Coercion-resistant cast-as-intended verifiability in electronic voting systems

Tamara Finogina

Supervisor: Dr. Javier Herranz Sotoca

Departamento de Matemática Aplicada IV

July 2023

Facultat de Matemàtiques i Estadística
Universitat Politècnica de Catalunya

Contents

Acknowledgements **7**

Preface **9**

1 Introduction **11**
1.1 State Of The Art . . . 12
1.1.1 Coercion-Resistance . . . 12
1.1.2 Receipt Freeness . . . 14
1.1.3 Cast-As-Intended Verification . . . 15
1.2 Thesis Contribution . . . 22
1.2.1 Publications Resulting From The Thesis . . . 23
1.3 Structure Of This Thesis . . . 23

2 Preliminaries **25**
2.1 Relations And Languages . . . 25
2.2 Groups And Hard Problems . . . 26
2.3 Homomorphic Public Key Encryption . . . 26
2.3.1 ElGamal Encryption . . . 27
2.4 Commitment Scheme . . . 28
2.5 Interactive Zero-Knowledge Systems . . . 30

3 Defining Coercion-Resistant Cast-As-Intended Verifability **33**
3.1 Can We Have One Standard Definition For Both Properties? . . . 33
3.2 Our Settings . . . 34
3.3 Parties And Syntax Of The E-Voting Protocol . . . 39
3.4 Formal Definitions Of Cast-As-Intended And Coercion-Resistance . . . 41
3.4.1 Formal Definition Of Cast-As-Intended (CAI) Verifiability . . . 41
3.4.2 Formal Definition Of Coercion-Resistance (CR) . . . 41
3.4.3 Comparison With Previous Definitions Of CAI And CR . . . 43
3.5 On The Necessary Number Of Rounds . . . 44

4 Instantiations Of CR-CAI Solutions **49**
4.1 Unsatisfactory Solution U_1 . . . 49
4.2 Solution S_1: Committing To Challenges . . . 50
4.2.1 Security Analysis Of The Protocol . . . 51
4.2.2 Detailed Protocol For ElGamal Ciphertexts . . . 53
4.3 Unsatisfactory Solution U_2 . . . 55

4.4 Solution S_2: Adding Interactive Proof Of The Trapdoor Knowledge . 56
4.4.1 Security Analysis Of The Protocol 58
4.4.2 Detailed Protocol For ElGamal Ciphertexts 60

5 Post-Quantum Solution **63**
5.1 Why We Cannot Use Our Solutions Directly In Lattice Settings? . . . 63
5.2 Basics Of Lattice-Based Cryptography 66
5.2.1 Polynomial Rings 66
5.2.2 The Ring Learning With Errors Problem 68
5.2.3 The Ring Short Integer Solution 69
5.2.4 Error Distribution And Rejection Sampling 69
5.2.5 Lattice-Based Public Key Encryption Scheme 70
5.2.6 Lattice-Based Proof Of A Short Integer Vector Knowledge . . 71
5.3 The Transformation 73
5.4 Security Analysis Of Our Transformation 76
5.4.1 Zero-Knowledge 76
5.4.2 Soundness 77
5.5 Possible Extensions Of Our Transformation 78
5.6 Use Case And Implementation 79

6 CR-CAI For A Computationally Limited Voter **83**
6.1 Parties And Syntax Of The Voting Protocol 83
6.2 Formal Definition Of CAI For A Computationally Limited Voter . . . 84
6.3 Formal Definition Of CR For A Computationally Limited Voter . . . 85
6.4 CR-CAI Solution For A Computationally Limited Voter 86
6.4.1 Security Analysis Of The Protocol 88
6.4.2 Comparison With Bingo Voting: On The Necessity Of OED . 90

7 Stronger Coercion Settings **93**
7.1 Stronger Coercion Settings 93
7.2 Why S_1 Is Vulnerable To Strong Coercion? 94
7.3 Solution S_3: Random Group Generator 95
7.3.1 Security Analysis Of The Protocol 96
7.4 Future Research 99

8 Conclusion **101**

A Brief Description Of Mentioned E-Voting Schemes **103**
A.1 Helios Voting System 103
A.2 Belenios Voting System 105
A.3 Demos Voting System 106
A.4 Demos-2 107
A.5 Swiss Post Voting System 109
A.6 Estonian Internet Voting Scheme 111
A.7 Hyperion Voting System 112

Bibliography **113**

List of Figures

2.1 DL game for the cyclic group $\mathbb{G}$ of the order q with a generator g. . . 26
2.2 DDH game . . . 27
2.3 A game against the hiding property of the commitment scheme Com. 28
2.4 A game against the binding property of the commitment scheme Com. 29
2.5 Σ-protocol for proving the statement x. . . . 31
2.6 Σ-protocol for proving the ciphertext encrypts the intended message. 32
2.7 Σ-protocol for proving the ElGamal encryption correctness. . . . 32

3.1 Experiment for coercion-resistance. . . . 42

4.1 Σ-protocol for proving the ciphertext encrypts the intended message. 50
4.2 The voting protocol from solution S_1. . . . 51
4.3 The ElGamal instantiation of the solution S_1. . . . 54
4.4 OR proof for proving the ciphertext encrypts the intended message. . 56
4.5 The voting protocol from solution S_2. . . . 57
4.6 The ElGamal instantiation of the solution S_2. . . . 60

5.1 Standard discrete logarithm-based identification scheme. . . . 64
5.2 Proof of knowledge of a ternary solution to a linear equation over $\mathbb{Z}_{\mathsf{q}}$. 74
5.3 Application of our transformation to the protocol from Figure 5.2. . . 80
5.4 Time distribution of the 1,000 executions of the first test. . . . 81
5.5 Percentage of executions suffering i aborts in the second test. . . . 81

6.1 Experiment for coercion-resistance for a computationally limited voter. 86
6.2 Solution S_{Lim}. . . . 87
6.3 ValidOption verification. . . . 88
6.4 ValidProof verification. . . . 88

7.1 The simplified version of the solution S_1 from Chapter 4. . . . 95
7.2 The solution S_3. The hash function is defined as $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_{\mathsf{q}}$ . . 96

A.1 A ballot verification based on Benaloh's challenge. . . . 104
A.2 A ballot verification from Belenios protocol. . . . 106
A.3 The vote casting protocol of Demos voting system. . . . 107
A.4 The vote-casting and voter-verification of the Demos-2 voting system. 108
A.5 The vote-casting and voter-verification of the Swiss Post voting system. 110
A.6 The vote-casting and voter-verification of Estonian voting system. . . 111
A.7 The vote-casting and voter-verification of Hyperion voting system. . . 113

List of Tables

4.1 Performance results of S_1 implementation in different groups. 54
4.2 Performance results of S_2 implementation in different groups. 61

Acknowledgements

First and foremost, I would like to express my sincere gratitude to Dr. Javier Herranz. Thank you for all the support, patience, and help, for all those emails, meetings, and calls during all (sometimes quite late) hours, especially during the tough weeks of finalizing the thesis. But, above all, thank you for keeping me sane throughout the Ph.D. process.

I also want to thank Jordi for giving me a chance to do this Ph.D. and for trusting me to be a part of the best research team. I am lucky to work with very talented people and to learn a lot from all of them. Thank you, team! Special thanks to Nuria for always replying to my never-ending questions (no matter how busy she was) and Enrique for reminding me to focus on my research at least once in a while. And, of course, thanks to the Scyphy team for their patience and flexibility, and thank you, Dani, for making the final implementation on short notice.

I want to thank the Mathematics Department at UPC for their help and flexibility with (some of) deadlines. Thanks to Xavier and Abraham for implementing the post-quantum solution. Finally, I would like to express my sincere gratitude to Generalitat for funding an industrial doctorate program that allowed me to complete this thesis.

Also, I would like to thank my friends and family for their love and support during this process. Without them, this journey would not have been possible.

Finally, I would also like to thank you, my reader: I hope you enjoy your reading.

Preface

In front of you is my Ph.D. thesis: 'Coercion-resistant cast-as-intended verifiability in electronic voting systems." It was written as a result of an industrial Ph.D. done at Scytl under the supervision of Dr. Javier Herranz Sotoca from the Department of Mathematics at UPC. The main objective of an industrial Ph.D. is to do applied research in the area of the company's expertise, which in this case is electronic voting. From the company side, my co-supervisors were my ex-colleague Dr. Enrique Larraia de Vega for the first three years of the research and, after that, my colleague Dr. Nuria Costa. Finally, I would like to mention master students Xavier Arnal and Abraham Cano from the Department of Mathematics at UPC, and my colleague Daniel Orihuela Rodríguez, who helped me with the implementation of some of the solutions this thesis presents.

During the entire Ph.D., I was part of the Research and Security team at Scytl, which supports the development team, evaluates security designs and changes, analyzes software bugs and proposes fixes, etc. This work allowed me to understand security requirements specific to electronic voting, learn about customers' expectations regarding privacy and verifiability, and get familiar with common security threats and their prevention methods. But more importantly, I noticed how different and (sometimes) conflicting approaches academia, industry, and legislation have regarding electronic voting properties. My colleagues and I published two papers on this topic: one - about some common electronic voting practices that could be dubious from a legal point of view [51]; another - about the compliance of the blockchain voting schemes with international standards [39]. While those publications do not directly contribute to this thesis, they helped me to realize the balance of all three approaches is crucial for making the result secure and practically applicable.

Another significant but indirect contribution to Ph.D. research came from the projects I was part of. The most memorable and challenging one was re-designing the Swiss Post voting system for a 100% certification process and writing the security proofs of verifiability and privacy for the new scheme [76],[105]. The entire project was an amazing experience of learning, featuring: working with an excellent team, constantly adapting the proof to the changing legislation, brainstorming of the fake confirmation fix (which developers managed to pull off before the release), adjustments due to code constraints, research of uncommon assumptions, etc. This project showed me how crucial trust assumptions are and how much everything is interlinked in a system as complex as the Swiss Post voting. It also allowed me to dive deep into the nuances of various cryptographic definitions and hardness assumptions. Another important project was a European Union project called PROMETHEUS. It helped me greatly expand my knowledge of lattice-based cryptography and post-quantum

security. Finally, I would like to mention a Spanish project Datamantium, which showed how difficult it is to implement anonymous channels in practice. There were other projects but their contribution to the thesis were more subtle.

The Ph.D. research started with an attempt to make a paperless cast-as-intended verification for a return code scheme. The resulting idea was based on generating multiple proofs and proving that only one (for the intended option) is real by modifying it with a voter's challenge [48]. Because it was essentially a Σ-proof, the voter had to be honest for the protocol to achieve zero knowledge. However, it raised several questions: Can we expect voters to behave honestly? Can we always assume voters have some pre-delivered data? All those questions led to the research for this thesis. Also, the same chameleon-based solution inspired the post-quantum transformation from Chapter 5 [13]. Hence, even though the paperless solution is not detailed in this thesis, it guided the research in the direction of the coercion-resistant cast-as-intended verification.

Finally, it is worth mentioning that coercion-resistance is becoming more and more desirable for customers of electronic voting solutions. Hence, the contributions from this thesis may be essential building blocks for future voting systems' designs.

Chapter 1

Introduction

When we hear about electronic voting, e-voting, we typically imagine a computer or another device somehow transmitting our vote directly to the election authorities and maybe helping them to speed up the result tabulation. The main benefit, of course, is convenience - voting from any place and at any time, faster result aggregation, and perhaps, even the possibility to change the mind or vote far away from prying eyes. One can think of e-voting as a tool with its strengths and weaknesses.

Unlike traditional polling place voting, which requires voters' physical presence and supervision of auditors, e-voting offers online participation and allows a single auditor to perform verification of 100% ballots multiple times. The security guarantees are based on mathematical assumptions rather than on the presence of observers. Plus, it is possible to audit operations that previously were not verified (e.g., eligibility of votes) or could be checked only once (e.g., unsealing envelopes). Unfortunately, e-voting has to run its code in an uncontrolled environment of a voting device, which extends possible attack scenarios as now an attacker is not limited to a physical location.

One of the most common fears is that a voting device will disregard the voter's intent and cast a different vote instead. An undetectable attack like that on a large scale will allow the adversary to control the election result completely. Therefore, ensuring that the ballot indeed contains the intended option is crucial. This check is known as cast-as-intended verification, a property that demands an e-voting scheme to prove that the cast ballot contains the voter's choice and was not altered by the system.

Cast-as-intended verification is not the only property the e-voting system has to comply with. For example, the e-voting system should also prove that a cast ballot arrived on the bulletin board unmodified (recorded-as-cast); no eligible and correct vote was modified or excluded from the tally (tally correctness); the tallying process was done correctly (universal verifiability); no ineligible votes were inserted; that voter's choice remains private and more. And things only get more complicated if we add everlasting privacy, receipt-freeness, or coercion-resistance.

Additional challenge comes from ambiguous definitions of those properties. For example, it is still unclear what the coercer can do or whether a forced abstention should be considered a coercion attack. Similarly, there is no standard definition for privacy or cast-as-intended verification; all existing ones are tailored to a particular

e-voting system. Things get even more complicated if we consider electoral laws. For example, e-voting is rarely the only voting channel, thus, it has to be possible to link a voter to their vote to prevent multiple voting. It implies we cannot use anonymous voting channels, but they are the only known way to ensure the coercer cannot force voters to abstain.

All in all, there are many tradeoffs and contradictions an e-voting scheme has to consider. One particularly interesting tradeoff is ensuring a voter cannot sell the vote willingly or under duress while at the same time preventing a malicious voting device from cheating. At first glance, the task seems easy: we need coercion resistance to limit (in)voluntary vote-selling and cast-as-intended verification to stop the voting device from cheating. However, those properties are contradictory.

Informally, the coercion-resistance property ensures that a voter cannot prove to anyone the vote content, which prevents vote selling and voting under duress. The cast-as-intended property states that a malicious voting device cannot cheat the voter and send a different intent from what the voter intended. One property requires outputting no information about the selection, and the other demands a proof - an extra piece of data.

Another curious observation is that the voting device is malicious but not evil enough to collaborate with the coercer. This goes against a common game-based approach, where each entity is either corrupted or honest, and there are no in-between states. Yet, now we are in a situation where the challenger has to team up with an adversary that controls the voting device to resist *another adversary* - the coercer. Such unusual cooperation, naturally, raises questions:

- Can we simultaneously achieve cast-as-intended and coercion-resistance? And if so, in which settings and under what assumptions?
- Will the answer be different for the post-quantum world?
- Are there any limits or conditions under which we cannot have both?

This thesis aims to look at all of those questions in order to find how we can provide coercion-resistant cast-as-intended verification.

1.1 State Of The Art

This section gives an extensive summary of the state-of-the-art for Coercion-Resistance (and the related notion of Receipt Freeness) and Cast-as-Intended verification. We show that both properties are tricky to define and are open to interpretation. Moreover, we illustrate the most popular coercion-resistant and cast-as-intended methods via existing e-voting scheme designs.

1.1.1 Coercion-Resistance

The first verifiable election protocol that focuses on preventing participants from selling their vote to a passive coercer was designed by Benaloh and Tuinstra in 1994 [108]. In the same work, they introduced the concepts of receipt-freeness and

uncoercibility. Later, in 1997, Okamoto proposed an alternative definition of receipt-freeness [88], which allowed the coercer to interact with the voter during the voting phase. Despite the name, it is essentially the first definition of what is now known as coercion-resistance. However, the first formal definition of coercion-resistance, known as the JCJ definition, appeared more than a decade later in the work of Juels *et al.* [67].

The JCJ definition is the most demanding definition of coercion-resistance. It captures the idea of anonymous credentials and accounts for intentional credentials selling and forced-abstention attacks. Unfortunately, any scheme that realizes it in practice [32, 12, 96, 103, 31] requires anonymous voting channels and puts trust in the voting device, as CAI verification is not possible. Moreover, it requires secure pre-delivery of secret keys and a system for managing real and fake credentials, which is challenging given that voters have trouble remembering passwords they use a few times per year [72]. Recently, in 2022, [37] found that the JCJ definition leaks too much data in case of revoting and hence is not coercion-resistant. The discovered attack resembles the "1009 attack" first mentioned in [102].

The simplest form of coercion-resistance technique is multiple voting [107]. However, it is not always legal as it gives an unfair advantage to e-voters [72]. Moreover, it implicitly assumes that the coercer cannot see the ballot box content nor observe voting channels. Otherwise, it is trivial to see whether the voter obeyed or not, which makes multiple voting the weakest form of coercion-resistance.

Another definition of coercion-resistance [56] challenges the adversary to distinguish between two ballots: one generated for the coercer's preference and another for the true intent (our definition for the CR property, in Section 3.4.2, will follow this idea). It separates the vote-casting part from the tallying, which, despite being criticized [59], allows excluding plaintext-based leakage: e.g., Italian attacks, specific write-ins, etc. Note that plaintext leakage is almost impossible to avoid (unless we use a homomorphic tally) and is present in any voting channel that permits free-form votes. Unfortunately, the definition in [56] prohibits any post-election communication between the coercer and the voters.

Delaune, Kremer, and Ryan [43] propose a formalization of coercion-resistance in the Dolev-Yao model and also formally prove that it implies receipt-freeness and hence, in turn, implies privacy. However, as noted in [94], this definition does not include the randomization attack and is unsuitable for weighted voting.

In [106] the coercion-resistance is defined as a non-cooperative game between the honest election authority representing society and the coercer. Unfortunately, only two simple coercion models are studied.

The family of UC-definition of coercion-resistance started in 1996 with a weaker notion of (post-factum) incoercibility [27]. The definition was allowing adversary to coerce voters only after voting phase was completed i.e. voter was not expected to receive any instructions or input before voting. The generalization of this property - receipt freeness given in [86] - permits coercers to corrupt voters at any time and enforce them to follow any strategy. Achieving receipt-freeness under the definition in [86], however, requires commitments with extremely strong hiding property that can only be implemented by physical means. Moreover, reactive protocols (where the input in one phase depends on the output of the other) are not modeled. The

work of Unruh and Müller-Quade [110] addressed the issue by defining Composable Incoercibility framework, which allows to model reactive protocols and gives universal composition. However, later work [59] found issues in the definition, thus it does not capture receipt-freeness nor coercion-resistance.

The only statistical approach to measuring coercion-resistance, σ-coercion resistance, was proposed in [75]. The definition assumes the coercer cannot perform at least one step of the voting protocol without the voter, requires prior knowledge of the choices distribution and the exact election configuration, and covers a multi-coercion case. Even though the definition permits the coercer to succeed with non-negligible probability, setting the coercion-resistance level requires conducting a complex combinatorial analysis. Also, as was pointed out in [100], the definition does not account for plausible deniability.

Other works on coercion-resistance include definition comparisons [94, 72, 59], analysis of coercion scenarios [77, 19], the coercion-resistance in social context [24] and studies of relations between privacy and coercion-resistance [89]. We can also mention an attempt to combine JCJ with the cast-as-intended mechanism from the Selene voting system [63]. However, the cast-as-intended verification is performed not during the voting phase but after the votes are mixed and decrypted.

1.1.2 Receipt Freeness

Nowadays, coercion is typically divided into passive and active coercions. The former is also known as receipt-freeness. There is no consensus over the boundary between receipt-freeness and coercion-resistance [44, 52]. However, intuitively the main distinction from the coercion definition is that the adversary cannot interact with the voter during the voting phase [28, 32].

Usually, we say that the receipt-freeness property guarantees that a voter cannot prove the vote content even when willingly deviates from the protocol attempting to sell the vote. However, there is no single opinion regarding how much the voter can misuse the protocol nor whether the coercer can receive anything from the voter *before* the voting. Nevertheless, the notion of receipt-freeness is (generally) stronger than privacy, which only guarantees secrecy to honest voters.

The very first definition of receipt-freeness [108] aimed to adopt the idea of a private voting booth for electronic voting. While inside, the voter cannot communicate with anyone or be under observation. The definition, however, assumes the existence of a trusted randomness source (beacon) used for challenging the system.

Typically, other definitions of receipt-freeness are similar in spirit. For example, more recent definitions [71] and [98] allow the coercer to demand proofs (but not voting cards or encryption randomness) from honest voters only *after* vote-casting and verification. Note that both definitions assume the existence of an untappable channel between voters and authority.

Contrary to traditional receipt-freeness, the strong receipt-freeness [28] allows voters to use coercer-selected ballot material (e.g., encryption randomness): i.e., receive instructions before the voting phase. Usually, such behavior is prohibited as no interaction is allowed. However, one may think of the coercer's pre-defined randomness or biased randomness generator as the forceful extraction of randomness

used by the voting devices. In other words, the strong receipt freeness allows the voter to tamper with the voting device to sell the vote. The scheme that implements the definition works under the assumption that the adversary cannot observe the interaction between the voter and the re-randomization server, which is trusted to modify the ballot before it can be published. CAI verification is incompatible with such demands.

A quantitative approach to receipt-freeness was proposed in [20]. The σ-receipt freeness is based on [28], but similar in spirit to [75]. It does not consider the leakage from tally results nor accounts for coercers that demand voters to cast invalid votes. Morover, the adversary is not allowed to observe the channel between the voter, the posting trustee and the voting system. However, it is the only definition that allows to measure the level of receipt-freeness.

To the best of our knowledge, the only comparison of receipt-freeness definitions is in [52].

1.1.3 Cast-As-Intended Verification

The first proposal for cast-as-intended verification (CAI) was published by Chaum in [30], followed by Neff's approach [87], published the same year. Both works expect the voter to cast an electronic vote in a polling place and require printing commitments to ensure the voter's choice is encrypted correctly. Chaum's proposal was based on visual cryptography and a particular printer, while Neff's utilized interactive zero-knowledge proofs that can be simulated. For coercion-resistance, Neff's scheme assumes that the voter selects challenges from a pre-generated set, and the voting device can print a short secret without revealing it. The voter must be honest since linking the challenge with the shown commitment would be fatal for the proposal.

Currently, there are numerous methods to ensure that the ballot contains the voter's intent and not something else. However, we can divide all those methods into several groups based on the origin of data used for the verification: probabilistic checks, a ballot cast assurance based on pre-delivered data, and simulatable transcripts.

1.1.3.1 Probabilistic Ballot Verification

The first CAI methods group relies on probabilistic checks and does not inspect the cast ballot per see. While it never gives a 100% guarantee, many schemes select probabilistic CAI verification due to its simplicity and support of any election type.

This group features the following CAI methods: the cast-or-challenge approach (e.g., [5]), partial checks (e.g., [33]), and one based on the inspection of all but one of the generated ballots (e.g., [70]). Notable examples of schemes proposing this type of CAI verification are Helios [5], Demos-2 [70], and Belenios [33].

Cast-or-challenge verification: The most famous CAI verification method is Benaloh's challenge, also known as the cast-or-challenge technique, [18], which is used by a voting scheme Helios [5] and its derivatives. The method works as follows:

a voter enters selected options into their voting device, then decides whether to cast or audit the encrypted ballot. In case of an audit, the voting device must reveal the encryption randomness (or any other proof of ballot correctness), which allows the voter to verify the ballot on another device. Else, the vote is submitted without any verification.

The verification works only if the voting device somehow commits to the encryption before the voter decides whether to audit or cast it. Otherwise, a malicious device will always pass the verification by outputting a freshly created ciphertext and the corresponding randomness. Also, the ballot check should be done on a separate device since asking the voting device to check itself is pointless. Finally, the number of audits before the cast should be unpredictable. Otherwise, a malicious voting device can guess when the voter stops the audit and still send a ballot with a different voting option. However, according to studies [3], only 43% of users would engage in cast-or-challenge verification; the rest would proceed without checking. Moreover, the majority of users do not understand this verification method [84].

Verification that relies on other honest voters: A verification method similar to the cast-or-challenge approach was proposed for the scheme called Demos-2 [70]. A voting device prepares two (supposedly) identical ballots and commits to them by showing corresponding hashes to the voter. The voter selects one of the two ballots for inspection; the unselected ballot will be completed with proof and cast immediately. The ballot inspection consists of reconstructing the ciphertext, ensuring it encrypts the intended voting option, and verifying that the hash matches the shown one. Of course, the voter needs a separate device for the verification; otherwise, it makes no sense since a malicious machine can always report a successful verification.

This technique is used by the Demos system family, which features Demos [71], Demos-2 [70], and D-Demos [1]. However, since the voter only gets a 50% chance of ballot correctness (a malicious voting device can guess an unchecked ballot and put there a different voting option), the method only works if enough voters verify. It is not clear what enough means in a general case because each election has a different number of votes separating the winner from the losers. However, one can do a rough estimation based on electoral pool results.

Nevertheless, strictly speaking, this verification is not a reliable cast-a-intended method as the voter has to assume others will perform checks and only has a 50% guarantee. Yet the expectation regarding voters' behaviour is a major drawback. While there are no studies regarding Demos-2, experiments for Demos show that about 85% of voters tend to select the first ballot for vote-casting and leave the second one for the audit [69].

Partial checks: Another way to probabilistically check the cast ballot is to do a partial check. An example would be a verification suggested for the Belenios voting system [33]. A voting device prepares three ciphertexts: for the selected voting option m, for a random value $\mathsf{m}_1 = \mathsf{t}$, and for the masked selection $\mathsf{m}_2 = \mathsf{m} - \mathsf{t}$. Also, it proves in zero-knowledge that the content of the first ciphertext is equal to the combination of the second and third ones. The voter receives both random value and masked selection in plain text (i.e., $\mathsf{m}_1, \mathsf{m}_2$) and ensures that their sum results in

the intended vote: i.e., $\mathsf{m}_1 + \mathsf{m}_2 = \mathsf{t} + \mathsf{m} - \mathsf{t} = \mathsf{m}$. Then the voter chooses whether to check the ciphertext corresponding to the random value m_1 or the masked selection m_2. Depending on the choice, the voting device reveals the randomness of one of the ciphertexts and posts it on the bulletin board for public verification.

While a partial check preserves privacy and allows inspection of the cast ballot, it gives only a 50% guarantee that the ciphertext contains the intended option. A malicious voting device can prepare a vote for an alternative voting option m^*, return values $\mathsf{m}_1 = \mathsf{t}$ and $\mathsf{m}_2 = \mathsf{m} - \mathsf{t}$, but encrypt $\mathsf{m}_1^* = \mathsf{m}^* - \mathsf{m} + \mathsf{t} \neq \mathsf{m}_1$ and m_2 instead. If the voter selects m_2 for testing, it would not detect the problem, and a modified ballot would be tallied.

Advantages and disadvantages of probabilistic verification: Probabilistic CAI verification is quite popular among e-voting scheme proposals. However, strictly speaking, it does not provide CAI since the sent ballot is never audited. True, it gives some chance to detect VD misbehavior, but a malicious VD may still cheat with non-negligible probability.

1.1.3.2 Transferable Proof

Another popular CAI method relies on giving the voter some feedback sufficient for the ballot content verification. While it guarantees that the intent has not been altered, the obtained proof can be given to anyone (transferred), thus enabling vote-selling.

This group features a variety of methods: revealing randomness used for ballot encryption, return codes, vote codes, and any verification based on pre-delivered voting material. This group is vast and includes any verification method that relies solely on data the voter has (not something the voter saw, selected, or did), and that data cannot be hidden, simulated, or denied. By undeniable data, we mean, for example, possession of: voting cards with return (e.g., [41]) or vote codes (e.g., [71]), outputting encryption randomness directly or as a QR-code (e.g., [107]), NIZK proofs of ballot correctness, etc.

Verification based on revealing randomness: One of the simplest ways to provide CAI verification is to give the voter the encryption randomness so the ballot can be recreated on another device. For example, the Estonian voting system [107] relies on this type of verification. The voter enters a selection into their voting device, which encrypts it and outputs a QR code containing the ciphertext's randomness. The voter can use a verification application and QR code to ensure that their ballot encrypts the intended option.

To protect voters' privacy, Estonian legislation permits multiple voting. Theoretically, a coercer can demand the voter's QR code, but the voter can re-write their vote later. However, it only works when the coercer cannot see the output of the voting device nor observe the ballot box. Otherwise, the mere existence of multiple voters would indicate disobedience.

The main benefit of revealing the encryption randomness is its simplicity. Such verification can be added to any e-voting scheme and support any election type.

However, it significantly facilitates vote-selling, especially when multiple voting is not allowed. Moreover, it is not coercion-resistant. Even if we overlook that most countries prohibit multiple voting as it gives an unfair advantage to voters voting electronically, the coercer still can ask voters to vote at the very last moment. Also, in most cases, the coercer can observe the ballot box, which would invalidate the benefit of multiple voting.

Verification based on return code: Another example of transferable CAI proof is verification based on return codes. The idea is quite simple and is based on comparing short alphanumerical values. Each voter receives a pre-printed voting card with a map between possible voting options and return codes. Then, after vote-casting, the voter must compare the code system returned with the one printed on their voting card. If the code does not match, the voter complains to the election authorities and, thus, invalidates the vote.

An example of an e-voting scheme relying on return codes for CAI would be the Swiss Post voting system [92]. Additionally, their system has a mandatory vote confirmation that allows voters who received incorrect codes to not confirm their ballot instead of contacting the electoral authorities.

The return-code method is one of the most user-friendly CAI verification techniques [85]. However, it requires pre-established secure channels for delivering voting cards. Also, for verification to work and privacy to hold, the printing entity should be trusted since it knows all codes of all voters.

Code-voting schemes: Code voting is quite similar to return codes. It also requires electoral authorities to generate, print, and securely distribute voting cards among voters before the election. However, instead of typing the voting option to the voting device, voters enter the vote code from the voting card corresponding to their preferred selection. The code can be sent without encryption and published on the bulletin board without privacy concerns. The only verification voter has to perform is to ensure the correct code appears on the public bulletin board.

For example, an e-voting scheme called Demos [71] relies on vote codes instead of expecting the voting device to encrypt the voter's selection. Of course, one has to trust (or verify) that the printing authority did not swap the codes hoping to alter the election result. Also, the part of the system that is responsible for linking the codes with candidates must be trusted or thoroughly audited to ensure that each code uniquely corresponds to the candidate. Otherwise, a malicious system can distribute voting cards where all codes result in the same selection.

Systems based on vote codes do not put any trust in voting devices. However, there should be secure channels between all voters and the printing authority for secure voting card distribution. Also, voters still need a second device to ensure the public bulletin board has the selected code. Otherwise, a malicious voting device can send nothing. Moreover, the printed voting cards and publicly available unencrypted codes facilitate vote-selling. Finally, the printing entity should be trusted for both privacy and CAI since it knows all the vote codes of all voters.

Finally, the CAI verification (checking the correct code is published) does not necessarily guarantee vote correctness. A malicious system can print one correspon-

dence between candidates and codes but tally the votes based on another mapping. Or assign multiple codes to the same candidate. Therefore, the audit is mandatory. However, in contrast, to the return-code-based schemes, the audit is more complicated as it has to preserve voter privacy. Yet, the most common solution that preserves privacy - partial audit - gives only probabilistic guarantees.

Advantages and disadvantages of transferable verification: This group includes many methods, and each has unique benefits and tradeoffs. For example, vote-code and return-code-based schemes require pre-printed and pre-delivered paper material. Delivering printed material implies the existence of trusted and reliable delivery channels.

Usually, verifications from this group are the most user-friendly and understandable. However, the proof voters obtain is transferable. It implies the voter can sell their vote (or be forced to do so).

An interesting question is whether we can expect voters to hide, destroy or alter the printed card with the vote or return codes. On the one hand, if voters can do it, then the verification method would no longer be transferable. However, it is quite an advanced voter behavioral requirement we cannot take for granted.

Some people claim that printed voting material can be faked by voters, who might print another indistinguishable card [71], but we do not believe it to be true. Imagine if anyone could create an authentically looking homemade voting card. How would the voter know that their card is authentic? What if someone already switched the original card with a modified copy? If there is no way to do so, then the voter never knows if the verification based on such a card is truthful. Otherwise, the coercer can always ask for proof.

1.1.3.3 Interactive Proofs

One more CAI method relies on the interaction between the voter and their voting device. However, the verification heavily relies on voters being honest.

This group features several CAI proposals: Σ-protocol-based proofs, Neff's proposal for polling-place verification known as MarkPledge [87], an adaptation of Neff's protocol to a Direct Recording Electronic (DRE) voting machines [86], an adaptation of MarkPledge [6] for incorporating a helper organization, successors of Neff's protocol called MarkPledge 2 [7], VerifyVotell [66], and a protocol based on chameleon hash [48].

Using Σ-protocols: One possibility is to consider an interactive Σ protocol between the voting device and the voter to let the voting device convince the voter that the ballot contains the intended option.

Recall that Σ-protocol is a three-move interactive protocol:

1. The voting device shows some commitment `a`.
2. The voter enters a random challenge `e`.
3. The voting device responds to the challenge with an answer `z`.

Unfortunately, the zero-knowledge property of a Σ-protocol is honest-verifier only. It means the voter must be honest, or the coercer must be limited in its

instructions. If the coercer forces the voter to choose the challenge e in a particular way, it will be computationally infeasible to simulate a valid transcript. For example, if the coercer demands to use $\mathsf{e} = \mathsf{H}(\mathsf{a})$, then the voter has to obey. This problem has been previously reported in the literature: a solution proposed in [83] to achieve receipt-freeness was claimed to be secure because the transcript of a Σ-protocol "is not transferable". As noted in footnote 1 of [67], this is not true when the voter (who plays the role of the verifier in the Σ-protocol) is dishonest/coerced.

Using interactive OR-proof: Neff's approach also relies on an interactive protocol, yet enhanced by a printer that physically commits to values by printing them on a tape. The printer is a standard one with the only addition of a "shield" that temporarily hides some of the printed values from the voter. The interactive protocol itself goes as follows:

1. After the voter makes selections, the printer computes a verifiable ballot commitment and prints it.
2. The voter selects challenges for unselected options from a fixed small challenges space.
3. The printer finalizes proofs for unselected options and commits to them.
4. The voter selects a challenge from a small fixed set.
5. The printer finishes the proof.

To check the proof, the voter compares the tape against the information published on the bulletin board, which requires only visual inspection. The rest of the verifications (proofs validity, commitments correctness, etc.) can be delegated to anyone without breaking the voter's privacy.

