C}$ initializes three empty lists $\mathsf{ID}_R, \mathsf{ID}_P, \mathsf{ID}_F$ such that $\mathsf{ID} = \mathsf{ID}_P \cup \mathsf{ID}_F$
2. **Registration phase**. The adversary may make one type of query.
   - **Register**$(\mathtt{id}, rid)$ query. The adversary provides a voter identity and a registrar identity such that $\mathtt{id} \notin \mathsf{ID}_R$. The challenger runs $\mathsf{Register}(\mathtt{id}, V, pk)$ to generate the public UCIV information $p^{\mathtt{id}}_{uciv}$ and the secret UCIV information $s^{\mathtt{id}}_{uciv}$. $\mathcal{C}$ provides $p^{\mathtt{id}}_{uciv}$ to $\mathcal{A}$ and publishes it on the bulletin board. $\mathtt{id}$ is added to ID_R.
3. **Voting phase**. The adversary may make two types of queries.

- **CorruptVotingDevice**($\mathtt{id}, v_{\mathtt{id}}$) queries. The adversary provides a voter identity $\mathtt{id} \notin \mathsf{ID}$ such that $\mathtt{id} \in \mathsf{ID}_R$ and a voting option $v_{\mathtt{id}}$ corresponding to such identity. Then, $\mathcal{C}$ provides $\sigma_{v_{\mathtt{id}}}(s_{uciv}^{\mathtt{id}})$ to $\mathcal{A}$. The challenger adds $\mathtt{id}$ to ID_P.
- **CorruptVoter**($\mathtt{id}$) queries. The adversary provides a voter identity $\mathtt{id} \notin \mathsf{ID}$ such that $\mathtt{id} \in \mathsf{ID}_R$. Then, $\mathcal{C}$ provides $s_{uciv}^{\mathtt{id}}$ to $\mathcal{A}$. The challenger adds $\mathtt{id}$ to ID_F.

4. **Output**. The adversary submits a pair $(b^*, \mathtt{id}^*)$. The output of the experiment is a bit δ^V, which is defined as 1 if (i) $\mathtt{id}^* \in \mathsf{ID}_P$, (ii) $\mathsf{ProcessBallot}(BB, b^*, \mathtt{id}^*) = 1$ and (iii) $\mathsf{Extract}(b^*, sk) \neq v_{\mathtt{id}^*}$, where $v_{\mathtt{id}^*}$ is the voting option submitted by the adversary in the PartialCorruptVoter query for $\mathtt{id}^*$. δ^V, is defined as 0 in any other case.

A voting protocol as defined in Section 6.3.2 has universal cast-as-intended verifiability with respect to a counting function ρ if there exists an algorithm $\mathsf{Extract}$ such that the following two conditions hold:

(i) for all sets of voting options V the voting protocol is strongly consistent with respect to $\rho, \mathsf{Extract}$.

(ii) for any probabilistic polynomial time (p.p.t.) adversary $\mathcal{A}$ and any set of voting options V, the following advantage is negligible as a function of λ.

$$\mathsf{Adv}_V^{\mathbf{uciv}}(\mathcal{A}) := \Pr[\delta^V = 1]$$

It has to be remarked that the universal cast-as-intended verifiability property does not rely on the secrecy of the private election key.

6.4 Protocol based on NIZK proofs

This is the first proposal for a protocol providing the universal cast-as-intended verifiability property. The voting device generates NIZK proofs that the content of the vote matches the voter's selections. These proofs will prove either that a voting option is in the set of encrypted options in the ballot, or that the voting device knows the corresponding secret provided by the voter. The description of the protocol can also be found in [52].

6.4.1 Protocol description

Primitives:

The voting protocol uses an homomorphic one-way function F. A one-way function is a function which is easy to compute but very difficult to invert. More formally, a function $F : X \to Y$ between two finite sets is said to be one-way if the following two properties are satisfied, where $k = \log |X|$:

(i) For each $x \in X$, the value $F(x)$ can be computed in time polynomial in k.

(ii) For any algorithm A running in time polynomial in k, and for $x \in X$ chosen with the uniform distribution, the probability that A, on input $F(x)$, outputs x is negligible

In an homomorphic one-way function between two groups $(X, +)$ and $(Y, \cdot)$, it is fulfilled that $F(x_1 + x_2) = F(x_1) \cdot F(x_2)$.

Additionally, the voting protocol uses an encryption scheme with algorithms ($\mathsf{Gen_e}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EncVerify}$), and a NIZKPK scheme ($\mathsf{GenCRS}, \mathsf{NIZKProve}, \mathsf{NIZKVerify}, \mathsf{NIZKSimulate}$), which proves the following:

- Consider c the output of Enc, r the randomness used in the encryption, and m the message to be encrypted. Then the relation $\mathcal{R}_{enc} = \{((c, m), r) | c = \mathsf{Enc}_r(m, pk_\mathsf{e})\}$ consists on the sets of ciphertexts, messages and randomness such that the ciphertext is the result of encrypting the message under a public key with that specific randomness. A ZK proof for this relation proves knowledge of the randomness used for such encryption.

- Consider then the one-way function f. Then the relation $\mathcal{R}_{owf} = \{(y, x) | y = f(x)\}$ consists of the pairs (image, pre-image) of such function. A ZK proof for such relation proves knowledge of the pre-image x, given the image y.

The following kind of relations are proven in our protocol:

$$\mathcal{R}_{OR} = \{((c, m), y, r, x) | (c = \mathsf{Enc}_r(m, pk_\mathsf{e})) \vee (y = f(x))\}$$

Specifically, a proof of the following type is generated for each possible message that can be encrypted in c. That is, each voting option in the election. The voter provides the pre-images corresponding to all the options, except to the one she selected. Therefore, if all the proves verify successfully, the statement $c = \mathsf{Enc}_r(m)$ will be true for the case when m matches the voter's selection, and the statement $y = f(x)$ will be true for the other cases. The complete relation is:

$$\begin{aligned}\mathcal{R}_{vote} = \{(c, (v_1, v_2, \ldots, v_n), (y_1, y_2, \ldots, y_n), r, (x_1, x_2, \ldots, x_n)) |\\ (c = \mathsf{Enc}_r(v_1, pk_\mathsf{e})) \vee (y_1 = f(x_1)) \wedge\\ (c = \mathsf{Enc}_r(v_2, pk_\mathsf{e})) \vee (y_2 = f(x_2)) \wedge\\ \ldots\\ (c = \mathsf{Enc}_r(v_n, pk_\mathsf{e})) \vee (y_n = f(x_n))\}\end{aligned}$$

Algorithms:

For defining the protocol, we consider that the counting function ρ is defined as the multiset function [23], and that the set of k voting options V are from the message space of the encryption scheme. Finally, an election where the voter can select 1 out of k options is considered. In [52], the authors explain how the protocol can be adapted in elections where more than one option can be selected.

In this protocol, the secret UCIV information provided to a voter is a set of pre-images $(x_1^{\mathtt{id}}, \ldots, x_k^{\mathtt{id}})$ of F, one for each voting option in V, and the public UCIV information is $(y_1^{\mathtt{id}}, \ldots, y_k^{\mathtt{id}}) = (F(x_1^{\mathtt{id}}), \ldots, F(x_k^{\mathtt{id}}))$, which are the images of each element in the secret UCIV information. Each pair $(x_i^{\mathtt{id}}, y_i^{\mathtt{id}})$ is related to the corresponding voting option v_i, and the relation between $y_i^{\mathtt{id}}$ and v_i is public. The function σ_{v_i} has as input the set of all pre-images $(x_1^{\mathtt{id}}, \ldots, x_k^{\mathtt{id}})$ and outputs the same values except for $x_i^{\mathtt{id}}$.

The protocol consists of the following algorithms:

$\mathsf{Setup}(1^\lambda)$ runs $\mathsf{Gen_e}$ to generate the encryption key pair $(pk_\mathsf{e}, sk_\mathsf{e})$ and GenCRS to generate the common reference string crs, and sets $pk = (pk_\mathsf{e}, \mathsf{crs})$, $sk = sk_\mathsf{e}$.

$\mathsf{Register}(\mathtt{id}, V, pk)$ selects, for each voting option $v_i \in V$, a random value $x_i^{\mathtt{id}} \in X$ and computes $y_i^{\mathtt{id}} = F(x_i^{\mathtt{id}})$. The secret UCIV information $s_{uciv}^{\mathtt{id}}$ is set to be $\{x_i^{\mathtt{id}} - v_i\}_{i=1}^k$, and the public UCIV information is the set of images of each element of the secret UCIV information: $p_{uciv}^{\mathtt{id}} = \{y_i^{\mathtt{id}} - v_i\}_{i=1}^k$.

$\mathsf{Vote}(\mathtt{id}, v_j, \sigma_v(s_{uciv}^{\mathtt{id}}), p_{uciv}^{\mathtt{id}}, pk)$ parses $\sigma_v(s_{uciv}^{\mathtt{id}})$ as the values $(x_1, \ldots, x_{j-1}, \perp, x_{j+1}, \ldots, x_k)$, parses pk as $(pk_\mathsf{e}, \mathsf{crs})$ and computes the encryption of the voter selection as $c \leftarrow \mathsf{Enc}_r(v_j, pk_\mathsf{e})$. Then it runs $\mathsf{NIZKProve}(\mathsf{crs}, (c, V, p_{uciv}^{\mathtt{id}}), (r, (x_1, \ldots, x_{j-1}, \perp, x_{j+1}, \ldots, x_k)))$ and obtains the proof π. The ballot b is set to be (c, π).