While the article claims the proposed CAI method resists coercion, we do not believe the protocol would work outside the polling place. Recall that the voter must select challenges from a fixed set. If there is no physical shield hiding the printed commitments from the voter, the voter might select challenges from a pre-defined set of possibilities based on the commitment values. In such a case, it would be trivial for the coercer to identify the voter's vote.

Using chameleon-hash: Another idea is to rely on chameleon hashes so that the voting device can simulate non-interactive proofs and create fake transcripts [48]. The idea is for the voting device to create multiple transcripts, among which only one is real, and then to use the voter's challenge to modify the real proofs. The modification is such that only the actual transcript can remain valid after the change.

First, the voting device generates a trapdoor for the chameleon hash. Then it creates non-interactive simulated Σ-proofs for all unselected options and one real proof for the selected one. After that, it shows the commitment to real transcript to the voter. The voter responds with a challenge, with which the voting device modifies the first move of the actual Σ-proof. Finally, the voting device finalizes the modified transcript.

Since the modification results in valid proof only for an actual transcript, the CAI method works as long as the voting device cannot predict the challenge. The voter only has to remember one commitment and, in case of coercion, can lie that a

different value was displayed. However, the voter should be honest as the protocol is honest verifier zero-knowledge only.

Advantages and disadvantages of interactive proofs: The interactive proof is quite an elegant solution to CAI verification. The voter witnesses something that no other party, including coercer, sees. In a way, it implies that the voter becomes a designated verifier. Moreover, there is no way to prove what exactly someone saw.

However, currently, this approach only works for honest voters as the challenge must be independent of the previous protocol steps. One might force voters to behave by physically hiding the transcript, but it only works in the safety of a polling place. In the remote settings, the coercer can always instruct the voter to select the challenge as a function of some previous messages and, by doing so, take away from voters the possibility to disobey.

1.1.3.4 Designated Proof

The last group of CAI methods is designated verification. The notion of designated verification appeared back in 1996 [64] and encapsulated the idea of proving a statement only for a specific person and no one else. Typically it is achieved by constructing an OR-proof of the form: "statement is correct" OR "I'm the designated verifier" - where the latter implies knowledge of a secret only the designated verifier should know.

In e-voting, the designated verifier is a voter, which is assumed to have a secret already. Usually, that secret comes from a trusted party via a reliable communication channel.

This group consists of different flavors of OR-proof construction. We show two CAI verification ideas, which, while relying on the same principle, lead to different approaches to ballot correctness verification.

Standard OR proofs: The most classic approach is based on standard OR proof of the following form: "ballot contains the intended option" OR "I know the secret" [58]. The voter is assumed to have a secret key unknown to the voting device. Therefore, the only way the voting device can convince the voter is to encrypt the correct option.

This CAI verification technique is coercion-resistant only if the voter has the secret key. If the voter does not receive the trapdoor key or gets it too late - the coercer wins. Therefore, the existence of secure channels between voters and the authority responsible for trapdoors distribution is mandatory.

Simulated tracking numbers: Another different idea is to verify the ballot content after it is decrypted based on the associated tracking number. An e-voting scheme called Selene [98] (as well as its successor Hyperion [99]) implements this type of verification.

The voter votes as usual but needs to return for the CAI verification after all ballots are anonymized and decrypted. To obtain the tracking number, the voter combines shares from all trustees and uses a secret key to compute it. After that,

the voter can fake the tracking number to deceive the coercer by simulating one of the trustees' shares.

All in all, this CAI verification provides strong protection against coercion. However, it requires voters to be technically sophisticated and understand the transcript simulation procedure. Moreover, it only works when voters receive their designated secret key. Otherwise, they have no power to simulate the proof and hide disobedience.

Advantages and disadvantages of designated proofs: The designated CAI verification method implicitly assumes that the voter has a secret and no one else knows it until after the verification. If the entity responsible for generation is not trusted and collaborates with the voting device, it can cheat and convince the voter of anything. If the coercer intercepts the secret, he might deny the voter the possibility of faking the transcript directly or indirectly by supplying an incorrect trapdoor. An alternative, where a voter generates a trapdoor, opens the door to active coercion or intentional vote selling if a malicious voter willingly chooses to participate in the protocol without any knowledge of a trapdoor.

1.2 Thesis Contribution

This thesis presents the results of exploring the two contradictory properties of electronic voting: cast-as-intended verification and coercion-resistance. The contributions can be roughly divided into three parts: (1) study in the standard settings, (2) exploration of post-quantum cryptography, and (3) practical constructions and search for the limitations of both properties.

In the first part, we give an extensive overview of the current state of the art in electronic voting literature regarding those properties. Then, we put forward two formal definitions for achieving coercion-resistant cast-as-intended verification in settings without pre-exchanged data. After that, we present two practical constructions and prove their security under the proposed definitions. We also show the efficiency of our proposals by providing proof of the concept implementations.

In the second part, we switch to post-quantum settings and identify the usability issues rooted in the lattice-based math affecting both proposed solutions. To address those issues, we present a generic transformation that departs from an interactive zero-knowledge system (that might require multiple re-runs to complete the protocol) and obtains a 3-move zero-knowledge system (without re-runs). The transformation combines the well-known Fiat-Shamir technique with several initially exchanged messages. The resulting 3-move system enjoys honest-verifier zero-knowledge and can be easily turned into a fully deniable proof using standard techniques.

In the final part, we focus on the practical aspects of the coercion-resistant cast-as-intended verification. First, we present the case of a computationally limited voter, which we consider the most realistic one. We show that even a computationally limited voter can enjoy coercion-resistant cast-as-intended verification, but a help of a simple aid device for nonce generation is required. Also, we demonstrate that our generic definition easily adapts to the constraints of the limited voter. After

that, we present ongoing work that focuses on the cases of extreme coercion based on new and unexplored mechanisms such as delay encryption and blockchain. We show an advanced coercive attack on our first construction and describe an improvement to the second solution that reduces the number of interactions to an optimal three rounds.

To summarize, we start by studying coercion-resistant cast-as-intended verification in standard settings, which results in formal definitions and two practical solutions. Then we move into the post-quantum world, where we learn that an extra step is needed to preserve the usability of our previously proposed constructions, which results in the generic transformation to avoid protocol re-runs. After that, we concentrate on a computationally limited voter, which leads to another simple solution and shows the adaptability of our original definitions. Finally, we explore the extreme coercion threats to see the limits of coercion-resistant cast-as-intended verification, which resulted in a new coercion attack on the first construction and an upgrade of the second solution.

1.2.1 Publications Resulting From The Thesis

1. T. Finogina, J. Herranz and E. Larraia. *How (not) to achieve both coercion resistance and cast-as-intended verifiability in remote eVoting.* Proceedings of CANS'2021, Lecture Notes in Computer Science, Vol. 13099, pp. 483-491, 2021 [50]
2. X. Arnal, A. Cano, T. Finogina and J. Herranz. *How to avoid repetitions in lattice-based deniable zero-knowledge proofs.* Proceedings of NordSec'2022, Lecture Notes in Computer Science, Vol. 13700, pp. 253-269, 2022 [13]
3. T. Finogina and J. Herranz. *Coercion-resistant cast-as-intended verifiability for computationally limited voters.* Proceedings of VOTING'2023 (workshop of Financial Cryptography 2023), Lecture Notes in Computer Science, 2023
4. T. Finogina and J. Herranz. *On Remote Electronic Voting with both Coercion-Resistance and Cast-as-Intended Verifiability.* Journal of Information Security and Applications, Volume 76, August 2023 [49]
5. (submitted) T. Finogina, N. Costa, J.Cucurull. *Selective comparison of E2E e-voting systems.* Preprint submitted to Journal of Systems and Software
6. (in preparation) P. Ronne, T. Finogina and J. Herranz. *Strong Coercion Settings in eVoting: Attacks and Solutions.*

1.3 Structure Of This Thesis

Chapter 2 introduces the notations used throughout the thesis and gives the necessary cryptography background for a better understanding of the following chapters. The chapter includes standard cryptographic definitions relevant to the thesis.

Chapter 3 formally defines coercion-resistant cast-as-intended verification in settings without secure delivery channels. Also, we prove that, for fulfilling both properties, a voting protocol requires at least three rounds of interaction.

Chapter 4 describes two satisfactory solutions for achieving coercion-resistant cast-as-intended verification. Also, the chapter shows how those solutions can be

instantiated in ElGamal settings. Finally, we give all the necessary implementational details and benchmarks.

Chapter 5 explores post-quantum settings and presents the necessary background for understanding lattice basics. Then we highlight a usability problem - protocol repetitions - that arises when the solutions from Chapter 4 are moved from ElGamal to lattice-based encryption. We present a simple generic transformation that removes protocol re-runs from any interactive lattice-based proof system. Finally, we describe an implementation and relevant benchmark to justify that our solution is not posing any performance issues.

Chapter 6 looks at the real-case applications where voters are severely limited in computational power. We provide a solution where the only things voters have to do are: remember and compare strings of numbers, on the one hand, and press a button at the appropriate moment, on the other hand.

Chapter 7 presents an ongoing work focusing on extreme coercion threats. It identifies an advanced attack on our first construction and shows an improvement in the number of rounds for the second solution.

In Chapter 8, we end this thesis, summarize the results of our research, and discuss some open questions.

For readers' convenience, Appendix A provides a detailed description of all e-voting schemes mentioned in the thesis.

Chapter 2

Preliminaries

We will use bold to denote vectors (column vectors by default) and bold capital letters for matrices. Also we write $\mathbf{e} \xleftarrow{\$} \mathbb{Z}_{\mathsf{q}}^n$ to mean that a vector $\mathbf{e}$ of n components is chosen uniformly at random from $\mathbb{Z}_{\mathsf{q}}$.

As usual, we say that a function ϵ is negligible in $n \in \mathbb{N}$, and write $\epsilon \in \mathsf{negl}(n)$, if for every polynomial $p(\cdot)$ there exists N such that for every $n \geq N$ it holds $\epsilon(n) \leq 1/p(n)$. By $x \hookleftarrow X$ we denote taking an element x from a finite set X uniformly at random.

2.1 Relations And Languages

Let $\mathcal{R}$ be a polynomial-time decidable binary relation defined on a pair of finite spaces. That is, $\mathcal{R}$ is a subset of $\mathcal{X} \times \mathcal{W}$ subject to the following conditions [57]:

- There exists a polynomial p such that if $(x, w) \in \mathcal{R}$ then $|w| \leq p(|x|)$
- There exists an efficient algorithm $\mathtt{G}$ that on input $\lambda \in \mathbb{N}$ outputs $(x, w) \in \mathcal{R}$ with $|x| \geq \lambda$.
- There exists an efficient algorithm $\mathtt{R}$ that outputs a bit such that $\mathtt{R}(x, w) = 1 \Leftrightarrow (x, w) \in \mathcal{R}$.

Such relation $\mathcal{R}$ is an NP-relation, and gives rise to the set of "yes"-instances defined as $\mathcal{L}_{\mathcal{R}} = \{x \in \mathcal{X} \mid \exists w \in \mathcal{W} \; s.t. \; \mathtt{R}(x, w) = 1\} \subset \mathcal{X}$, known as the language of $\mathcal{R}$. Also, $\mathcal{R}$ is believed to be hard if there are no known efficient algorithms that can recover w from x with non-negligible probability on λ.

The language $\mathcal{L}_{\mathcal{R}} = \{\mathsf{x} \in \mathcal{X} \mid \exists \omega \in \mathcal{W} \; s.t. \; \mathtt{R}(\mathsf{x}, \omega) = 1\} \subset \mathcal{X}$ associated to a binary relation $\mathcal{R}$ is hard-to-distinguish inside the space $\mathcal{X}$ if, given a random element in $\mathcal{X}$, it is hard to distinguish in polynomial time if the element belongs to $\mathcal{L}_{\mathcal{R}}$ or not. That is, for any polynomial time distinguisher $\mathcal{D}_{\mathcal{R}}$, it holds

$$|\Pr[1 \leftarrow \mathcal{D}_{\mathcal{R}}(\mathsf{x}) \mid \mathsf{x} \hookleftarrow \mathcal{L}_{\mathcal{R}}] \; - \; \Pr[1 \leftarrow \mathcal{D}_{\mathcal{R}}(\mathsf{x}') \mid \mathsf{x}' \hookleftarrow \mathcal{X} - \mathcal{L}_{\mathcal{R}}]| \in \mathsf{negl}(\lambda)$$

where λ is the security parameter of the system.

2.2 Groups And Hard Problems

Definition 1 (Groups)**.** A cyclic group $\mathbb{G}$ is a group in which every element can be generated by a single element g called a group generator. In other words, one can write every element in $\mathbb{G}$ in the form g^i, where i is some integer and g^i is a self-apply operation (e.g., scalar multiplication for elliptic curves and modular exponentiation for Schnorr groups). The order of $\mathbb{G}$ is the smallest positive integer q such that $\mathsf{g}^{\mathsf{q}} = 1$.

Definition 2 (The Discrete Logarithm Problem)**.** The discrete logarithm problem (DL) in a cyclic group $\mathbb{G}$ of order q with a generator g, informally, requires finding an integer x such that for the given element $\mathsf{a} \in \mathbb{G}$ it holds that $\mathsf{g}^{\mathsf{x}} = \mathsf{a}$. Figure 2.1 describes the DL game formally.

DL is hard in the group $\mathbb{G}$ if $\mathsf{Adv}_{\mathcal{A}}^{\mathbb{G},\mathsf{DL}} = \Pr\left[1 \leftarrow \mathsf{DL}_{\mathcal{A}}^{\mathbb{G}}(\lambda)\right] \approx 0$.

$\underline{\mathsf{DL}_{\mathcal{A}}(\mathbb{G})}$:
1. $\mathsf{x} \xleftarrow{\$} \mathbb{Z}_{\mathsf{q}}$
2. $\mathsf{a} = \mathsf{g}^{\mathsf{x}}$
3. $\mathsf{x}' \leftarrow \mathcal{A}(\mathsf{g}, \mathsf{a})$
4. Output 1 if $\mathsf{x} = \mathsf{x}'$. Else output 0.

Figure 2.1: DL game for the cyclic group $\mathbb{G}$ of the order q with a generator g.

The typical examples of groups where the discrete logarithm problem is believed to be hard are:

- subgroups of order q of $\mathbb{Z}_{\mathsf{p}}^*$, for q and p both being primes.
- subgroups of q points of an elliptic curve.

Definition 3 (The Decisional Diffie-Hellman Problem)**.** The decisional Diffie-Hellman problem (DDH) in a cyclic group $\mathbb{G}$ of order q with a generator g, informally, requires the indistinguishability of $\mathsf{g}_1^{\mathsf{s}}$ from a random element, given $(\mathsf{g}^{\mathsf{s}}, \mathsf{g}, \mathsf{g}_1)$ for random $\mathsf{s} \in \mathbb{Z}_q^*$ and random $\mathsf{g}_1 \in \mathbb{G}$. Figure 2.2 describes the DDH game formally.

The group $\mathbb{G}$ is DDH-hard if it holds:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathbb{G},\mathsf{DDH}} = |\Pr\left[1 \leftarrow \mathsf{DDH}_{\mathcal{A}}^{\mathbb{G}}(\lambda)\right] - \frac{1}{2}| \approx 0.$$

The typical examples of groups where the DDH problem is believed to be hard are:

- groups of quadratic residues $\mathbb{Q}_{\mathsf{p}} \subset \mathbb{Z}_{\mathsf{p}}^*$ of prime order q, where modulus $\mathsf{p} = 2\mathsf{q} + 1$ a safe prime of a sufficient length λ.
- subgroups of q points of an elliptic curve.

2.3 Homomorphic Public Key Encryption

A probabilistic public key encryption scheme $\mathbb{E}$ consist of three protocols $\mathbb{E} = (\mathbb{E}.\mathsf{KG}, \mathbb{E}.\mathsf{Enc}, \mathbb{E}.\mathsf{Dec})$ defined for a message space $\mathcal{M}_{\mathbb{E}}$, a randomness space $\mathcal{RS}_{\mathbb{E}}$ and a ciphertext space $\mathcal{C}_{\mathbb{E}}$ as follows:

$\underline{\mathsf{DDH}_{\mathcal{A}}(\mathbb{G})}$:
1. $\mathsf{b} \leftarrow \{0,1\}$
2. $\mathsf{s} \leftarrow \mathbb{Z}_{\mathsf{q}}$
3. $\mathsf{s}_0 = \mathsf{s}_1 = \mathsf{s}$
4. $\mathsf{g}_1 \leftarrow \mathbb{G}$ // Sampled at random
5. if $\mathsf{b} = 1$ then
   6.1 $\mathsf{s}_0, \mathsf{s}_1 \leftarrow \mathbb{Z}_{\mathsf{q}}$ // Sampled at random
6. $\mathsf{b}' \leftarrow \mathcal{A}(\mathsf{g}, \mathsf{g}_1, \mathsf{g}^{\mathsf{s}_0}, \mathsf{g}_1^{\mathsf{s}_1})$
7. Output 1 if $\mathsf{b} = \mathsf{b}'$. Else output 0.

Figure 2.2: DDH game for the cyclic group $\mathbb{G}$ of the order q with a generator g.

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathbb{E}.\mathsf{KG}(\lambda)$ is a key pair generation algorithm that takes as input a security parameter λ and outputs a public pk and secret sk keys of the encryption scheme $\mathbb{E}$.
- $\mathtt{ctxt} \leftarrow \mathbb{E}.\mathsf{Enc}(\mathsf{m}, \mathsf{r})$ is an encryption algorithm that takes as input a message $\mathsf{m} \in \mathcal{M}_{\mathbb{E}}$ and randomness $\mathsf{r} \in \mathcal{RS}_{\mathbb{E}}$ and generates a ciphertext $\mathtt{ctxt} \in \mathcal{C}_{\mathbb{E}}$. We omit pk as a well-known input for simplicity.
- $\mathsf{m}' \leftarrow \mathbb{E}.\mathsf{Dec}(\mathtt{ctxt}, \mathsf{sk})$ is a decryption algorithm that takes as input a ciphertext $\mathtt{ctxt} \in \mathcal{C}_{\mathbb{E}}$ and a secret key sk and outputs the result of the decryption $\mathsf{m}' \in \mathcal{M}_{\mathbb{E}}$ or an error if $\mathsf{m}' \notin \mathcal{M}_{\mathbb{E}}$.

Security properties: We say that $\mathbb{E}$ has a *perfect correctness* if for any given pair $(\mathsf{pk}, \mathsf{sk})$ generated by the algorithm KG and any randomness $\mathsf{r} \in \mathcal{RS}_{\mathbb{E}}$, it holds that $\mathbb{E}.\mathsf{Dec}\big(\mathbb{E}.\mathsf{Enc}(\mathsf{m}, \mathsf{r}), \mathsf{sk}\big) = \mathsf{m}$ for any $\mathsf{m} \in \mathcal{M}_{\mathbb{E}}$.

Another important property for a public key encryption scheme is *semantic security*. Roughly speaking, this property says that any adversary that sees a public key, chooses a pair of plaintexts $\mathsf{m}_0, \mathsf{m}_1$ and receives an encryption of one of the two (chosen at random) cannot distinguish which is the encrypted plaintext with better probability than $1/2$. In particular, a public key encryption scheme with such a property must be probabilistic: there exists a randomness space $\mathcal{RS}_{\mathbb{E}}$ and the encryption protocol $\mathbb{E}.\mathsf{Enc}$ chooses some random value(s) in $\mathcal{RS}_{\mathbb{E}}$ in the process of computing the ciphertext.

Finally, an encryption scheme is *homomorphic* if for a public key pk, messages $\mathsf{m}_1, \mathsf{m}_2$ and randomnesses $\mathsf{r}_1, \mathsf{r}_2$ the encryption function $\mathbb{E}.\mathsf{Enc}$ satisfies $\mathbb{E}.\mathsf{Enc}(\mathsf{m}_1 \cdot \mathsf{m}_2; \mathsf{r}_1 + \mathsf{r}_2) = \mathbb{E}.\mathsf{Enc}(\mathsf{m}_1; \mathsf{r}_1) \cdot \mathbb{E}.\mathsf{Enc}(\mathsf{m}_2; \mathsf{r}_2)$.

Binary relation: For such a probabilistic encryption scheme $\mathbb{E}$, we can consider the binary relation $\mathcal{R}_{\mathbb{E}}$ defined on sets $\mathcal{X} = \mathcal{M}_{\mathbb{E}} \times \mathcal{C}_{\mathbb{E}}$ and $\mathcal{W} = \mathcal{RS}_{\mathbb{E}}$ as $(\mathsf{x}, \omega) \in \mathcal{R}_{\mathbb{E}} \Leftrightarrow \mathtt{ctxt} = \mathbb{E}.\mathsf{Enc}(\mathsf{m}, \mathsf{r})$, where $\mathsf{x} = (\mathsf{m}, \mathtt{ctxt}), \omega = \mathsf{r}$. Note that, if $\mathbb{E}$ is semantically secure, then the associated language $\mathcal{L}_{\mathbb{E}}$ is hard-to-distinguish in a quite specific way: elements $\mathsf{x} = (\mathsf{m}, \mathtt{ctxt}) \in \mathcal{L}_{\mathbb{E}}$ are indistinguishable from elements $\mathsf{x}' = (\mathsf{m}', \mathtt{ctxt}) \in \mathcal{X} - \mathcal{L}_{\mathbb{E}}$, even if the second component (the ciphertext) is the same.

2.3.1 ElGamal Encryption

We use ElGamal encryption scheme [46], which defines the triple of algorithms $(\mathbb{E}.\mathsf{KG}, \mathbb{E}.\mathsf{Enc}, \mathbb{E}.\mathsf{Dec})$ as follows:

$\mathbb{E}$.KG: Defines a cyclic group $\mathbb{G}_{\mathsf{q}}$ of prime order q with a generator g. Then the algorithm samples a secret key $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_{\mathsf{q}} \setminus \{0\}$ uniformly at random and computes a public key $\mathsf{pk} = \mathsf{g}^{\mathsf{sk}} \in \mathbb{G}_{\mathsf{q}}$.

$\mathbb{E}$.Enc: To encrypt a message $\mathsf{m} \in \mathbb{G}_{\mathsf{q}}$, the algorithm samples a randomness $\mathsf{r} \xleftarrow{\$} \mathbb{Z}_{\mathsf{q}}$ uniformly at random and computes the ciphertext $\mathtt{ctxt} = (\mathsf{c}_1, \mathsf{c}_2) = (\mathsf{g}^{\mathsf{r}}, \mathsf{m} \cdot \mathsf{pk}^{\mathsf{r}})$.

$\mathbb{E}$.Dec: To recover the message $\mathsf{m}^* \in \mathbb{G}_{\mathsf{q}}$, the algorithm computes the following $\mathsf{m}^* = \mathsf{c}_2 \cdot (\mathsf{c}_1^{\mathsf{sk}})^{-1}$.

The ElGamal encryption scheme is *perfectly correct*, *semantically secure* (assuming the hardness of the DDH problem in $\mathbb{G}$) and *homomorphic* [46].

2.4 Commitment Scheme

A commitment scheme Com consist of three protocols ($\mathsf{Com.Gen}$, $\mathsf{Com.Eval}$, $\mathsf{Com.Open}$) defined for a message space $\mathcal{M}_{\mathsf{Com}}$, a randomness space $\mathcal{RS}_{\mathsf{Com}}$, and a commitment space $\mathcal{C}_{\mathsf{Com}}$ as follows:

- $\mathsf{pms}_{\mathsf{Com}} \leftarrow \mathsf{Com.Gen}(\lambda)$ is a PPT setup algorithm that takes as input a security parameter λ and outputs public parameters $\mathsf{pms}_{\mathsf{Com}}$.
- $\mathsf{cmt} \leftarrow \mathsf{Com.Eval}(\mathsf{m}, \mathsf{r})$ is a PPT evaluation protocol that takes as input a plaintext $\mathsf{m} \in \mathcal{M}_{\mathsf{Com}}$ and randomness $\mathsf{r} \in \mathcal{RS}_{\mathsf{Com}}$, and outputs a commitment $\mathsf{cmt} \in \mathcal{C}_{\mathsf{Com}}$. We omit $\mathsf{pms}_{\mathsf{Com}}$ as a well-known input for simplicity.
- $\mathsf{b} \leftarrow \mathsf{Com.Open}(\mathsf{cmt}, \mathsf{m}, \mathsf{r})$ is a deterministic opening protocol that takes as input a commitment $\mathsf{cmt} \in \mathcal{C}_{\mathsf{Com}}$, a plaintext $\mathsf{m} \in \mathcal{M}_{\mathsf{Com}}$ and randomness $\mathsf{r} \in \mathcal{RS}_{\mathsf{Com}}$, and outputs a bit $\mathsf{b} \in \{0, 1\}$.

Secutity properties of commitments: Informally, we say that a commitment scheme is *hiding* if a commitment cmt does not leak any information about the committed message m.

Definition 4 (Hiding)**.** A commitment scheme is ϵ-hiding if for all algorithms $\mathcal{A}$

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hide,Com}} = |\Pr\left[1 \leftarrow \mathsf{Hide}_{\mathcal{A}}^{\mathsf{Com}}(\lambda)\right] - \frac{1}{2}| < \epsilon$$

$\underline{\mathsf{Hide}_{\mathcal{A}}^{\mathsf{Com}}(1^{\lambda})}$:
1. $\mathsf{pms}_{\mathsf{Com}} \leftarrow \mathsf{Com.Gen}(\lambda)$
2. $(\mathsf{m}_0, \mathsf{m}_1) \leftarrow \mathcal{A}(\mathsf{pms}_{\mathsf{Com}})$
3. $\mathsf{b} \xleftarrow{\$} \{0, 1\}$
4. $\mathsf{r} \xleftarrow{\$} \mathcal{RS}_{\mathsf{Com}}$
5. $\mathsf{cmt} \leftarrow \mathsf{Com.Eval}(\mathsf{m}_{\mathsf{b}}, \mathsf{r})$
6. $\mathsf{b}' \leftarrow \mathcal{A}(\mathsf{pms}_{\mathsf{Com}}, \mathsf{cmt})$
7. Output 1 if $\mathsf{b} = \mathsf{b}'$. Else output 0.

Figure 2.3: A game against the hiding property of the commitment scheme Com.

If the algorithm $\mathcal{A}$ is restricted to polynomial-time algorithms, then the scheme is called *computationally hiding*. If there are no restrictions on the running time of such algorithms (i.e., $\mathcal{A}$ is computationally unbounded), then the scheme is *statistically hiding*. If $\mathcal{A}$ is computationally unbounded and $\epsilon = 0$, then the commitment is *perfectly hiding*.

Informally we say that a commitment scheme is *binding* if an adversary cannot open a commitment in two different ways.

Definition 5 (Binding)**.** A commitment scheme is ϵ-binding if for all algorithms $\mathcal{A}$

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Bind},\mathsf{Com}} = \Pr\left[1 \leftarrow \mathsf{Hide}_{\mathcal{A}}^{\mathsf{Com}}(\lambda)\right] < \epsilon$$

$\underline{\mathsf{Hide}_{\mathcal{A}}^{\mathsf{Com}}(1^{\lambda})}$:
1. $\mathsf{pms}_{\mathsf{Com}} \leftarrow \mathsf{Com.Gen}(\lambda)$
2. $(\mathsf{m}, \mathsf{m}', \mathsf{r}, \mathsf{r}', \mathsf{cmt}) \leftarrow \mathcal{A}(\mathsf{pms}_{\mathsf{Com}})$
3. If $\mathsf{Com.Open}(\mathsf{cmt}, \mathsf{m}, \mathsf{r}) = \mathsf{Com.Open}(\mathsf{cmt}, \mathsf{m}', \mathsf{r}') = 1$ and $\mathsf{m} \neq \mathsf{m}'$, output 1. Else output 0.

Figure 2.4: A game against the binding property of the commitment scheme Com.

If the algorithm $\mathcal{A}$ is restricted to polynomial-time algorithms, then the scheme is called *computationally binding*. If there are no restrictions on the running time of such algorithms (i.e., $\mathcal{A}$ is computationally unbounded), then the scheme is *statistically binding*. If $\mathcal{A}$ is computationally unbounded and $\epsilon = 0$, then the commitment is *perfectly binding*.

A commitment scheme cannot be perfectly hiding and perfectly binding at the same time.

Finally, we say that commitment scheme is *homomorphic*, if it holds that $\mathsf{Com.Eval}(\mathbf{a}+\mathbf{b}; \mathsf{r}+\mathsf{s}) = \mathsf{Com.Eval}(\mathbf{a}; \mathsf{r})\mathsf{Com.Eval}(\mathbf{b}; \mathsf{s})$ for messages $\mathbf{a}, \mathbf{b}$, a commitment key h and random values r, s.

Pedersen Commitment Scheme: We use the Pedersen commitment scheme [90], which defines the triple of algorithms $(\mathsf{Com.Gen}, \mathsf{Com.Eval}, \mathsf{Com.Open})$ as follows:

$\mathsf{Com.Gen}$: Creates a public commitment key $\mathsf{h} = (\mathbb{G}_{\mathsf{q}}, \mathsf{g}, \mathsf{h}_1, \ldots, \mathsf{h}_n)$, where $\mathsf{g}, \mathsf{h}_1, \ldots, \mathsf{h}_n$ are generators of the group $\mathbb{G}_{\mathsf{q}}$ computed in a verifiable manner.

$\mathsf{Com.Eval}$: To commit to $\mathbf{m} = (\mathsf{m}_1, \ldots, \mathsf{m}_n) \in \mathbb{Z}_{\mathsf{q}}^n$, we sample the randomness $\mathsf{r} \xleftarrow{\$} \mathbb{Z}_{\mathsf{q}}$ and compute $\mathsf{com} = \mathsf{g}^{\mathsf{r}} \prod_{i=1}^{n} \mathsf{h}_i^{\mathsf{m}_i}$. We can also commit to a vector $\mathbf{m}$ with less than n elements by padding it with 0s.

$\mathsf{Com.Open}$: To verify that a commitment com conrresponds to the vector of messages $\mathbf{m} = (\mathsf{m}_1, \ldots, \mathsf{m}_n) \in \mathbb{Z}_{\mathsf{q}}^n$ and randomness r, the verifier computes $\mathsf{com}^* = \mathsf{g}^{\mathsf{r}} \prod_{i=1}^{n} \mathsf{h}_i^{\mathsf{m}_i}$ and checks that $\mathsf{com}^* = \mathsf{com}$.

The Pedersen commitment scheme is *perfectly hiding*, *computationally binding* (assuming the hardness of the Discrete Logarithm problem in $\mathbb{G}$) and *homomorphic* [90].

2.5 Interactive Zero-Knowledge Systems

Let $\mathcal{R}$ be a binary relation parameterized with a security parameter λ. An interactive protocol $\langle \mathtt{P}, \mathtt{V} \rangle$ between a prover $\mathtt{P}$ and a verifier $\mathtt{V}$, both being PPT algorithms, is said to be a *zero-knowledge* proof system for $\mathcal{R}$ if the following holds [57]:

Completeness: On common input x, and Prover's private input ω, if $(\mathsf{x}, \omega) \in \mathcal{R}$ it holds $\Pr[\langle \mathtt{P}(\mathsf{x}, \omega), \mathtt{V}(\mathsf{x}) \rangle = 1] = 1$.

Soundness: For every $\mathsf{y} \notin \mathcal{L}_{\mathcal{R}}$ it holds $\Pr[\langle \mathtt{P}(\mathsf{y}), \mathtt{V}(\mathsf{y}) \rangle = 1] \in \mathsf{negl}(\lambda)$.

Perfect zero-knowledge: For every verifier $\mathtt{V}^*$ there exists a PPT simulator $\mathtt{Sim}_{\mathtt{V}^*}$ such that for every $(\mathsf{x}, \omega) \in \mathcal{R}$ the output $\langle \mathtt{P}(\mathsf{x}, \omega), \mathtt{V}^*(\mathsf{x}) \rangle$ is identically distributed to the output $\mathtt{Sim}_{\mathtt{V}^*}(\mathsf{x})$. This property can be relaxed requiring that the outputs are only statistically or computationally close.

Simulating Execution of ZK Proof Systems for Instances Out of the Language: Let us assume that there exists a proof system for $\mathcal{R}$ enjoying zero-knowledge: for every verifier $\mathtt{V}^*$ there exists a PPT simulator $\mathtt{Sim}_{\mathtt{V}^*}$ such that for every $(\mathsf{x}, \omega) \in \mathcal{R}$ the distribution $\langle \mathtt{P}(\mathsf{x}, \omega), \mathtt{V}^*(\mathsf{x}) \rangle$ is (perfectly / statistically / computationally) indistinguishable to the distribution $\mathtt{Sim}_{\mathtt{V}^*}(\mathsf{x})$.

We claim that such a simulator $\mathtt{Sim}_{\mathtt{V}^*}$ must also work when the input is an element $\mathsf{x}' \in \mathcal{X} - \mathcal{L}_{\mathcal{R}}$ out of the language $\mathcal{L}_{\mathcal{R}}$, if $\mathcal{L}_{\mathcal{R}}$ is a hard-to-distinguish language, and that the joint distributions $(\mathsf{x}, \mathtt{Sim}_{\mathtt{V}^*}(\mathsf{x}))_{\mathsf{x} \hookleftarrow \mathcal{L}_{\mathcal{R}}}$ and $(\mathsf{x}', \mathtt{Sim}_{\mathtt{V}^*}(\mathsf{x}'))_{\mathsf{x}' \hookleftarrow \mathcal{X} - \mathcal{L}_{\mathcal{R}}}$ are indistinguishable.

Lemma 2.5.1. The two joint distributions $(\mathsf{x}, \mathtt{Sim}_{\mathtt{V}^*}(\mathsf{x}))_{\mathsf{x} \hookleftarrow \mathcal{L}_{\mathcal{R}}}$ and $(\mathsf{x}', \mathtt{Sim}_{\mathtt{V}^*}(\mathsf{x}'))_{\mathsf{x}' \hookleftarrow \mathcal{X} - \mathcal{L}_{\mathcal{R}}}$ are indistinguishable.