$\mathsf{ProcessBallot}(\mathsf{BB}, b, \mathtt{id})$ parses the ballot b as (c, π) and checks whether a ballot for that $\mathtt{id}$, or another ballot (c', π') with $c' = c$, already exists . If any of such cases is found, the algorithm halts and returns 0. Otherwise, the public UCIV information corresponding to that $\mathtt{id}$, $p_{uciv}^{\mathtt{id}}$, is recovered from the bulletin board, $\mathsf{NIZKVerify}(\mathsf{crs}, (c, V, p_{uciv}^{\mathtt{id}}), \pi)$ is run, and the result is provided as output.

$\mathsf{Tally}(\mathsf{BB}, sk)$ executes $\mathsf{ProcessBallot}$ for all the pairs $(b, \mathtt{id})$ in the bulletin board. Then, each individual ballot b is decrypted $\tilde{v} = \mathsf{Dec}(C, sk)$ and ρ is applied to the resulting decryptions $\{\tilde{v}\}$. The output of ρ is defined as the result and the proof of correct tabulation is defined to be the empty string ε.

6.4.2 Security analysis

This instantiation has been proven in [52] to provide *ballot privacy*, *strong consistency* and *universal cast-as-intended verifiability* under the following premises:

- $(\mathsf{Gen_e}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EncVerify})$ is a NM-CPA secure encryption scheme, and $(\mathsf{GenCRS}, \mathsf{NIZKProve}, \mathsf{NIZKVerify}, \mathsf{NIZKSimulate})$ is a NIZKPK scheme which has the property of zero-knowledge, which implies *witness indistinguishability* [54], and the soundness property.

- ρ is the counting function which outputs its inputs randomly permuted, filtering invalid votes (voting options v_x such that $v_x \notin V$).

- F is a homomorphic one-way function
- The number of voting options is polynomial in the security parameter. This is necessary to provide a tight security reduction in the analysis of universal cast-as-intended, when the attacker does not target a specific voter and voting option during registration, but does it during the voting phase.

6.4.3 Implementation

The protocol can be instantiated using the Signed ElGamal encryption scheme over finite fields. We use the exponentiation function $F(x) = g^x$, where $x \in \mathbb{Z}_q$ and g is the generator of the same cyclic group $\mathbb{G}$ defined for ElGamal, as a one-way homomorphic function. The NIZKPK scheme is constructed following the Fiat-Shamir heuristic for non-interactive proofs and the techniques in [42] for construction OR-proofs:

Consider $c = (c_1, c_2, c_3, c_4)$ to be the output of $\mathsf{Enc}_r(m, pk_\mathsf{e})$, where $c_1 = g^r, c_2 = h^r \cdot m$ and c_3, c_4 are the Schnorr signature over the ciphertext components c_1, c_2. Then the algorithms $(\mathsf{GenCRS}, \mathsf{NIZKProve}, \mathsf{NIZKVerify})$ are defined as follows:

GenCRS outputs $\mathsf{crs} = (\mathbb{G}, g)$

$\mathsf{NIZKProve}$ receives crs, the statement $(c, (v_1, v_2, \ldots, v_k), (y_1, y_2, \ldots, y_k))$ for which $c \leftarrow \mathsf{Enc}_r(v_j, pk_\mathsf{e})$ and $c = (c_1, c_2, c_3, c_4)$; and the witness $(r, (x_1, x_2, \ldots, x_k))$ for which $x_j = \perp$. Then it computes:

1. $\forall i = 1 \ldots k$, $i \neq j$, the simulation of the $c = \mathsf{Enc}_r(v_i, pk_\mathsf{e})$ statement: chooses $e_1^{(i)}, z_1^{(i)}$ at random from $\mathbb{Z}_q$ and computes $a_1^{(i)} = g^{z_1^{(i)}} \cdot c_1^{-e_1^{(i)}}$, $b_1^{(i)} = pk_\mathsf{e}^{z_1^{(i)}} \cdot (c_2/v_i)^{-e_1^{(i)}}$.
2. For the index j, the simulation of the $y_j = f(x_j)$ statement: chooses $e_2^{(j)}, z_2^{(j)}$ at random from $\mathbb{Z}_q$ and computes $a_2^{(j)} = g^{z_2^{(j)}} \cdot y_j^{-e_2^{(j)}}$.
3. $\forall i = 1 \ldots k$, $i \neq j$, the first part of the proof for the $y_i = f(x_i)$ statement: chooses $\beta^{(i)} \in \mathbb{Z}_q$ at random and computes $a_2^{(i)} = g^{\beta^{(i)}}$.
4. For the index j, the first part of the proof for the $c = \mathsf{Enc}_r(v_j, pk_\mathsf{e})$ statement: chooses $\alpha^{(j)} \in \mathbb{Z}_q$ at random and computes $a_1^{(j)} = g^{\alpha^{(j)}}$ and $b_1^{(j)} = pk_\mathsf{e}^{\alpha^{(j)}}$.
5. Then computes the hash value $h = H((c_1, c_2, c_3, c_4), \{a_1^{(i)}, b_1^{(i)}, a_2^{(i)}\}_{i=1}^k)$.
6. Finally, $\forall i$, $i \neq j$, computes $e_2^{(i)} = h - e_1^{(i)} \bmod q$, $z_2^{(i)} = \beta^{(i)} + e_2^{(i)} \cdot x_i$, and for $i = j$ computes $e_1^{(j)} = h - e_2^{(j)} \bmod q$, $z_1^{(j)} = \alpha^{(j)} + e_1^{(j)} \cdot r$.

The proof result is $\pi = (h, \{e_1^{(i)}, z_1^{(i)}, z_2^{(i)}\}_{i=1}^k)$

$\mathsf{NIZKVerify}$ receives crs, the statement $(c, (v_1, v_2, \ldots, v_k), (y_1, y_2, \ldots, y_k))$ and the proof π, and does the following for $i = 1, \ldots, k$:

1. Computes $e_2^{(i)} = h - e_1^{(i)} \mod q$.
2. Computes $A_1^{(i)} = c_1^{-e_1^{(i)}} \cdot g^{z_1^{(i)}}$, $B_1^{(i)} = pk_{\mathtt{e}}^{z_1^{(i)}} \cdot (c_2/v_i)^{-e_1^{(i)}}$, $A_2^{(i)} = y^{-e_2^{(i)}} \cdot g^{z_2^{(i)}}$.

Finally, it checks that $H((c_1, c_2, c_3, c_4), \{A_1^{(i)}, B_1^{(i)}, A_2^{(i)}\}_{i=1}^{k}) = h$.

6.5 UCIV with return codes

In this section we propose another approach for achieving the universal cast-as-intended verifiability property, inspired by the mechanisms of return codes and for confirming a vote of the system used in Neuchâtel, described in Chapter 4. This proposal has different privacy requirements, since an adversary knowing a voter's private UCIV information could break the voter privacy. On the other hand, the computation costs at the voting device are reduced, since they are proportional to the number of options the voter selects, and not to the total in the election, as in the previous approach.

6.5.1 Overview of the solution

As we have explained in the overview, in this approach we make a twist on the return code mechanism. Instead of receiving return codes, the voter has to enter them into the voting device, as in code voting proposals: indeed, the voter, besides her selections, has to enter the return codes in her verification card related to them. A reference value has been generated during registration for each combination of voting option - return code. This reference value can be used to universally verify that the combination of the content of the ballot generated by the voting device, with the return code provided by the voter, is a valid one (which happens only if the pair matches). Given that the contents of the verification card are unknown to the voting device, it cannot guess a return code such that its combination with a ballot, containing a voting option different than the one selected by the voter, is considered valid according to the published reference values.

The voter privacy is preserved by maintaining the [return code - voting option] and [reference value - voting option] links in secret. In our description, we will assume that the information generated during registration is correct, and that secret values are indeed kept as secret values. We will discuss how to reduce these assumptions in Section 6.6. Finally, we first describe the system for a single voting scenario, and then explain how it could be extended to multiple voting in Section 6.5.5.

6.5.2 Protocol description

In this protocol, the secret UCIV information for a voter is a set of return codes randomly chosen from the return code space RC linked to voting options, $\{v_i - \mathtt{RC}_i^{\mathtt{id}}\}_{i=1}^{k}$, as well as a voter private key $sk_{\mathtt{id}}$. The public UCIV information is a set of signatures, each one computed over the combination of the partial return code

and the return code corresponding to one voting option: $\{\psi_i\}_{i=1}^{k}$. The function σ_{v_j} outputs the return code $\mathtt{RC}_j^{\mathtt{id}}$ corresponding to the voting option v_j, and the voter private key $sk_{\mathtt{id}}$.

The system uses an encryption scheme ($\mathsf{Gen_e}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EncVerify}$), a digital signature scheme ($\mathsf{Gen_s}, \mathsf{Sign}, \mathsf{SignVerify}$), and a NIZKPK scheme ($\mathsf{ProveEq}, \mathsf{VerifyEq}, \mathsf{SimEq}$). We define also a generic operation $\oplus$, and use a keyed pseudo-random function $w_k()$ with homomorphic properties. Additionally, an aggregation function ϕ which takes advantage of the homomorphic properties of $w_k()$ and a hash function H are also used in the protocol.