Proof. Let us assume the existence of a distinguisher algorithm $\mathcal{D}$ that can distinguish these two distributions. We use $\mathcal{D}$ to build a distinguisher $\mathcal{D}_{\mathcal{R}}$ against the hard-to-distinguish property of $\mathcal{L}_{\mathcal{R}}$, as follows.

$\mathcal{D}_{\mathcal{R}}$ receives as input an element $\tilde{\mathsf{x}} \in \mathcal{X}$, and its goal is to distinguish if $\tilde{\mathsf{x}} \in \mathcal{L}_{\mathcal{R}}$ or if $\tilde{\mathsf{x}} \in \mathcal{X} - \mathcal{L}_{\mathcal{R}}$. What $\mathcal{D}_{\mathcal{R}}$ does is to run the simulator $\mathtt{Sim}_{\mathtt{V}^*}(\tilde{\mathsf{x}})$. If this simulator aborts, $\mathcal{D}_{\mathcal{R}}$ answers $\tilde{\mathsf{x}} \in \mathcal{X} - \mathcal{L}_{\mathcal{R}}$. Otherwise, $\mathcal{D}_{\mathcal{R}}$ runs the distinguisher $\mathcal{D}$ with the tuple $(\tilde{\mathsf{x}}, \mathtt{Sim}_{\mathtt{V}^*}(\tilde{\mathsf{x}}))$ as input, and outputs the same answer as $\mathcal{D}$ outputs. □

Definition 6 (Public coin)**.** An interactive protocol between a prover $\mathtt{P}$ and a verifier $\mathtt{V}$ is public-coin if all the messages sent by the verifier to the prover are randomly and independently sampled from the messages sent by the prover (that is, random coins from the verifier are publicly available).

Σ-protocols: Σ-protocol $\langle \mathtt{P}(\mathsf{x}, \omega), \mathtt{V}(\mathsf{x}) \rangle$ is a popular and powerful subset of interactive protocols that require three rounds of communication between the prover $\mathtt{P}$ in possession of the witness ω and the verifier $\mathtt{V}$ for proving the validity of a statement x as described in Figure 2.5. During the first round, the prover runs Σ_1 to sample a random opening op and commit to it (the resulting commitment denoted as com). Then it sends com to the verifier, who replies with a random challenge e from the

challenge space Chl (with size exponential in λ). Finally, the prover runs Σ_2 to reply to the challenge with the answer z. The validity of the resulting transcript $(\mathsf{com}, \mathsf{e}, \mathsf{z})$ can be checked by anyone running the verification algorithm Verify, which will output $\top$ in case of success and $\bot$ otherwise.

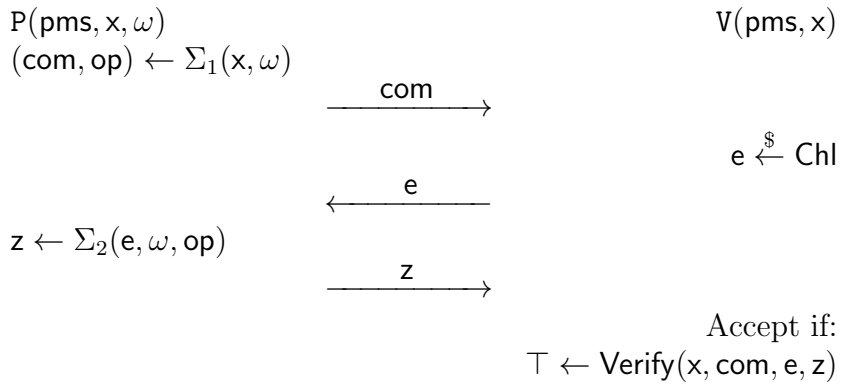


Figure 2.5: Σ-protocol for proving the statement x.

The required properties for a Σ-*protocol*, other than completeness, are [40]:

Special soundness: There exists a PPT algorithm $\mathtt{K}$ such that

$$\Pr[(\mathsf{x}, \omega) \in \mathcal{R} \mid \omega \leftarrow \mathtt{K}(\mathsf{x}, (\mathsf{com}, \mathsf{e}, \mathsf{z}), (\mathsf{com}, \mathsf{e}', \mathsf{z}'))] \geq 1 - \mathsf{negl}(t),$$

for any $\mathsf{x} \in \mathcal{L}_{\mathcal{R}}$ and any two accepting transcripts $(\mathsf{com}, \mathsf{e}, \mathsf{z})$, $(\mathsf{com}, \mathsf{e}', \mathsf{z}')$.

Special honest-verifier zero-knowledge: There exists a PPT algorithm $\mathtt{Sim}_{\mathcal{R}}$ such that, for any $(\mathsf{x}, \omega) \in \mathcal{R}$, on input x and a random e, outputs an accepting transcript $(\mathsf{com}, \mathsf{e}, \mathsf{z})$ with the same probability distribution as $\langle \mathtt{P}(\mathsf{x}, \omega), \mathtt{V}(\mathsf{x}) \rangle$.

Proof of the correct encryption: We can define a relation for proving the correctness of encryption using Σ-protocol $\langle \mathcal{VD}, \mathcal{V} \rangle$ between the voter and the voting device as follows:

$$\mathcal{L}_{\mathbb{E}} = \{(\mathsf{m}, \mathtt{ctxt}) \mid \mathtt{ctxt} \text{ is encryption of } \mathsf{m}\}$$

If the homomorphic encryption scheme $\mathbb{E}$ is IND-CPA secure, then $\mathcal{R} = \mathcal{R}_{\mathbb{E}}$ is a hard relation, and Lemma 2.5.1 ensure that $\mathtt{Sim}$ also works for instances out of $\mathcal{L}_{\mathbb{E}}$. It means that we can use $\mathtt{Sim}(\mathsf{m}^*, \mathtt{ctxt}, \mathsf{e}^*)$ to produce an indistinguishable transcript $(\mathsf{com}^*, \mathsf{e}^*, \mathsf{z}^*)$ for a given $\mathsf{e}^* \in \mathsf{Chl}$ and for m^* such that $\mathbb{E}.\mathsf{Dec}(\mathtt{ctxt}, \mathsf{sk}) \neq \mathsf{m}^*$. The Σ-protocol for proving the encryption correctness is detailed in Figure 2.6.

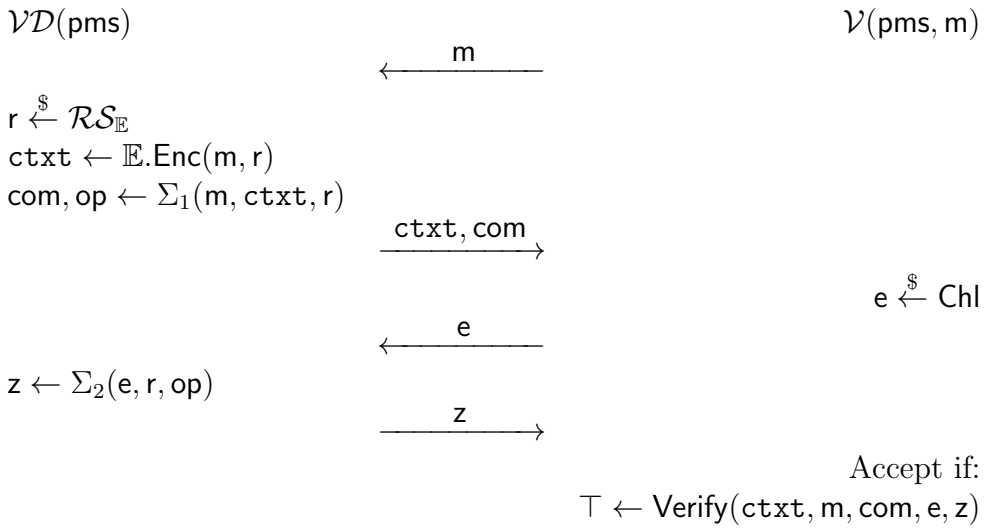


Figure 2.6: Σ-protocol for proving the ciphertext encrypts the intended message.

For the case of ElGamal encryption, the detailed protocol is presented in Figure 2.7. The public parameters pms contain the group $\mathbb{G}$, the group generator g, the public key pk, etc. Both prover and verifier know the message m and the ciphertext $\mathtt{ctxt} = (\mathsf{c}_1, \mathsf{c}_2)$. However, only the prover has the encryption randomness r.

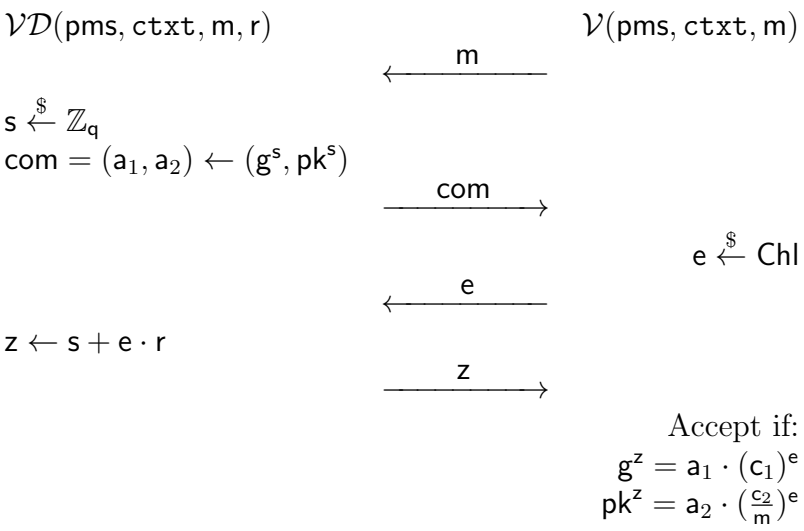


Figure 2.7: Σ-protocol for proving the ElGamal encryption correctness.

$\mathtt{P}$ chooses $\mathsf{s} \xleftarrow{\$} \mathbb{Z}_{\mathsf{q}}$ at random and sends the commitment $\mathsf{com} = (\mathsf{a}_1, \mathsf{a}_2)$ to $\mathtt{V}$, where $\mathsf{a}_1 = \mathsf{g}^{\mathsf{s}}$ and $\mathsf{a}_2 = \mathsf{pk}^{\mathsf{s}}$. After that, $\mathtt{V}$ selects and sends a random challenge $\mathsf{e} \in \mathsf{Chl} = \mathbb{Z}_{\mathsf{q}}$. Finally, $\mathtt{P}$ computes and sends the last element $\mathsf{z} = \mathsf{s} + \mathsf{e} \cdot \mathsf{r}$. $\mathtt{V}$ accepts the proof if and only if $\mathsf{g}^{\mathsf{z}} = \mathsf{a}_1 \cdot \mathsf{c}_1^{\mathsf{e}}$ and $\mathsf{pk}^{\mathsf{z}} = \mathsf{a}_2 \cdot (\frac{\mathsf{c}_2}{\mathsf{m}})^{\mathsf{e}}$.

The protocol enjoys the honest-verifier zero-knowledge property: the simulator $\mathtt{Sim}$, when given as input an element $\mathsf{x} = (\mathsf{g}, \mathsf{pk}, \mathsf{c}_1, \frac{\mathsf{c}_2}{\mathsf{m}^*})$ for m^* such that $\mathbb{E}.\mathsf{Dec}(\mathtt{ctxt}, \mathsf{sk}) \neq \mathsf{m}^*$ and a random challenge e^*, can simulate the fake transcript $(\mathsf{com}^*, \mathsf{e}^*, \mathsf{z}^*)$, by taking $\mathsf{z}^* \xleftarrow{\$} \mathbb{Z}_{\mathsf{q}}$ at random and computing $\mathsf{a}_1^* = \mathsf{g}^{\mathsf{z}^*} \cdot \mathsf{c}_1^{-\mathsf{e}^*}$ and $\mathsf{a}_2^* = \mathsf{pk}^{\mathsf{z}^*} \cdot \frac{\mathsf{c}_2}{\mathsf{m}^*}^{-\mathsf{e}^*}$.

Chapter 3

Defining Coercion-Resistant Cast-As-Intended Verifability

In this chapter, we formally define coercion-resistance and cast-as-intended verification. Before putting forward the definitions, in Section 3.1, we discuss why we cannot cover all possible e-voting scheme designs and have to limit ourselves to some subset with given characteristics. Then, in Section 3.2, we discuss the settings we selected and the reasons behind our choice. After that, in Section 3.3, we talk about the syntax and protocols we require. Finally, we give definitions in Section 3.4 and discuss how to fulfill them in Section 3.5, focusing on the number of interactions between the voter and the voting device.

3.1 Can We Have One Standard Definition For Both Properties?

As we mentioned in the introduction (see Chapter 1), there are dozens of ways to define adversaries against the freedom of will ranging from the strongest variant of coercion-resistance, which covers forced abstention attacks [67], to the weakest form of receipt-freeness, which is barely different from ballot privacy [108]. Similarly, there are dozens of mechanisms to provide cast-as-intended verification. Yet, our goal is to give one more definition, which would combine both properties. The natural question to ask is: Why do we need another definition?

If we look closely, we will see that each of the existing definitions varies in the adversary strength, trust assumptions, and capabilities of voters. The JCJ definition [67] says that the coercer can stop the voter from voting and even demand voting credentials. The cast-or-challenge verification method [18] requires voters to have a verification device and understand the verification process. The vote verification proposed in Demos [71] trust that voters will make a random choice between two ballot sides. The return-code-based verification [105] works only if there are secure delivery channels between the entity that prints ballots and the voters. The list goes on and on, and each definition seems to cover a specific coercer or verification tailored to a particular voting system.

No definition is currently accepted as the standard one for coercion-resistance

or cast-as-intended verifiability. Some approaches intersect, while others do not seem to agree on the key ideas. We observe that mainly just a few key elements cause the differences: the existence of reliable delivery channels, the trustworthiness of the voting device, reliance on auditors for cast-as-intended verification, and the possibility of the coercer to stop the voter from voting or buy credentials. One may wonder if it is possible to cover all those differences to obtain the standard definition.

Unfortunately, it is not that easy to formally define some properties without implicitly restricting possible voting scheme designs. For example, should the voting device be trusted for coercion-resistance? On the one hand, assuming that the voting device is trustworthy favors the encryption-based type of voting schemes, i.e., those which require voters to enter the selection in plaintext. On the other hand, not trusting the voting device leaves only solutions, which either delegate vote encryption or encoding to a third party or perform re-randomization before publishing the ballot. Hence, either choice results in eliminating a voting system class.

An implicit assumption about the voting system design is quite common in definitions. For example, the most famous privacy definition by Benaloh [17] requires the voting system to have a public bulletin board and doesn't permit running tally multiple times. In practice, however, recounts are sometimes permitted, and the ballot box is private or distributed, which results in definition adjustments (see Swiss Post voting system privacy definition for details [105]). Similarly, the coercion-resistance definition by Juels *et al.* [67] works only if there is a way to fake credentials. Another example is the verifiability definition in Demos system [71], which assumes enough voters are honest, and the adversary can slightly modify the election results.

As we can see, formalizing any intuitively understandable property requires restricting the voting system designs by, at least, deciding if the voting device can be trusted. What is more, we need to agree if the coercer wants to stop the voter from voting, change the intended vote or only cause trouble. Finally, we need to determine whether we will be satisfied with some probability of casting the correct ballot and getting away from the coercion or whether we want the mechanism to work in the overwhelming number of cases.

All those decisions imply some restrictions and will unavoidably leave out some solutions. However, without them, it is impossible to be concrete enough to formalize coercion-resistant cast-as-intended verification. We can hardly combine, for example, relying and not relying on pre-delivered data for cast-as-intended verification without creating multiple cases. Any definition with numerous ifs and else would be too difficult to understand and use. Hence, we must carefully select our settings to cover as many potential voting systems as possible.

3.2 Our Settings

As explained in the previous section, we cannot define any property unless we establish the setting. Those settings include assumptions about the voting system, expectations regarding voters' behavior, trusted parties, what the adversary can and cannot do, the existing communication channels, etc.

Since we are looking at coercion-resistant cast-as-intended voting schemes, it

makes sense first to clarify what we mean by cast-as-intended verification, what we consider coercion, and which voting schemes we focus on. In other words, what we implicitly assume about definitions and the e-voting system.

Assumptions regarding cast-as-intended verification: Cast-as-intended verification is a type of verification that gives voters some guarantees that their vote was not altered. Basically, it ensures that whichever entity prepared the ballot didn't change the voter's intent. In most cases, the entity responsible for ballot preparation is a voting device, yet in some cases, it is done by the election authority (e.g., vote-code-based schemes and QR-code-based schemes).

There are dozens of ways to provide cast-as-intended verification, each offering different guarantees. Some rely on repetitive checks before actual ballot casting; others allow auditing the sent ballot. Some methods require the majority of voters to be honest, while others focus only on a particular voter. Since all those details affect the e-voting scheme design and trust assumptions, we cannot support all verification methods and should focus on a subset only.

First, we assume that the voter can always verify the cast ballot. Alternative approaches rely on variations of the challenge-or-cast method [4] or partial checks [33], which still leave a voting device a non-negligible chance of cheating. One might argue that verifying only prior ballots and leaving the sent vote unchecked is not cast-as-intended verification because a malicious voting device can guess the number of verification attempts and cast an altered ballot.

Second, we assume the cast-as-intended verification proves the ballot's correctness with a negligible chance of error. This excludes all probabilistic verification methods that only partially inspect the vote. While the requirement might seem too strong, we believe it is a reasonable request.

Third, we assume that voter does not have to expect other voters to be honest for the cast-as-intended verification to work. It means we exclude designs similar to the Demos system, where enough voters should randomly select the voting card side for the Σ-proof (showing election authority behaved as expected) to be sound.

Finally, we assume that honest voters always perform the verification as instructed. In other words, we do not expect them to deviate from the verification scenario or omit some steps.

To conclude, we assume the following about cast-as-intended mechanisms:

1. Voters verify the cast ballot itself;
2. Verification guarantees that ballot contains the intended selection under given trust assumptions except for negligible probability;
3. Voter is not expected to rely on the honesty of other voters for cast-as-intended verification;
4. Honest voter always performs verification as instructed.

Assumptions regarding coercion-resistance: The task of defining coercion is not as trivial as it may seem. While cast-as-intended verification is more straightforward, coercion-resistance remains a mostly intuitively understandable notion. Anything that might prevent voters from expressing their will freely due to the fear of

consequences might be considered coercion. Yet, it's impossible to protect voters from every possible threat - thus, we need to define coercion more specifically.

We can firmly agree that the coercer cannot control a voter all the time [43]; else, the voter has no choice but to comply [108]. Also, we can agree that a voter cannot cast a vote on their own and requires the assistance of a voting device. We assume that voters can be either honest but under duress or malicious and willing to sell their vote. Finally, we can safely presume that the coercer would not use extreme threats such as putting a gun to the voter's head. Otherwise, coercion-resistance makes no sense.

So far, we have established that coercion-resistance makes sense only when the coercer has limited observational power and cannot use extreme threats. However, what do "extreme threats" mean? Everyone is different, and it's hard to predict when someone will consider it safe to disobey. Intuition says: the simpler and more secure the coercion-resistance strategy is, the more voters will opt for it. This intuition was proven to be correct in [65].

However, we cannot measure voters' presumptions of risks, so we focus on defining the coercer's limitations. Mainly, we will focus on the coercer's motivation, passive observation powers, and moments when voters are left uncontrolled.

First, we assume that the coercer wants voters to vote in a specific way rather than not vote at all (a type of coercion known as a forced-abstention attack). Our assumption is reasonable since the only known way to hide participation is to use anonymous voting channels [67], which are famously hard to implement. Moreover, forced-abstention attacks are not unique to e-voting and can be run against any voting channel. The common countermeasure is to offer voters multiple ways to vote in hopes that the coercer cannot control all of them.

Second, we assume that voters can vote only once. Typically, the simplest coercion-resistance strategy is to allow voters to change their choice at any moment. However, this strategy is not ideal. Not all electoral laws permit multiple voting as it gives an unfair advantage to voters voting electronically as opposed to any other voter. Also, the strategy implicitly assumes that the coercer cannot determine which ballots came from the same voter. Unfortunately, preventing coercer from detecting all voter's ballots implies he cannot see the bulletin board, which severely limits possible designs of an e-voting system. Therefore, we exclude multiple voting and assume the one-voter one-vote rule.

Third, we assume that voters are left unobserved during the vote-casting. Specifically, the voter is free to interact with the voting system privately. The coercer might issue instructions of any complexity but is not allowed to observe or interact with the voter from the moment the voter enters their intent until the ballot appears on the bulletin board.

One might note that leaving the voter unobserved for the duration of vote-casting is quite a strong assumption. However, recall that voters are only active during the authentication or voting phase, which combines vote-casting and (optionally) vote-verification [1]. Moreover, we already established that the coercer doesn't control

[1]Separating vote-casting and vote-verification would not help. Privacy during the vote verification alone would not help resist coercion as it merely ensures the vote-casting phase was correct. Therefore, we can combine voting and verification into one phase and keep calling it the voting

voters all the time. Hence, we can assume that voters are left unobserved sometimes during the authentication or vote-casting part of the voting protocol.

One option is for the coercer to let voters authenticate privately. By authentication, we mean action throught which voter proves their identity and obtains voting credentials. Anything beyond that is not part of authentication. Therefore, if voters only can privately authenticate but not vote, they can only invalidate their vote by asking for fake credentials. However, it is not enough for coercion-resistance.

Another possibility is to grant privacy during both phases. It would allow voters to ask for multiple fake credentials to fool the coercer. However, since we already excluded multiple voting, we can strengthen our model and assume that only voting (including vote verification) happens privately.

Finally, we assume that the coercer cannot ask the voter to record the voting process since it is equivalent to having total control over the voter's actions.

To recap, we assume the following about the coercer and voters:

1. Voters can disobey without facing high safety risks;
2. Coercer wants voters to vote in a specific way rather than force them to abstain;
3. Each voter can vote only once;
4. Coercer doesn't observe or communicate with voters during vote-casting and vote-verification.

Assumptions about voting system design: Now we can discuss assumptions regarding e-voting system design. As we saw earlier, there are numerous ways to design an e-voting system. Some e-voting system designs rely on vote codes, while others perform encryption in the voting device. Some rely on pre-delivered voting material, while others don't.

Since we do not wish to limit our coercion-resistant cast-as-intended definition to specific e-voting systems only, we need to set basic assumptions regarding the voting and voter-verification processes. Otherwise, it would be impossible to come up with any formalization at all.

First, we will focus only on vote-casting and cast-as-intended verification parts of the e-voting scheme. Thus, we will forget other protocols and properties of the global e-voting system, like tallying, result consolidation, etc. At first glance, it might seem like a disadvantage. Our approach fails to capture adversaries that exploit the final result of the election, faults of anonymization procedure, or problems with decryption key handling. However, most information leakage during the tally phase can be fixed with proper result anonymization and control over entities performing the tally.

Our local approach makes perfect sense in big elections where we want to ensure that: (i) honest voters will not be cheated by a dishonest voting device, (ii) dishonest voters cannot prove how they voted to anybody else, and thus they cannot sell the vote or suffer from coercion. One may point out that if multiple-choice is permitted, a coercer (or vote-seller) might run an Italian attack[2], which means the tally leakage

phase for simplicity.

[2]An Italian attack[108] is a coercive attack where voters are forced to vote for a unique (or very unlikely) permutation or combination of voting options. As a result, the coercer can always verify the published results to see if the voter obeyed. Since the choice combination or order is easily

might harm privacy. However, it can only be prevented if we use a homomorphic tally or allow only a small number of selections that would severely limit the result application.

Similarly, we assume that the size of the potential voting options set is polynomial in the security parameter. While it does exclude the acceptance of completely random strings as the voting options, we argue it is natural for an election to have only a limited number of valid selections. Moreover, if we assume otherwise, a coercer might force the voter to submit a random answer, which would be excluded from the counting (effectively implying abstention) or require a non-public ballot box to protect the voter (thus excluding public result verifiability).

Second, we expect no pre-established secure channels between voters and electoral authorities running the election. It implies that voters do not have pre-delivered voting cards, signing keys, or any other material necessary for either cast-as-intended verification or coercion-resistance. Voters might have pre-established authentication data such as electronic identity cards, passwords, etc., but we leave authentication out of our scope.

As a consequence of excluding pre-delivered data, we only focus on systems where the voting device receives choice in clear and encrypts it. This covers many types of e-voting schemes but, unfortunately, completely excludes code-voting (e.g., Demos [71]), voting with QR codes (e.g., BeleniosVS [38]), or any other way of ballot pre-encryption.

Since the voting device performs encryption (or equivalent operation for preserving the voter's privacy), we expect it will not collaborate with the coercer. Otherwise, there is nothing the voter can do to hide disobedience. Fortunately, trusting the voting device for ballot privacy (and consequently for receipt-freeness and coercion-resistance) is quite common. For example, the voting device is regarded as trustworthy in Benaloh's and BPRIV's definitions of privacy and Swiss and Estonian legislations.

Finally, we assume that voters cannot forcefully extract from the voting device more information than the system outputs. It includes encryption randomness, witnesses for zero-knowledge proofs, etc. Some say such an expectation is unrealistic since the voting device is just a computer that can always be exploited. However, we note that: first, data extraction requires technical knowledge, which an average voter might not have; second, it implies that an honest voting device alone cannot protect voters' privacy and that the e-voting system should be trusted.

To summarize, we assume the following about the e-voting system design:

1. Voter has no pre-delivered material;
2. Voting device receives voting option from a polynomially large set in clear and encrypts it;
3. Voting device does not collaborate with the coercer;
4. Voter cannot forecefully extract more information from the voting device than it outputs.

Settings summary: We consider only e-voting schemes, where a voting device receives a voter's intent and prepares a ballot. The voting device might be malicious

identifiable, shuffling ballots before publishing would not protect against such coercion.

and willing to change a voter's choice, but it does not collaborate with the coercer.

Multiple voting is not allowed - each voter has only one vote. At least one voter is honest, but there is no guarantee that a majority or even a substantial number of voters is trustworthy. Honest voters are willing to disobey and would not face life-threatening consequences if caught.

An honest voter has no pre-delivered information and will always verify cast-as-intended proof of the sent vote, which ensures the ballot contains the correct intent except for a negligible probability. Voters can be malicious or trusted, but they won't be able to extract from the voting device more information than it's willing to give.

The coercer cannot prevent voters from voting or interact with them during the voting phase. He also cannot observe voters while they vote, which includes asking for video recordings, control over the voting device, real-time feedback, etc. The coercion goal is a vote for a specific option or a candidate. The coercer can see the public output of the voting device (i.e., ballot, publicly available proofs, etc.) and knows if the given voter voted or abstained.

3.3 Parties And Syntax Of The E-Voting Protocol

As explained before (see Section 3.2), we focus on a specific voter $\mathcal{V}$ who casts a vote through a voting device $\mathcal{VD}$, possibly under the coercion of an adversary $\mathcal{C}$. The voting device might be under the control of an adversary $\mathcal{A}$ aiming to change the voter's selection undetectable. However, that adversary is not the coercer nor collaborates with the one, i.e., $\mathcal{A} \neq \mathcal{C}$.

We will forget other protocols and properties of the global voting system, like the authentication of voters, the mixing or homomorphic tally step, the final decryption and publication of the result, etc. We only assume that someone, possibly electoral authorities $\mathcal{EA}$, generated public parameters pms known to all parties. Those parameters (might) include but are not limited to:

- election configuration, which includes election duration, questions, eligible voting options $\mathcal{M}$, etc.
- security parameter λ;
- hash functions;
- the public-key encryption scheme $\mathbb{E}$ with at least encryption Enc and decryption Dec algorithms;
- the public key pk of the election.

We assume that the validity of the public parameters can be verified by anyone and does not contain any secret information.

To generate pms, one would run an initial protocol:

$$\mathsf{pms} \leftarrow \mathsf{Setup}(\lambda)$$

Setup may eventually output other information, which is not intended to be published, such as secret keys, used randomness, etc. However, since we focus only on cases where no information is pre-delivered to voters, we simplify the model by saying that only public parameters are generated and published.

To cast a vote, the voter $\mathcal{V}$ interacts with their voting device $\mathcal{VD}$. Apart from the public parameters pms, the initial input of $\mathcal{VD}$ is empty, while the voter $\mathcal{V}$ also has a voting option $\mathsf{m} \in \mathcal{M}$, a coercer's voting preference $\mathsf{m}_{\mathcal{C}} \in \mathcal{M}$ and an element coerc, which is either empty (in case of no coercion) or specifies some instructions given to $\mathcal{V}$ by the coercer $\mathcal{C}$ on how to behave during the voting process. Those instructions may include specific rules about values the voter should enter (e.g., only use odd numbers), the output restrictions (e.g., if the ballot does not start with a bit '0', abort and repeat voting until it does), the selection criteria (e.g., open the left ciphertext), etc.

Since we do not limit the coercer's freedom, we must assume that the coercer's instructions coerc might affect the resulting ballot regardless of whether the voter obeys. Therefore those instructions should be part of the voting protocol input even if the voter chooses to vote for their preference m instead of the coercer's choice $\mathsf{m}_{\mathcal{C}}$. For example, consider a simple protocol where no proof of ballot content is given - the voter enters the voting option, receives back the ciphertext, and can either confirm the vote or try again. If the coercer demands the voter only to accept the ballot when the ciphertext starts with a bit '0', then not following this instruction would be obvious.

To vote for an option m while following the instructions from coerc, the voter $\mathcal{V}$ jointly with the voting device $\mathcal{VD}$ execute the voting protocol:

$$(\mathsf{b}_{\mathcal{V}}, \mathsf{b}_{\mathcal{VD}}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}}) \leftarrow \mathsf{Vote}^{\langle \mathcal{V}, \mathcal{VD} \rangle}(\mathsf{pms}, \mathsf{m}, \mathsf{coerc})$$

We denote as $\mathsf{rand}_{\mathcal{V}}$ the set of all values explicitly selected by $\mathcal{V}$ during the execution of the voting protocol (e.g., challenges), and as Trc the list of messages exchanged between $\mathcal{V}$ and $\mathcal{VD}$ in such an execution.

In particular, Trc includes a ciphertext $\mathtt{ctxt}$, which is supposed to be (if $\mathcal{VD}$ is honest) encryption of the voting option m under the public key pk of the election and using randomness r; that is $\mathtt{ctxt} = \mathsf{Enc}(\mathsf{m}, \mathsf{r})$, where we intentionally omit pk as a well-known input of the encryption protocol Enc.

Additionally, the voting protocol has two private output bits:

- a bit $\mathsf{b}_{\mathcal{V}}$ describing if the voter $\mathcal{V}$ accepts the interaction as valid ($\mathsf{b}_{\mathcal{V}} = 1$) or not ($\mathsf{b}_{\mathcal{V}} = 0$),

- a bit $\mathsf{b}_{\mathcal{VD}}$ describing if the voting device $\mathcal{VD}$ accepts the interaction as valid ($\mathsf{b}_{\mathcal{VD}} = 1$) or not ($\mathsf{b}_{\mathcal{VD}} = 0$),

If both $\mathsf{b}_{\mathcal{V}} = \mathsf{b}_{\mathcal{VD}} = 1$, the encrypted vote ($\mathtt{ctxt} \in \mathsf{Trc}$) generated by $\mathcal{VD}$ is deemed valid. If $\mathsf{b}_{\mathcal{V}} = 0$, the voter would complain to election authorities $\mathcal{EA}$ against the voting device, which leads to the vote invalidation. Similarly, if $\mathsf{b}_{\mathcal{VD}} = 0$, then the ballot box (as well as the voting device) would not accept the vote as valid.

In practice, these two bits are outputs of some verification functions run locally by $\mathcal{V}$ and $\mathcal{VD}$. The voter $\mathcal{V}$ might keep some part of the interaction with the voting device $\mathcal{VD}$ secret and use this knowledge for the cast-as-intended verification.

3.4 Formal Definitions Of Cast-As-Intended And Coercion-Resistance

We will use syntax from Section 3.3 to formally define when the protocol Vote enjoys cast-as-intended (see Definition 7) and coercion-resistance (see Definition 8) properties.

3.4.1 Formal Definition Of Cast-As-Intended (CAI) Verifiability

Informally, we want to ensure that a not coerced voter who runs the voting protocol with a voting option m and accepts the interaction as valid can be confident that $\mathtt{ctxt}$ indeed contains encryption of m. The property is thus formalized by considering a dishonest voting device that tries to cheat the voter and sends to the ballot box a ciphertext $\mathtt{ctxt}$ that decrypts to $\mathsf{m}' \neq \mathsf{m}$. We want the probability of such a cheating behavior to happen without being detected by the voter is negligible.

The corresponding event Cheat is defined as follows:

$$\mathsf{Cheat} = \begin{cases} \mathsf{pms} \leftarrow \mathsf{Setup}(\lambda) \\ (\mathsf{b}_{\mathcal{V}}, \mathsf{b}_{\mathcal{VD}}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}}) \leftarrow \mathsf{Vote}^{\langle \mathcal{V}, \mathcal{VD} \rangle}(\mathsf{pms}, \mathsf{m}, \emptyset) \\ \mathtt{ctxt} \in \mathsf{Trc} \\ \mathsf{b}_{\mathcal{V}} = 1 \text{ (voter accepts the proof)} \\ \mathsf{m} \neq \mathsf{Dec}(\mathtt{ctxt}, \mathsf{sk}) \text{ (but } \mathtt{ctxt} \text{ does not contain the intent)} \end{cases}$$

where sk is the secret key of the election, matching the public key pk included in pms, and Dec is the decryption algorithm.

Definition 7 (Cast-as-intended)**.** The protocol Vote enjoys Cast-as-Intended (CAI) verifiability if for any $\mathsf{m} \in \mathcal{M}$ the probability of event Cheat is a negligible function of the security parameter λ, for any polynomial-time voting device $\mathcal{VD}$.

3.4.2 Formal Definition Of Coercion-Resistance (CR)

In our setting, a coercer $\mathcal{C}$ (e.g., a vote buyer) may interact with the voter $\mathcal{V}$ before the execution of the voting protocol and force the voter to vote for some option $\mathsf{m}_{\mathcal{C}}$ and to follow the instructions in coerc while executing Vote. The coercer cannot observe the interaction between the voter $\mathcal{V}$ and the voting device $\mathcal{VD}$ (otherwise, it would be impossible to achieve any meaningful coercion-resistance property). However, the coercer has access to the public channel through which the ciphertext $\mathtt{ctxt}$ is sent by $\mathcal{VD}$ to the ballot box.