The following implementation is proposed for the algorithms presented in Section 6.3.2 :

> $\mathsf{Setup}(1^\lambda)$ runs $\mathsf{Gen_e}$ to generate a key pair $(pk_{\mathtt{e}}, sk_{\mathtt{e}})$, which is set to be the election public and private keys, pk and sk, respectively. Alternatively, sk may consist of the shares of $sk_{\mathtt{e}}$ if there are several trustees.
>
> $\mathsf{Register}(\mathtt{id}, V, pk)$ runs $\mathsf{Gen_s}$ to create a signing key pair $(pk_{\mathtt{s}}, sk_{\mathtt{s}})$. For each voting option $v_i \in V$, it chooses a return code at random from RC. Then it runs $\mathsf{Gen_e}$ to generate a voter keypair $(pk_{\mathtt{id}}, sk_{\mathtt{id}})$, and computes a partial return code for each voting option v_i as $\mathtt{pRC}_i^{\mathtt{id}} = \mathsf{w}_{sk_{\mathtt{id}}}(v_i)$. $\mathsf{Sign}(H(\mathtt{RC}_i^{\mathtt{id}} \oplus \mathtt{pRC}_i^{\mathtt{id}}), sk_{\mathtt{s}})$ is run for each pair return code - partial return code corresponding to each voting option, obtaining the signature ψ_i. The resulting set $\{H(\mathtt{RC}_i^{\mathtt{id}} \oplus \mathtt{pRC}_i^{\mathtt{id}}), \psi_i\}_{i=1}^{k}$ is ordered alphabetically and, together with the signing public key $pk_{\mathtt{s}}$ and the voter public key $pk_{\mathtt{id}}$, is set to be the public UCIV information $p_{uciv}^{\mathtt{id}}$. The set of pairs $\{v_i - \mathtt{RC}_i^{\mathtt{id}}\}_{i=1}^{k}$, together with the voter's private key $sk_{\mathtt{id}}$ is set to be the private UCIV information $s_{uciv}^{\mathtt{id}}$. The private key $sk_{\mathtt{s}}$ has to be *forgotten* at this point.
>
> $\mathsf{Vote}(\mathtt{id}, v, \sigma_v(s_{uciv}^{\mathtt{id}}), p_{uciv}^{\mathtt{id}}, pk)$ is very similar to the $\mathsf{CreateVote}$ algorithm from Neuchâtel's system, described in Section 4.4: it parses v as the voter's selections $\{v_{j_1}, \ldots, v_{j_t}\}$, sets $v_{tot} \leftarrow \phi(\{v_{j_1}, \ldots, v_{j_t}\})$, and runs Enc from the encryption scheme, using the election public key pk and v_{tot} to get the ciphertext c. The algorithm then parses $\sigma_v(s_{uciv}^{\mathtt{id}})$ as the set of return codes corresponding to the entered voting options $\{\mathtt{RC}_{j_1}^{\mathtt{id}}, \ldots, \mathtt{RC}_{j_t}^{\mathtt{id}}\}$, and the voter private key $sk_{\mathtt{id}}$. The partial return codes are computed as $\{\mathtt{pRC}_{j_l}^{\mathtt{id}}\}_{l=1}^{t} \leftarrow (\mathsf{w}_{sk_{\mathtt{id}}}(v_{j_1}), \ldots, \mathsf{w}_{sk_{\mathtt{id}}}(v_{j_t}))$. The value $\mathsf{w}_{sk_{\mathtt{id}}}(c)$ and the proofs π_1, π_2 are computed as in Neuchâtel's protocol:

- $\pi_1 = \mathsf{ProveEq}(g, c, pk_{\mathtt{id}}, \mathsf{w}_{sk_{\mathtt{id}}}(c), sk_{\mathtt{id}})$ proves that $\mathsf{w}_{sk_{\mathtt{id}}}(c)$ is computed with the private key $sk_{\mathtt{id}}$ corresponding to the public key $pk_{\mathtt{id}}$.
- $\pi_2 = \mathsf{ProveEq}(g, pk, \mathsf{w}_{sk_{\mathtt{id}}}(c)/\phi(\mathtt{pRC}_{j_1}^{\mathtt{id}}, \ldots, \mathtt{pRC}_{j_t}^{\mathtt{id}}), r \cdot sk_{\mathtt{id}})$ proves that the value $\mathsf{w}_{sk_{\mathtt{id}}}(c)$ is equivalent to the encryption of the aggregation ϕ of partial return codes $\{\mathtt{pRC}_{j_l}^{\mathtt{id}}\}_{l=1}^{t}$, under the election public key pk (note that r denotes the encryption randomness used to compute c).

The ballot b, composed by $c, \{\mathtt{pRC}^{\mathtt{id}}_{jl}\}^{t}_{l=1}, \mathsf{w}_{sk_{\mathtt{id}}}(c), \pi_1, \pi_2$ and $\{\mathtt{RC}^{\mathtt{id}}_{jl}\}^{t}_{l=1}$ is returned.

$\mathsf{ProcessBallot}(\mathsf{BB}, b, \mathtt{id})$ checks if there already exists a ballot for the identity $\mathtt{id}$ in the ballot box, or that there is not another vote in BB with the same ciphertext value c. If so, it stops and outputs 0. Otherwise, it runs $\mathsf{VerifyEq}$ to validate the NIZKPK proofs π_1, π_2 from the ballot b, using the voter's public key $pk_{\mathtt{id}}$, and runs $\mathsf{EncVerify}$ to check that the ciphertext c is correctly formed. Finally, it computes $\zeta_l = \mathtt{RC}^{\mathtt{id}}_{jl} \oplus \mathtt{pRC}^{\mathtt{id}}_{jl}$ for $l = 1, \ldots, t$ and checks that all $H(\zeta_l)$ values match with a signature in $p^{\mathtt{id}}_{uciv}$, running $\mathsf{SignVerify}$ with the signing public key $pk_{\mathtt{s}}$. In case all the validations are successful, 1 is returned. Otherwise, the algorithm returns 0.

$\mathsf{Tally}(\mathsf{BB}, sk)$ executes $\mathsf{ProcessBallot}$ for all the pairs $(b, \mathtt{id})$ in the bulletin board. Then, each individual ballot b which passed the previous verification is decrypted: $\tilde{v}_{tot} = \mathsf{Dec}(c, sk)$. $\phi^{-1}(\tilde{v}_{tot})$ outputs the individual voting options v_i composing $\tilde{v}_{tot}$, for which it is tested that $v_i \in V$. Otherwise, the whole set of voting options is discarded. The result r, composed of the values v_i recovered from each vote is provided as the output, and the proof of correct tabulation is defined to be the empty string ε.

6.5.3 Security Analysis

In this section we show that the protocol presented in the previous section satisfies the properties of ballot privacy, strong consistency, strong correctness and universal cast-as-intended verifiability, according to the definitions in Section 6.3.3. The analysis is similar to that provided for the Neuchâtel system in Section 4.5. We use again the $\mathsf{Enc2Vote}$ scheme for making a security reduction in the case of privacy.

Ballot Privacy

Given that the universal cast-as-intended verification mechanism is not based in a witness-indistinguishable NIZKPK, the level of privacy provided by the return code variant is different: in fact, we have to restrict the access of the adversary to the secrect UCIV information. We therefore use a slightly different definition of ballot privacy than that provided in Section 6.3.3: the original experiment for ballot privacy, in which $\mathcal{A}$ had to distinguish between two experiments parameterized by $\beta = (0, 1)$, allowed the adversary $\mathcal{A}$ to access all the registration information, for both honest and corrupt voters. We modify this definition in order to allow $\mathcal{A}$ to have access to such registration information only for corrupt voters, and restrict the Vote oracle to be used only once per honest voter. The experiments are parametrized by the set of voting options V and an algorithm $\mathsf{SimProof}(BB, r)$ such that, given a bulletin board and a result, it simulates a proof of correct tabulation.

1. **Setup phase**. The challenger sets up two empty bulletin boards L and R. It runs the $\mathsf{Setup}(1^\lambda)$ protocol to obtain the election public key pk and the election private key sk. It then posts pk on both bulletin boards. The adversary is

given read access to either L if $\beta = 0$ or R if $\beta = 1$. In addition, $\mathcal{C}$ initializes the empty lists ID, ID_c, ID_h.

2. **Registration phase**. The adversary may make one type of query.

   - **Register**($\mathtt{id}$) query. The adversary provides a voter identity such that $\mathtt{id} \notin \mathsf{ID}$. The challenger runs $\mathsf{Register}$ on inputs $(\mathtt{id}, V, pk)$ to generate the public UCIV information $p^{\mathtt{id}}_{uciv}$ and the secret UCIV information $s^{\mathtt{id}}_{uciv}$. $\mathcal{C}$ keeps $s^{\mathtt{id}}_{uciv}$ and publishes $p^{\mathtt{id}}_{uciv}$ on both bulletin boards. The identity $\mathtt{id}$ is added to ID.

3. **Voting phase**. The adversary may make two types of queries.

   - **Vote**($\mathtt{id}, v_L, v_R$) queries. The adversary provides a voter identity $\mathtt{id}$ such that $\mathtt{id} \in \mathsf{ID}$, $\mathtt{id} \notin \mathsf{ID}_c$, $\mathtt{id} \notin \mathsf{ID}_h$ and two votes $v_L, v_R \in V$. The challenger runs $\mathsf{Vote}(\mathtt{id}, v_L, \sigma_{v_L}(s^{\mathtt{id}}_{uciv}), p^{\mathtt{id}}_{uciv}, pk)$, which outputs b_L, and $\mathsf{Vote}(\mathtt{id}, v_R, \sigma_{v_R}(s^{\mathtt{id}}_{uciv}), p^{\mathtt{id}}_{uciv}, pk)$, which outputs b_R. $\mathcal{C}$ then obtains new versions of the boards L and R by running $\mathsf{ProcessBallot}(L, b_L, \mathtt{id})$ and $\mathsf{ProcessBallot}(R, b_R, \mathtt{id})$ and updating the boards accordingly. Finally it adds $\mathtt{id}$ to ID_h.
   - $\mathcal{O}$**getVotingData**($\mathtt{id}$): $\mathcal{A}$ provides an identity $\mathtt{id} \in \mathsf{ID}$, $\mathtt{id} \notin \mathsf{ID}_h$. $\mathcal{C}$ provides $s^{\mathtt{id}}_{uciv}$ to $\mathcal{A}$ and adds $\mathtt{id}$ to ID_c.
   - **Ballot**($b, \mathtt{id}$) queries. These are queries made on behalf of corrupt voters. Here the adversary provides a ballot b and an identity $\mathtt{id}$ such that $\mathtt{id} \in \mathsf{ID}$. The challenger runs $\mathsf{ProcessBallot}(L, b, \mathtt{id})$ and if the process accepts it also runs $\mathsf{ProcessBallot}(R, b, \mathtt{id})$ and updates the boards accordingly.

4. **Tallying phase**. The challenger evaluates $\mathsf{Tally}(L, sk)$, obtaining the result r and the proof of correct tabulation Π. If $\beta = 0$, the challenger posts (r, Π) on the bulletin board L. If $\beta = 1$, the challenger runs $\mathsf{SimProof}(R, r)$, obtaining a simulated proof Π', and posts (r, Π') on the bulletin board R.

5. **Output**. The adversary $\mathcal{A}$ outputs a bit $\alpha^{\mathcal{A},V}$.

A voting protocol as defined in Section 6.3.2 provides ballot privacy, if there exists an algorithm $\mathsf{SimProof}$ such that for any probabilistic polynomial time (p.p.t.) adversary $\mathcal{A}$ and any set of voting options V, the following advantage is negligible in the security parameter λ.

$$\mathsf{Adv}^{\mathbf{priv}}_V(\mathcal{A}) := |\Pr[\alpha^{\mathcal{A},V} = 1|\beta = 1] - \Pr[\alpha^{\mathcal{A},V} = 1|\beta = 0]|$$

Theorem 6.1. *Let* $(\mathsf{Gen_e}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EncVerify})$ *be an NM-CPA secure encryption scheme and* $(\mathsf{ProveEq}, \mathsf{VerifyEq}, \mathsf{SimEq})$ *be a NIZKPK scheme with the zero-knowledge property. Then the protocol presented in Section 6.5.2 satisfies the ballot privacy property.*

Now we prove that the return code based protocol presented in the previous section provides ballot privacy according to this new definition, through the following steps: in a first step we prove in Lemma 6.1 that the experiments Exp_β and $\mathsf{Exp}_{\beta'}$, where in $\mathsf{Exp}_{\beta'}$ the NIZK proofs inside the ballot b are simulated, instead of honestly generated, are indistinguishable by $\mathcal{A}$. In a second step we prove that this scenario is indistinguishable from one where the partial return codes are generated at random, through Lemma 6.2. Finally, in Lemma 6.3 we make a reduction of the last experiment to the ballot privacy of the Enc2Vote scheme.

Next, let's consider $\mathsf{SimVote}(\mathtt{id}, v, \sigma_v(s^{\mathtt{id}}_{uciv}), p^{\mathtt{id}}_{uciv}, pk)$ be a modification of the algorithm $\mathsf{Vote}(\mathtt{id}, v, \sigma_v(s^{\mathtt{id}}_{uciv}), p^{\mathtt{id}}_{uciv}, pk)$ where, instead of running ProveEq for the generation of the proof π_2, the algorithm SimEq is run to obtain the simulated proof π'_2 with the same distribution than the non-simulated one. Then define $\mathsf{Exp}_{\beta'}$ as the experiment in which the challenger runs SimVote instead of Vote. The following lemma is straightforward to prove:

Lemma 6.1. The experiments Exp_β and $\mathsf{Exp}_{\beta'}$ are computationally indistinguishable for $\beta \in \{0, 1\}$, given a $(\mathsf{ProveEq}, \mathsf{VerifyEq}, \mathsf{SimEq})$ zero-knowledge NIZKPK scheme.

In the following transformation, we define $\mathsf{SimVote}_2(\mathtt{id}, p^{\mathtt{id}}_{uciv}, pk)$ to be a modification of SimVote where, instead of computing the partial return code value as $\mathsf{w}_{sk_{\mathtt{id}}}(v)$, it is selected at random from the same value space. Then we consider the experiment $\mathsf{Exp}_{\beta''}$ where, when the adversary submits the query $\mathbf{Vote}(\mathtt{id}, v_L, v_R)$, the challenger executes $\mathsf{SimVote}_2$ instead of SimVote. We also define an oracle $\mathcal{O}$ which, when $\mathcal{A}$ makes a query to compute $H(\mathtt{RC}^{\mathtt{id}} \oplus \mathtt{pRC}^{\mathtt{id}})$, it returns one of the values in the public UCIV information $p^{\mathtt{id}}_{uciv}$. The following lemma can be easily proven:

Lemma 6.2. The experiments $\mathsf{Exp}_{\beta'}$ and $\mathsf{Exp}_{\beta''}$ are computationally indistinguishable for $\beta \in \{0, 1\}$, given the pseudorandom function $\mathsf{w}()$ and the oracle $\mathcal{O}$.

Next, we proceed to reduce the ballot privacy property of our scheme to the ballot privacy property of the Enc2Vote scheme.

Lemma 6.3. Let $\mathcal{A}$' be a p.p.t. adversary that interacts with a challenger $\mathcal{C}$, such that | Pr[$\mathsf{Exp}_{0''}$ = 1] - Pr[$\mathsf{Exp}_{1''}$ = 1] | is non-negligible. Then, there exists an adversary $\mathcal{A}$" that breaks the ballot privacy property of the Enc2Vote scheme.

Proof. In the reduction, we use $\mathcal{A}$" as the challenger for $\mathcal{A}$', and $\mathcal{A}$" interacts with $\mathcal{C}$ in the same way as in the experiment defined in [25]. The reduction is as follows:

In the **Setup phase**, $\mathcal{C}$ sets up two empty bulletin boards BB_0 and BB_1, runs the $\mathsf{Gen_e}$ algorithm and keeps the sk^{e2v} private key for itself, while it publishes the public key pk^{e2v} key on both bulletin boards. In turn, $\mathcal{A}$" publishes $pk = pk^{e2v}$ on the bulletin board visible by $\mathcal{A}$'.

In the **Registration phase**, when $\mathcal{A}$' makes the **Register**($\texttt{id}$), $\mathcal{A}$" runs the $\mathsf{Register}(\texttt{id}, V, pk)$ algorithm from our protocol, keeps the secret UCIV information $s^{\texttt{id}}_{uciv}$ and publishes the public UCIV information $p^{\texttt{id}}_{uciv}$ on the bulletin board visible by $\mathcal{A}'$.

During the **Voting phase**, when $\mathcal{A}$' submits the **Vote**$(\texttt{id}, v_L, v_R)$ query, $\mathcal{A}$" submits the **Vote** query to $\mathcal{C}$, who responds by publishing a ballot b_{e2v} to the bulletin board visible by $\mathcal{A}$". $\mathcal{A}$" parses b_{e2v} as c, computes $\mathsf{w}_{sk_{\texttt{id}}}(c)$, $\pi_1 = \mathsf{ProveEq}(g, c, pk_{\texttt{id}}, \mathsf{w}_{sk_{\texttt{id}}}(c), sk_{\texttt{id}})$, picks v'_i at random from the $\mathsf{w}()$ output message space and computes $\pi'_2 = \mathsf{SimEq}(g, pk, \mathsf{w}_{sk_{\texttt{id}}}(c)/v'_i)$. Then $\mathcal{A}$" chooses at random one of the return codes $\texttt{RC}^{\texttt{id}}_x$ in $s^{\texttt{id}}_{uciv}$. The resulting ballot $b = (b_{e2v}, v'_i, \mathsf{w}_{sk_{\texttt{id}}}(c), pk_{\texttt{id}}, \pi_1, \pi'_2, \texttt{RC}^{\texttt{id}}_x)$ is published in the bulletin board visible by $\mathcal{A}$', together with the voter identity $\texttt{id}$.

When $\mathcal{A}$' submits the $\mathcal{O}$getVotingData($\texttt{id}$) query, $\mathcal{A}$" just returns the private UCIV information $s^{\texttt{id}}_{uciv}$ to $\mathcal{A}$'. When $\mathcal{A}$' submits the **Ballot**$(b, \texttt{id})$ query, $\mathcal{A}$" parses b as $(c, \texttt{pRC}^{\texttt{id}}_i, \mathsf{w}_{sk_{\texttt{id}}}(c), pk_{\texttt{id}}, \pi_1, \pi_2, \texttt{RC}^{\texttt{id}}_i)$ and submits a **Ballot(**c**)** query to $\mathcal{C}$. Then $\mathcal{A}$" puts $\texttt{id}, b$ in the bulletin board visible by $\mathcal{A}$'.

In the **Tallying phase**, $\mathcal{C}$ posts the result of evaluating $\mathsf{Tally}(\mathsf{BB}_0, sk^{e2v})$ on the bulletin board visible to $\mathcal{A}$". $\mathcal{A}$" in turn publishes the same result on the bulletin board visible by $\mathcal{A}$'. Both tabulation proofs (the real and the simulated) are set to be the empty string ε.