After the protocol Vote terminates, $\mathcal{C}$ expects to receive a voter's view of the interaction view, which includes Trc along with the set of random values $\mathsf{rand}_{\mathcal{V}}$ selected by the voter during Vote.

The intuition is that for resisting coercion, the voter $\mathcal{V}$ must always have the possibility to run the protocol Vote with its voting option m and later simulate all values (including the challenges) to make $\mathcal{C}$ believe that Vote was run with $\mathsf{m}_{\mathcal{C}}$ as

input. If the coercer $\mathcal{C}$ cannot distinguish whether the voter followed the instructions or not with probability more than random guessing, coercion-resistance holds. We formalize this intuition in the experiment $\mathsf{Exp}^{\mathsf{CR}}_{\mathcal{C},\mathcal{V}}(\mathsf{pms}, \mathsf{m}_{\mathcal{C}}, \mathsf{m}, \mathsf{coerc})$ in Figure 3.1.

In the experiment, the challenger, playing on behalf of the honest voter and trustworthy voting device, flips a coin β. If $\beta = 0$, the challenger obeys the coercer, follows all instructions, votes for $\mathsf{m}_{\mathcal{C}}$, and gives the coercer an authentic view of the interaction. However, if $\beta = 1$, the challenger votes for the voter's choice m and uses polynomial-time simulator algorithm $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ to produce a fake view. The experiment returns 1 whenever the coercer can guess β and 0 otherwise.

We say that coercion-resistant holds for the protocol Vote if the coercer $\mathcal{C}$ can distinguish whether the voter followed the instructions and voted for $\mathsf{m}_{\mathcal{C}}$ or provided simulated values with probability $\frac{1}{2}$. The formalization of this property is given in the Definition 8.

$\underline{\mathsf{Exp}^{\mathsf{CR}}_{\mathcal{C},\mathcal{V}}(\mathsf{pms}, \mathsf{m}_{\mathcal{C}}, \mathsf{m}, \mathsf{coerc})}$:

1. $\mathsf{pms} \leftarrow \mathsf{Setup}(\lambda)$
2. $\beta \xleftarrow{\$} \{0,1\}$
3. If $\beta = 0$:
   - $(\mathsf{b}^{(0)}_{\mathcal{V}}, \mathsf{b}^{(0)}_{\mathcal{VD}}, \mathsf{Trc}^{(0)}, \mathsf{rand}^{(0)}_{\mathcal{V}}) \leftarrow \mathsf{Vote}^{\langle \mathcal{V}, \mathcal{VD} \rangle}(\mathsf{pms}, \mathsf{m}_{\mathcal{C}}, \mathsf{coerc})$
   - If $\mathsf{b}^{(0)}_{\mathcal{VD}} = 0$, abort the experiment.
   - Define $\mathsf{view}^{(0)} = (\mathsf{b}^{(0)}_{\mathcal{V}}, \mathsf{Trc}^{(0)}, \mathsf{rand}^{(0)}_{\mathcal{V}})$
4. Else if $\beta = 1$:
   - $\mathsf{coerc}^* \leftarrow \mathsf{Sim}_1(\mathsf{coerc}, \mathsf{m}, \mathsf{m}_{\mathcal{C}})$
   - $(\mathsf{b}_{\mathcal{V}}, \mathsf{b}_{\mathcal{VD}}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}}) \leftarrow \mathsf{Vote}^{\langle \mathcal{V}, \mathcal{VD} \rangle}(\mathsf{pms}, \mathsf{m}, \mathsf{coerc}^*)$, with $\mathtt{ctxt} \in \mathsf{Trc}$
   - If $\mathsf{b}^{(0)}_{\mathcal{VD}} = 0$, abort the experiment.
   - $(\mathsf{b}^{(1)}_{\mathcal{V}}, \mathsf{Trc}^{(1)}, \mathsf{rand}^{(1)}_{\mathcal{V}}) \leftarrow \mathsf{Sim}_2(\mathsf{b}_{\mathcal{V}}, \mathsf{b}_{\mathcal{VD}}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}}, \mathsf{m}, \mathsf{m}_{\mathcal{C}}, \mathsf{coerc})$, with $\mathtt{ctxt} \in \mathsf{Trc}^{(1)}$
   - Define $\mathsf{view}^{(1)} = (\mathsf{b}^{(1)}_{\mathcal{V}}, \mathsf{Trc}^{(1)}, \mathsf{rand}^{(1)}_{\mathcal{V}})$
5. $\mathsf{b} \leftarrow \mathcal{C}(\mathsf{view}^{(\beta)})$
6. If $\mathsf{b} = \beta$, return 1. Else, return 0.

Figure 3.1: Experiment for coercion-resistance.

Definition 8 (Coercion-resistance)**.** The protocol Vote enjoys coercion-resistance (CR) if for any polynomial-time coercer $\mathcal{C}$, any coercion $(\mathsf{m}_{\mathcal{C}}, \mathsf{coerc})$ and any voting option m, there exists a polynomial-time simulator algorithm $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that $\left|\Pr[1 \leftarrow \mathsf{Exp}^{\mathsf{CR}}_{\mathcal{C},\mathcal{V}}(\mathsf{pms}, \mathsf{m}_{\mathcal{C}}, \mathsf{m}, \mathsf{coerc})] - \frac{1}{2}\right| = \mathsf{negl}(\lambda)$ is a negligible function of the security parameter λ.

Since the experiment $\mathsf{Exp}^{\mathsf{CR}}_{\mathcal{C},\mathcal{V}}(\mathsf{pms}, \mathsf{m}_{\mathcal{C}}, \mathsf{m}, \mathsf{coerc})$ aborts when the coercion chosen by $\mathcal{C}$ makes the (honest) voting device reject the execution of the protocol Vote, our definition does not cover some very strong types of coercion: (1) the coercer forces a voter not to participate in the election, (2) the coercer forces a voter to misbehave

during the interaction with $\mathcal{VD}$, which results in $\mathcal{VD}$ aborting the protocol and not sending any ciphertext to the ballot box.

Under our assumption that the adversary can see the ciphertext `ctxt` that is sent by $\mathcal{VD}$ to the ballot box, these stronger types of coercion could be addressed only with solutions that allow $\mathcal{VD}$ to send a ciphertext (encryption of a dummy/special plaintext) to the ballot box even in the case when a voter abstains or when a voter misbehaves during the voting protocol. This is explicitly prohibited by some countries that run electronic elections today. Furthermore, this would open the door to possible attacks by a dishonest $\mathcal{VD}$ which, in case of abstention, does not send an encryption of the dummy/special plaintext, but encryption of a specific valid voting option.

3.4.3 Comparison With Previous Definitions Of CAI And CR

Our CAI definition is game-based and similar in spirit to that in [97]. The main difference is that we are interested in yes/no CAI verification: the goal is that a malicious $\mathcal{VD}$ has a negligible probability of cheating a voter. In contrast, a more general and quantified definition was provided in [97] to measure or bound the (non-necessarily negligible) probability that $\mathcal{VD}$ cheats a voter without being detected.

The comparison between our CR definition and other existing ones is more complex due to the wide variety of settings defined for CR in the literature. First, we consider only e-voting systems without secure channels between authorities and voters, which is very relevant and constrains our CR definition. In particular, in such a setting, it is clear that a voter $\mathcal{V}$ must send its voting choice m to the voting device $\mathcal{VD}$ during the vote-casting phase; therefore, we must assume that $\mathcal{VD}$ is trusted for privacy and coercion-resistance. Otherwise, it would be impossible for an honest voter to deceive a coercer colluding with $\mathcal{VD}$. This is in contrast to other definitions of CR, in particular, the JCJ one [67] and variants thereof [37].

In our CR definition, the analysis is local: we assume that one single voter is under coercion (by a real coercer or by itself, in the case of vote-selling). In other words, we do not worry about the behavior of other voters or other authorities of the e-voting system. This simplification allows us to concentrate only on the vote-casting phase of the election, the only one where the voter plays an active role. This is common to the model in [56] but differs from many other CR definitions and models, which deal with more global adversaries [67, 37, 110, 11].

We believe a local approach makes perfect sense, in particular for typical elections where the rest of the phases (after vote casting) are done in a verifiable way: verifiable shuffle of ciphertexts or verifiable homomorphic tally, verifiable decryption of the final results, etc., possibly executed by a threshold of servers.

A direct consequence of our local setting for CR is that we can consider a (simpler to use) game-based definition: the goal of the (real) adversary is to distinguish between two possible (real) executions of the protocol Vote. In contrast, in the (conceptually more complicated, in our opinion) UC-based setting, where two different worlds (the real and an ideal world) are considered, the goal is to show that the behavior of a real adversary (coercer, in this case) in a real-world can be simulated

in the ideal world, without anybody (the environment) being able to distinguish between both. Works proposing CR models and proofs in the UC-based setting include [37, 86, 110, 11].

3.5 On The Necessary Number Of Rounds

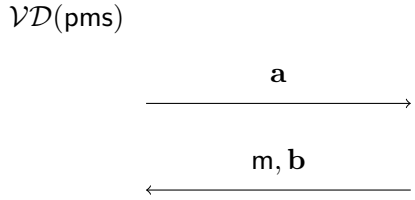
We have formally defined coercion-resistance and cast-as-intended verification. Now the question is whether we can fulfill them. Do we necessarily need interaction between the voter and the voting device? If so, how many rounds of communication do we need? Is there a minimum number?

We expect the ciphertext (at least partially) to be created by the voting device. Otherwise, the voter can easily demonstrate to the coercer the ballot content by revealing the randomness used to compute it.

One round is not enough: The nature of voting implies that voters should communicate their choice m, which is not known to the system in advance. Otherwise, there will be no need to conduct elections. Moreover, the voter must receive the ciphertext $\mathtt{ctxt}$ after voting. Hence, at least two rounds of interaction are necessary.

Two rounds are not enough: We show that two rounds are not sufficient. Consider an arbitrary protocol with two interactions. There are only two possible ways to start the protocol - it can be done either by the voting device or the voter. Hence, we have two possibilities to analyze.

Case 1: The voting device starts the interaction: The ultimate goal of the voting device use is to produce ciphertext $\mathtt{ctxt}$ encrypting the voting option m and the proof π that the said voting option is what the voter intended it to be. Since we have only two rounds and the voting device starts, it can only return some data $\mathbf{a}$ independent of the voter's selection. After that, the voter replies with the selected voting option m and (maybe) some value denoted as $\mathbf{b}$.



The voting protocol must send the ciphertext $\mathtt{ctxt}$ to the voting system at the end of the voting process. If the protocol has cast-as-intended verifiability, there should be some function $\mathsf{Verify}(\mathbf{a}, \mathbf{b}, \mathsf{m}, \mathsf{rand}_{\mathcal{V}}, \mathtt{ctxt}) \rightarrow \{0, 1\}$ which allows the voter to ensure that the published $\mathtt{ctxt}$ is valid and contains the intended option m.

The coercer can control the voter's input $\mathbf{b}$. However, we don't know if the protocol permits $\mathbf{b}$ to be selected more or less arbitrarily or if $\mathbf{b}$ must be chosen only from a relatively small set of possible values. Since difficult-to-resist coercion strategies would differ, we must consider both cases separately.

Case 1.A: Suppose, in our two-round protocol, the value $\mathbf{b}$ should be chosen from a relatively small (polynomially large in the security parameter) set of possible values. The coercer can instruct the voter to use a specific value $\mathbf{b} = \mathbf{b}_{\mathcal{C}}$ or utilize some heuristic for selecting $\mathbf{b}_{\mathcal{C}}$ adaptively. Since the set of possible values is relatively small, relying on heuristic is equivalent to expecting a specific $\mathbf{b}_{\mathcal{C}}$. Thus we focus on coercion strategies where $\mathbf{b} = \mathbf{b}_{\mathcal{C}}$.

For such coercion, the coercion-resistance only holds if the simulator $\mathtt{Sim}$ is capable of generating a value $\mathbf{a}^{(1)}$ based on the "disobeying coercion" transcript $(\mathbf{a}, \mathbf{b}, \mathtt{ctxt})$ in a way that the simulated transcript $(\mathbf{b}_{\mathcal{C}}, \mathbf{a}^{(1)}, \mathtt{ctxt})$ is indistinguishable from the "obeying coercion" transcript $(\mathbf{a}_{\mathcal{C}}, \mathbf{b}_{\mathcal{C}}, \mathtt{ctxt}_{\mathcal{C}})$[3], where $\mathtt{ctxt}_{\mathcal{C}}$ encrypts $\mathsf{m}_{\mathcal{C}}$ and $\mathtt{ctxt}$ encrypts m. The cast-as-intended verification should hold for the case of following the coercion $1 \leftarrow \mathsf{Verify}(\mathbf{b}_{\mathcal{C}}, \mathbf{a}_{C}, \mathsf{m}_{\mathcal{C}}, \mathtt{ctxt}_{\mathcal{C}})$ and disobedience $1 \leftarrow \mathsf{Verify}(\mathbf{b}, \mathbf{a}, \mathsf{m}, \mathtt{ctxt})$. Also, the coercer that attempts to verify a simulated transcript should pass it successfully too. Otherwise, the coercer can easily distinguish transcripts. Therefore, $1 \leftarrow \mathsf{Verify}(\mathbf{b}_{\mathcal{C}}, \mathbf{a}^{(1)}, \mathsf{m}_{\mathcal{C}}, \mathtt{ctxt})$ must be successful.

Unfortunately, if such $\mathtt{Sim}$ exists, the voting device can use $\mathtt{Sim}$ to break cast-as-intended verification with a non-negligible probability. For that, the voting device selects $(\mathbf{a}_{\mathcal{VD}}, \mathbf{b}_{\mathcal{VD}}, \mathsf{m}_{\mathcal{VD}})$ and generates $\mathtt{ctxt}_{\mathcal{VD}}$ as an encryption of $\mathsf{m}_{\mathcal{VD}}$, then runs $\mathtt{Sim}$ to generate $\mathbf{a}^{(1)}$ for $\mathsf{m}_{\mathcal{V}}$ and starts the protocol with an honest voter by sending $\mathbf{a}^{(1)}$ and hopes that the voter would reply with $\mathbf{b} = \mathbf{b}_{\mathcal{VD}}$ and $\mathsf{m} = \mathsf{m}_{\mathcal{V}}$. Since the set of possible $\mathbf{b}$ values is relatively small and the set of possible voting options m naturally has polynomial size as well, there is a non-negligible chance that the voting device guessed correctly, i.e.: $\mathbf{b} = \mathbf{b}_{\mathcal{VD}}$ and $\mathsf{m} = \mathsf{m}_{\mathcal{V}}$. If this is the case, the voting device sends $\mathtt{ctxt}_{\mathcal{VD}}$ to the ballot box or voting server. Because we assume that the protocol is coercion-resistant, the voter cannot distinguish between "voting device cheats" transcripts $(\mathbf{a}^{(1)}, \mathbf{b} = \mathbf{b}_{\mathcal{VD}}, \mathtt{ctxt}_{\mathcal{VD}})$ and "voting device behaves honestly" transcript $(\mathbf{a}, \mathbf{b}, \mathtt{ctxt})$, where $\mathtt{ctxt}$ encrypts the voter's choice m and $\mathtt{ctxt}_{\mathcal{VD}}$ encryps $\mathsf{m}_{\mathcal{VD}} \neq \mathsf{m}$. Hence, the voter will accept the interaction $1 \leftarrow \mathsf{Verify}(\mathbf{b}_{\mathcal{VD}}, \mathbf{a}^{(1)}, \mathsf{m}, \mathtt{ctxt}_{\mathcal{VD}})$ and believe that $\mathtt{ctxt}_{\mathcal{VD}}$ encrypts m, while in reality, $\mathtt{ctxt}_{\mathcal{VD}}$ encrypts $\mathsf{m}_{\mathcal{VD}}$.

Therefore, in **Case 1.A**, our two-round protocol enjoys the cast-as-intended verification but cannot resist coercion where forces $\mathbf{b} = \mathbf{b}_{\mathcal{C}}$ or vice versa.

Case 1.B: Suppose, in our two-round protocol, the value $\mathbf{b}$ can be selected from a large set of possible values. In such a case, a coercer can force the voter to use adaptively computed $\mathbf{b} = \mathcal{F}(\mathbf{a})$, where $\mathcal{F}$ is some pseudo-random function of the coercer's choice.

The coercion-resistance only holds if the simulator $\mathtt{Sim}$ is capable of generating a fake transcript $(\mathcal{F}(\mathbf{a}^{(1)}), \mathbf{a}^{(1)}, \mathtt{ctxt})$ based on the actual transcript $(\mathbf{a}, \mathbf{b}, \mathtt{ctxt})$ that is indistinguishable from the "obeying the coercer" case $(\mathbf{a}_{\mathcal{C}}, \mathcal{F}(\mathbf{a}_{\mathcal{C}}), \mathtt{ctxt}_{\mathcal{C}})$, where $\mathtt{ctxt}$ encrypts the voter's option m and $\mathtt{ctxt}_{\mathcal{C}}$ encrypts the coercer's choice $\mathsf{m}_{\mathcal{C}}$. The cast-as-intended verification should hold for the case of following the coercion $1 \leftarrow \mathsf{Verify}(\mathcal{F}(\mathbf{a}_{\mathcal{C}}), \mathbf{a}_{\mathcal{C}}, \mathsf{m}_{\mathcal{C}}, \mathtt{ctxt}_{\mathcal{C}})$ and disobedience $1 \leftarrow \mathsf{Verify}(\mathbf{a}, \mathbf{b}, \mathsf{m}, \mathtt{ctxt})$. Also, the coercer that attempts to verify a simulated transcript should pass it successfully too. Otherwise, the coercer can easily distinguish transcripts. Therefore, $1 \leftarrow \mathsf{Verify}(\mathcal{F}(\mathbf{a}^{(1)}), \mathbf{a}^{(1)}, \mathsf{m}_{\mathcal{C}}, \mathtt{ctxt})$ must be sucessful.

[3]Since $\mathbf{b}$ is fixed, $\mathsf{rand}_{\mathcal{V}}$ is empty. Hence, we omit it from the list of simulated values.

Lemma 3.5.1. With overwelming probability $\mathtt{Sim}$ generates $\mathbf{a}^{(1)}$ before computing $\mathbf{b}^{(1)} = \mathcal{F}(\mathbf{a}^{(1)})$.

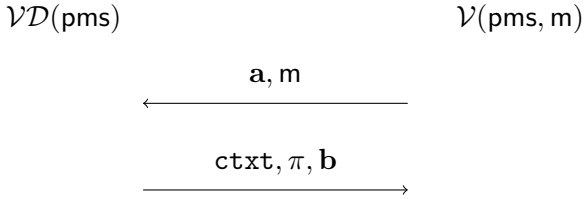
Proof. Suppose, our assumption is wrong. In such a case, the $\mathtt{Sim}$ that can generate $\mathbf{b} = \mathcal{F}(\mathbf{a})$ first and then find corresponding $\mathbf{a}$ can be used to find pre-images of the one-way function $\mathcal{F}$. However, it is considered to be infeasible. □

Lemma 3.5.2. If cast-as-intended verifiability holds, then for the given $\mathbf{a}^{(1)}$ and $\mathtt{ctxt}$ that does not encrypt $\mathsf{m}_{\mathcal{C}}$ the probability that a random $\mathbf{b}^{(1)}$ results in a sucessful veriifcation $1 \leftarrow \mathsf{Verify}(\mathbf{b}^{(1)}, \mathbf{a}^{(1)}, \mathsf{m}_{\mathcal{C}}, \mathtt{ctxt})$ must be negligible.

Proof. Suppose our assumption is wrong. In such a case, the voting device can break cast-as-intended verification with a non-negligible chance. For that, the voting device sends a random $\mathbf{a}^{(1)}$ to the voter and computes $\mathtt{ctxt}$ as an encryption of $\mathsf{m}_{\mathcal{VD}}$, which is likely different from the voter's intent m. When the voter replies with $\mathbf{b}$, there is a non-negligible probability that the verification $1 \leftarrow \mathsf{Verify}(\mathbf{b}, \mathbf{a}^{(1)}, \mathsf{m}, \mathtt{ctxt})$ will pass, which implies a non-negligible probability of breaking cast-as-intended verification. □

However, now we can conclude that such a simulator $\mathtt{Sim}$ can succeed with only a negligible probability of succeeding. By Lemma 3.5.1, $\mathtt{Sim}$ should simulate $\mathbf{a}^{(1)}$ before computing $\mathbf{b}^{(1)} = \mathcal{F}(\mathbf{a}^{(1)})$; however the probability that the random output $\mathbf{b}^{(1)}$ of a pseudo-random function $\mathcal{F}$ is such that $1 \leftarrow \mathsf{Verify}(\mathbf{a}^{(1)}, \mathbf{b}^{(1)}, \mathsf{m}_{\mathcal{C}}, \mathtt{ctxt})$ for a ciphertex that encrypts $\mathsf{m} \neq \mathsf{m}_{\mathcal{C}}$ is negligible per Lemma 3.5.2. Hence, in **Case 1.B**, our two-round protocol enjoys the cast-as-intended verification but cannot resist coercion where forces $\mathbf{b} = \mathcal{F}(\mathbf{a})$ or vice versa.

Case 2: The voter starts the interaction: The voter inputs a voting option m and some value denoted as $\mathbf{a}$. The voting device replies with the ciphertext $\mathtt{ctxt}$, the proof π and (maybe) some value $\mathbf{b}$.



Suppose the protocol is coercion-resistant. According to our definition, it means the voter can vote for m, obtain the actual transcript $(\mathbf{a}, \mathtt{ctxt}, \pi, \mathbf{b})$, and then run a simulator $\mathtt{Sim}$ to get an indistinguishable transcript $(\mathbf{a}^{(1)}, \mathtt{ctxt}, \pi^{(1)}, \mathbf{b}^{(1)})$. The coercer should not notice that values $(\mathbf{a}^{(1)}, \pi^{(1)}, \mathbf{b}^{(1)})$ were simulated.

However, in our two-round protocol, the coercer can control the voter's choice of $\mathbf{a}$ and even force a specific $\mathbf{a} = \mathbf{a}_{\mathcal{C}}$. Therefore, the coercion-resistance only holds if the simulator $\mathtt{Sim}$ is capable of generating a fake transcript $(\mathbf{a}_{\mathcal{C}}, \mathtt{ctxt}, \pi^{(1)}, \mathbf{b}^{(1)})$ based on the actual transcript $(\mathbf{a}, \mathtt{ctxt}, \pi, \mathbf{b})$ in a way that it is indistinguishable from the "obeying the coercion" case $(\mathbf{a}_{\mathcal{C}}, \mathtt{ctxt}_{\mathcal{C}}, \pi_{\mathcal{C}}, \mathbf{b}_c)$, where $\mathtt{ctxt}$ encrypts the voter's choice m and $\mathtt{ctxt}_{\mathcal{C}}$ contains the coercer's choice. The cast-as-intended verification

should hold for the case of following the coercion $1 \leftarrow \mathsf{Verify}(\mathbf{a}_{\mathcal{C}}, \mathsf{m}_{\mathcal{C}}, \mathtt{ctxt}_{\mathcal{C}}, \pi_{\mathcal{C}}, \mathbf{b}_{\mathcal{C}})$ and disobedience $1 \leftarrow \mathsf{Verify}(\mathbf{a}, \mathsf{m}, \mathtt{ctxt}, \pi, \mathbf{b})$. Also, the coercer that attempts to verify a simulated transcript should pass it successfully too. Otherwise, the coercer can easily distinguish transcripts. Therefore, $1 \leftarrow \mathsf{Verify}(\mathbf{a}^{(1)}, \mathsf{m}_{\mathcal{C}}, \mathtt{ctxt}, \pi^{(1)}, \mathbf{b}^{(1)}))$ must be sucessful.

Unfortunately, if such `Sim` exists, the voting device can use `Sim` to break cast-as-intended verification. For that, the voting device selects $\mathbf{a}_{\mathcal{VD}}$, encrypts $\mathsf{m}_{\mathcal{VD}}$, honestly generates $(\pi_{\mathcal{VD}}, \mathtt{ctxt}_{\mathcal{VD}}, \mathbf{b}_{\mathcal{VD}})$ for $(\mathsf{m}_{\mathcal{VD}}, \mathbf{a}_{\mathcal{VD}})$, then runs `Sim` to generate $\pi^{(1)}$ and $\mathbf{b}^{(1)}$ corresponding to the voter's input m and $\mathbf{a}$. Because the protocol is coercion-resistant, the voter cannot distinguish between $(\mathbf{a}_{\mathcal{VD}}, \pi_{\mathcal{VD}}, \mathbf{b}_{\mathcal{VD}})$ and $(\mathbf{a}, \pi^{(1)}, \mathbf{b}^{(1)})$, and cast-as-intended verifiability is broken.

Number of rounds in our constructions: As we proved, achieving a CR-CAI solution with two rounds of interaction between the voter and the voting device is impossible. Therefore, the minimum number of rounds is three. In Chapter 4, we describe and analyze two solutions with four rounds of communication; these are (modifications of) the first solutions that we found in the thesis (paper in CANS'21 [50]). Two years later, when dealing with more powerful coercions, we found another solution with three rounds of communication (and so, optimal in terms of round complexity). The three-round solution is described and analyzed in Chapter 7. The first solution in Chapter 4 works in the plain model, whereas the second solution and its enhancement (the three-rounds solution in Chapter 7) are proven secure in the Random Oracle Model. It is an open problem to find a three-round solution in the plain model or to prove that such a solution is impossible.

Chapter 4

Instantiations Of Coercion Resistant Cast-As-Intended Verifiable Voting Protocols

In this chapter, we present two protocols for achieving coercion-resistant cast-as-intended verification. Before giving the solutions, explained in Sections 4.2 and 4.4, we recall why a standard interactive Σ-protocol and interactive OR-proof are not sufficient in Sections 4.1 and 4.3. As we explain, the problem lies in a gap between honest-verifier zero knowledge and full zero-knowledge. Then, we focus on solving this gap and outlining two coercion-resistant cast-as-intended protocols. Both solutions are generic and can be described for general binary relations, zero-knowledge systems, etc. Also, we provide a security analysis of both solutions to show they achieve coercion-resistant cast-as-intended verification in Sections 4.2.1 and 4.4.1. Finally, we give specific instantiations for the ElGamal encryption case and comment on the implementations' performance in Sections 4.2.2 and 4.4.2.

4.1 Unsatisfactory Solution U_1

Before describing the first solution S_1, we recall the original Σ-protocol idea from Section 1.1.3.3 as it will be the basis for our first coercion-resistant cast-as-intended protocol. We also argue that the gap between unsatisfactory and satisfactory protocols is closely related to the gap between the honest verifier and the full zero-knowledge property of cryptographic zero-knowledge systems.

Given a public-key homomorphic encryption scheme $\mathbb{E} = (\mathbb{E}.\mathsf{KG}, \mathbb{E}.\mathsf{Enc}, \mathbb{E}.\mathsf{Dec})$ from Section 2.3 and a Σ-protocol for proving the ciphertext encrypts the intended message from Section 2.5, we can construct a protocol described in Figure 4.1. The resulting proof achieves receipt-freeness as an honest voter (who selects the challenge at random) can generate a fake transcript indistinguishable from the actual one. For details, please, refer to the simulator description for instances out of the language described in Section 2.5.

However, since the proof is based on a Σ-protocol, it is not coercion-resistant. A coercer can force the voter to select a specific challenge $\mathsf{e} = \mathcal{F}(\mathsf{com})$ for some function $\mathcal{F}$. Such choice of e would allow the coercer to ensure that the voter

obeyed as the probability of a random e^* from a simulated proof to satisfy such constraint is negligible.

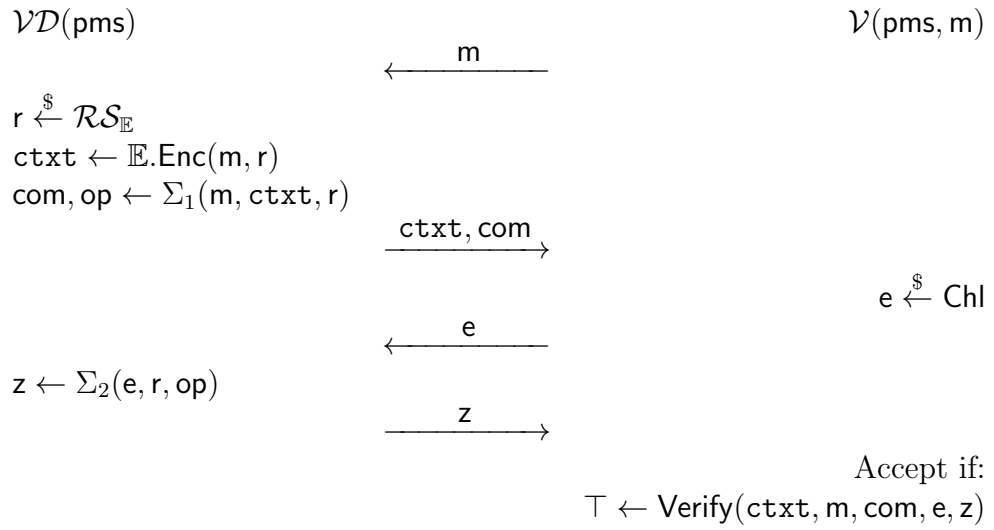


Figure 4.1: Σ-protocol for proving the ciphertext encrypts the intended message.

Therefore, to achieve CR-CAI verification, one has to assume that the voter is potentially dishonest. This, in turn, implies that the resulting protocol must have full zero knowledge.

4.2 Solution S_1: Committing To Challenges

The departing point for the first solution S_1 is the unsatisfactory generic solution U_1 based on a Σ-protocol described in the previous section (see Section 4.1). The problem with that solution was that a Σ-protocol is only an honest-verifier zero-knowledge but not full zero-knowledge. This allows the coercer to force a specific distribution of the challenges so that the actual proof transcripts cannot be simulated for instances out of the language.

Luckily, we can easily solve this problem by using a full zero-knowledge interactive protocol for the language $\mathcal{L}_{\mathbb{E}} = \{(\mathtt{ctxt}, \mathsf{m}) \mid \mathtt{ctxt} \text{ encryption of } \mathsf{m}\}$. To do so, we can use a well-known technique [60]: make the voter $\mathcal{V}$ commit to the challenge e before receiving anything from the voting device by using a perfectly hiding commitment scheme $\mathsf{Com} = (\mathsf{Com.Gen}, \mathsf{Com.Eval}, \mathsf{Com.Open})$ (see Section 2.4). In turn, the voting device $\mathcal{VD}$ will accept the challenge e only if it corresponds to the commitment previously received from $\mathcal{V}$.

The setup algorithm $\mathsf{pms} \leftarrow \mathsf{Setup}(\lambda)$ runs $\mathsf{Com.Gen}(\lambda)$ resulting in public commitment scheme parameters $(\mathsf{pms}_{\mathsf{Com}}, \mathcal{M}_{\mathsf{Com}}, \mathcal{RS}_{\mathsf{Com}}, \mathcal{C}_{\mathsf{Com}})$ and generates public-key encryption scheme parameters $\mathbb{E}.\mathsf{KG}(\lambda)$ resulting in key pair $(\mathsf{pk}, \mathsf{sk})$ for the encryption scheme $\mathbb{E}$, along with spaces $\mathcal{M}_{\mathbb{E}}, \mathcal{C}_{\mathbb{E}}, \mathcal{RS}_{\mathbb{E}}$. Note that the secret key sk is kept private. Then it defines challenge space for the Σ-protocol $\mathsf{Chl} \subseteq \mathcal{M}_{\mathbb{E}} \subseteq \mathcal{M}_{\mathsf{Com}}$ and sets public parameters as follows:

$$\mathsf{pms} = (\mathsf{pms}_{\mathsf{Com}}, \mathcal{M}_{\mathsf{Com}}, \mathcal{RS}_{\mathsf{Com}}, \mathcal{C}_{\mathsf{Com}}, \mathsf{pk}, \mathcal{M}_{\mathbb{E}}, \mathcal{C}_{\mathbb{E}}, \mathcal{RS}_{\mathbb{E}}, \mathsf{Chl})$$

The voting protocol $(\mathsf{b}_{\mathcal{V}}, \mathsf{b}_{\mathcal{VD}}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}}) \leftarrow \mathsf{Vote}^{\langle \mathcal{V}, \mathcal{VD} \rangle}(\mathsf{pms}, \mathsf{m}, \emptyset)$ with $\mathsf{coerc} = \emptyset$ works as described in Figure 4.2.

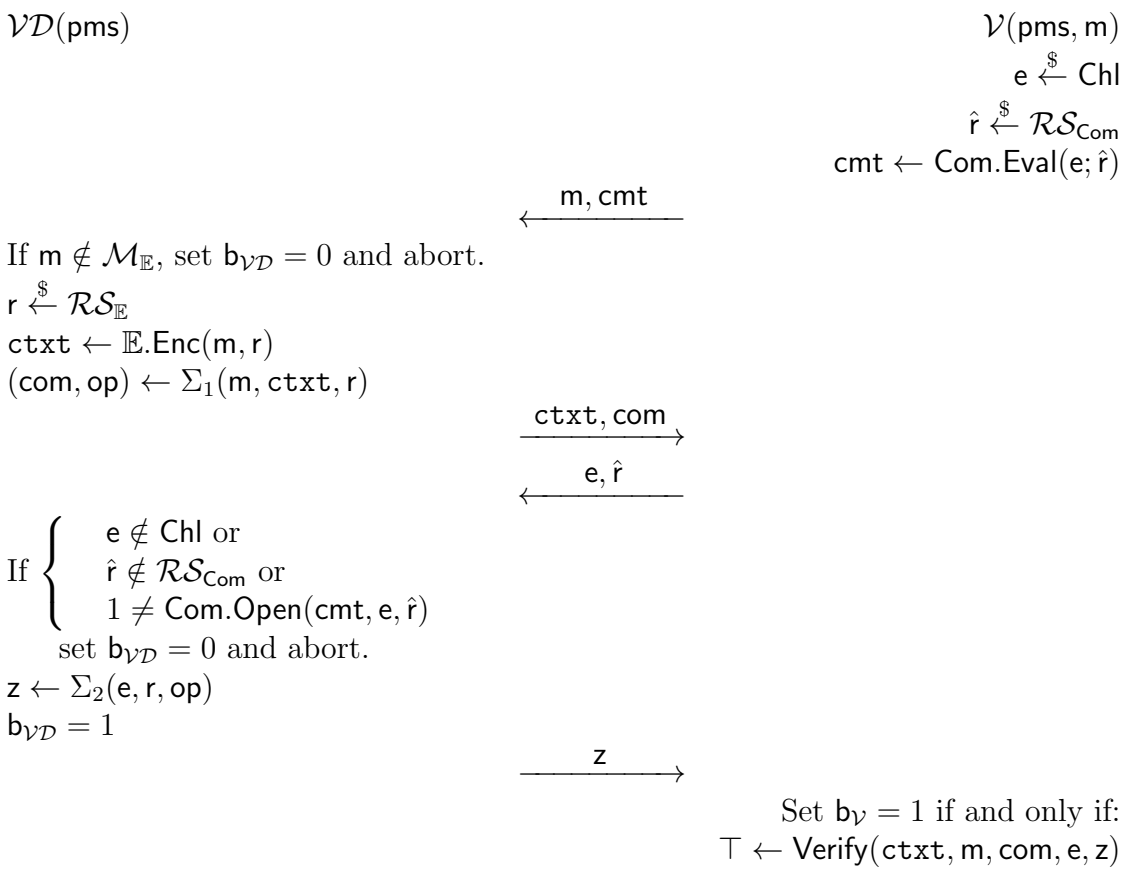


Figure 4.2: The voting protocol from solution S_1.

If $\mathsf{b}_{\mathcal{VD}} = 1$, then the voting device $\mathcal{VD}$ accepts the interaction as correct. Similarly, if $\mathsf{b}_{\mathcal{V}} = 1$, the voter $\mathcal{V}$ trusts that the ballot contains the intended option. The vote is only deemed valid when $\mathsf{b}_{\mathcal{V}} = \mathsf{b}_{\mathcal{VD}} = 1$. Otherwise, whenever $\mathsf{b}_{\mathcal{VD}} = 0$ - the voting device $\mathcal{VD}$ aborts the vote-casting procedure, and if $\mathsf{b}_{\mathcal{V}} = 0$ - the voter complains to authorities which invalidates the vote.

Following the notation introduced in Section 3.3, we have $\mathsf{rand}_{\mathcal{V}} = (\mathsf{e}, \hat{\mathsf{r}})$ and $\mathsf{Trc} = (\mathsf{m}, \mathtt{ctxt}, \mathsf{com}, \mathsf{e}, \mathsf{z}, \mathsf{cmt}, \hat{\mathsf{r}}, \mathsf{b}_{\mathcal{VD}}, \mathsf{b}_{\mathcal{V}})$.

4.2.1 Security Analysis Of The Protocol

Theorem 4.2.1 (Solution S_1 achieves CAI)**.** Assuming the special soundness of the Σ-protocol for $\mathcal{R}_{\mathbb{E}}$ and the perfect hiding property of Com, the protocol described in Figure 4.2 achieves the Cast-as-Intended (CAI) property (Definition 7).

Proof. Remember that the CAI property is satisfied if the probability of Cheat is

negligible in the security parameter λ, where Cheat is defined as follows:

$$\mathsf{Cheat} = \begin{cases} \mathsf{pms} \leftarrow \mathsf{Setup}(\lambda) \\ (\mathsf{b}_{\mathcal{V}}, \mathsf{b}_{\mathcal{VD}}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}}) \leftarrow \mathsf{Vote}^{\langle \mathcal{V}, \mathcal{VD} \rangle}(\mathsf{pms}, \mathsf{m}, \emptyset) \\ \mathtt{ctxt} \in \mathsf{Trc} \\ \mathsf{b}_{\mathcal{V}} = 1 \text{ (voter accepts the proof)} \\ \mathsf{m} \neq \mathsf{Dec}(\mathtt{ctxt}, \mathsf{sk}) \text{ (but } \mathtt{ctxt} \text{ does not contain the intent)} \end{cases}$$

Since the commitment scheme Com is perfectly hiding, the voting device $\mathcal{VD}$ does not gain any information on the challenge e before choosing the first message com of the Σ-protocol. In such a case, the protocol is a zero-knowledge proof with soundess error $\frac{1}{|\mathsf{Chl}|} \in \mathsf{negl}(\lambda)$ (see for instance Theorem 6.5.2 in [60]). Therefore, we directly get that the probability of Cheat happening is negligible in λ, which implies that a transcript for an out-of-the-language instance (e.g., $\mathtt{ctxt}$ encrypts m' instead of m) will not be accepted: i.e., $\mathsf{b}_{\mathcal{V}} \neq 1$. □

Theorem 4.2.2 (Solution S_1 achieves CR)**.** Assuming the special honest-verifier zero-knowledge property of the Σ-protocol for $\mathcal{R}_{\mathbb{E}}$, the IND-CPA security of $\mathbb{E}$ and the binding property of Com, the protocol described in Figure 4.2 achieves the Coercion-Resistance (CR) property (Definition 8).

Proof. Other than choosing its preferred voting option $\mathsf{m}_{\mathcal{C}}$, the coercer $\mathcal{C}$ may eventually force the voter to deviate from the prescribed execution of (some of) the steps in the voting protocol. This part of the coercion may affect the distribution of the elements $\mathsf{e}, \hat{\mathsf{r}}, \mathsf{cmt}$, which are the only values generated/sent by the voter. We will model this fact with three functions $\mathcal{F}_{\mathsf{e}}$, $\mathcal{F}_{\hat{\mathsf{r}}}$, $\mathcal{F}_{\mathsf{cmt}}$ that describe how these three elements are to be computed by the coerced voter. These functions may depend on the voting option $\mathsf{m}_{\mathcal{C}}$ and values previously exchanged during the protocol. For instance, $\mathcal{F}_{\mathsf{e}}$ might be a function of $(\mathsf{m}_{\mathcal{C}}, \mathtt{ctxt}, \mathsf{com}, \mathsf{cmt})$.

Let us consider first coercions where the voter is forced to choose e or $\hat{\mathsf{r}}$ (in round 3 of the protocol Vote) as a function of the values $\mathsf{com}, \mathtt{ctxt}$ sent by $\mathcal{VD}$ in round 2. This means the value cmt sent by $\mathcal{V}$ in round 1 has been computed before the values $\mathsf{e}, \hat{\mathsf{r}}$ unless the value com has been guessed in advance, which happens with negligible probability. In such a situation, it must be $1 \neq \mathsf{Com.Open}(\mathsf{cmt}, \mathsf{e}, \hat{\mathsf{r}})$; otherwise, such a coercer could be used to break the binding property of the commitment scheme Com. Therefore, the CR experiment for these coercions will lead to $\mathcal{VD}$ rejecting the Vote protocol without answering the final z and setting $\mathsf{b}_{\mathcal{VD}} = 0$. Thus, the CR experiment will abort.

For the rest of coercions, the first execution of the Vote protocol results in $\mathtt{ctxt}^{(0)}$ being an encryption of $\mathsf{m}_{\mathcal{C}}$, where both $\mathsf{m}_{\mathcal{C}}$ and $\mathtt{ctxt}^{(0)}$ are part of $\mathsf{Trc}^{(0)}$.

The not-coerced execution of Vote run by Sim_1 with input message m, results in $\mathtt{ctxt}^{(1)}$ being an encryption of m, where both m and $\mathtt{ctxt}^{(1)}$ are part of Trc. Finally, what Sim_2 does the following:

1. since now the values of $\mathsf{e}, \hat{\mathsf{r}}$ do not depend on the values $\mathsf{com}, \mathtt{ctxt}$, values $\mathsf{e}^{(1)}, \hat{\mathsf{r}}^{(1)}, \mathsf{cmt}^{(1)}$ can be chosen first by Sim_2, following the instructions in $\mathcal{F}_{\mathsf{cmt}}, \mathcal{F}_{\mathsf{e}}, \mathcal{F}_{\hat{\mathsf{r}}}$. Since we are assuming that the CR experiment does not abort, these values satisfy $1 = \mathsf{Com.Open}(\mathsf{cmt}^{(1)}, \mathsf{e}^{(1)}, \hat{\mathsf{r}}^{(1)})$.

2. Sim_2 runs the simulator $\mathtt{Sim}_{\mathcal{R}_\mathbb{E}}$ associated to the zero-knowledge property of the Σ-protocol for $\mathcal{R}_\mathbb{E}$ with input $(\mathsf{m}_\mathcal{C}, \mathtt{ctxt}^{(1)})$, which is an element out of the language $\mathcal{L}_\mathbb{E}$, for the challenge $\mathsf{e}^{(1)}$. Let $(\mathsf{com}^{(1)}, \mathsf{e}^{(1)}, \mathsf{z}^{(1)})$ be the output of $\mathtt{Sim}_{\mathcal{R}_\mathbb{E}}$.
3. Sim_2 ends by defining $\mathsf{Trc}^{(1)} = (\mathsf{m}_\mathcal{C}, \mathtt{ctxt}^{(1)}, \mathsf{com}^{(1)}, \mathsf{e}^{(1)}, \mathsf{z}^{(1)}, \mathsf{cmt}^{(1)}, \hat{\mathsf{r}}^{(1)})$.

The computational indistinguishability between the distributions of $\mathsf{view}^{(0)}$ and $\mathsf{view}^{(1)}$ is directly implied by the zero-knowledge property of the Σ-protocol for $\mathcal{R}_\mathbb{E}$ and the IND-CPA security of $\mathbb{E}$, as stated in Lemma 2.5.1. □

4.2.2 Detailed Protocol For ElGamal Ciphertexts

Now we describe an instantiation of the generic construction from Figure 4.2 with the ElGamal public key encryption [54] and the Pedersen commitment [90] schemes. Also, we show how one can fake the transcript after the voting.

Recall that the ElGamal encryption scheme is a public-key encryption scheme (see Section 2.3.1) instantiated over a cyclic group $\mathbb{G}_\mathsf{q}$ of order q with generator g, where the decisional Diffie-Hellman (DDH) problem is believed to be hard (see Section 3). The default spaces for the messages, ciphertext, and randomness are $\mathbb{G}$, $\mathbb{G}$, and $\mathbb{Z}_\mathsf{q}$ respectively: i.e., $\mathcal{M}_\mathbb{E} = \mathbb{G}, \mathcal{C}_\mathbb{E} = \mathbb{G}, \mathcal{RS}_\mathbb{E} = \mathbb{Z}_\mathsf{q}$.

We will use the Pedersen commitment scheme with a commitment key $\mathsf{h} = (\mathbb{G}, \mathsf{h})$, where the element h is a generator of the group $\mathbb{G}$ created in a verifiable manner. By default, the parameters include the commitment key, the group and the group generator (i.e., $\mathsf{pms}_\mathsf{Com} = (\mathbb{G}, \mathsf{h}, \mathsf{g})$), and both the message and randomness spaces are equal to $\mathbb{Z}_\mathsf{q}$: i.e., $\mathcal{M}_\mathsf{Com} = \mathbb{Z}_\mathsf{q}, \mathcal{RS}_\mathsf{Com} = \mathbb{Z}_\mathsf{q}$.

Thus, the public parameters are $\mathsf{pms} = (\mathbb{G}, \mathsf{h}, \mathsf{pk}, \mathsf{g}, \mathbb{Z}_\mathsf{q})$.

The resulting protocol, with four rounds of communication, works as described in Figure 4.3.

To fake a transcript for the coercer after voting, the voter must select $\mathsf{z}^* \xleftarrow{\$} \mathbb{Z}_\mathsf{q}$ at random, then compute $\mathsf{com}^* = (\mathsf{a}_1^*, \mathsf{a}_2^*) \leftarrow (\mathsf{g}^{\mathsf{z}^*}\mathsf{c}_1^{-\mathsf{e}}, \mathsf{pk}^{\mathsf{z}^*}(\frac{\mathsf{c}_2}{\mathsf{m}_\mathcal{C}})^{-\mathsf{e}})$. The resulting transcript $(\mathsf{cmt}, \mathtt{ctxt}, \mathsf{com}^*, \mathsf{e}, \hat{\mathsf{r}}, \mathsf{z}^*)$ would be indistinguishable from the real transcript, even though $\mathtt{ctxt}$ encrypts the voter's intent m instead of the coercer's wish $\mathsf{m}_\mathcal{C}$.

We implemented the solution S_1 in Rust for four different cyclic groups: two groups of integers (based on safe primes of 2048 and 3072 bits, respectively) and two Twisted Edwards elliptic curves (Curve25519 and Ed448-Goldilock, respectively). According to NIST[1] recommendations, the estimated security strength for factoring discrete logarithms in the group of 2048 bits is 112 and 128 for groups of 3072 bits. The selected elliptic curves Curve25519 and Ed448-Goldilock are estimated to have security strengths of approximately 126 and 223, respectively.

The tests were performed on an Intel Core i7-1165G7 CPU. Typical performance results are presented in Table 4.1. As expected, elliptic curves demonstrate better performance results due to the smaller element size. Nevertheless, the protocol is lightweight and would not pose usability issues if used in practice for any of the presented groups.

[1]The abbreviation NIST stands for the National Institute of Standards and Technology. It is an agency of the United States Department of Commerce founded in 1901 that aims to promote security standards and guidelines.

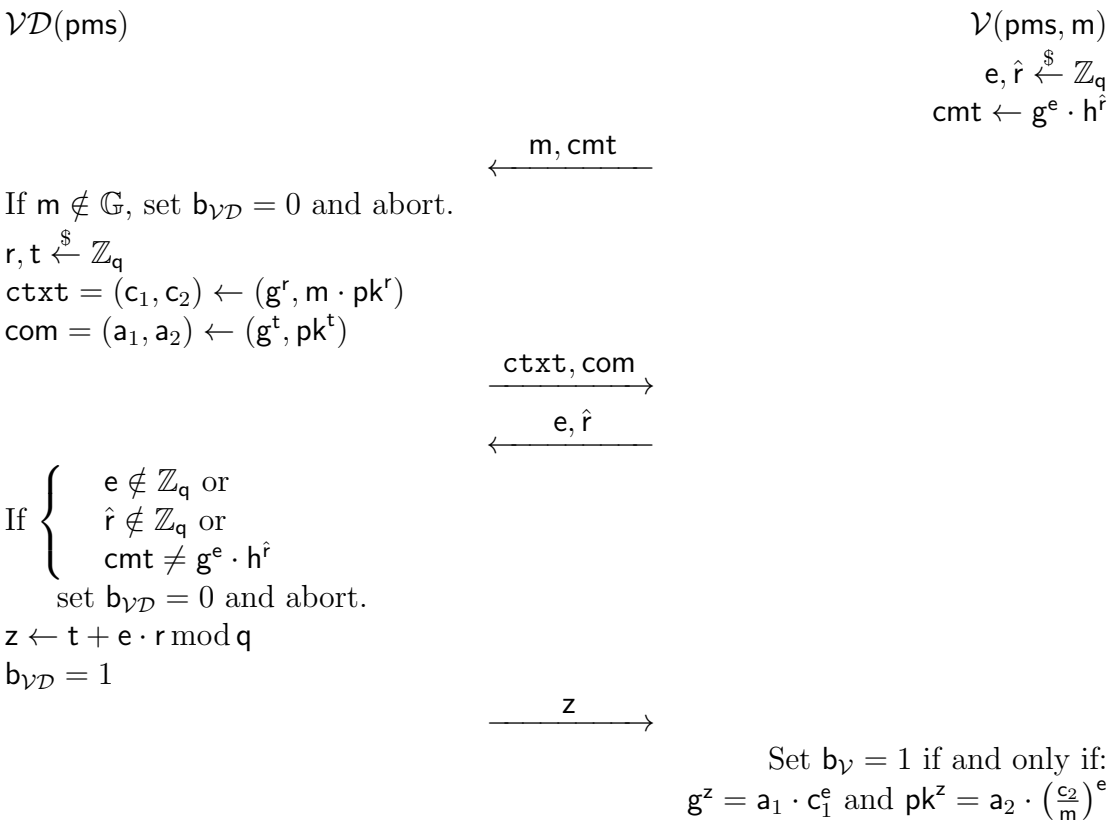


Figure 4.3: The ElGamal instantiation of the solution S_1.

Steps \ Group	$\mathsf{p} = 2\mathsf{q} + 1$ p is 2048 bits q is 2047 bits	$\mathsf{p} = 2\mathsf{q} + 1$ p is 3072 bits q is 3071 bits	Elliptic curve Curve25519	Elliptic curve Ed448 (Goldilocks)
Step 1: $\mathcal{V}$ computes cmt (commits to e)	3.039102ms	8.510232ms	82.335µs	419.701µs
Step 2: $\mathcal{VD}$ starts the proof (gets $\mathtt{ctxt}$ and com)	12.503429ms	34.602619ms	481.415µs	2.347005ms
Step 4: $\mathcal{VD}$ finishes the proof (computes z)	2.919297ms	9.618284ms	49.082µs	316.121µs
Step 5: $\mathcal{V}$ runs Verify (verifies the proof)	11.72786ms	33.579814ms	183.512µs	1.406092ms
$\mathcal{V}$'s total time	14.766962ms	42.090046ms	265.847µs	1.825793ms
$\mathcal{VD}$'s total time	15.422726ms	44.220903ms	530.497µs	2.663126ms

Table 4.1: Performance results of S_1 implementation in different groups.

4.3 Unsatisfactory Solution U_2

Before describing the second solution S_2, we recall the original OR-proof idea from Section 1.1.3.4 as it will be the basis for our second coercion-resistant cast-as-intended protocol, see Figure 4.4 for details.

Recall a public-key encryption scheme $\mathbb{E}$ from Section 2.3 that consists of three algorithms $(\mathbb{E}.\mathsf{KG}, \mathbb{E}.\mathsf{Enc}, \mathbb{E}.\mathsf{Dec})$. Then define an OR-relation that we wish to prove using Σ-protocol $\langle \mathcal{VD}, \mathcal{V} \rangle$ between the voter and the voting device for some one-way function OWF:

$$\mathcal{R}_{\text{OR}} = \{(\mathtt{ctxt}, \mathsf{m}, \mathsf{y}) \mid \mathtt{ctxt} \text{ encrypts } \mathsf{m} \text{ OR I know } \mathsf{x} \text{ s.t. } \mathsf{y} = \mathsf{OWF}(\mathsf{x})\}$$

The voting device $\mathcal{VD}$ uses the simulator $\mathtt{Sim}$ of the Σ-protocol to create a transcript for the statement with an unknown witness (i.e., I know x s.t, $\mathsf{y} = \mathsf{OWF}(\mathsf{x})$) for a randomly sampled challenge e_1. Then it runs the first step of the Σ-protocol for the other statement (i.e., $\mathtt{ctxt}$ encrypts m) for which it knows the witness x (i.e., randomness r used to compute $\mathtt{ctxt}$). Thus after the first step, the voting device $\mathcal{VD}$ obtains $(\mathsf{com}_1, \mathsf{z}_1) \leftarrow \mathtt{Sim}(\mathsf{y}, \mathsf{e}_1)$ and $(\mathsf{com}_0, \mathsf{op}_0) \leftarrow \Sigma_1(\mathtt{ctxt}, \mathsf{m}, \mathsf{r})$.

In the interactive case, the voting device sends $\mathsf{com}_0, \mathsf{com}_1$ to the voter $\mathcal{V}$, which replies with a challenge $\mathsf{e} \in \mathsf{Chl}$. However, in the non-interactive proof, the voting device computes the challenge e by applying a Fiat-Shamir transformation to values $\mathsf{y}, \mathsf{com}_0, \mathsf{com}_1$ using a collision-resistant hash $\mathsf{H}_{\text{OR}} : \{0,1\}^* \to \mathsf{Chl}$. After obtaining the value e, the voting device computes a challenge to the non-simulated transcript as $\mathsf{e}_0 \leftarrow \mathsf{e} \oplus \mathsf{e}_1$ and finishes the Σ-protocol $\mathsf{z}_0 \leftarrow \Sigma_2(\mathsf{e}_0, \mathsf{op}_0, \mathsf{r})$, where $\oplus$ is an XOR operation. The resulting proof π is $(\mathsf{com}_0, \mathsf{e}_0, \mathsf{z}_0, \mathsf{com}_1, \mathsf{e}_1, \mathsf{z}_1)$.

To verify the proof, the voter $\mathcal{V}$ will need to validate both Σ-proofs $(\mathsf{com}_0, \mathsf{e}_0, \mathsf{z}_0)$ and $(\mathsf{com}_1, \mathsf{e}_1, \mathsf{z}_1)$ from the proof π and then ensure that $\mathsf{e}_0 \oplus \mathsf{e}_1 = \mathsf{e}$.

Hence, the resulting protocol is simple and described in Figure 4.4. During the first round, the voter samples a trapdoor x and applies one-way function OWF to obtain y. Then it sends y and the selected option m to the voting device, which replies with a non-interactive OR-proof of the form: "$\mathtt{ctxt}$ encrypts m OR I know x s.t. $\mathsf{y} = \mathsf{OWF}(\mathsf{x})$".

We denote as pms all public parameters required for the public-key encryption scheme $\mathbb{E}$ and a Σ-protocol. Those parameters might include the mathematical group, group generator, hash functions, etc.

The OR-proof proposed in Figure 4.4 achieves receipt-freeness since a voter (who knows the trapdoor x) can simulate proof. However, it is not coercion-resistant as two-round protocols cannot simultaneously satisfy coercion-resistance and cast-as-intended definitions, which was proven in Section 3.5 Recall that a coercer can generate an instance (x, y) and force the voter to use y without knowing the corresponding trapdoor x. Since OWF is a one-way function, the chance that the voter would guess x and simulate the transcript is negligible.

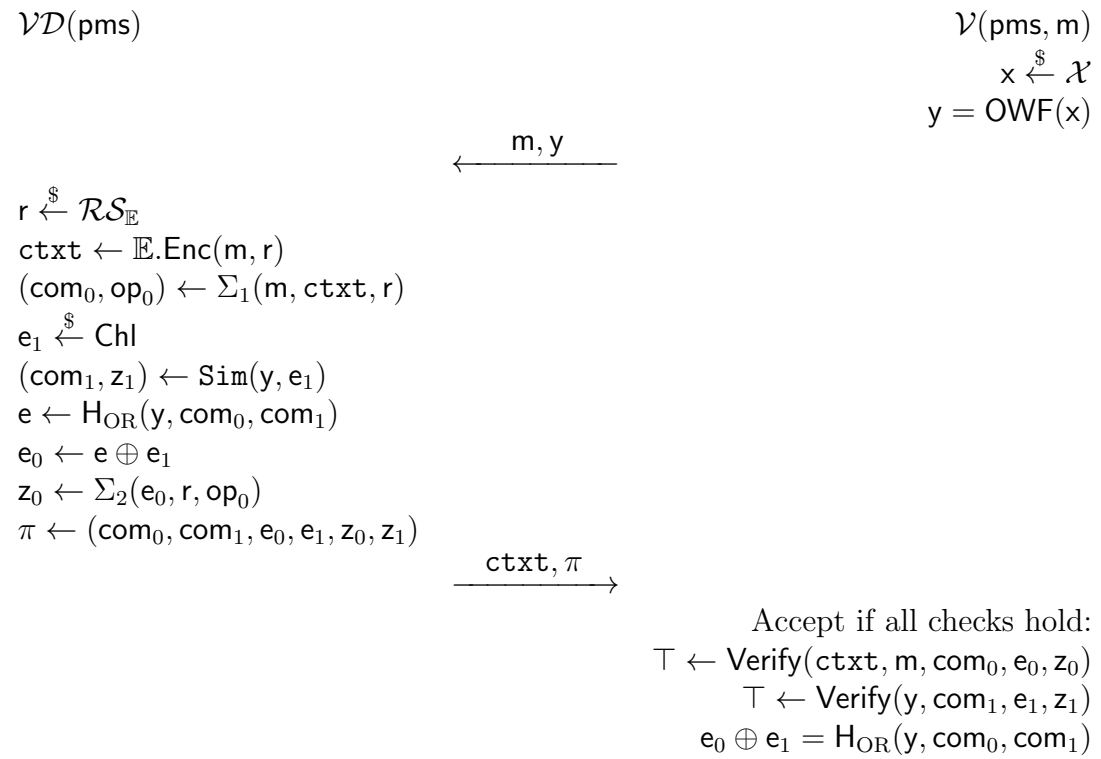


Figure 4.4: OR proof for proving the ciphertext encrypts the intended message.

4.4 Solution S_2: Adding Interactive Proof Of The Trapdoor Knowledge

The departing point for the solution S_2 described in Figure 4.5 is the unsatisfactory OR-proof-based solution U_2 described in Section 4.3. The problem with U_2 was that the coercer could give the voter only y (and optionally a non-interactive proof of x knowledge) without revealing x, which would prevent the voter from simulating the transcript.

However, we can easily fix U_2 with the idea proposed in [50]. The solution is to add a proof of knowledge of $\mathsf{x} \in \mathcal{X}$ but ensure that such proof cannot be pre-computed by the coercer in advance. We require interaction between $\mathcal{VD}$ and $\mathcal{V}$. However, a standard Σ protocol has only an honest verifier zero-knowledge, which is not enough because the verifier, $\mathcal{VD}$ in this case, can be malicious. Hence, we cannot assume that the resulting proof is zero-knowledge and leaks nothing about the trapdoor x, which is bad for cast-as-intended verifiability. Our solution is a modified interactive protocol, which combines Fiat-Shamir for a standard Σ-protocol with a random challenge aux; it enjoys the soundness and the full zero-knowledge in the Random Oracle Model. As a result, we need a Σ-protocol $\langle \mathtt{P}, \mathtt{V} \rangle$ for the relation $\mathcal{R}_{\mathsf{OWF}} = \{(\mathsf{x}, \mathsf{y}) \in \mathcal{X} \times \mathcal{Y} \mid \mathsf{y} = \mathsf{OWF}(\mathsf{x})\}$ with the challenge space $\mathsf{Chl}_{\mathsf{OWF}} \subseteq \mathcal{X}$ (with size exponential in λ) and a collision-resistant hash function defined as $\mathsf{H}_{\mathsf{OWF}} : \{0,1\}^* \to \mathsf{Chl}_{\mathsf{OWF}}$. Additionally, we require a collision resistant hash function $\mathsf{H} : \{0,1\}^* \to \mathcal{C}_{\mathsf{Com}}$, where $\mathcal{C}_{\mathsf{Com}}$ is an appropriate commitment space.

Since now both the voter $\mathcal{V}$ and the voting device $\mathcal{VD}$ run Σ-protocols, we will

use the superscript of a player to differentiate between them. Thus, the voter $\mathcal{V}$ would run $\Sigma^{\mathcal{V}} = (\Sigma_1^{\mathcal{V}}, \Sigma_2^{\mathcal{V}})$ to prove the trapdoor x knowledge, while the voting device $\mathcal{VD}$ would use $\Sigma_{\mathrm{OR}}^{\mathcal{VD}} = (\Sigma_{1,\mathsf{r}}^{\mathcal{VD}}, \Sigma_{2,\mathsf{r}}^{\mathcal{VD}}, \Sigma_{1,\mathsf{x}}^{\mathcal{VD}}, \Sigma_{2,\mathsf{x}}^{\mathcal{VD}})$ and a collision-resistant hash $\mathsf{H}_{\mathrm{OR}} : \{0,1\}^* \to \mathsf{Chl}$ to non-interactively prove that it either knows the randomness r or the trapdoor x. To simulate proof of the trapdoor x knowledge, the $\mathcal{VD}$ would run $\mathtt{Sim}_{\mathsf{x}}^{\mathcal{VD}}$. Similarly, it would run $\mathtt{Sim}_{\mathsf{r}}^{\mathcal{VD}}$ to simulate the knowledge of the randomness r.

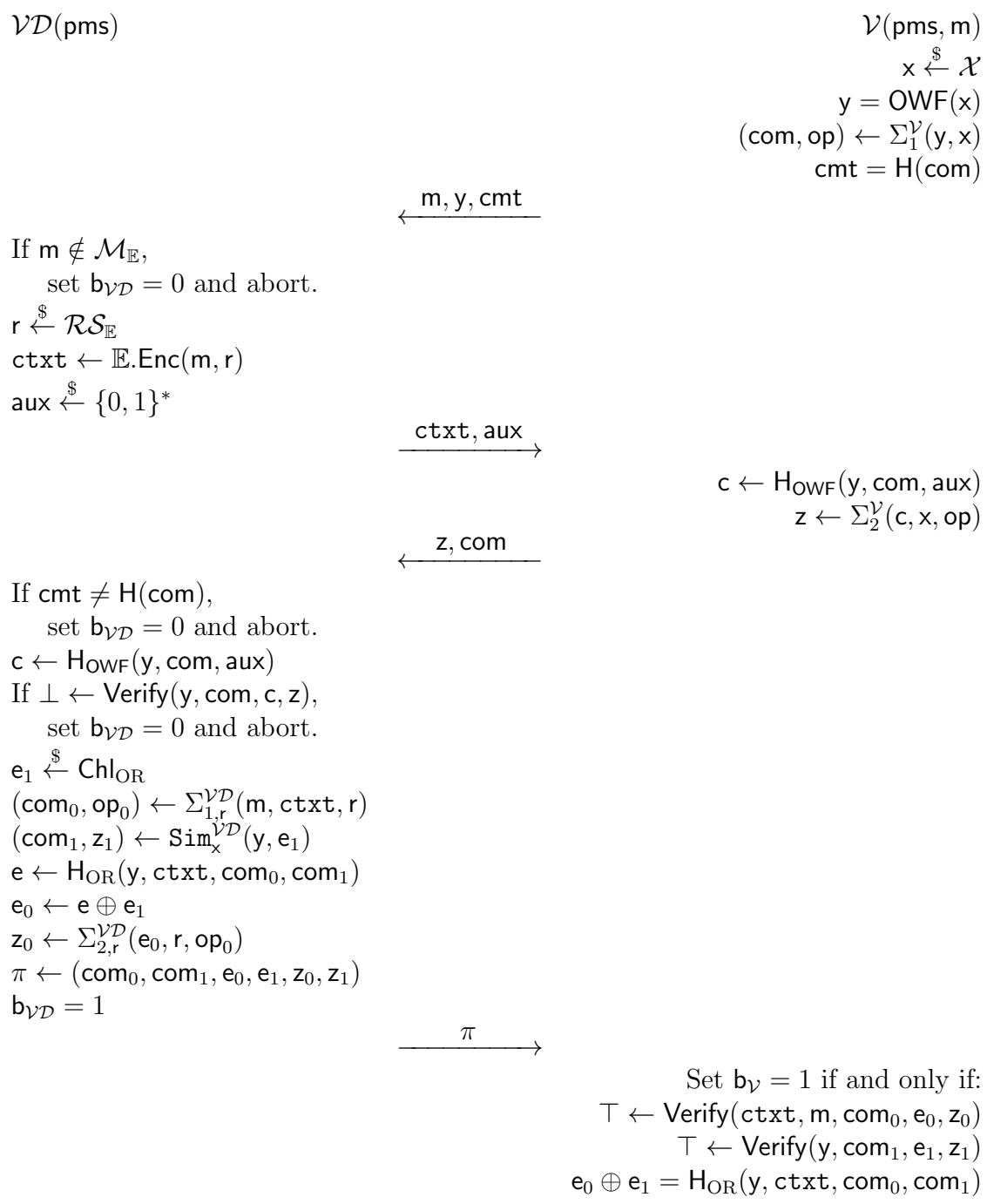


Figure 4.5: The voting protocol from solution S_2.

The setup algorithm $\mathsf{pms} \leftarrow \mathsf{Setup}(\lambda)$ runs $\mathbb{E}.\mathsf{KG}(\lambda)$ resulting in key pair $(\mathsf{pk}, \mathsf{sk})$

for the encryption scheme $\mathbb{E}$, along with spaces $\mathcal{M}_{\mathbb{E}}, \mathcal{C}_{\mathbb{E}}, \mathcal{RS}_{\mathbb{E}}$. Note that the secret key sk is kept private. Then it defines challenge space for the Σ-protocol $\mathsf{Chl}_{\mathrm{OR}} \subseteq \mathcal{M}_{\mathbb{E}}$ and sets public parameters as follows:

$$\mathsf{pms} = (\mathsf{pk}, \mathcal{M}_{\mathbb{E}}, \mathcal{C}_{\mathbb{E}}, \mathcal{RS}_{\mathbb{E}}, \mathsf{Chl}_{\mathsf{OWF}}, \mathsf{Chl}_{\mathrm{OR}}, \mathsf{H}_{\mathsf{OWF}}, \mathsf{H}_{\mathrm{OR}}, \mathsf{H})$$

The voting protocol $(\mathsf{b}_{\mathcal{V}}, \mathsf{b}_{\mathcal{VD}}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}}) \leftarrow \mathsf{Vote}^{\langle \mathcal{V}, \mathcal{VD} \rangle}(\mathsf{pms}, \mathsf{m}, \emptyset)$ with $\mathsf{coerc} = \emptyset$ works as described in Figure 4.5.

If $\mathsf{b}_{\mathcal{VD}} = 1$, then the voting device $\mathcal{VD}$ accepts the interaction as correct. Similarly, if $\mathsf{b}_{\mathcal{V}} = 1$, the voter $\mathcal{V}$ trusts that the ballot contains the intended option. The vote is only deemed valid when $\mathsf{b}_{\mathcal{V}} = b_{\mathcal{VD}} = 1$. Otherwise, whenever $\mathsf{b}_{\mathcal{VD}} = 0$ - the voting device $\mathcal{VD}$ aborts the vote-casting procedure, and if $\mathsf{b}_{\mathcal{V}} = 0$ - the voter complains to authorities which invalidates the vote.