At the end of the experiment, $\mathcal{A}$' outputs a bit and $\mathcal{A}$" outputs the same bit. The outputs of $\mathcal{A}$"as a result of the interaction with $\mathcal{A}$' have the same distribution as in the ballot privacy experiment in [25]. Therefore, the reduction is sound. □

Strong Consistency

The following theorem is straightforward to prove:

Theorem 6.2. *Let ρ be the counting function which outputs its inputs randomly permuted and filtering invalid votes. Let* $\mathsf{Extract}(b, sk) = \mathsf{Dec}(c, sk_{\mathsf{e}})$*. Then the protocol defined in Section 6.5.2 has strong consistency with respect to* $\mathsf{Extract}$*, ρ.*

Strong Correctness

The following theorem is straightforward to prove:

Theorem 6.3. *Let* $(\mathsf{Gen}_{\mathsf{e}}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EncVerify})$ *be a randomized encryption scheme. Then the protocol defined in Section 4.4 has the property of strong correctness.*

In the protocol defined in Section 6.5.2, the only condition for which $\mathsf{ProcessBallot}$ $(\mathsf{BB}, \texttt{id}, b)$ may output 0, given a ballot b produced by an honest registered voter that was not already registered by the adversary, is that a previous entry with the same ciphertext c is already present in the bulletin board. Given that $(\mathsf{Gen}_{\mathsf{e}}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EncVerify})$ is a probabilistic encryption scheme, this probability is negligible in the security parameter λ.

Universal cast-as-intended verifiability

Theorem 6.4. *Let* (ProveEq, VerifyEq, SimEq) *be a sound NIZKPK scheme,* ($\mathsf{Gen_s}$, Sign, SignVerify) *be an unforgeable signature scheme and* H *be a collision-resistant hash function. Then the protocol presented in Section 6.5.2 satisfies the universal cast-as-intended verifiability property.*

According to the definition presented in 6.3.3, the adversary succeeds if it is able to generate a ballot for which ProcessBallot successfully verifies, when the ballot contains a different voting option (the value produced by $\mathsf{Extract}(b, sk)$ is different) than the one claimed. In order to do that, the adversary can follow different strategies:

Let's consider a voting option v_i which is used by $\mathcal{A}$ to do the CorruptVotingDevice query, for which it gets from the challenger the private key $sk_{\mathtt{id}}$ and the return code $\mathtt{RC}_i^{\mathtt{id}}$. Then let's consider that $\mathcal{A}$ provides at the output of the experiment $b^* = \left(c, \mathtt{pRC}_j^{\mathtt{id}}, \mathsf{w}_{sk_{\mathtt{id}}}(c), pk_{\mathtt{id}}, \pi_1, \pi_2, \mathtt{RC}_i^{\mathtt{id}}\right)$, such that $\mathsf{Extract}(b^*, sk)$ outputs $v_j \neq v_i$. Recall that ProcessBallot computes $\zeta = \mathtt{RC}_i^{\mathtt{id}} \oplus \mathtt{pRC}_j^{\mathtt{id}}$ and checks that $H(\zeta)$ matches an entry with a valid signature in $p_{uciv}^{\mathtt{id}}$. Clearly, given the collision resistance of the hash function and the unforgeability of the signature scheme, this strategy suceeds with negligible probability. An alternative approach for $\mathcal{A}$ is to guess the value $\mathtt{RC}_j^{\mathtt{id}}$ such that $H(\zeta_j)$ is a valid entry in $p_{uciv}^{\mathtt{id}}$, for which it has a probability of success of $\frac{1}{q}$, where q is the size of the space RC.

Then let's consider that $\mathcal{A}$ provides at the output of the experiment the ballot $b^* = (c, \mathtt{pRC}_i^{\mathtt{id}}, \mathsf{w}_{sk_{\mathtt{id}}}(c), pk_{\mathtt{id}}, \pi_1, \pi_2, \mathtt{RC}_i^{\mathtt{id}})$, such that $\mathsf{Extract}(b^*, sk)$ outputs $v_j \neq v_i$. Recall that ProcessBallot verifies the proofs π_1, π_2 from the ballot b before accepting. The probability that such proofs are verified successfully, when the voting option v_j encrypted in c does not match the one used to compute the partial return code, is negligible given the soundness property of the NIZKPK scheme.

6.5.4 Implementation

This protocol can be instantiated using the same primitives as in Neuchâtel's system:

- The Signed Elgamal encryption scheme, which is NM-CPA secure.
- The RSA-FDH signature scheme.
- The NIZKPK schemes EqDL for proving equality of discrete logarithms.
- The exponentiation function $w_k(x) = x^k$, which is discussed to be pseudorandom under some assumptions in [60].
- The product operation ϕ and the factorization operation ϕ^{-1}, where the voting options v_i are small primes.
- The concatenation operation $\oplus$.
- The SHA-256 hash function H.

The number of operations to be done at the voting device is the same as in the Neuchâtel scheme: for a *t-out-of-k* scenario where t voting options can be encrypted together in the same ciphertext, the cost of the Signed ElGamal encryption is 3 exponentiations. t more exponentiations are required for computing the partial return codes, while the NIZK proofs and intermediate values cost 7 exponentiations.

In case of a *1-out-of-k* scenario, 11 exponentiations have to be computed by the voting device. The cost of the previous approach in the same scenario was of $3k+4$ exponentiations: $3k+1$ for the NIZK proof for the UCIV property, and 3 for the Signed ElGamal encryption. Therefore, we can see that with this second approach we are improving on the number of operations, since they are constant in the number of options in the election. However, this second approach has drawbacks such as the fact that the knowledge of the UCIV private information has to remain secret in order to keep voter privacy, while it was not required for the NIZKPK variant.

Another important drawback of this second approach is that multiple voting allows to break voter privacy, or at least it lets an adversary to detect when a voter votes twice for the same voting option.

6.5.5 Extension to multiple voting

An alternative for extending this protocol for multiple voting, is that the partial return codes, and the return code the voter sends in her ballot (that is, the deterministic values), are encrypted with a public key corresponding to the bulletin board manager. The output of the bulletin board manager would still have to be restricted. Otherwise an adversary could detect when two ballots cast by the same voter match the same signature on the public UCIV information, for example. However, then the system could not be considered to be universally cast-as-intended verifiable, since it would be restricted to entities holding the bulletin board manager private key. An open topic of research is to investigate the use of signature schemes which are *commutable* with encryption schemes, in order to be able to prove that an encrypted content matches one encrypted signed value from a set, without revealing which one. Techniques presented in [30] could be used for this purpose.

Another approach is to make the voter re-register, and provide her a new set of public and private UCIV values each time she wants to re-vote. Several sets of public and private UCIV values may be provided to the voter when registering, in order to allow, in advance, to vote up to a pre-defined number of times.

6.6 Distributed generation of UCIV information

A key point of the universal cast-as-intended verifiable schemes presented in this chapter is the generation of the public and private UCIV information. A leak of private UCIV information in the NIZK proofs approach may result in the voting device being able to deceive the cast-as-intended verification. Such leak in the return codes approach additionally results on a lost of privacy for the voter. In both

cases, not well formed UCIV public and private data results in a deception of the cast-as-intended verification. Therefore, it is very important to provide mechanisms for distributing the task of generating such UCIV information, in order to maintain the properties of the scheme in front of a subset of malicious registrars.

In the return code approach, measures for distributed verification card generation, similar to those described for the Neuchâtel's system in Section 4.8.2, can be applied. A set of registrars generates in a distributed way the partial return codes corresponding to the voting options in the election: each one (denoted by the index m) computes a part of the voter's secret key $sk_{\texttt{id}}(m)$ and of the partial return code associated to a voting option, $v_i - \texttt{pRC}_i^{\texttt{id}}(m)$. The set of possible return codes in each verification card is shuffled with a random permutation, using a verifiable mixnet, and the output return codes are assigned to voting options: $\{v_i - \texttt{RC}_i^{\texttt{id}}\}_{i=1}^{k}$. This operation can be done by the printing service, since at the end it is unavoidable that it sees the correspondence between return code and voting option, to print the contents of the verification card. The printing service therefore can be also in charge of receiving the *shares* of the voter private key and of partial return codes from the multiple registrars, putting them together, and computing the UCIV public values as the signature of each relation $\texttt{RC}_i^{\texttt{id}} \oplus \texttt{pRC}_i^{\texttt{id}}$ assigned to the same voting option v_i.

The operations of the printing service can be audited by randomly selecting some verification cards to audit, and checking that an honestly formed ballot, with the verification card parameters, is successfully verified according to the signatures in the UCIV public values.

In order to avoid a single point of failure, multiple printing services can be used, each one generating its own mapping $(\texttt{RC}_i^{\texttt{id}} \oplus \texttt{pRC}_i^{\texttt{id}})_j$ and signing this relation with a different private key (owned by each printing service). The voter then receives as many verification cards as printing services, and she is required to enter all the return codes assigned to the voting option v_i she has selected into the voting device. The ProcessBallot algorithm will check that each of the multiple combinations of partial return code and return code in the ballot matches one of the signatures published by each printing service.

In the NIZK proofs approach, authors in [52] propose that the voter is able to select the registrars with which she wants to register. Each registrar `rid` selected by the voter provides to her a set of private UCIV information $s_{uciv}^{\texttt{id,rid}}$ and a set of public UCIV information $p_{uciv}^{\texttt{id,rid}}$. The voting device puts together the private UCIV information, related to the voter's selected voting option, from all the registrars the voter has registered with, to compute the NIZKPK proofs (a more detailed description is provided in [52]). One honest registrar, in the set of registrars selected by the voter, is enough for ensuring that the private UCIV information is kept secret and that the cast-as-intended verification mechanism is not deceived by the voting device.