Following the notation introduced in Section 3.3, we have $\mathsf{rand}_{\mathcal{V}} = (\mathsf{x}, \mathsf{y}, \mathsf{com}, \mathsf{op})$ and $\mathsf{Trc} = (\mathsf{y}, \mathsf{m}, \mathtt{ctxt}, \mathsf{e}, \mathsf{z}, \pi, \mathsf{b}_{\mathcal{VD}}, \mathsf{b}_{\mathcal{V}})$.

4.4.1 Security Analysis Of The Protocol

Theorem 4.4.1 (Solution S_2 achieves CAI)**.** Assuming the honest verifier zero-knowledge of the $\Sigma^{\mathcal{V}}$-protocol for $\mathcal{R}_{\mathsf{OWF}}$, the special soundness of the $\Sigma^{\mathcal{VD}}_{\mathrm{OR}}$-protocol for $\mathcal{R}_{\mathrm{OR}}$, and that OWF is a one-way function and the output com of the 1st round of $\Sigma^{\mathcal{V}}$-protocol is a uniform element in a set with exponential size in the security parameter λ, the protocol described in Figure 4.5 achieves the Cast-as-Intended (CAI) property in the Random Oracle Model as per Definition 7.

Proof. Remember that the CAI property is satisfied if the probability of Cheat is negligible in the security parameter λ, where Cheat is defined as follows:

$$\mathsf{Cheat} = \begin{cases} \mathsf{pms} \leftarrow \mathsf{Setup}(\lambda) \\ (\mathsf{b}_{\mathcal{V}}, \mathsf{b}_{\mathcal{VD}}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}}) \leftarrow \mathsf{Vote}^{\langle \mathcal{V}, \mathcal{VD} \rangle}(\mathsf{pms}, \mathsf{m}, \emptyset) \\ \mathtt{ctxt} \in \mathsf{Trc} \\ \mathsf{b}_{\mathcal{V}} = 1 \text{ (voter accepts the proof)} \\ \mathsf{m} \neq \mathsf{Dec}(\mathtt{ctxt}, \mathsf{sk}) \text{ (but } \mathtt{ctxt} \text{ does not contain the intent)} \end{cases}$$

Since com is a uniform element in a set with exponential size and H behaves as a random oracle, the polynomial-time $\mathcal{VD}$ obtains no information on com from the hash cmt. Thus, the auxiliary string aux is selected independently of com.

The three-moves protocol in Steps 1, 2, and 3 where $\mathcal{V}$ proves to $\mathcal{VD}$ the knowledge of x is full zero-knowledge (even for malicious verifiers $\mathcal{VD}$) in the Random Oracle Model. In a nutshell, for any verifier (in this case $\mathcal{VD}'$) and any choice of aux (possibly depending on y and cmt), a transcript of this sub-protocol can be simulated for any y, without knowing x as following: choose z, c at random, simulate appropriate com (using the HVZK property of the underlying $\Sigma^{\mathcal{V}}$ protocol), then compute $\mathsf{cmt} = \mathsf{H}(\mathsf{com})$ and, for any aux picked by $\mathcal{VD}'$, program random oracle of $\mathsf{H}_{\mathsf{OWF}}$ s.t. $\mathsf{H}_{\mathsf{OWF}}(\mathsf{y}, \mathsf{cmt}, \mathsf{aux}) = \mathsf{c}$.

Zero-knowledge of the proof generated by $\mathcal{V}$ and the fact that OWF is a one-way function imply that the voting device does not learn the trapdoor x. In turn, it

means that, for passing voter's verification, the voting device must generate a valid OR proof without knowing x and so, without knowing the witness for one of the two statements of the OR language. The special soundness property of the $\Sigma_{\mathrm{OR}}^{\mathcal{VD}}$-proof for $\mathcal{R}_{\mathrm{OR}}$ guarantees soundness of the resulting OR proof with the soundness error $\frac{1}{|\mathsf{Chal}_{\mathrm{OR}}|} \in \mathsf{negl}(\lambda)$ (see for instance Proposition 6.2.3 in [60]).

Therefore, the probability of Cheat happening is negligible in λ, which implies that a transcript for an out-of-the-language instance (e.g., $\mathtt{ctxt}$ encrypts m' instead of m) will not be accepted: i.e., $\mathsf{b}_{\mathcal{V}} \neq 1$. □

Theorem 4.4.2 (Solution S_2 achieves CR)**.** Assuming the zero-knowledge of $\Sigma_{\mathrm{OR}}^{\mathcal{VD}}$-protocol for $\mathcal{R}_{\mathrm{OR}}$, special soundness property of the $\Sigma^{\mathcal{V}}$-protocol for $\mathcal{R}_{\mathsf{OWF}}$, collision-resistance of the hash functions H, H_{OR} and $\mathsf{H}_{\mathsf{OWF}}$ and the IND-CPA security of $\mathbb{E}$ the protocol described in Figure 4.5 achieves the Coercion-Resistance (CR) property as per Definition 8.

Proof. Other than choosing its preferred voting option $\mathsf{m}_{\mathcal{C}}$, the coercer $\mathcal{C}$ may eventually force the voter to deviate from the correct execution of (some of) the steps in the voting protocol.

Recall that the voter only generates/sends the following values: $(\mathsf{cmt}, \mathsf{y}, \mathsf{com}, \mathsf{z})$. Because $(\mathsf{y}, \mathsf{cmt})$ are sent in the first round, the coercer cannot make them dependent on what is sent later. To pass the voting device's check, com must be such that $\mathsf{H}(\mathsf{com}) = \mathsf{cmt}$, which means it cannot depend on anything sent in the second round. Hence, the coercer cannot enforce dependency between the voter $\mathcal{V}$ values $(\mathsf{cmt}, \mathsf{y}, \mathsf{com})$ and the voting device choice of auxiliary string aux in Step 2.

Moreover, in Step 3, the voter must successfully reply to any aux sent by the voting device. Otherwise, a voting device would abort the execution of the voting protocol. Therefore, the voter must know the witness x, or else the special soundness property of the underlying proof of x knowledge would be broken.

Now we show how to simulate the transcript for the coercer. The coerced execution of the Vote protocol results in $\mathtt{ctxt}^{(0)}$ being an encryption of $\mathsf{m}_{\mathcal{C}}$, where both $\mathsf{m}_{\mathcal{C}}$ and $\mathtt{ctxt}^{(0)}$ are part of $\mathsf{Trc}^{(0)}$. The not-coerced execution of Vote with new instructions provided by Sim_1 for the voter's voting option m, results in $\mathtt{ctxt}^{(1)}$ being an encryption of m, where both m and $\mathtt{ctxt}^{(1)}$ are part of Trc. Then, Sim_2 does the following:

1. sets values $(\mathsf{cmt}, \mathsf{y}, \mathsf{com}, \mathsf{aux}, \mathsf{z})$ to be identical in both views;
2. runs $\Sigma_{1,\mathsf{x}}^{\mathcal{VD}}(\mathsf{y}, \mathsf{x})$ to obtain $(\mathsf{com}_1^{(1)}, \mathsf{op}_1^{(1)})$.
3. runs the simulator $\mathtt{Sim}_{\mathsf{r}}^{\mathcal{VD}}$ associated to the zero-knowledge property of the $\Sigma_{\mathrm{OR}}^{\mathcal{VD}}$-protocol for a randomly selected challenge $\mathsf{e}_0^{(1)}$ to obtain $(\mathsf{com}_0^{(1)}, \mathsf{z}_0^{(1)})$.
4. computes $\mathsf{e}^{(1)} = \mathsf{H}_{\mathrm{OR}}(\mathsf{y}, \mathtt{ctxt}^{(1)}, \mathsf{com}_0^{(1)}, \mathsf{com}_1^{(1)})$.
5. sets $\mathsf{e}_1^{(1)} \leftarrow \mathsf{e}^{(1)} \oplus \mathsf{e}_0^{(1)}$
6. runs $\Sigma_{2,\mathsf{x}}^{\mathcal{V}}(\mathsf{e}_1^{(1)}, \mathsf{x}, \mathsf{op}_1^{(1)})$ to get $\mathsf{z}_1^{(1)}$.
7. defines $\pi^{(1)}$ as $(\mathsf{com}_0^{(1)}, \mathsf{e}_0^{(1)}, \mathsf{z}_0^{(1)}, \mathsf{com}_1^{(1)}, \mathsf{e}_1^{(1)}, \mathsf{z}_1^{(1)})$.

Sim_2 ends by defining $\mathsf{Trc}^{(1)} = (\mathsf{cmt}, \mathsf{y}, \mathsf{com}, \mathtt{ctxt}^{(1)}, \mathsf{aux}, \mathsf{z}, \pi^{(1)}, \mathsf{b}_{\mathcal{VD}}, \mathsf{b}_{\mathcal{V}})$. The computational indistinguishability between the distributions of $\mathsf{view}^{(0)}$ and $\mathsf{view}^{(1)}$ is directly implied by the zero-knowledge property of the non-interactive Σ-protocol for $\mathcal{R}_{\mathrm{OR}}$ and the IND-CPA security of $\mathbb{E}$. □

4.4.2 Detailed Protocol For ElGamal Ciphertexts

$\mathcal{VD}(\mathsf{pms})$ $\qquad$ $\mathcal{V}(\mathsf{pms}, \mathsf{m})$

$\mathsf{x}, \mathsf{t} \xleftarrow{\$} \mathbb{Z}_{\mathsf{q}}$

$\mathsf{y} \leftarrow \mathsf{g}^{\mathsf{x}}$

$\mathsf{com} \leftarrow \mathsf{g}^{\mathsf{t}}$

$\mathsf{cmt} = \mathsf{H}(\mathsf{com})$

$\xleftarrow{\mathsf{m}, \mathsf{y}, \mathsf{cmt}}$

If $\mathsf{m} \notin \mathbb{G}$, set $\mathsf{b}_{\mathcal{VD}} = 0$ and abort.

$\mathsf{aux} \xleftarrow{\$} \{0,1\}^*$

$\mathsf{r} \xleftarrow{\$} \mathbb{Z}_{\mathsf{q}}$

$\mathtt{ctxt} = (\mathsf{c}_1, \mathsf{c}_2) \leftarrow (\mathsf{g}^{\mathsf{r}}, \mathsf{m} \cdot \mathsf{pk}^{\mathsf{r}})$

$\xrightarrow{\mathtt{ctxt}, \mathsf{aux}}$

$\mathsf{c} = \mathsf{H}(\mathsf{y}, \mathsf{com}, \mathsf{aux})$

$\mathsf{z} \leftarrow \mathsf{t} + \mathsf{c} \cdot \mathsf{x} \bmod \mathsf{q}$

$\xleftarrow{\mathsf{z}, \mathsf{com}}$

If $\mathsf{cmt} \neq \mathsf{H}(\mathsf{com})$
  set $\mathsf{b}_{\mathcal{VD}} = 0$ and abort.

$\mathsf{c} = \mathsf{H}(\mathsf{y}, \mathsf{com}, \mathsf{aux})$

If $\mathsf{g}^{\mathsf{z}} \neq \mathsf{com} \cdot \mathsf{y}^{\mathsf{c}}$
  set $\mathsf{b}_{\mathcal{VD}} = 0$ and abort.

$\mathsf{t}_0, \mathsf{z}_1, \mathsf{e}_1 \xleftarrow{\$} \mathbb{Z}_{\mathsf{q}}$

$\mathsf{a}_0 = (\mathsf{a}_{01}, \mathsf{a}_{02}) \leftarrow (\mathsf{g}^{\mathsf{t}_0}, \mathsf{pk}^{\mathsf{t}_0})$

$\mathsf{a}_1 \leftarrow \mathsf{g}^{\mathsf{z}_1} \cdot \mathsf{y}^{-\mathsf{e}_1}$

$\mathsf{e} = \mathsf{H}(\mathsf{y}, \mathtt{ctxt}, \mathsf{a}_0, \mathsf{a}_1)$

$\mathsf{e}_0 = \mathsf{e} - \mathsf{e}_1 \bmod \mathsf{q}$

$\mathsf{z}_0 = \mathsf{t}_0 + \mathsf{e}_0 \cdot \mathsf{r}$

$\pi = (\mathsf{z}_0, \mathsf{z}_1, \mathsf{e}_0, \mathsf{e}_1)$

$\mathsf{b}_{\mathcal{VD}} = 1$

$\xrightarrow{\pi}$

$\mathsf{a}_0 = (\mathsf{g}^{\mathsf{z}_0} \cdot \mathsf{c}_1^{-\mathsf{e}_0}, \mathsf{pk}^{\mathsf{z}_0} \cdot \left(\frac{\mathsf{c}_2}{\mathsf{m}}\right)^{-\mathsf{e}_0})$

$\mathsf{a}_1 = \mathsf{g}^{\mathsf{z}_1} \cdot \mathsf{y}^{-\mathsf{e}_1}$

Set $\mathsf{b}_{\mathcal{V}} = 1$ if and only if:

$\mathsf{e}_0 + \mathsf{e}_1 \bmod \mathsf{q} = \mathsf{H}(\mathsf{y}, \mathtt{ctxt}, \mathsf{a}_0, \mathsf{a}_1)$

Figure 4.6: The ElGamal instantiation of the solution S_2.

Now we describe an instantiation of the generic construction from Figure 4.5 with the ElGamal public key encryption [54]. Recall that the ElGamal encryption scheme

is a public-key encryption scheme (see Section 2.3.1) instantiated over a cyclic group $\mathbb{G}_\mathsf{q}$ of order q with generator g, where the decisional Diffie-Hellman (DDH) problem is believed to be hard (see Section 3). The default spaces for the messages, ciphertext, and randomness are $\mathbb{G}$, $\mathbb{G}$, and $\mathbb{Z}_\mathsf{q}$ respectively: i.e., $\mathcal{M}_\mathbb{E} = \mathbb{G}, \mathcal{C}_\mathbb{E} = \mathbb{G}, \mathcal{RS}_\mathbb{E} = \mathbb{Z}_\mathsf{q}$. For the simplicity of notations, both hash functions H_{OR} and $\mathsf{H}_{\mathsf{OWF}}$ are defined as $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_\mathsf{q}$. Also, we will define OWF function as follows $\mathsf{OWF}(\mathsf{x}) : \mathsf{y} = \mathsf{g}^\mathsf{x}$, therefore $\mathcal{X} = \mathbb{Z}_\mathsf{q}$ and $\mathcal{Y} = \mathbb{G}$. Thus, the public parameters are $\mathsf{pms} = (\mathbb{G}, \mathsf{pk}, \mathbb{Z}_\mathsf{q}, \mathsf{g}, \mathsf{OWF}, \mathsf{H})$.

The resulting protocol, with four rounds of communication, works as described in Figure 4.6.

To fake a transcript for the coercer after voting, the voter must use the same $(\mathsf{aux}, \mathsf{cmt}, \mathsf{com}, \mathsf{z})$, then select $\mathsf{z}_0, \mathsf{e}_0, \mathsf{t}_1 \xleftarrow{\$} \mathbb{Z}_\mathsf{q}$ at random, and compute $\mathsf{a}_0 = (\mathsf{g}^{\mathsf{z}_0} \cdot \mathsf{c}_1^{-\mathsf{e}_0}, \mathsf{pk}^{\mathsf{z}_0} \cdot \left(\frac{\mathsf{c}_2}{\mathsf{m}_\mathcal{C}}\right)^{-\mathsf{e}_0})$ and $\mathsf{a}_1 = \mathsf{g}^{\mathsf{t}_1}$. Then define $\mathsf{e} = \mathsf{H}(\mathsf{y}, \mathtt{ctxt}, \mathsf{a}_0, \mathsf{a}_1)$, set the challenge $\mathsf{e}_1 \leftarrow \mathsf{e} - \mathsf{e}_0 \bmod \mathsf{q}$ and compute $\mathsf{z}_1 = \mathsf{t}_1 + \mathsf{e}_1 \cdot \mathsf{x} \bmod \mathsf{q}$. Finally, the voter sets $\pi = (\mathsf{z}_0, \mathsf{z}_1, \mathsf{e}_0, \mathsf{e}_1)$. The resulting transcript $(\mathsf{m}_C, \mathsf{cmt}, \mathsf{y}, \mathsf{com}, \mathtt{ctxt}, \mathsf{aux}, \mathsf{z}, \pi)$ would be indistinguishable from the real transcript, even though $\mathtt{ctxt}$ encrypts the voter's intent m instead of the coercer's wish $\mathsf{m}_\mathcal{C}$.

Steps \ Group	$\mathsf{p} = 2\mathsf{q} + 1$ p is 2048 bits q is 2047 bits	$\mathsf{p} = 2\mathsf{q} + 1$ p is 3072 bits q is 3071 bits	Elliptic curve Curve25519	Elliptic curve Ed448 (Goldilocks)
Step 1: $\mathcal{V}$ computes com and y (start the proof)	7.384294ms	19.041588ms	1.079912ms	4.503275ms
Step 2: $\mathcal{VD}$ computes $\mathtt{ctxt}$ and aux (challenges $\mathcal{V}$)	7.275616ms	19.799465ms	2.010838ms	11.971917ms
Step 3: $\mathcal{V}$ computes z (finishes the proof)	449.281µs	776.145µs	181.422µs	277.99µs
Step 4: $\mathcal{VD}$ computes π (finishes the OR-proof)	19.23375ms	54.285146ms	2.117673ms	13.602927ms
Step 5: $\mathcal{V}$ runs Verify (verifies the proof)	20.337231ms	55.966615ms	2.214394ms	14.064987ms
$\mathcal{V}$'s total time	28.170806ms	75.784348ms	3.475728ms	18.846252ms
$\mathcal{VD}$'s total time	26.509366ms	74.084611ms	4.128511ms	25.574844ms

Table 4.2: Performance results of S_2 implementation in different groups.

We implemented the solution S_2 in Rust for four different cyclic groups: two groups of integers (based on safe primes of 2048 and 3072 bits, respectively) and two Twisted Edwards elliptic curves (Curve25519 and Ed448-Goldilock, respectively). According to NIST recommendations, the estimated security strength for factoring discrete logarithms in the group of 2048 bits is 112 and 128 for groups of 3072 bits.

The selected elliptic curves Curve25519 and Ed448-Goldilock are estimated to have security strengths of approximately 126 and 223, respectively.

The tests were performed on an Intel Core i7-1165G7 CPU. Typical performance results are presented in Table 4.2. As expected, the S_2 protocol is a bit slower than S_1 due to the computation of a heavier OR-proof. Nevertheless, it would not pose usability issues if used in practice for any of the presented groups.

Chapter 5

Post-Quantum Solution

In this chapter, we explain how to translate the solutions presented in the previous chapter to the post-quantum settings. We start by highlighting a usability problem - the possibility of multiple protocol repetitions - that arises if we simply switch to lattice-based primitives without any adjustments in Section 5.1. Then, in Section 5.2, we give the lattice-based cryptography basics. After that, in Section 5.3, we propose a generic transformation, which would solve the need for protocol re-runs. We prove that transformation is secure in Section 5.4 and discuss the possible improvements and extensions in Section 5.5. Finally, in Section 5.6, we present an implementation to confirm that our proposal is efficient.

5.1 Why We Cannot Use Our Solutions Directly In Lattice Settings?

Standard cryptography relies on mathematical problems (like discrete logarithm, DDH, factoring, etc.) requiring so much computing force to solve them that (with the correct parameters) they are computationally unbreakable. However, the pending arrival of quantum algorithms will change that and drastically reduce the solving time for these problems. Therefore, the cryptographic world is looking for quantum-resistant replacements.

One of the most promising settings for achieving security in front of quantum computers is lattice-based cryptography. There are many algorithmic problems related to lattices, some of which are hard to solve even for quantum computers. Among many, there is the Shortest Vector Problem (SVP), the Shortest Independent Vector Problem (SIVP), the Bounded Distance Decoding Problem (BDD), etc. What is more, for most cryptographic schemes, we have no proof that a random instance of a problem is as hard to solve as the hardest one. Hence, if we derive a random problem instance, some may be easier to solve than others. However, for lattices, it was proven that the average case is just as hard to solve as the worst case. It means that for an adversary to succeed, it must be capable of solving the problem for all instances with non-negligible probability. This gives lattice-based cryptography strong security guarantees.

The rise of lattice-based cryptography started with the work of Ajtai, who showed

that it is possible to construct one-way functions based on the worst-case hardness of some lattice problems [8]. However, the early primitives were highly inefficient. For example, evaluating a collision-resistant hash function required $\mathcal{O}(n^2)$ time and space [8] (where the lattice dimension n is typically selected as 2^9 or 2^{10}), and in public-keys cryptosystems, the keys were the order of megabytes [9]. It was not until 2005, when the new hardness assumption - Learning With Errors - appeared [93], that lattice-based cryptography became efficient and became wide-spread research topic. It is believed that those problems are hard to solve even for a sizable quantum computer.

When trying to instantiate the solutions S_1 and S_2 in lattice-based settings, we need to use a lattice-based Σ protocol to prove that `ctxt` contains the option m and to prove knowledge of a pre-image x of $\mathsf{y} = \mathsf{OWF}(\mathsf{x})$. However, in the lattice-based setting, Σ-protocols are not trivial to construct: the security of the Learning With Errors (LWE) and the Short Integer Solution (SIS) problems require that the solution not only has a specific structure but also is *small*. It is a significant difference from the cyclic groups we have dealt with till now.

To understand how the "must be small" requirement affects the proof, consider a typical discrete logarithm-based identification scheme, which works fine in the standard settings (rings and cyclic groups) from Figure 5.1. Now we try to convert this scheme into a similar one in the lattice-based settings following the example used by Lyubashevsky in [79].

First of all, the goal of the protocol is to prove to the verifier $\mathcal{V}$ that the prover $\mathcal{P}$ knows the secret key $\mathsf{sk} \in \mathbb{Z}_\mathsf{q}$ that corresponds to the public key $\mathsf{pk} = \mathsf{g}^{\mathsf{sk}} \in \mathbb{G}$. The public parameters contain a public key pk, the cyclic group $\mathbb{G}$ defined over finite field $\mathcal{F}_\mathsf{q}$, the cyclic group generator g, and the cyclic group order q.

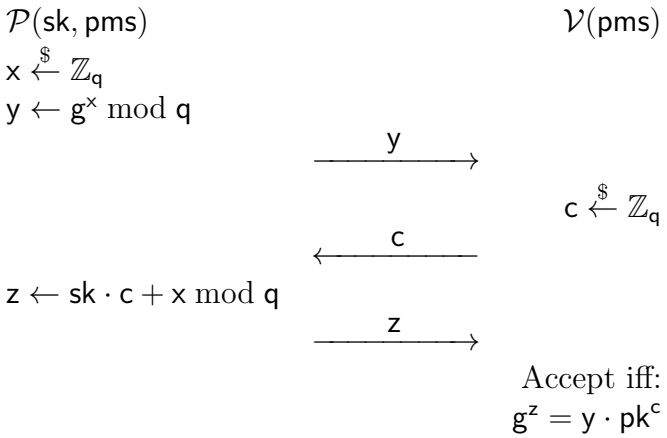


Figure 5.1: Standard discrete logarithm-based identification scheme.

As you can see, the prover `P` replies with a re-randomized discrete logarithm problem instance $\mathsf{z} \leftarrow \mathsf{sk} \cdot \mathsf{c} + \mathsf{x} \bmod \mathsf{q}$, which does not expose sk. If one would send only $\mathsf{sk} \cdot \mathsf{c}$, it would immediately reveal the secret; hence a masking term x is necessary. Another important observation is that we can safely pick x uniformly from $\mathbb{Z}_\mathsf{q}$ and expect that the result z, as well as operation $\mathsf{sk} \cdot \mathsf{c}$ to belong to the same group $\mathbb{Z}_\mathsf{q}$.

Suppose we have mapped the cyclic group $\mathbb{G}$, the finite field $\mathcal{F}_{\mathsf{q}}$, and all the group operations to the corresponding analogs from lattice-based cryptography based on the hardness of the SIS problem. Can we essentially run the same identification protocol? Unfortunately, no.

It turns out that in the lattice settings, we cannot hide the secret sk using the same approach. Recall that standard cryptography works with a field $\mathbb{Z}_{\mathsf{q}}$ closed under addition and multiplication. Yet, the lattice-based one deals with a subset of a ring (it includes only elements with a small Euclidian norm), which is not closed under addition or multiplication.

So, for a lattice-based case, one can end up with z that does not have a small norm: i.e., outside of the ring subset. However, we can prove the protocol security only if z is small because, only in this case, breaking the scheme would lead to solving an instance of the SIS problem, which is believed to be hard in lattices. Making z small requires both c and x being small. Therefore, x might not completely hide the secret, which unavoidably leaks parts of it after several protocol repetitions.

A solution to this problem was found by Lyubashevsky in [79]. He proposed the smart idea of a (possibly) aborting prover, using rejection sampling to ensure that the answer's distribution is independent of the secret. So, whenever the z might leak the secret sk, the prover does not return it.

The rejection sampling allowed to ensure correctness and security and led to many fundamental cryptographic constructions: canonical identification scheme and signatures (e.g. [79]), zero-knowledge proofs (e.g. [80]), blind signatures (e.g. **BLAZE**+ [10]), and others.

The main downside of the idea is the possibility of multiple protocol repetitions that may be very undesirable in practical applications. Real people expect to interact with a system only once and always receive the (correct) result at the end. Hence, an unpredictable number of repeats due to rejection sampling makes deniable protocols unpractical, at least for e-voting.

In our settings, the voting device is malicious and needs to prove to the voter that the ciphertext contains the intended voting option. However, in the lattice-based protocol, the voting device might intend to always abort the protocol in hopes that the voter would lose patience and leave without verifying the CAI property.

Unfortunately, eliminating aborts in lattice-based zero-knowledge proofs is quite challenging. Behnia *et al.* [16] studied rejection conditions in Lyubashevsky's canonical identification scheme and found a way to remove one of the two conditions. However, they concluded that full elimination of rejection sampling is problematic.

An alternative is to reduce the occurrence of protocol re-runs. One of the simplest methods to ensure the protocol terminates after a fixed number of repetitions $M \geq 2$ (with a high probability) is to use a large enough distribution over $\mathbb{Z}$. However, this comes at the cost of increased execution time and proof size.

Another method to decrease the occurrence of aborts is to run several protocols in parallel. Usually, parallel repetition aims to reduce the knowledge error rather than the completeness error. However, we can use it for decreasing rejects as well. Prover starts N independent instances of the protocol and sends N commitments to the verifier, replying only with the first proof that did not cause an abort. While this increases the probability of successful protocol termination, it significantly increases

communication and computational complexity (by a factor of N) and does not eliminate aborts completely.

Another work by Attema *et al.* [14] proposes a s-out-of-t threshold parallel repetition, where the verifier accepts if s out of t of the parallel instances are accepting. Unfortunately, the completeness error is still $\geq \rho^t$, where ρ is the completeness error of a single protocol run. Therefore, achieving a negligible probability of aborts would require a substantial t and would result in an increased proof size and proving time.

An improvement over parallel repetition — a generic construction for reducing aborts in 3-move protocols — was proposed in [10]. This construction builds on top of the idea of ℓ parallel repetitions and uses (unbalanced) binary hash trees to reduce the size of the first answer, from ℓ-commitments to a tree root.

An alternative to the aborts approach is probabilistically checkable proofs (PCPs) and interactive oracle proofs (IOPs) cleverly combined with lattice-based algebraic techniques. For example, [22] presents a zero-knowledge system for proving knowledge of Learning With Errors (LWE) pre-images, which does not involve aborts. Unfortunately, this solution is more efficient than a general lattice-based system (with aborts) only for some specific settings, for instance, when proving at the same time knowledge of a lot of LWE pre-images with the same matrix $\boldsymbol{A}$. In other cases, the initial commitment and Merkle paths result in bigger proofs than possible alternatives, especially considering that we need several iterations to achieve negligible soundness.

All in all, the currently most efficient and compact interactive zero-knowledge systems in the lattice-based setting are those with aborts, and (so far) there is no efficient way to eliminate repetitions therein.

5.2 Basics Of Lattice-Based Cryptography

We will use bold to denote vectors (column vectors by default) and bold capital letters for matrices. Also, we write $\mathbf{e} \xleftarrow{\$} \mathbb{Z}_\mathsf{q}^m$ to mean that the m components of a vector $\mathbf{e}$ are chosen uniformly at random from $\mathbb{Z}_\mathsf{q}$ and independently from each other.

As usual, we say that an infinity norm or ℓ_∞-norm of a vector is the largest magnitude among each element of the vector. For example, for a vector $\mathbf{e} = \{-6, 0, 1\}$, the infinity norm is 6. We will denote it as $||\mathbf{e}||_\infty = 6$.

For any positive integer β, we define the set $[\beta]$ as $\{-\beta, \ldots, -1, 0, 1, \ldots, \beta\}$. We write $\mathbf{e} \xleftarrow{\$} [\beta]^m$ or $||\mathbf{e}||_\infty \leq \beta$ to indicate that a vector $\mathbf{e}$ of m elements is chosen in such a way that each component is selected from $[\beta]$ uniformly at random.

5.2.1 Polynomial Rings

The cryptography we used before is based on cyclic groups over $\mathbb{Z}_\mathsf{q}$. Unfortunately, lattice-based primitives are inefficient over $\mathbb{Z}_\mathsf{q}$. For example, if we construct a public key encryption scheme based on the LWE problem over $\mathbb{Z}_\mathsf{q}$, we would need quite a large ciphertext of the form $(\mathbf{u}, v) \in \mathbb{Z}_\mathsf{q}^m \times \mathbb{Z}_\mathsf{q}$ for encrypting just one bit [78]. For

the concrete security parameters, say $\mathsf{q} \approx 2^{13}$ and $m \approx 700$, we will get that only $\mathbf{u}$ requires $m \log \mathsf{q} \approx 9100$ bits. We can slightly reduce the ciphertext size at the expense of the public key size, but ultimately we cannot make efficient primitives over the group $\mathbb{Z}_{\mathsf{q}}$. To build a better cryptographic primitive, we need to switch to a different group - higher-degree polynomial rings.

The polynomial ring $(\mathbb{Z}[X], +, \times)$ consists of the elements of the form $a(X) = \sum_{i=0}^{\infty} a_i X^i$ for $a_i \in \mathbb{Z}$ and indeterminate X with the usual polynomial addition and multiplication.

We denote the degree of the polynomial $a(X)$ as $\deg(a)$, which is equal to the largest i for which $a_i \neq 0$. We say that $a(X)$ is monic if the biggest coefficient $a_{\deg(a)} = 1$ and is irreducible if it cannot be written as $a(X) = b(X)c(X)$, where $b(X), c(X) \in \mathbb{Z}[X]$ are both polynomials of a smaller degree than $a(X)$.

For cryptography, we focus only on $(\mathcal{R}_f, +, \times)$, where $f \in \mathbb{Z}[X]$ is a monic polynomial of the degree n. Sometimes, we will refer to that ring as $\mathcal{R}_f = \mathbb{Z}[X]/\langle f(X) \rangle$, where $\mathcal{R}_f$ is a generalization of the q-ary integer lattices. Also, one can think of the usual ring $(\mathbb{Z}, +, \times)$ as a special instantiation of the ring $(\mathcal{R}_f, +, \times)$ where $f = X$ or $f = X - \alpha$ for $\alpha \in \mathbb{Z}$.

The elements of $\mathcal{R}_f$ are the polynomials $a = \sum_{i=0}^{n-1} a_i X^i$, where $a_i \in \mathbb{Z}$. The sum of two polynomials means simply summing the corresponding coefficients in $\mathbb{Z}$ as $a + b = \sum_{i=0}^{n-1} (a_i + b_i) X^i$. Multiplication by scalar k is equivalent to multiplying the vector by k: $ka = \sum_{i=0}^{n-1} k a_i X^i$. The multiplication of two polynomials in $\mathcal{R}_f$ means doing a normal polynomial multiplication followed by a reduction modulo f. Reduction modulo f means $a \bmod f = (bf + r) \bmod f = r$, where $b, r \in \mathcal{R}_f$ and $\deg(r) < n$. We can use a simple algorithm for computing a reduction modulo f - multiply f by an appropriate monomial αX^i, subtract it from a to create a polynomial of a smaller degree and continue until getting a polynomial with a degree less than n.

To better understand the polynomial multiplication modulo f, consider the following example $(2X^2-1)(X^2-X+2) \bmod (X^3-X+1)$. First, we multiply polynomials to obtain $2X^4-2X^3+3X^2+X-2 \bmod (X^3-X+1)$. Then we mulptiply $f = X^3-X+1$ by $2X$ to get $2X^4-2X^2+2X$ and subtract it from $2X^4-2X^3+3X^2+X-2$, which gives us polynomial of a smaller degree $-2X^3+5X^2-X-2 \bmod (X^3-X+1)$. However, $-2X^3 + 5X^2 - X - 2$ has the same degree as f, which means we should repeat the process. This time we multiply $f = X^3 - X + 1$ by -2, then subtract the result from $-2X^3 + 5X^2 - X - 2$ to get $5X^2 - 3X$, which finally is of degree smaller than f. Therefore, $(2X^2 - 1)(X^2 - X + 2) \bmod (X^3 - X + 1) = 5X^2 - 3X$.

One can simplify the process by observing that polynomial multiplication modulo f can be written as a multiplication of a matrix $\mathbb{Z}^{n \times n}$ with a vector in $\mathbb{Z}^n$:

$$ab \bmod f = a(\sum_{i=0}^{n-1} b_i X^i) \bmod f = \sum_{i=0}^{n-1} (aX^i \bmod f) b_i$$

Since each $aX^i \bmod f$ is a polynomial of degree less than n, it can be interpreted as a vector in $\mathbb{Z}^n$. Therefore, multiplication ab can be seen as a matrix-vector multiplication.

Therefore, we can re-write our example $(2X^2-1)(X^2-X+2) \bmod (X^3-X+1) = 5X^2 - 3X$ as:

$$\begin{bmatrix} -1 & -2 & 0 \\ 0 & 1 & -2 \\ 2 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -3 \\ 5 \end{bmatrix}$$

From now on, we will use the following notation when referring to a vector of polynomials:

$$\mathbf{a} = \begin{bmatrix} a_1 \\ \dots \\ a_m \end{bmatrix} \in \mathcal{R}_f^m$$

Similarly for a matrix $\mathbf{A}$ of polynomials we will write:

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & \dots & a_{1,m} \\ \dots & \dots & \dots \\ a_{n,1} & \dots & a_{n,m} \end{bmatrix} \in \mathcal{R}_f^{n \times m}$$

One can compute different norms of a ring element $f \in \mathcal{R}_f$, where $f = \sum_i f_i X^i$ as follows:

$$\ell_1 : ||f||_1 = \sum_i |f_i|$$

$$\ell_2 : ||f||_2 = (\sum_i |f_i|^2)^{1/2}$$

$$\ell_\infty : ||f||_\infty = \max_i |f_i|$$

5.2.2 The Ring Learning With Errors Problem

Most of the lattice-based cryptography is based on the Learning With Errors (LWE) problem [93] presented in Definition 9 for a polynomial ring $\mathcal{R}_{\mathsf{q},f}$, which is like the ring $\mathcal{R}_f$, but with coefficients in $\mathbb{Z}_\mathsf{q}$ instead of $\mathbb{Z}$. On the high level, the problem asks to distinguish between a randomly sampled vector $\mathbf{b}$ and the result of a vector $\mathbf{a}$ multiplication with a short secret s blinded by random noise vector $\mathbf{e}$.

In the original definition of the LWE problem, the error distribution was chosen as rounded Gaussian to show that the average case of LWE is at least as hard as some of the worst-case lattice problems. However, later it was shown to be unnecessary [78]. One can use a uniform or binomial distribution to generate the errors since it is much more efficient than the Gaussian distribution.

Definition 9 (The Ring Learning With Errors (RLWE) Problem)**.** In the Ring Learning with Errors problem RLWE_m with parameter $m \geq 1$, an adversary $\mathcal{A}$ tries to distinguish between the two following distributions:

1. $(\mathbf{a}, \mathbf{a}s + \mathbf{e})$, where $\mathbf{a} \xleftarrow{\$} \mathcal{R}_{\mathsf{q},f}^m$, $s \xleftarrow{\$} [\beta]$, and $\mathbf{e} \xleftarrow{\$} [\beta]^m$.
2. $(\mathbf{a}, \mathbf{b})$, where $\mathbf{a} \xleftarrow{\$} \mathcal{R}_{\mathsf{q},f}^m$ and $\mathbf{b} \xleftarrow{\$} \mathcal{R}_{\mathsf{q},f}^m$

We say that an algorithm $\mathcal{A}$ has advantage ϵ in solving RLWE_m problem if

$$\Big| \Pr[\mathsf{b} = 1 | \mathbf{a} \xleftarrow{\$} \mathcal{R}_{\mathsf{q},f}^m; s \xleftarrow{\$} [\beta]; \mathbf{e} \xleftarrow{\$} [\beta]^m; \mathsf{b} \leftarrow \mathcal{A}(\mathbf{a}, \mathbf{a}s + \mathbf{e})] - \Pr[\mathsf{b} = 1 | \mathbf{a} \xleftarrow{\$} \mathcal{R}_{\mathsf{q},f}^m; \mathbf{b} \xleftarrow{\$} \mathcal{R}_{\mathsf{q},f}^m; \mathsf{b} \leftarrow \mathcal{A}(\mathbf{a}, \mathbf{b})] \Big| \geq \epsilon$$

5.2.3 The Ring Short Integer Solution

One of the most fundamental computational tasks in lattices is finding a short non-zero vector in a random lattice, known as the Short Integer Solution (SIS) problem, which is stated in Definition 10 for a polynomial ring $\mathcal{R}_{\mathsf{q},f}$. Informally, the problem asks to find an integer linear dependence between a given set of vectors where the integer scalars are all small.

It appeared in 1996, long before the LWE problem, but was not actively used until it became possible to perform operations with lattices of high dimensions. Similarly to the LWE, it has a reduction showing that a random instance of the SIS problem is as hard as some of the worst-case lattice problems.

Definition 10 (The Ring Short Integer Solution (RSIS))**.** The Ring Short Integer Solution problem $\mathsf{RSIS}_{k,B}$ with parameters $k \geq 1$ and $B > 0$ is solved by finding a short, non-zero vector $\mathbf{s} \in \mathcal{R}_f^{k+1}$ such that $(\mathbf{1}, \mathbf{a}^\intercal) \cdot \mathbf{s} \equiv \mathbf{0}$ over $\mathcal{R}_{\mathsf{q},f}$. We say that an algorithm $\mathcal{A}$ has advantage ϵ in solving $\mathsf{RSIS}_{k,B}$ problem if

$$\Pr[||\mathbf{s}||_2 \leq B \wedge (1, \mathbf{a}^\intercal) \cdot \mathbf{s} \equiv 0 \wedge \mathbf{s} \neq \mathbf{0}^{k+1} | \mathbf{a} \xleftarrow{\$} \mathcal{R}_{\mathsf{q},f}^k; \mathbf{s} \leftarrow \mathcal{A}(\mathbf{a})] \geq \epsilon$$

where $1, 0 \in \mathcal{R}_f$ and $\mathbf{0}^{k+1} \in \mathcal{R}_f^{k+1}$.

5.2.4 Error Distribution And Rejection Sampling

Error Distribution: Recall that, for sampling an RLWE tuple, we need to define an error distribution $[\beta]$. The main challenge is to control the noise term size during addition, multiplication, and other operations with RLWE samples. For example, for correct decryption, the q must be large enough so that the accumulated error does not wrap around modulo q. On the other hand, the bigger q is, the larger are keys and ciphertexts.

For general cyclotomic rings $\mathcal{R}_{\mathsf{q},f} = \mathbb{Z}_\mathsf{q}[X]/(f(X))$ with $f(X)$ being the monic irreducible polynomial with degree $n = \phi(m)$, sampling error polynomials sometimes can be challenging as the cyclotomic polynomial $f(X)$ can have quite "irregular" structure. Therefore, in practice power-of-two cyclotomics, where $m = 2^k$ for $k \geq 1$ are preferable. Especially efficient arithmetics over the ring is achieved for $f(X) = X^n + 1$ because it allows us to use the classical n-dimensional Fast Fourier transform (FFT) [82].

It turns out that for the power-of-two cyclotomics, the sampling of polynomial coefficients directly is more efficient and secure. Since we also need to bound the norm of error polynomials, we will use a uniform or binomial distribution instead of a small one-dimensional discrete Gaussian as they give slightly better bounds.

For the implementation in Section 5.6 we use the same distribution $[\beta] = \{-1, 0, 1\}$ as in [23], where both ± 1 have probability $5/16$ and 0 has probability $6/16$. Hence, if $\mathbf{v} \xleftarrow{\$} [\beta]^2$, then for $0 < \sigma \leq 1$ we have:

$$\Pr\left[||\mathbf{v}||_2 < \sqrt{(1+\sigma)\frac{10}{16}n}\right] \geq 1 - \exp(-\frac{\sigma^2}{3}\frac{10}{16}d)$$

Rejection Sampling: Sometimes we need to compute $\mathbf{z} = \mathbf{y} + \mathsf{f}\mathbf{r}$ and do not reveal $\mathbf{r}$ when outputting $\mathbf{z}$. To remove any dependency of $\mathbf{z}$ and $\mathbf{r}$, we use the rejection sampling technique [79]. The rejection sampling technique can be realized in one of two ways: (1) the masking vector is sampled uniformly from a bounded region, or (2) the masking vector is sampled using a discrete Gaussian distribution. In the lattices of high dimensions, the Gaussian sampling is far superior and gives a more acceptable rejection probability.

Definition 11 (The discrete Gaussian distribution)**.** The discrete Gaussian distribution on $\mathcal{R}_f^k$ centered around $\mathbf{v} \in \mathcal{R}_f^k$ with standard deviation $\sigma > 0$ is given by

$$\mathcal{D}_{\mathbf{v},\sigma}^{kn}(\mathbf{z}) = \frac{e^{-||\mathbf{z}-\mathbf{v}||_2^2/2\sigma^2}}{\sum_{\mathbf{z}' \in \mathcal{R}_f^k} e^{-||\mathbf{z}'||_2^2/2\sigma^2}}$$

When $\mathbf{v} = \mathbf{0}$ we refer to the distribution as $\mathcal{D}_{\sigma}^{kn}$.

Lemma 5.2.1 (Bootle *et al.* [23], Lemma 2.3)**.** Let $\mathbf{z} \xleftarrow{\$} \mathcal{D}_{\sigma}^{kn}$, where $\mathcal{D}_{\sigma}^{kn}$ is the discrete Gaussian distribution on $\mathcal{R}_f^k$ centered around $\mathbf{0} \in \mathcal{R}_f^k$, then

$$\Pr\left[||\mathbf{z}||_2 \leq \sigma\sqrt{2kn}\right] \geq 1 - 2^{-\log(e/2)kn/4}$$

Algorithm 1 $\mathsf{RejSampl}(\mathbf{z}, \mathbf{v}, \sigma)$

$\mathbf{u} \xleftarrow{\$} [0, 1)$
if $\mathbf{u} < \frac{1}{12}\exp\left(\frac{-2\langle \mathbf{z}, \mathbf{v}\rangle + ||\mathbf{v}||_2}{2\sigma^2}\right)$ **then**
    return 0
else
    return 1
end if

Lemma 5.2.2 (Bootle *et al.* [23], Lemma 2.4)**.** Let $\rho : \mathcal{R}_f^k \to [0, 1]$ be a probabilistic distribution such that, for some $T > 0$, $\rho(\{\mathbf{v} \in \mathcal{R}_f^k \mid ||\mathbf{v}||_2 \leq T\}) \geq 1 - 2^{-101}$ and let $\sigma \geq 5T$. Sample $\mathbf{v} \xleftarrow{\$} \rho$ and $\mathbf{y} \xleftarrow{\$} \mathcal{D}_{\sigma}^{kn}$, set $\mathbf{z} = \mathbf{y} + \mathbf{v}$, and run $\mathsf{b} \leftarrow \mathsf{RejSampl}(\mathbf{z}, \mathbf{v}, \sigma)$. Then the probability that $\mathsf{b} = 0$ is at least $1/12 - 2^{-104}$ and the distribution of $(\mathbf{v}, \mathbf{z})$, conditioned on $\mathsf{b} = 0$, is within statistical distance of 2^{-100} of the product distribution $\rho \times \mathcal{D}_{\sigma}^{kn}$.

5.2.5 Lattice-Based Public Key Encryption Scheme

Let $\mathcal{R}_{\mathsf{q},f}$ be the ring of integer polynomials modulo both $f(\mathsf{x}) = (\mathsf{x}^n + 1)$ and q, where n is the power of 2 and q is a sufficiently large public prime modulus such that $\mathsf{q} = 1 \bmod 2n$. The additive homomorphic lattice-based encryption scheme proposed by Lyubashevsky *et al.* in [81] is defined by a triple of algorithms $(\mathbb{E}.\mathsf{KG}, \mathbb{E}.\mathsf{Enc}, \mathbb{E}.\mathsf{Dec})$ as follows:

$\mathbb{E}$.KG: On input of the security parameter κ, the algorithm samples uniformly random $a_E \xleftarrow{\$} \mathcal{R}_{\mathsf{q},f}$ and draws two *small* elements s, e from the error distribution χ. Then it sets $\mathsf{pk} = (a_E, b_E) = (a_E, a_E \cdot s + e) \in \mathcal{R}_{\mathsf{q},f} \times \mathcal{R}_{\mathsf{q},f}$ and the secret key as $\mathsf{sk} = s$.

$\mathbb{E}$.Enc: To encrypt an n-bit message $z \in \{0,1\}^n$ (identified as a polynomial of degree $n-1$ with coefficients 0 or 1), the algorithm samples three random small elements $r_E, e_{E,u}, e_{E,v} \in \mathcal{R}_{\mathsf{q},f}$ from the error distribution χ. Then it computes the encryption $(u, v) = (a_E \cdot r_E + e_{E,u} \bmod \mathsf{q}, b_E \cdot r_E + e_{E,v} + \lfloor \frac{\mathsf{q}}{2} \rceil z \bmod \mathsf{q}) \in \mathcal{R}_{\mathsf{q},f} \times \mathcal{R}_{\mathsf{q},f}$.

$\mathbb{E}$.Dec: To recover the message z, the algorithm computes the following:

$$\begin{aligned} v - u \cdot s &= r_E \cdot b_E + e_{E,v} - s(a_E \cdot r_E + e_{E,u}) + \lfloor \frac{\mathsf{q}}{2} \rceil z \bmod \mathsf{q} \\ &= r_E \cdot (a_E \cdot s + e) + e_{E,v} - s(a_E \cdot r_E + e_{E,u}) + \lfloor \frac{\mathsf{q}}{2} \rceil z \bmod \mathsf{q} \\ &= r_E \cdot e - s \cdot e_{E,u} + e_{E,v} + \lfloor \frac{\mathsf{q}}{2} \rceil z \bmod \mathsf{q} \\ &\approx \lfloor \frac{\mathsf{q}}{2} \rceil z \end{aligned}$$

A decryption error happens only if the coefficients of $(r_E \cdot e - s \cdot e_{E,u} + e_{E,v})$ have a magnitude greater than $\mathsf{q}/4$.

We can generalize the cryptosystem to encrypt messages $z \in \{0, 1, \ldots, k-1\}^n$. For that will need to map the n-symbol message z to a polynomial of $\mathcal{R}_{\mathsf{q},f}$ by scaling it with a factor of $\lfloor \frac{\mathsf{q}}{k} \rceil$, instead of $\lfloor \frac{\mathsf{q}}{2} \rceil$. In the decryption step, the message z can be recovered by rounding each coefficient of $v - u \cdot s$ back to $i\lfloor \frac{\mathsf{q}}{k} \rceil$ for $i = \{0, \ldots, k-1\}$ whichever is closest modulo q. The resulting decryption will be correct if the coefficients of $(r_E \cdot e - s \cdot e_{E,u} + e_{E,v}) \in \mathcal{R}_{\mathsf{q},f}$ have magnitude less than $q/2k$.

5.2.6 Lattice-Based Proof Of A Short Integer Vector Knowledge

The voting device $\mathcal{VD}$ needs to prove to the voter $\mathcal{V}$ that the ciphertext (u, v) is an encryption of the plaintext z. If we write the RLWE encryption in matrix form, we notice that proving knowledge of the plaintext is equivalent to proving knowledge of $r_E = (r_{E,1}, \ldots, r_{E,n})$, $e_{E,u} = (e_{E,u,1}, \ldots, e_{E,u,1})$ and $e_{E,v} = (e_{E,v,1}, \ldots, e_{E,v,n})$. However, that is equivalent to proving the knowledge of a small vector $\mathbf{s}$ with $\|\mathbf{s}\| \leq \beta$.

$$\begin{pmatrix} u_1 \\ \vdots \\ u_n \\ v_1 \\ \vdots \\ v_n \end{pmatrix} = \lfloor \frac{\mathsf{q}}{2} \rceil \begin{pmatrix} 0 \\ \vdots \\ 0 \\ z_1 \\ \vdots \\ z_n \end{pmatrix} + \begin{pmatrix} a_1 & -a_n & \cdot & -a_2 \\ \ldots & \vdots & \ddots & \vdots \\ a_n & a_{n-1} & \cdot & -a_1 \\ b_1 & -b_n & \cdot & -b_2 \\ \ldots & \vdots & \ddots & \vdots \\ b_n & b_{n-1} & \cdot & b_1 \end{pmatrix} \begin{pmatrix} r_{E,1} \\ \vdots \\ r_{E,n} \\ r_{E,1} \\ \vdots \\ r_{E,n} \end{pmatrix} + \begin{pmatrix} e_{E,u,1} \\ \vdots \\ e_{E,u,n} \\ e_{E,v,1} \\ \vdots \\ e_{E,v,n} \end{pmatrix}$$

We can re-write it in the form $\mathbf{u} = \tilde{\mathbf{A}}\mathbf{s}$ as follows:

$$\begin{pmatrix} u_1 \\ \vdots \\ u_n \\ v_1 - \lfloor \frac{\mathsf{q}}{2} \rceil z_1 \\ \vdots \\ v_n - \lfloor \frac{\mathsf{q}}{2} \rceil z_n \end{pmatrix} = \begin{pmatrix} \mathbf{A} & \mathbf{I}_n & \mathbf{0}_n \\ \mathbf{B} & \mathbf{0}_n & \mathbf{I}_n \end{pmatrix} \begin{pmatrix} r_{E,1} \\ \vdots \\ r_{E,n} \\ e_{E,u,1} \\ \vdots \\ ae_{E,u,n} \\ e_{E,v,1} \\ \vdots \\ e_{E,v,n} \end{pmatrix}$$

Hence, to show the plaintext correctness, we want to prove knowledge of a short integer vector $\mathbf{s}$ that is a solution to the linear equation $\mathbf{As} = \mathbf{u}$ over $\mathbb{Z}_\mathsf{q}$ with public metrix $\mathbf{A} \in \mathbb{Z}_\mathsf{q}^{n \times m}$ and vector $\mathbf{u} \in \mathbb{Z}_\mathsf{q}^m$, where $\mathbf{s} = (s_1, \ldots, s_n)$ has coefficients in $\{-S, \ldots, S\}$.

The main difficulty with proving $\mathbf{As} = \mathbf{u}$ is the "vector $\mathbf{s}$ is small" part. It implies a sort of range proof, which is harder and computationally more challenging to construct than simply proving a knowledge of an element in the group (or ring) satisfying some relation.

There are several ways to do such a proof. The most direct approach proves the exact knowledge of $\mathbf{s}$ via the proof of knowledge of nearby codewords, which is an adaptation of Stern's protocol [104]. Unfortunately, it has the soundness error 2/3; thus, it has to be repeated 219 times to achieve the soundness error 2^{-128}. For a typical modulus $\mathsf{q} \approx 2^{30}$, the resulting proof would have the size of several megabytes [104].

Another approach is to give a "relaxed" proof, for example: $\mathbf{As}' = \mathsf{c}\mathbf{u}$ for a bigger vector $||\mathbf{s}'|| > ||\mathbf{s}||$ and some extra factor c, which is also useful in some situation [79]. Those proofs are usually efficient and have a negligible soundness error after just one run. However, those proofs always prove the knowledge of a bigger vector than $\mathbf{s}$, which forces modulus q to be larger for security reasons.

Recently, a new efficient exact proof technique appeared. It relies on the connection between the coefficient and the Number Theoretic Transform (NTT) representation in the polynomial ring [23]. The idea is to provide the knowledge of $\mathbf{s}$ such that $\mathbf{As} = \mathbf{u}$ for $\mathbf{s}$ with coefficients from $\{0, 1, 2\}$ first, and then extend it to a bigger range.

First, we rewrite $\mathbf{s} = \mathbf{Gs}'$, where $\mathbf{s}'$ has coefficients in $\{0, 1, 2\}$. Then we prove the knowledge of $\mathbf{s}'$ such that $\mathbf{A}'\mathbf{s}' = \mathbf{u}$ for $\mathbf{A}' = \mathbf{AG}$.

The easiest way to rewrite $\mathbf{s}$ is to write its coefficients as

$$s_i = s'_{i,0} + 3s'_{i,1} + \cdots + 3^{r-1}s'_{i,r-1} - 3^r s'_{i,r} = \mathbf{g}^\intercal \mathbf{s}'_i$$

where $r = \lfloor \log_3 S + 1 \rfloor$ and $\mathbf{g}^\intercal = (1, 3, \ldots, 3^{r-1}, -3^r)^\intercal$. The resulting coefficients $s'_{i,j}$ will be in $\{0, 1, 2\}$ and the new matrix $\mathbf{A}'$ is constructed as $\mathbf{A}' = \mathbf{A}(\mathbf{I}_n \otimes \mathbf{g}^\intercal) \in \mathbb{Z}_\mathsf{q}^{m \times rn}$

Showing that coefficients of $\mathbf{s}'$ are in $\{0, 1, 2\}$ is equivalent to proving

$$\mathbf{s}' \circ (\mathbf{s}' - \mathbf{1}) \circ (\mathbf{s}' - \mathbf{1}) = \mathbf{0} \bmod \mathsf{q} \tag{5.1}$$

where $\circ$ denotes component-wise multiplication.

However if $\mathbf{s}' \in \mathcal{R}_{\mathsf{q},f}$ satisfies 5.1 is means $\hat{\mathbf{s}}' \circ (\hat{\mathbf{s}}' - \hat{\mathbf{1}}) \circ (\hat{\mathbf{s}}' - \hat{\mathbf{1}}) = \hat{\mathbf{0}} \bmod \mathsf{q}$ also holds, where $\hat{\mathbf{s}}'$ is NTT (or FTT) representation of $\mathbf{s}'$.

Now we can prove that we know $\mathbf{s} \in \mathcal{R}_{\mathsf{q},f}$ such that

$$\mathbf{s}' \circ (\mathbf{s}' - \mathbf{1}) \circ (\mathbf{s}' - \mathbf{1}) = \mathbf{0} \bmod \mathsf{q} \text{ and } \mathbf{A}\hat{\mathbf{s}} = \mathbf{u}$$

which is equivalent to demonstrating the knowledge of $\mathbf{s}$ such that $\mathbf{As} = \mathbf{u}$ for $\mathbf{s}$ with coefficients from $\{0, 1, 2\}$. The proof is presented in Figure 5.2.

To commit to messages, we need the public parameters $\mathbf{B} \in \mathcal{R}_{\mathsf{q},f}^{5\times 6}$, which is a matrix with five row vectors $\mathbf{b}_1^\intercal, \ldots, \mathbf{b}_5^\intercal$ defined as follows:

$$\mathbf{B} = \begin{bmatrix} \mathbf{b}_1^c \\ \mathbf{b}_2^\intercal \\ \mathbf{b}_3^\intercal \\ \mathbf{b}_4^\intercal \\ \mathbf{b}_5^\intercal \end{bmatrix} = \begin{bmatrix} 1 & \mathsf{b}_{1,2} & \mathsf{b}_{1,3} & \mathsf{b}_{1,4} & \mathsf{b}_{1,5} & \mathsf{b}_{1,6} \\ 0 & 1 & 0 & 0 & 0 & \mathsf{b}_{2,6} \\ 0 & 0 & 1 & 0 & 0 & \mathsf{b}_{3,6} \\ 0 & 0 & 0 & 1 & 0 & \mathsf{b}_{4,6} \\ 0 & 0 & 0 & 0 & 1 & \mathsf{b}_{5,6} \end{bmatrix}$$

where the polynomials $\mathsf{b}_{i,j} \in \mathcal{R}_{\mathsf{q},f}$ are chosen uniformly at random.

The protocol also uses two challenges $\mathsf{c} \in \mathbb{Z}_\mathsf{q}$ to show $\mathbf{A}\hat{\mathbf{s}} = \mathbf{u}$ and $\mathsf{f} \in \mathcal{C}$ to embed the commitment randomness into a masked value. We define $\mathcal{C} \subset \mathcal{R}_f$ as $\mathcal{C} = \{0, X^i | 0 \leq i < \ell\}$. The crucial property of this set is that the difference between any two members is invertible in $\mathcal{R}_f$, and the multiplication of any element in $\mathcal{R}_f$ by any member of the set does not increase the norm of the element.

For the implementation we use the same power-of-two cyclotomic ring $\mathcal{R}_{\mathsf{q},f} = \mathbb{Z}_\mathsf{q}[X]/(X^{2048} + 1)$ with $\mathsf{q} \approx 2^{32}$ as in the article [23].

Theorem 5.2.3 (Bootle *et al.* [23], Theorem 3.1)**.** The protocol in Figure 5.2 is complete, computational honest verifier zero-knowledge if RLWE_5 is hard and generated special sound if $\mathsf{RSIS}_{5,8B}$ is hard.

More precisely, the honest prover `P` convinces the honest verifier `V` with probability $\epsilon \approx 1/12$.

For zero-knowledge, there exists a simulator `Sim`, that, without access to secret information, outputs a simulation of a non-aborting transcript of the protocol between `P` and `V`. Then for every algorithm $\mathcal{A}$ that has advantage ϵ in distinguishing the simulated transcript from an actual transcript, there is an algorithm $\mathcal{A}'$ with the same running time that has advantage $\epsilon - 2^{-100}$ in distinguishing RLWE_5.

For knowledge-soundness, there is an extractor `K` with the following properties. When given rewindable black-box access to a deterministic prover $\mathtt{P}^*$ that convinces `V` with probability $\epsilon > 2/\mathsf{q} + 2/\ell$, `K` either outputs a solution $\mathbf{s}^* \in \{0, 1, 2\}^n$ to $\mathbf{As}^* = \mathbf{u}$, or a $\mathsf{RSIS}_{5,8B}$ solution for $\mathbf{b}_1^\intercal$ in expected time at most $144/(\epsilon - 2/\mathsf{q} - 2/\ell)$ when running $\mathtt{P}^*$ once is assumed to take unit time.

5.3 The Transformation

One may notice that aborts are only an issue when interactions with the voter are involved. Otherwise, surely it slows the proof generation down but does not pose

$\mathcal{P}(\mathbf{A}, \hat{\mathsf{s}}, \mathbf{u}, \mathbf{B})$ | | $\mathcal{V}(\mathbf{A}, \mathbf{u}, \mathbf{B})$

$\mathsf{y} \xleftarrow{\$} \mathcal{R}_{\mathsf{q},f}$

$\mathbf{r} \xleftarrow{\$} [\beta]^6$

$$\mathbf{t} = \begin{bmatrix} \mathbf{b}_1^\intercal \\ \mathbf{b}_2^\intercal \\ \mathbf{b}_3^\intercal \\ \mathbf{b}_4^\intercal \\ \mathbf{b}_5^\intercal \end{bmatrix} \cdot \mathbf{r} + \begin{bmatrix} 0 \\ \mathsf{y} \\ \mathsf{s} \\ \mathsf{y}(2\mathsf{s}-3) \\ \mathsf{y}^2(\mathsf{s}-3) \end{bmatrix}$$

$\mathbf{w} = \mathbf{A}\hat{\mathsf{y}}$ | $\xrightarrow{\mathbf{t}, \mathbf{w}}$ |

| | $\mathsf{c} \xleftarrow{\$} \mathbb{Z}_\mathsf{q}$

| $\xleftarrow{\mathsf{c}}$ |

$\mathsf{z} = \mathsf{y} + \mathsf{cs}$

$\mathbf{y}' \xleftarrow{\$} \mathcal{D}_\sigma^{6n}$

$\mathsf{w}' = \mathbf{b}_1^\intercal \mathbf{y}'$

$\mathsf{x}_1 = (\mathbf{b}_2^\intercal + \mathsf{c}\mathbf{b}_3^\intercal)\mathbf{y}'$

$\mathsf{x}_2 = \Big((\mathsf{z}-\mathsf{c})(\mathsf{z}-2\mathsf{c})\mathbf{b}_3^\intercal - \mathsf{z}\mathbf{b}_4^\intercal + \mathbf{b}_5^\intercal\Big)\mathbf{y}'$

| $\xrightarrow{\mathsf{z}, \mathsf{w}', \mathsf{x}_1, \mathsf{x}_2}$ |

| | $\mathsf{f} \xleftarrow{\$} \mathcal{C}$

| $\xleftarrow{\mathsf{f}}$ |

$\mathbf{z}' = \mathbf{y}' + \mathsf{f}\mathbf{r}$

If $\top \leftarrow \mathsf{RejSampl}(\mathbf{z}', \mathsf{f}\mathbf{r}, \sigma)$, abort. | $\xrightarrow{\mathbf{z}'}$ |

| | $||\mathbf{z}'|| \leq B = \sigma\sqrt{12n}$

| | $\mathbf{A}\hat{\mathsf{z}} \stackrel{?}{=} \mathbf{w} + \mathsf{c}\mathbf{u}$

| | $\mathbf{b}_1^\intercal \mathbf{z}' \stackrel{?}{=} \mathsf{w}' + \mathsf{f}\mathsf{t}_1$

| | $(\mathbf{b}_2^\intercal + \mathsf{c}\mathbf{b}_3^\intercal)\mathbf{z}' + \mathsf{f}\mathsf{z}$

| | $\stackrel{?}{=} \mathsf{x}_1 + \mathsf{f}(\mathsf{t}_2 + \mathsf{c}\mathsf{t}_3)$

| | $\Big((\mathsf{z}-\mathsf{c})(\mathsf{z}-2\mathsf{c})\mathbf{b}_3^\intercal - \mathsf{z}\mathbf{b}_4^\intercal + \mathbf{b}_5^\intercal\Big)\mathbf{z}'$

| | $\stackrel{?}{=} \mathsf{x}_2 + \mathsf{f}(\mathsf{z}-\mathsf{c})(\mathsf{z}-2\mathsf{c})\mathsf{t}_3 - \mathsf{z}\mathsf{t}_4 + \mathsf{t}_5)$

Figure 5.2: Proof of knowledge of a ternary solution to a linear equation over $\mathbb{Z}_\mathsf{q}$.

any problem. Therefore, what we want is to avoid protocol re-runs rather than aborts per see.

The most obvious solution is to use non-interactive proof, which can be repeated multiple times until it does not abort. However, as we mentioned earlier, the non-interactive proof is transferable and cannot be simulated by someone who does not control the random oracle.

The idea of our construction is to generate the proof non-interactively (via the Fiat-Shamir transformation) but then turn it back into an interactive one with the help of a simple trapdoor commitment. Note that aborts are not eliminated: they will still happen during the NIZK (Fiat-Shamir) generation. However, the interaction with the verifier (voter, in our use case) will not involve aborts or repetitions. Hence, from the voter's point of view, the protocol has a single run, as desired.

Let us define two hash functions $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\ell$ and $\mathsf{H}_1 : \{0,1\}^\ell \to \mathcal{C}$ and denote component-wise XOR operation between two strings of ℓ bits as $\oplus$. Then the new protocol goes as follows:

1. To prove a statement st, $\mathtt{P}$ samples a value $\mathsf{r} \in \{0,1\}^\ell$ at random and sends it to the verifier $\mathtt{V}$.

2. $\mathtt{V}$ sends to $\mathtt{P}$ a random challenge $\gamma \in \{0,1\}^\ell$.

3. $\mathtt{P}$ runs a non-interactive version of the proof of a short integer vector knowledge scheme (if necessary, re-running it until abort does not happen) to get a typical proof π; but the non-interactive challenge $\mathbf{e}$ is defined as $\mathbf{e} = \mathsf{H}_1\big(\mathsf{r} \oplus \mathsf{H}(\mathsf{st}, \mathsf{com}, \gamma)\big)$ instead of the usual $\mathsf{H}(\mathsf{st}, \mathsf{com})$ in the Fiat-Shamir transformation, where $\mathsf{st} = (\mathbf{A}, \mathbf{u}, \mathbf{B})$ and $\mathsf{com} = (\mathbf{t}, \mathbf{w}, \mathsf{c}, \mathbf{z}, \mathbf{w}', \mathbf{x}_1, \mathbf{x}_2)$ in the protocol from Figure 5.2.

Intuitively, we see that security is inherited from the non-interactive version of the protocol. On the one hand, $\mathtt{P}$ commits to r prior to receiving the verifier's challenge γ, thus it cannot manipulate the non-interactive challenge f. Therefore the resulting proof behaves as a standard non-interactive version of the initial (interactive) protocol. On the other hand, thanks to the use of the simple trapdoor commitment, anyone can generate a simulated transcript that is indistinguishable from the real one.

The generic transformation: Now we define our transformation for a generic case without limiting the number of rounds or protocol purposes.

Let $\Pi = \langle \mathtt{P}(\mathsf{x}, \omega), \mathtt{V}(\mathsf{x}) \rangle_\Pi$ be a public coin $(2\mu+1)$-rounds interactive proof system for language $\mathcal{L}_\mathcal{R}$. We denote as a_i the message sent by $\mathtt{P}$ to $\mathtt{V}$ in round $2i-1$, for $i = 1, \ldots, \mu$, and as z the last message sent by $\mathtt{P}$ in round $2\mu+1$. The message sent by $\mathtt{V}$ in round $2i$ is a random challenge $\mathsf{c}_i \in \mathcal{C}_i$, for some challenge space $\mathcal{C}_i$, for $i = 1, \ldots, \mu$.

Let us consider $1+\mu$ hash functions: on the one hand $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\ell$ and on the other hand $\mathsf{H}_i : \{0,1\}^\ell \to \mathcal{C}_i$, for $i = 1, \ldots, \mu$. One of the hash functions, H_1, is necessary for applying Fiat-Shamir transform: the resulting non-interactive challenge should belong to the challenge space $\mathcal{C}_i$. The other one, H, allows us to perform an XOR operation between r and the previous transcript.

We construct a 3-rounds interactive proof system $\Sigma = \langle \mathtt{P}(\mathsf{x}, \omega), \mathtt{V}(\mathsf{x}) \rangle_\Sigma$ for the same language $\mathcal{L}_\mathcal{R}$, as follows.

1. For $i = 1, \ldots, \mu$, $\mathtt{P}$ chooses $\mathsf{r}_i \in \{0,1\}^\ell$ uniformly at random. These values $\mathsf{r}_1, \ldots, \mathsf{r}_\mu$ are sent to $\mathtt{V}$.

2. $\mathtt{V}$ chooses a challenge $\gamma \in \{0,1\}^\ell$ uniformly at random and sends it to $\mathtt{P}$.

3. $\mathtt{P}$ runs an execution of the system Π by using inputs (x, ω), and playing also the role of the verifier, by defining the challenges as $\mathsf{c}_i = \mathsf{H}_i(\mathsf{r}_i \oplus \mathsf{h}_i)$, where $\mathsf{h}_i = \mathsf{H}(\mathsf{x}, \mathsf{a}_1, \ldots, \mathsf{a}_i, \mathsf{c}_1, \ldots, \mathsf{c}_{i-1}, \gamma)$, for $i = 1, \ldots, \mu$. The resulting transcript $(\mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_\mu, \mathsf{z})$ is sent by $\mathtt{P}$ to $\mathtt{V}$.