6.7 Setup

The setups of both schemes (the one based on NIZK proofs, and the one based on return codes) are very similar. Voters receive a set of codes prior to the voting phase, each one related to a voting option in the election. In the return code-based scheme they additionally receive a private key. In a remote voting scenario, voters may receive these values by postal mail in a sealed paper card. In a poll-site voting scenario, the paper card may be printed on-demand after the voter is successfully identified by the poll worker.

During the voting phase, the voter makes her selections on her voting device and enters the codes in the card, corresponding to the selected voting options. In the return code approach, the voter additionally enters her private key, also printed in the card.

For usability purposes, the codes in the voting card may be shorter that what is required by the security parameter for the UCIV secret values. These codes can be used to recover the indicated UCIV secret values, which may be downloaded to the voting device in a password-protected key container (for example, a PKCS#12 keystore [69]), or provided in a hardware token (this option is more suitable for poll-site voting scenarios).

The ballot is generated by the voting device and sent to the bulletin board manager, where it is verified prior to being published on the bulletin board. Auditors can be connected to the bulletin board manager to additionally verify the incoming ballots. They may provide their approval by signing the ballot and posting their signatures also on the bulletin board.

During the counting phase, only votes which have been successfully verified are taken into account for the tally. Although the description of a verifiable counting phase has been left from the formal description of the algorithms and properties of the scheme, proofs of correct mixing and decryption, which can be audited by the auditors, can be generated by the tally algorithm as in the systems presented in previous chapters.

Chapter 7

Conclusions

This work has focused on individual verifiability, which is a key component on building trust and ensuring reliability in electronic voting systems. This thesis has been done as part of the job of the author as a researcher at Scytl, and should be seen as the result of ongoing research for improving the electronic voting systems to be used across many countries in the world.

This work combines application and research, including descriptions of electronic voting protocols which have already been used (or currently are) in national level binding elections, providing an analysis of their security properties and proposing improvements for future versions. Moreover, new protocols which emerge from the needs at the industrial level and the application of new cryptographic techniques are also presented in this thesis.

Although research is something that never ends, there is a moment when you have to sit down and put together everything you have done. This undoubtedly makes one realize the amount of pending problems and details to solve, and of the new lines of research that may have been opened.

One obvious pending topic is the case of usability studies: proposals for individual verification cannot be only based on the security characteristics they provide, but also on the feasibility of voters performing them, and the value they obtain. Although some considerations have been taken into account when making the protocol proposals (for example, by assuming that a user may not be able to enter a 2048-bit value in the voting device, but she can do so with a 4-digit string), usability studies with real voters have to be performed in order to learn more about the users interaction with the system.

Another pending topic is to consolidate the proposals for distributed generation of voter information, at the registration phase. Some solutions have been suggested throughout this work. However, further research can be done in, for example, techniques for distributed printing processes. The formal security analysis can also be extended in a future work in order to cover the distributed generation scenarios.

The possible use of identity-based cryptography has been a constant during the research composing this thesis. My supervisor and I always had the feeling that some of the key management issues that impacted the voter in some way could be reduced by using these kinds of cryptographic primitives. We will certainly continue making research in this line.

Finally, the novel concept of universal cast-as-intended presented in this work comes at a cost of the usability and practicality of the electronic voting protocols providing this property. We foresee here a new line of research focused on the improvement of such systems.

Bibliography

[1] Belenios verifiable online voting system. Available at http://belenios.gforge.inria.fr/

[2] MinID. Available at http://eid.difi.no/en/minid

[3] Crytographic key length recommendation. Available at http://www.keylength.com (2015)

[4] Helios ballot external verifier. Available at https://github.com/google/pyrios (2015)

[5] Helios ballot verifier. Available at https://vote.heliosvoting.org/booth/single-ballot-verify.html (2015)

[6] Helios voting system. Available at https://vote.heliosvoting.org/ (2015)

[7] Verificatum mix-net. Available at http://www.verificatum.org/ (2015)

[8] Wombat voting system. Available at http://www.wombat-voting.com/ (2015)

[9] Adida, B.: Advances in cryptographic voting systems. PhD Thesis. Available at http://groups.csail.mit.edu/cis/theses/adida-phd.pdf (2006)

[10] Adida, B.: Helios: Web-based open-audit voting. In: Proceedings of the 17th Conference on Security Symposium. pp. 335–348. SS'08, USENIX Association, Berkeley, CA, USA (2008)

[11] Adida, B., Rivest, R.L.: Scratch & vote: self-contained paper-based cryptographic voting. In: Juels, A., Winslett, M. (eds.) Proceedings of the 2006 ACM Workshop on Privacy in the Electronic Society, WPES 2006, Alexandria, VA, USA, October 30, 2006. pp. 29–40. ACM (2006)

[12] Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: Camellia: A 128-bit block cipher suitable for multiple platforms — design and analysis. In: Stinson, D., Tavares, S. (eds.) Selected Areas in Cryptography, Lecture Notes in Computer Science, vol. 2012, pp. 39–56. Springer Berlin Heidelberg (2001)

[13] Bayer, S., Groth, J.: Efficient zero-knowledge argument for correctness of a shuffle. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012, Lecture Notes in Computer Science, vol. 7237, pp. 263–280. Springer Berlin Heidelberg (2012)

[14] Bell, S., Benaloh, J., Byrne, M.D., DeBeauvoir, D., Eakin, B., Fisher, G., Kortum, P., McBurnett, N., Montoya, J., Parker, M., et al.: Star-vote: A secure, transparent, auditable, and reliable voting system. The USENIX Journal of Election Technology Systems, 1 (1) pp. 18–37 (2013)

[15] Bellare, M.: New proofs for NMAC and HMAC: Security without collision-resistance. In: Dwork, C. (ed.) Advances in Cryptology - CRYPTO 2006, Lecture Notes in Computer Science, vol. 4117, pp. 602–619. Springer Berlin Heidelberg (2006)

[16] Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the 1st ACM Conference on Computer and Communications Security. pp. 62–73. CCS '93, ACM, New York, NY, USA (1993)

[17] Bellare, M., Rogaway, P.: Optimal asymmetric encryption. In: Santis, A.D. (ed.) Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings. Lecture Notes in Computer Science, vol. 950, pp. 92–111. Springer (1994)

[18] Bellare, M., Rogaway, P.: The exact security of digital signatures-how to sign with RSA and Rabin. In: Proceedings of the 15th Annual International Conference on Theory and Application of Cryptographic Techniques. pp. 399–416. EUROCRYPT'96, Springer-Verlag, Berlin, Heidelberg (1996)

[19] Ben-Nun, J., Fahri, N., Llewellyn, M., Riva, B., Rosen, A., Ta-Shma, A., Wikström, D.: A new implementation of a dual (paper and cryptographic) voting system. In: 5th International Conference on Electronic Voting, EVOTE 2012, Lochau / Bregenz, Austria, July 11-14, 2012. pp. 315–329 (2012)

[20] Benaloh, J.: Simple verifiable elections. In: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006. pp. 5–5. EVT'06, USENIX Association, Berkeley, CA, USA (2006)

[21] Benaloh, J., Jones, D., Lazarus, E.L., Lindeman, M., Stark, P.B.: Soba: Secrecy-preserving observable ballot-level audit. In: Proceedings of the 2011 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections. pp. 13–13. EVT/WOTE'11, USENIX Association, Berkeley, CA, USA (2011)

[22] Benaloh, J., Tuinstra, D.: Receipt-free secret-ballot elections (extended abstract). In: Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing. pp. 544–553. STOC '94, ACM, New York, NY, USA (1994)

[23] Bernhard, D., Cortier, V., Galindo, D., Pereira, O., Warinschi, B.: SoK: A comprehensive analysis of game-based ballot privacy definitions. In: IEEE Symposium on Security and Privacy 2015. IEEE Computer Society (5 2015)

[24] Bernhard, D., Cortier, V., Pereira, O., Smyth, B., Warinschi, B.: Adapting Helios for provable ballot privacy. In: Atluri, V., Díaz, C. (eds.) Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6879, pp. 335–354. Springer (2011)

[25] Bernhard, D., Pereira, O., Warinschi, B.: On necessary and sufficient conditions for private ballot submission. IACR Cryptology ePrint Archive, Report 2012/236 (2012)

[26] Boneh, D., Franklin, M.: Efficient generation of shared RSA keys. In: Kaliski, Burton S., J. (ed.) Advances in Cryptology — CRYPTO '97, Lecture Notes in Computer Science, vol. 1294, pp. 425–439. Springer Berlin Heidelberg (1997)

[27] Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. Journal of Computer and System Sciences 37(2), 156 – 189 (1988)

[28] Budurushi, J., Neumann, S., Olembo, M.M., Volkamer, M.: Pretty understandable democracy - A secure and understandable internet voting scheme. In: 2013 International Conference on Availability, Reliability and Security, ARES 2013, Regensburg, Germany, September 2-6, 2013. pp. 198–207. IEEE Computer Society (2013)

[29] Bulens, P., Giry, D., Pereira, O.: Running mixnet-based elections with Helios. In: Shacham, H., Teague, V. (eds.) 2011 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '11, San Francisco, CA, USA, August 8-9, 2011. USENIX Association (2011)

[30] Camenisch, J., Chaabouni, R., Shelat, A.: Efficient protocols for set membership and range proofs. In: Pieprzyk, J. (ed.) Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5350, pp. 234–252. Springer (2008)

[31] Carback, R., Chaum, D., Clark, J., Conway, J., Essex, A., Herrnson, P.S., Mayberry, T., Popoveniuc, S., Rivest, R.L., Shen, E., Sherman, A.T., Vora, P.L.: Scantegrity II municipal election at Takoma Park: The first E2E binding governmental election with ballot privacy. In: 19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings. pp. 291–306. USENIX Association (2010)

[32] Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. In: Communications of the ACM. vol. 24, pp. 84–90. ACM, New York, NY, USA (Feb 1981)

[33] Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982. pp. 199–203. Plenum Press, New York (1982)

[34] Chaum, D.: Surevote: Technical report. Available at `http://www.iavoss.org/mirror/wote01/pdfs/surevote.pdf` (2001)

[35] Chaum, D., Carback, R.T., Clark, J., Essex, A., Popoveniuc, S., Rivest, R.L., Ryan, P.Y., Shen, E., Sherman, A.T., Vora, P.L.: Scantegrity II: End-to-end verifiability by voters of optical scan elections through confirmation codes. Information Forensics and Security, IEEE Transactions on 4(4), 611–627 (2009)

[36] Chaum, D., Essex, A., Carback, R., Clark, J., Popoveniuc, S., Sherman, A., Vora, P.: Scantegrity: End-to-end voter-verifiable optical-scan voting. Security & Privacy, IEEE 6(3), 40–46 (2008)

[37] Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings. Lecture Notes in Computer Science, vol. 740, pp. 89–105. Springer (1992)

[38] Chaum, D., Ryan, P.Y.A., Schneider, S.A.: A practical voter-verifiable election scheme. In: di Vimercati, S.D.C., Syverson, P.F., Gollmann, D. (eds.) Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3679, pp. 118–139. Springer (2005)

[39] Chen, X., Wu, Q., Zhang, F., Tian, H., Wei, B., Lee, B., Lee, H., Kim, K.: New receipt-free voting scheme using double-trapdoor commitment. Information Sciences 181(8), 1493–1502 (2011)

[40] Cortier, V., Galindo, D., Glondu, S., Izabachène, M.: A generic construction for voting correctness at minimum cost - application to Helios. IACR Cryptology ePrint Archive, Report 2013/177 (2013)

[41] Cortier, V., Galindo, D., Glondu, S., Izabachène, M.: Election verifiability for Helios under weaker trust assumptions. In: Kutylowski, M., Vaidya, J. (eds.) Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II. Lecture Notes in Computer Science, vol. 8713, pp. 327–344. Springer (2014)

[42] Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 839, pp. 174–187. Springer (1994)

[43] Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. In: Fumy, W. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 1233, pp. 103–118. Springer (1997)

[44] Culnane, C., Ryan, P.Y.A., Schneider, S.A., Teague, V.: vVote: A verifiable voting system. In: ACM Transactions on Information and System Security (TISSEC). vol. 18, p. 3 (2015)

[45] Damgård, I.: Commitment schemes and zero-knowledge protocols. In: Damgård, I. (ed.) Lectures on Data Security, Lecture Notes in Computer Science, vol. 1561, pp. 63–86. Springer Berlin Heidelberg (1999)

[46] Damgård, I.: On σ-protocols. Available at `http://www.cs.au.dk/~ivan/Sigma.pdf` (2010)

[47] Damgård, I., Mikkelsen, G.L.: Efficient, robust and constant-round distributed RSA key generation. In: Micciancio, D. (ed.) Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings. Lecture Notes in Computer Science, vol. 5978, pp. 183–200. Springer (2010)

[48] Demirel, D., Henning, M., van de Graaf, J., Ryan, P.Y.A., Buchmann, J.A.: Prêt à voter providing everlasting privacy. In: Heather, J., Schneider, S.A., Teague, V. (eds.) E-Voting and Identity - 4th International Conference, Vote-ID 2013, Guildford, UK, July 17-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7985, pp. 156–175. Springer (2013)

[49] Diffie, W., Hellman, M.E.: New directions in cryptography. In: IEEE Transactions on Information Theory. vol. 22, pp. 644–654. IEEE (1976)

[50] Dolev, D., Dwork, C., Naor, M.: Non-malleable cryptography. In: Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing. pp. 542–552. STOC '91, ACM, New York, NY, USA (1991)

[51] ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO. Lecture Notes in Computer Science, vol. 196, pp. 10–18. Springer (1984)

[52] Escala, A., Guasch, S., Herranz, J., Morillo, P.: Universal cast-as-intended verifiability. To be published.

[53] Federal Voting Assistance Program: The Uniformed and Overseas Citizens Absentee Voting Act. Available at `http://www.fvap.gov/info/laws/uocava`

[54] Feige, U., Shamir, A.: Witness indistinguishable and witness hiding protocols. In: Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing. pp. 416–426. STOC '90, ACM, New York, NY, USA (1990)

[55] Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer (1986)

[56] Fujioka, A., Okamoto, T., Ohta, K.: A practical secret voting scheme for large scale elections. In: Seberry, J., Zheng, Y. (eds.) Advances in Cryptology - AUSCRYPT '92, Workshop on the Theory and Application of Cryptographic Techniques, Gold Coast, Queensland, Australia, December 13-16, 1992, Proceedings. Lecture Notes in Computer Science, vol. 718, pp. 244–251. Springer (1992)

[57] Fujisaki, E., Okamoto, T., Pointcheval, D., Stern, J.: RSA-OAEP is secure under the RSA assumption. In: Journal of Cryptology. vol. 17, pp. 81–104 (2004)

[58] Gerlach, J., Gasser, U.: Three case studies from Switzerland: E-voting. Berkman Center Research Publication No. 2009-03.1 (2009)

[59] Gharadaghy, R., Volkamer, M.: Verifiability in electronic voting - explanations for non security experts. In: 4th International Conference on Electronic Voting: Verifying the Vote, EVOTE 2010, Lochau / Bregenz, Austria, July 21 - 24, 2010. pp. 151–162 (2010)

[60] Gjøsteen, K.: Analysis of an internet voting protocol. IACR Cryptology ePrint Archive, Report 2010/380 (2010)

[61] Gjøsteen, K.: The Norwegian internet voting protocol. IACR Cryptology ePrint Archive, Report 2013/473 (2013)

[62] Goldwasser, S., Micali, S.: Probabilistic encryption & how to play mental poker keeping secret all partial information. In: Proceedings of the fourteenth annual ACM symposium on Theory of computing. pp. 365–377. ACM (1982)

[63] Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of computer and system sciences 28(2), 270–299 (1984)

[64] Grewal, G.S., Ryan, M.D., Chen, L., Clarkson, M.R.: Du-vote: Remote electronic voting with untrusted computers. In: Fournet, C., Hicks, M.W., Viganò, L. (eds.) IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015. pp. 155–169. IEEE (2015)

[65] Groth, J.: Extracting witnesses from proofs of knowledge in the random oracle model. BRICS Report Series. RS-01-52 (2001)

[66] Heiberg, S., Lipmaa, H., van Laenen, F.: On e-vote integrity in the case of malicious voter computers. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6345, pp. 373–388. Springer (2010)

[67] Heiberg, S., Willemson, J.: Verifiable internet voting in Estonia. In: Krimmer, R., Volkamer, M. (eds.) 6th International Conference on Electronic Voting: Verifying the Vote, EVOTE 2014, Lochau / Bregenz, Austria, October 29-31, 2014. pp. 1–8. IEEE (2014)

[68] Ian Brightwell, Jordi Cucurull, D.G., Guasch, S.: An overview of the iVote 2015 voting system. Available at `https://www.elections.nsw.gov.au/about_us/plans_and_reports/ivote_reports` (2015)

[69] IETF: PKCS#12: Personal information exchange syntax v1.1. Available at `https://tools.ietf.org/html/rfc7292`

[70] IETF: PKCS#5: Password-based cryptography specification version 2.0. Available at `https://tools.ietf.org/html/rfc2898`

[71] International Organization for Standardization: ISO/IEC 18033-2. Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric Ciphers (2006)

[72] Jakobsson, M., Juels, A.: Mix and match: Secure function evaluation via ciphertexts. In: Okamoto, T. (ed.) Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1976, pp. 162–177. Springer (2000)

[73] Jakobsson, M., Juels, A., Rivest, R.L.: Making mix nets robust for electronic voting by randomized partial checking. In: Proceedings of the 11th USENIX Security Symposium. pp. 339–353. USENIX Association, Berkeley, CA, USA (2002)

[74] Jakobsson, M., Sako, K., Impagliazzo, R.: Designated verifier proofs and their applications. In: Maurer, U.M. (ed.) Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding. Lecture Notes in Computer Science, vol. 1070, pp. 143–154. Springer (1996)

[75] Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant electronic elections. In: Atluri, V., di Vimercati, S.D.C., Dingledine, R. (eds.) Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005, Alexandria, VA, USA, November 7, 2005. pp. 61–70. ACM (2005)

[76] Krawczyk, H., Rabin, T.: Chameleon hashing and signatures. IACR Cryptology ePrint Archive, Report 1998/010 (1998)

[77] Lipmaa, H.: A simple cast-as-intended e-voting protocol by using secure smart cards. Cryptology ePrint Archive, Report 2014/348 (2014)

[78] Malkhi, D., Margo, O., Pavlov, E.: E-voting without 'cryptography'. In: Blaze, M. (ed.) Financial Cryptography, Lecture Notes in Computer Science, vol. 2357, pp. 1–15. Springer Berlin Heidelberg (2003)

[79] de Marneffe, O., Pereira, O., Quisquater, J.: Electing a university president using open-audit voting: Analysis of real-world use of Helios. In: Jefferson, D., Hall, J.L., Moran, T. (eds.) 2009 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '09, Montreal, Canada, August 10-11, 2009. USENIX Association (2009)

[80] Moran, T., Naor, M.: Receipt-free universally-verifiable voting with everlasting privacy. In: Dwork, C. (ed.) Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4117, pp. 373–392. Springer (2006)

[81] Naor, M., Yung, M.: Public-key cryptosystems provably secure against chosen ciphertext attacks. In: Proceedings of the twenty-second annual ACM symposium on Theory of computing. pp. 427–437. ACM (1990)

[82] National Institute of Standards and Technology: FIPS-46-3. Data Encryption Standard (DES) (archived) (1999)

[83] National Institute of Standards and Technology: FIPS-197. Advanced Encryption Standard (AES) (2001)

[84] National Institute of Standards and Technology: SP800-38a. Recommendation for block cipher modes of operation – Modes and techniques (2001)

[85] National Institute of Standards and Technology: SP800-38D. Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC (2007)

[86] National Institute of Standards and Technology: SP800-67-Rev1. Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher (2012)

[87] Neuchâtel: Guichet Unique citizen portal. Available at `https://www.guichetunique.ch/`

[88] Neumann, S., Feier, C., Sahin, P., Fach, S.: Pretty understandable democracy 2.0. IACR Cryptology ePrint Archive, Report 2014/625 (2014)

[89] Olembo, M.M., Schmidt, P., Volkamer, M.: Introducing verifiability in the POLYAS remote electronic voting system. In: Sixth International Conference on Availability, Reliability and Security, ARES 2011, Vienna, Austria, August 22-26, 2011. pp. 127–134. IEEE Computer Society (2011)

[90] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) Advances in Cryptology — EUROCRYPT '99, Lecture Notes in Computer Science, vol. 1592, pp. 223–238. Springer Berlin Heidelberg (1999)

[91] Pedersen, T.: A threshold cryptosystem without a trusted party. In: Davies, D. (ed.) Advances in Cryptology — EUROCRYPT '91, Lecture Notes in Computer Science, vol. 547, pp. 522–526. Springer Berlin Heidelberg (1991)

[92] Pedersen, T.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) Advances in Cryptology — CRYPTO '91, Lecture Notes in Computer Science, vol. 576, pp. 129–140. Springer Berlin Heidelberg (1992)

[93] Peng, K.: A hybrid e-voting scheme. In: Bao, F., Li, H., Wang, G. (eds.) Information Security Practice and Experience, 5th International Conference, ISPEC 2009, Xi'an, China, April 13-15, 2009, Proceedings. Lecture Notes in Computer Science, vol. 5451, pp. 195–206. Springer (2009)

[94] Peng, K., Bao, F.: Efficient multiplicative homomorphic e-voting. In: Burmester, M., Tsudik, G., Magliveras, S.S., Ilic, I. (eds.) Information Security - 13th International Conference, ISC 2010, Boca Raton, FL, USA, October 25-28, 2010, Revised Selected Papers. Lecture Notes in Computer Science, vol. 6531, pp. 381–393. Springer (2010)

[95] Pinault, T., Courtade, P.: E-voting at expatriates' MPs elections in France. In: 5th International Conference on Electronic Voting, EVOTE 2012, Lochau / Bregenz, Austria, July 11-14, 2012. pp. 189–195 (2012)

[96] Popoveniuc, S., Hosp, B.: An introduction to PunchScan. In: Chaum, D., Jakobsson, M., Rivest, R., Ryan, P., Benaloh, J., Kutylowski, M., Adida, B. (eds.) Towards Trustworthy Elections, Lecture Notes in Computer Science, vol. 6000, pp. 242–259. Springer Berlin Heidelberg (2010)

[97] Popoveniuc, S., Stanton, J.: Undervote and pattern voting: Vulnerability and a mitigation technique. In: In Preproceedings of the 2007 IAVoSS Workshop on Trustworthy Elections (WOTE 2007). Citeseer (2007)

[98] Provos, N., Mazières, D.: A future-adaptive password scheme. In: Proceedings of the USENIX Annual Technical Conference. pp. 32–32. ATEC '99, USENIX Association, Berkeley, CA, USA (1999)

[99] Puiggalí, J., Guasch, S.: Universally verifiable efficient re-encryption mixnet. In: 4th International Conference on Electronic Voting: Verifying the Vote, EVOTE 2010, Lochau / Bregenz, Austria, July 21 - 24, 2010. pp. 241–254 (2010)

[100] Puiggalí, J., Guasch, S.: Internet voting system with cast as intended verification. In: Kiayias, A., Lipmaa, H. (eds.) VOTE-ID. Lecture Notes in Computer Science, vol. 7187, pp. 36–52. Springer (2011)

[101] Puiggalí, J., Guasch, S.: Cast-as-intended verification in Norway. In: 5th International Conference on Electronic Voting, EVOTE 2012, Lochau / Bregenz, Austria, July 11-14, 2012. pp. 49–63 (2012)

[102] Rackoff, C., Simon, D.: Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In: Feigenbaum, J. (ed.) Advances in Cryptology — CRYPTO '91, Lecture Notes in Computer Science, vol. 576, pp. 433–444. Springer Berlin Heidelberg (1992)

[103] Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. In: Communications of the ACM. vol. 21, pp. 120–126. ACM (1978)

[104] RSA Laboratories: PKCS#1: RSA cryptography standard version 2.1 (2002)

[105] RSA Laboratories: What is a blind signature scheme? Available at `http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/what-is-a-blind-signature-scheme.htm` (2002)

[106] Ryan, P.Y.A.: A variant of the Chaum voter-verifiable scheme. In: Meadows, C. (ed.) Proceedings of the POPL 2005 Workshop on Issues in the Theory of Security, WITS 2005, Long Beach, California, USA, January 10-11, 2005. pp. 81–88. ACM (2005)

[107] Ryan, P.Y.A.: Prêt à voter with Paillier encryption. In: Mathematical and Computer Modelling. vol. 48, pp. 1646–1662. Elsevier Science Publishers B. V. (2008)

[108] Ryan, P.Y.A., Schneider, S.A.: Prêt à voter with re-encryption mixes. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security, Hamburg, Germany, September 18-20, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4189, pp. 313–326. Springer (2006)

[109] Ryan, P.Y.A., Teague, V.: Pretty good democracy. In: Christianson, B., Malcolm, J.A., Matyas, V., Roe, M. (eds.) Security Protocols XVII, 17th International Workshop, Cambridge, UK, April 1-3, 2009. Revised Selected Papers. Lecture Notes in Computer Science, vol. 7028, pp. 111–130. Springer (2009)

[110] Sako, K., Kilian, J.: Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In: Guillou, L.C., Quisquater, J.J. (eds.) EUROCRYPT. Lecture Notes in Computer Science, vol. 921, pp. 393–403. Springer (1995)

[111] Sandler, D., Derr, K., Wallach, D.S.: Votebox: A tamper-evident, verifiable electronic voting system. In: Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA. pp. 349–364 (2008)

[112] Sandler, D., Wallach, D.S.: Casting votes in the auditorium. In: Martinez, R., Wagner, D. (eds.) 2007 USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'07, Boston, MA, USA, August 6, 2007. USENIX Association (2007)

[113] Santis, A.D., Persiano, G.: Zero-knowledge proofs of knowledge without interaction (extended abstract). In: 33rd Annual Symposium on Foundations of Computer Science, FOCS, Pittsburgh, Pennsylvania, USA, 24-27 October 1992. pp. 427–436. IEEE Computer Society (1992)

[114] Schnorr, C.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings. Lecture Notes in Computer Science, vol. 435, pp. 239–252. Springer (1989)

[115] Shoup, V.: A proposal for an ISO standard for public key encryption. IACR Cryptology ePrint Archive, Report 2001/112 (2001)

[116] Smyth, B., Frink, S., Clarkson, M.R.: Computational election verifiability: Definitions and an analysis of Helios and JCJ. IACR Cryptology ePrint Archive, Report 2015/233 (2015)

[117] Stark, P., Lindeman, M.: A gentle introduction to risk-limiting audits. In: IEEE Security and Privacy. vol. 10, pp. 42–49. IEEE (Sep 2012)

[118] Storer, T.W.: Practical pollsterless remote electronic voting. PhD Thesis. University of St Andrews (2007)

[119] Swiss Federal Chancellery: Explications relatives à l'ordonnance de la Chancellerie Fédérale sur le vote électronique (OVotE). Available at `http://www.bk.admin.ch/themen/pore/evoting/07979` (2013)

[120] W3C: HTML5 specification. Available at `http://www.w3.org/TR/html5/`

[121] Wikström, D.: A commitment-consistent proof of a shuffle. IACR Cryptology ePrint Archive, Report 2011/168 (2011)

[122] Zagórski, F., Carback, R., Chaum, D., Clark, J., Essex, A., Vora, P.L.: Remotegrity: Design and use of an end-to-end verifiable remote voting system. In: Jr., M.J.J., Locasto, M.E., Mohassel, P., Safavi-Naini, R. (eds.) Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7954, pp. 441–457. Springer (2013)

[123] Zetter, K.: Diebold hack hints at wider flaws. Wired Magazine. Available at `http://archive.wired.com/politics/security/news/2005/12/69893` (2005)