$\mathtt{V}$ accepts the interaction as valid if $(\mathsf{a}_1, \mathsf{c}_1, \mathsf{a}_2, \mathsf{c}_2, \ldots, \mathsf{a}_\mu, \mathsf{c}_\mu, \mathsf{z})$ is an accepting transcript for Π with input x, where $\mathsf{c}_i = \mathsf{H}_i(\mathsf{r}_i \oplus \mathsf{h}_i)$ and $\mathsf{h}_i = \mathsf{H}(\mathsf{x}, \mathsf{a}_1, \ldots, \mathsf{a}_i, \mathsf{c}_1, \ldots, \mathsf{c}_{i-1}, \gamma)$, for $i = 1, \ldots, \mu$.

5.4 Security Analysis Of Our Transformation

The completeness property of Σ is trivially satisfied, assuming the public coin $(2\mu + 1)$-rounds interactive proof system Π enjoys completeness. Now we prove that zero-knowledge and soundness properties of Π are also inherited from Σ.

5.4.1 Zero-Knowledge

Theorem 5.4.1. Assume the public coin $(2\mu+1)$-rounds interactive proof system Π enjoys the honest-verifier zero-knowledge (HVZK) property, then the new interactive system Σ also enjoys the HVZK property.

Proof. The goal is to show that, for any $(\mathsf{x}, \omega) \in \mathcal{L}_\mathcal{R}$, a simulator $\mathtt{Sim}_\Sigma$ can, on input x and any (honest) random challenge $\gamma \in \{0,1\}^\ell$, produce transcripts $(\mathsf{r}_1, \ldots, r_\mu, \gamma, \mathsf{a}_1, \ldots, \mathsf{a}_\mu, \mathsf{z})$ indistinguishable from those produced by an execution of $\langle \mathtt{P}(\mathsf{x}, \omega), \mathtt{V}(\mathsf{x}) \rangle_\Sigma$ with a honest verifier $\mathtt{V}$ which takes that γ uniformly at random in $\{0,1\}^\ell$.

By hypothesis, there is a simulator $\mathtt{Sim}_\Pi$ for Π. What the simulator $\mathtt{Sim}_\Sigma$ does first is to choose uniformly at random μ values $\mathsf{v}_1, \ldots, \mathsf{v}_\mu \xleftarrow{\$} \{0,1\}^\ell$ and to compute $\mathsf{c}_i = \mathsf{H}_i(\mathsf{v}_i)$ for $i = 1, \ldots, \mu$. Then $\mathtt{Sim}_\Sigma$ runs simulator $\mathtt{Sim}_\Pi$ with input x and challenges $\mathsf{c}_1, \ldots, \mathsf{c}_\mu$, which results in an accepting transcript $(\mathsf{a}_1, \mathsf{c}_1, \mathsf{a}_2, \mathsf{c}_2, \ldots, \mathsf{a}_\mu, \mathsf{c}_\mu, \mathsf{z})$, indistinguishable from those produced by $\langle \mathtt{P}(\mathsf{x}, \omega), \mathtt{V}(\mathsf{x}) \rangle_\Pi$. After that $\mathtt{Sim}_\Sigma$ computes the values $\mathsf{h}_i = \mathsf{H}_i(\mathsf{x}, \mathsf{a}_1, \ldots, \mathsf{a}_i, \mathsf{c}_1, \ldots, \mathsf{c}_{i-1}, \gamma)$ and $\mathsf{r}_i = \mathsf{v}_i \oplus \mathsf{h}_i$, for $i = 1, \ldots, \mu$.

It is easy to check that the transcript has the same distribution as those produced in a real execution of $\langle \mathtt{P}(\mathsf{x}, \omega), \mathtt{V}(\mathsf{x}) \rangle_\Sigma$ where γ is the challenge chosen by the honest verifier.

Assuming hash functions H_i are pseudo-random functions, the values $\mathsf{c}_i = \mathsf{H}_i(\mathsf{v}_i)$ generated by $\mathtt{Sim}_\Sigma$ and given as inputs to $\mathtt{Sim}_\Pi$ are random and uniform elements in $\mathcal{C}_i$. □

5.4.2 Soundness

Theorem 5.4.2. Assume the public coin $(2\mu+1)$-rounds interactive proof system Π has ϵ-soundness and if ℓ is big enough, then the new interactive system Σ has the ϵ'-soundness, in the (classical) Random Oracle Model, where $\epsilon' \leq \epsilon \cdot Q^{\mu}$ and Q is an upper bound on the number of hash queries that a prover of Σ can make.

Proof. The proof of this result works in a similar way as the well-known (in its naive, non-optimized version) proof that the Fiat-Shamir transformation of a public-coin interactive system with soundness results in a secure non-interactive system: the idea is to rewind the adversary several (in our case, μ times), by fixing the randomness and the answers to the hash queries up to a specific point, and then to use the Forking Lemma [91] to ensure that, with non-negligible probability, all the instances of the adversary will lead to forgeries with the desired outputs (that have been fixed in the rewinds).

First of all, if ℓ is big enough, then the probability $2^{-\ell}$ of breaking soundness by guessing the challenge $\gamma \in \{0,1\}^{\ell}$ is negligible. In that setting, let us assume that Σ still does not have ϵ'-soundness. Thus, there exists a prover $\mathtt{P}_{\Sigma}$ that is accepted with probability $> \epsilon'$, when run with some instance $\mathsf{x}' \notin \mathcal{L}_{\mathcal{R}}$. We are going to construct a prover $\mathtt{P}_{\Pi}$ against the soundness of Π, running thus with the same $\mathsf{x}' \notin \mathcal{L}_{\mathcal{R}}$.

As its first instruction, $\mathtt{P}_{\Pi}$ starts running $\mathtt{P}_{\Sigma}$, which sends its first message $(\mathsf{r}_1, \ldots, \mathsf{r}_{\mu})$. Now $\mathtt{P}_{\Pi}$ chooses at random $\gamma \in \{0,1\}^{\ell}$ and sends it to $\mathtt{P}_{\Sigma}$. We remark that $(\mathsf{r}_1, \ldots, \mathsf{r}_{\mu})$ and γ are going to be fixed for all the calls that $\mathtt{P}_{\Pi}$ makes to $\mathtt{P}_{\Sigma}$. In this first call, $\mathtt{P}_{\Sigma}$ gives its final answer $(\mathsf{a}_1^{(1)}, \ldots, \mathsf{a}_{\mu}^{(1)}, \mathsf{z}^{(1)})$, which is valid with probability $\geq \epsilon'$.

During this and the other executions of $\mathtt{P}_{\Sigma}$, our new prover $\mathtt{P}_{\Pi}$ has to answer the hash queries made by $\mathtt{P}_{\Sigma}$. This is done in the usual way, by keeping track of all previous queries, selecting a random output for new queries, storing the (input,output) relations in a table, etc. With overwhelming probability, a successful prover $\mathtt{P}_{\Sigma}$ will have made all the key queries $\mathsf{h}_i \leftarrow \mathsf{H}(\mathsf{x}', \mathsf{a}_1^{(1)}, \ldots, \mathsf{a}_i^{(1)}, \mathsf{c}_1^{(1)}, \ldots, \mathsf{c}_{i-1}^{(1)}, \gamma)$ and $\mathsf{H}_i(\mathsf{r}_i \oplus \mathsf{h}_i)$, for $i = 1, \ldots, \mu$.

After the first execution, $\mathtt{P}_{\Pi}$ sends the value $\mathsf{a}_1^{(1)}$ to its verifier $\mathtt{V}_{\Pi}$, which then sends a challenge c_1. With overwhelming probability, it will be the case that $\mathsf{c}_1 \neq \mathsf{H}_1(\mathsf{r}_1 \oplus \mathsf{h}_1)$. What $\mathtt{P}_{\Pi}$ does now is to rewind: it starts a new running of $\mathtt{P}_{\Sigma}$, with the same random tape and the same answers to the hash queries, up to the point where the query $\mathsf{H}_1(\mathsf{r}_1 \oplus \mathsf{h}_1)$ is made; this time, the answer to this query is defined as c_1. The Forking Lemma ensures that, with non-negligible probability, this second execution of $\mathtt{P}_{\Sigma}$ will produce a valid transcript $(\mathsf{a}_1^{(1)}, \mathsf{a}_2^{(2)}, \ldots, \mathsf{a}_{\mu}^{(2)}, \mathsf{z}^{(2)})$ with the same value $\mathsf{a}_1^{(1)}$ as in the first execution (because, with overwhelming probability, the value $\mathsf{a}_1^{(1)}$ had been queried to hash oracle H to produce h_1, before the key query $\mathsf{H}_1(\mathsf{r}_1 \oplus \mathsf{h}_1)$ was made). At this point, $\mathtt{P}_{\Pi}$ sends the value $\mathsf{a}_2^{(2)}$ to its verifier $\mathtt{V}_{\Pi}$, which then sends a challenge c_2.

The same rewind argument is done again, with the same random tape and hash answers as in the second execution, but now defining $\mathsf{H}_2(\mathsf{r}_2 \oplus \mathsf{h}_2)$ to be c_2. Again with overwhelming probability this query, which depends on h_2 which depends on c_1, must have been made after the query $\mathsf{H}_1(\mathsf{r}_1 \oplus \mathsf{h}_1)$, which is again answered as c_1. With non-negligible probability, this third execution of $\mathtt{P}_{\Sigma}$ produces a valid

transcript $(\mathsf{a}_1^{(1)}, \mathsf{a}_2^{(2)}, \mathsf{a}_3^{(3)}, \ldots, \mathsf{a}_\mu^{(3)}, \mathsf{z}^{(3)})$.

Repeating this argument μ times, letting $\mathtt{P}_\Pi$ send $\mathsf{a}_i^{(i)}$ to its verifier $\mathtt{V}_\Pi$ in round i, getting c_i as answer and rewinding $\mathtt{P}_\Sigma$ accordingly, at the end we eventually finish, after $\mu + 1$ executions of $\mathtt{P}_\Sigma$, with a valid transcript $(\mathsf{a}_1^{(1)}, \mathsf{a}_2^{(2)}, \mathsf{a}_3^{(3)}, \ldots, \mathsf{a}_\mu^{(\mu)}, \mathsf{z}^{(\mu+1)})$ satisfying $\mathsf{c}_i = \mathsf{H}_i(\mathsf{r}_i \oplus \mathsf{h}_i)$, where $\mathsf{h}_i = \mathsf{H}_i(\mathsf{x}', \mathsf{a}_1^{(1)}, \ldots, \mathsf{a}_i^{(i)}, \mathsf{c}_1, \ldots, \mathsf{c}_{i-1}, \gamma)$. Thus, our $\mathtt{P}_\Pi$ has convinced its verifier $\mathtt{V}_\Pi$ with non-negligible probability ϵ. By the iterated use of the Forking Lemma, the relation between ϵ and ϵ' is essentially $\epsilon \approx \frac{\epsilon'}{Q^\mu}$. □

5.5 Possible Extensions Of Our Transformation

We see several potential ways to improve our transformation in terms of security and usability. Without going into many details, we present these improvements below so one can get inspiration for future research.

Knowledge soundness: The same idea as in the proof for soundness can be applied to prove that knowledge soundness of Π implies knowledge soundness of Σ.

Full Zero-Knowledge: We can obtain full zero-knowledge in the plain model if we combine this 3-round and HVZK protocol Σ with a trapdoor and perfectly hiding commitment scheme (as described in Section 4.2) or interactive authentication (as described in Section 4.4). The resulting protocol would enjoy coercion-resistance and cast-as-intended verification and be safe against the quantum computer threat.

Switch to Quantum Random Oracle Model: The soundness property of Σ is obtained in the classical Random Oracle Model. Achieving soundness in the Quantum Random Oracle Model, where the adversary has quantum access to an oracle that computes a random function, is more complicated and is left as an open problem. The first thing one can try is to use quantum-oriented versions of Fiat-Shamir, either generic [109, 45] or specific for lattice-based systems [68], that have been proposed in the last years.

Improve the soundness loss factor: The naive reduction in our proof for the soundness property implies a loss factor Q^μ which is exponential in the number of rounds of Π. This problem can be solved by using the results in [15], whenever the starting protocol Π enjoys $(k_1, \ldots, k_\mu)$-special soundness. We stress that most (if not all) popular interactive systems Π enjoy this property, including lattice-based ones.

Remove an extra hash for closed challenge spaces: If the challenge spaces $\mathcal{C}_i$ of the interactive protocol Π are closed spaces for some mathematical operation (that we denote for simplicity as +), then a small modification to our construction is possible, basically choosing $r_i \leftarrow_R \mathcal{C}_i$ and then defining $c_i = r_i + h_i$, where $h_i = H_i(x, a_1, \ldots, a_i, h_1, \ldots, h_{i-1}, \gamma)$, being now $\mathsf{H}_i : \{0,1\}^* \to \mathcal{C}_i$. With this modification, the random oracle model assumption is not needed in the proof of the honest-verifier

zero-knowledge property. This situation happens for instance when Π is the protocol in [111]: the challenge space contains integers modulo a prime p.

5.6 Use Case And Implementation

Recall that the protocol described in Figure 5.3 can be used for proving knowledge of the small vectors $r_E = (r_{E,1}, \ldots, r_{E,n})$, $e_{E,u} = (e_{E,u,1}, \ldots, e_{E,u,1})$ and $e_{E,v} = (e_{E,v,1}, \ldots, e_{E,v,n})$ such that the ciphertext $(u, v) \in \mathcal{R}_{\mathsf{q},f} \times \mathcal{R}_{\mathsf{q},f}$ produced by RLWE public-key encryption scheme contains the plaintext z.

$$\begin{pmatrix} u_1 \\ \vdots \\ u_n \\ v_1 - \lfloor \frac{\mathsf{q}}{2} \rceil z_1 \\ \vdots \\ v_n - \lfloor \frac{\mathsf{q}}{2} \rceil z_n \end{pmatrix} = \begin{pmatrix} \mathbf{A} & \mathbf{I}_n & \mathbf{0}_n \\ \mathbf{B} & \mathbf{0}_n & \mathbf{I}_n \end{pmatrix} \begin{pmatrix} r_{E,1} \\ \vdots \\ r_{E,n} \\ e_{E,u,1} \\ \vdots \\ ae_{E,u,n} \\ e_{E,v,1} \\ \vdots \\ e_{E,v,n} \end{pmatrix}$$

This is equivalent to showing the knowledge of the randomness r s.t. $\mathtt{ctxt} = \mathbb{E}.\mathsf{Enc}(\mathsf{m}, \mathsf{r})$ we used in the ElGamal settings for S_1. Hence, it is straightforward to obtain the lattice-based construction for our solution S_1.

We did not implement the entire S_1 solution. Instead, we focused on demonstrating that our transformation is usable and its application does not slow down the proof generation process. We applied our transformation to the 5-round protocol of Bootle et al. [23] (see Figure 5.2 for the protocol details). The resulting interactive proof of a short integer vector knowledge scheme that always terminates after a single run is described in Figure 5.3. In case of aborting, fresh $\mathsf{y}, \mathbf{r}, \mathbf{y}'$ are sampled and the process is repeated until $\mathsf{RejSampl}(\mathbf{z}', \mathsf{f}\mathbf{r}, \sigma)$ accepts, ensuring $\mathbf{z}'$ does not leak $\mathbf{r}$.

The implementation was done using a custom-built library for polynomial operations over $\mathbb{Z}_\mathsf{q}[X]/\langle X^n + 1\rangle$, along with the RustCrypto library for computing SHA2 hashes in Rust.

The tests were performed on an Intel Core i7-10750H CPU. We have performed 1,000 tests over the protocol with and without the transformation. We have found that, when using the parameters proposed by Bootle et al. [23], the mean execution time increases from 20.6 to 21.5 seconds ($\sigma < 0.3$), amounting to an increase of about 5% in execution time. In Figure 5.4 we can see the time distribution of the protocol over the executions with (orange) and without (blue) the transformation.

$\mathcal{P}(\mathbf{A}, \hat{\mathsf{s}}, \mathbf{u}, \mathbf{B})$ | $\mathcal{V}(\mathbf{A}, \mathbf{u}, \mathbf{B})$

$\mathsf{r} \xleftarrow{\$} \{0,1\}^{\ell}$

$\xrightarrow{\quad \mathsf{r} \quad}$

$\boldsymbol{\gamma} \xleftarrow{\$} \{0,1\}^{\ell}$

$\xleftarrow{\quad \boldsymbol{\gamma} \quad}$

$\mathsf{y} \xleftarrow{\$} \mathcal{R}_{\mathsf{q},f}$

$\mathbf{r} \xleftarrow{\$} [\beta]^6$

$$\mathbf{t} = \begin{bmatrix} \mathbf{b}_1^\intercal \\ \mathbf{b}_2^\intercal \\ \mathbf{b}_3^\intercal \\ \mathbf{b}_4^\intercal \\ \mathbf{b}_5^\intercal \end{bmatrix} \cdot \mathbf{r} + \begin{bmatrix} 0 \\ \mathsf{y} \\ \mathsf{s} \\ \mathsf{y}(2\mathsf{s}-3) \\ \mathsf{y}^2(\mathsf{s}-3) \end{bmatrix}$$

$\mathbf{w} = \mathbf{A}\hat{\mathsf{y}}$

$\mathsf{c} = \mathsf{H}_{\mathsf{q}}(\mathbf{A}, \mathbf{u}, \mathbf{B}, \mathbf{t}, \mathbf{w}, \mathsf{aux})$

$\mathsf{z} = \mathsf{y} + \mathsf{cs}$

$\mathbf{y}' \xleftarrow{\$} \mathcal{D}_{\sigma}^{6n}$

$\mathsf{w}' = \mathbf{b}_1^\intercal \mathbf{y}'$

$\mathsf{x}_1 = (\mathbf{b}_2^\intercal + \mathsf{c}\mathbf{b}_3^\intercal)\mathbf{y}'$

$\mathsf{x}_2 = \Big((\mathsf{z}-\mathsf{c})(\mathsf{z}-2\mathsf{c})\mathbf{b}_3^\intercal - \mathsf{z}\mathbf{b}_4^\intercal + \mathbf{b}_5^\intercal\Big)\mathbf{y}'$

$\mathsf{com} = (\mathbf{t}, \mathbf{w}, \mathsf{c}, \mathsf{z}, \mathsf{w}', \mathsf{x}_1, \mathsf{x}_2)$

$\mathsf{f} = \mathsf{H}_1\big(\mathsf{r} \oplus \mathsf{H}\big(\mathbf{A}, \mathbf{u}, \mathbf{B}, \mathsf{com}, \mathsf{aux}, \gamma\big)\big)$

$\mathbf{z}' = \mathbf{y}' + \mathsf{f}\mathbf{r}$

If $\mathsf{RejSampl}(\mathbf{z}', \mathsf{f}\mathbf{r}, \sigma)$ does not accept, repeat.

$\xrightarrow{\quad \mathbf{t}, \mathsf{z}, \mathbf{z}' \quad}$

$\mathbf{w} = \mathbf{A}\hat{\mathsf{z}} - \mathsf{c}\mathbf{u}$

$\mathsf{w}' = \mathbf{b}_1^\intercal \mathbf{z}' - \mathsf{f}\mathsf{t}_1$

$\mathsf{x}_1 = (\mathbf{b}_2^\intercal + \mathsf{c}\mathbf{b}_3^\intercal)\mathbf{z}' + \mathsf{f}\mathsf{z} - \mathsf{f}(\mathsf{t}_2 + \mathsf{c}\mathsf{t}_3)$

$\mathsf{x}_2 = \Big((\mathsf{z}-\mathsf{c})(\mathsf{z}-2\mathsf{c})\mathbf{b}_3^\intercal - \mathsf{z}\mathbf{b}_4^\intercal + \mathbf{b}_5^\intercal\Big)\mathbf{z}' -$

$-\mathsf{f}(\mathsf{z}-\mathsf{c})(\mathsf{z}-2\mathsf{c})\mathsf{t}_3 - \mathsf{z}\mathsf{t}_4 + \mathsf{t}_5)$

$\mathsf{c} = \mathsf{H}_{\mathsf{q}}(\mathbf{A}, \mathbf{u}, \mathbf{B}, \mathbf{t}, \mathbf{w}, \mathsf{aux})$

$\mathsf{com} = (\mathbf{t}, \mathbf{w}, \mathsf{c}, \mathsf{z}, \mathsf{w}', \mathsf{x}_1, \mathsf{x}_2)$

Accept if:

$||\mathbf{z}'|| \leq B = \sigma\sqrt{12n}$

$\mathsf{f} \stackrel{?}{=} \mathsf{H}_1\big(\mathsf{r} \oplus \mathsf{H}\big(\mathbf{A}, \mathbf{u}, \mathbf{B}, \mathsf{com}, \mathsf{aux}, \gamma\big)\big)$

Figure 5.3: Application of our transformation to the protocol from Figure 5.2.

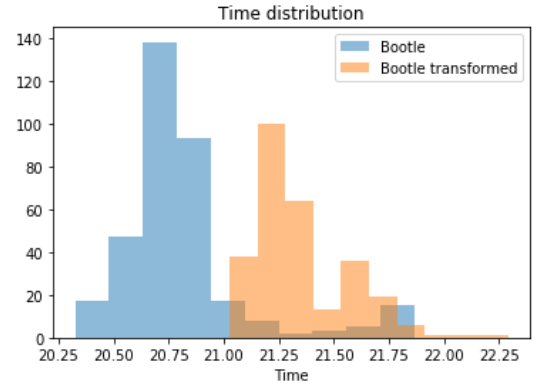


Figure 5.4: Time distribution of the 1,000 executions of the first test.

While we have obtained an expected decrease in the performance of a single run of the protocol, we have been able to avoid the need for re-runs and thus achieve an improvement over the whole testing.

For completeness, we performed a second series of tests to see how often aborts happen in a standard (non-transformed) 5-round protocol of Bootle et al. [23]. Figure 5.5 gives the distribution of the number of aborts produced in 10,000 tests of the non-transformed protocol with faster parameters ($n = 16$). As you can see, more than 6% of the executions required 10 or more repetitions of the protocol; this may be very undesirable in some real-life interactive protocols. In 1.5% of the executions, we had 20 repetitions with aborts.

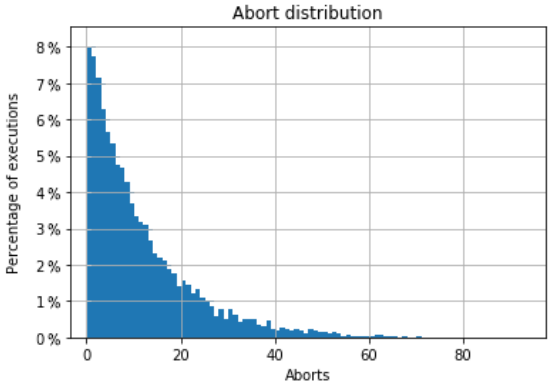


Figure 5.5: Percentage of executions suffering i aborts in the second test.

Chapter 6

Coercion-Resistant Cast-As-Intended Verifiability For A Computationally Limited Voter

In this chapter, we would like to focus on voters who are computationally limited and cannot do cryptographic operations, one-way function computations, and such without requiring the assistance of an aid device. The only capability voters have is remembering and comparing strings they see - an assumption similar in spirit to [87, 108]. For simplicity, we do not restrict the length of memorized data; however, we realize the amount of information the voter can safely remember is limited.

Therefore, we consider the following scenario: a realistic but potentially malicious voter is coerced to vote for a specific option but left without supervision for (some part) of the vote-casting process. We start by presenting an updated syntax in Section 6.1. Then, we show how to simplify our cast-as-intended and coercion-resistance definitions from Chapter 3 for computationally limited voters in Sections 6.2 and 6.3. After that, in Section 6.4, we present a simple solution for achieving both: cast-as-intended and coercion-resistance. Finally, in Section 6.4.1, we prove that our solution is secure.

6.1 Parties And Syntax Of The Voting Protocol

As before, in Section 3.3, we focus on a specific voter $\mathcal{V}$ who wishes to cast a vote for the intent m through a voting device $\mathcal{VD}$, possibly under the coercion of a coercer $\mathcal{C}$ that prefers another option $\mathsf{m}_{\mathcal{C}} \neq \mathsf{m}$. Similarly, we use pms to denote the public election parameters, which can be generated by running an initial protocol $\mathsf{pms} \leftarrow \mathsf{Setup}(\lambda)$.

In the restricted setting we consider in this paper, we assume that the voter can only generate, memorize, and compare strings of numbers. Let Cpb denote the class of operations the voter $\mathcal{V}$ is supposed to be capable of doing.

Since people are bad at generating random values, it makes perfect sense to consider the possibility that $\mathcal{V}$ gets the help of another entity or device, which we will refer to as an official election device (OED for short). It may participate in the protocol but can only generate strings and nothing else. The OED is trusted to run

the prescribed steps of the protocol correctly, and its actions are free from coercion.

Previously, we defined an execution of the voting protocol as:

$$(\mathsf{b}_{\mathcal{V}}, \mathsf{b}_{\mathcal{VD}}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}}) \leftarrow \mathsf{Vote}^{\langle \mathcal{V}, \mathcal{VD} \rangle}(\mathsf{pms}, \mathsf{m}, \mathsf{coerc})$$

However, now that we know the voter is computationally limited, we can simplify the protocol.

Even though the voter $\mathcal{V}$ can only generate random strings but not do any cryptography, the voting device $\mathcal{VD}$ still might do some checks of the voter's input. Therefore we keep the bit $\mathsf{b}_{\mathcal{VD}}$.

Second, a computationally limited voter cannot validate the complete proof as it requires, at the very least, to check that ciphertext belongs to the correct mathematical group, which is a mathematical operation humans cannot do in their minds. It is also true that we cannot delegate verification of the voting option's correctness to anyone without breaking the voter's privacy. Therefore, we split the bit $\mathsf{b}_{\mathcal{V}}$ into two bits $\mathsf{b}_{\mathcal{V},\mathsf{priv}}, \mathsf{b}_{\mathcal{V},\mathsf{pub}}$, where the voter has to only check $\mathsf{b}_{\mathcal{V},\mathsf{priv}}$ and $\mathsf{b}_{\mathcal{V},\mathsf{pub}}$ is determined by a public verification later.

Finally, we know that the voter $\mathcal{V}$ can interact with the secure device OED.

Thus, the updated execution of the voting protocol looks as follows:

$$(\mathsf{b}_{\mathcal{VD}}, \mathsf{b}_{\mathcal{V},\mathsf{priv}}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}}) \leftarrow \mathsf{Vote}_{\mathsf{Lim}}^{\langle \mathcal{V}^{\mathsf{OED}}, \mathcal{VD} \rangle}(\mathsf{pms}, \mathsf{m}, \mathsf{coerc})$$

Note that all instructions the coercer gives should be within the voter's capabilities; in other words, the actions of coerc are restricted to the class Cpb.

If both $\mathsf{b}_{\mathcal{V},\mathsf{priv}} = \mathsf{b}_{\mathcal{V},\mathsf{pub}} = 1$, the ciphertext ($\mathtt{ctxt} \in \mathsf{Trc}$) generated by $\mathcal{VD}$ is deemed valid. If $\mathsf{b}_{\mathcal{V},\mathsf{priv}} = 0$, the voter would complain to election authorities $\mathcal{EA}$ against the voting device, which leads to the vote invalidation. Similarly, if $\mathsf{b}_{\mathcal{V},\mathsf{pub}} = 0$, then the ballot box would not accept the vote as valid.

In our setting, the bit $\mathsf{b}_{\mathcal{V},\mathsf{priv}}$ (and similarly $\mathsf{b}_{\mathcal{V},\mathsf{pub}}$) is set to 1 if the corresponding verification function $\mathsf{ValidOption}$ (and similarly $\mathsf{ValidProof}$) returns $\top$.

The first (public) verification is done by anyone (with enough computational capabilities) by running a protocol

$$\{\bot, \top\} \leftarrow \mathsf{ValidProof}(\mathsf{pms}, \mathsf{Trc})$$

The second (private) verification is done by $\mathcal{V}$ by running the protocol below, whose operations must belong to class Cpb:

$$\{\bot, \top\} \leftarrow \mathsf{ValidOption}(\mathsf{pms}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}})$$

6.2 Formal Definition Of CAI For A Computationally Limited Voter

Recall that in Section 3.4.1 we formally defined cheating event Cheat for cast-as-intended verification as follows:

$$\mathsf{Cheat} = \begin{cases} \mathsf{pms} \leftarrow \mathsf{Setup}(\lambda) \\ (\mathsf{b}_{\mathcal{V}}, \mathsf{b}_{\mathcal{VD}}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}}) \leftarrow \mathsf{Vote}^{\langle \mathcal{V}^{\mathsf{OED}}, \mathcal{VD} \rangle}(\mathsf{pms}, \mathsf{m}, \emptyset) \\ \mathtt{ctxt} \in \mathsf{Trc} \\ \mathsf{b}_{\mathcal{V}} = 1 \text{ (voter accepts the proof)} \\ \mathsf{m} \neq \mathsf{Dec}(\mathtt{ctxt}, \mathsf{sk}) \text{ (but } \mathtt{ctxt} \text{ does not contain the intent)} \end{cases}$$

Now that we know the voter is computationally limited, we can define event $\mathsf{Cheat}_{\mathsf{Lim}}$ as follows:

$$\mathsf{Cheat}_{\mathsf{Lim}} = \begin{cases} \mathsf{pms} \leftarrow \mathsf{Setup}(\lambda) \\ (\mathsf{b}_{\mathcal{VD}}, \mathsf{b}_{\mathcal{V},\mathsf{priv}}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}}) \leftarrow \mathsf{Vote}_{\mathsf{Lim}}^{\langle \mathcal{V}^{\mathsf{OED}}, \mathcal{VD} \rangle}(\mathsf{pms}, \mathsf{m}, \emptyset) \\ \mathtt{ctxt} \in \mathsf{Trc} \\ \mathsf{b}_{\mathcal{V},\mathsf{pub}} = 1 \text{ i.e. } \top \leftarrow \mathsf{ValidProof}(\mathsf{pms}, \mathsf{Trc}) \\ \mathsf{b}_{\mathcal{V},\mathsf{priv}} = 1 \text{ i.e. } \top \leftarrow \mathsf{ValidOption}(\mathsf{pms}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}}) \\ \mathsf{m} \neq \mathsf{Dec}(\mathtt{ctxt}, \mathsf{sk}) // \text{ but } \mathtt{ctxt} \text{ does not contain the intent} \end{cases}$$

Definition 12 (Cast-as-intended for a computationally limited voter)**.** The protocol $\mathsf{Vote}_{\mathsf{Lim}}$ enjoys Cast-as-Intended (CAI) verifiability if the probability of event $\mathsf{Cheat}_{\mathsf{Lim}}$ is a negligible function of the security parameter λ, for any polynomial-time voting device $\mathcal{VD}$ assuming an official election device OED is reliable.

6.3 Formal Definition Of CR For A Computationally Limited Voter

Recall that in Section 3.4.1 we defined the experiment $\mathsf{Exp}^{\mathsf{CR}}_{\mathcal{C},\mathcal{V}}(\mathsf{pms}, \mathsf{m}_{\mathcal{C}}, \mathsf{m}, \mathsf{coerc})$ in Figure 3.1. Now that we know the voter is computationally limited, we can define experiment $\mathsf{Exp}^{\mathsf{CR}}_{\mathcal{C},\mathcal{V}^{\mathsf{OED}}}(\mathsf{pms}, \mathsf{m}_{\mathcal{C}}, \mathsf{m}, \mathsf{coerc})$ as defined in Figure 6.1.

Note that since $\mathsf{coerc} \in \mathsf{Cpb}$, we do not require to run $\mathtt{Sim}_1$ to account for the coercion when voting for the voter's option m. Therefore, it suffices to have only one simulator, which we denote as $\mathtt{Sim}$.

$\underline{\mathsf{Exp}^{\mathsf{CR}}_{\mathcal{C},\mathcal{V}^{\mathsf{OED}}}(\mathsf{pms}, \mathsf{m}_{\mathcal{C}}, \mathsf{m}, \mathsf{coerc})}$:

1. $\mathsf{pms} \leftarrow \mathsf{Setup}(\lambda)$
2. $\beta \xleftarrow{\$} \{0,1\}$

3. If $\beta = 0$:

   $(\mathsf{b}^{(0)}_{\mathcal{VD}}, \mathsf{b}^{(0)}_{\mathcal{V},\mathsf{priv}}, \mathsf{Trc}^{(0)}, \mathsf{rand}^{(0)}_{\mathcal{V}}) \leftarrow \mathsf{Vote}^{\langle \mathcal{V}^{\mathsf{OED}}, \mathcal{VD} \rangle}_{\mathsf{Lim}}(\mathsf{pms}, \mathsf{m}_{\mathcal{C}}, \mathsf{coerc})$
   If $\mathsf{b}^{(0)}_{\mathcal{VD}} = 0$, abort the experiment.
   Define $\mathsf{view}^{(0)} = (\mathsf{b}^{(0)}_{\mathcal{VD}}, \mathsf{b}^{(0)}_{\mathcal{V},\mathsf{priv}}, \mathsf{Trc}^{(0)}, \mathsf{rand}^{(0)}_{\mathcal{V}})$

4. Else if $\beta = 1$:

   $(\mathsf{b}_{\mathcal{VD}}, \mathsf{b}_{\mathcal{V},\mathsf{priv}}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}}) \leftarrow \mathsf{Vote}^{\langle \mathcal{V}^{\mathsf{OED}}, \mathcal{VD} \rangle}_{\mathsf{Lim}}(\mathsf{pms}, \mathsf{m}, \emptyset)$, with $\mathtt{ctxt} \in \mathsf{Trc}$
   If $\mathsf{b}_{\mathcal{VD}} = 0$, abort the experiment.
   $(\mathsf{b}^{(1)}_{\mathcal{VD}}, \mathsf{b}^{(1)}_{\mathcal{V},\mathsf{priv}}, \mathsf{Trc}^{(1)}, \mathsf{rand}^{(1)}_{\mathcal{V}}) \leftarrow \mathtt{Sim}(\mathsf{b}_{\mathcal{V},\mathsf{priv}}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}}, \mathsf{m}, \mathsf{m}_{\mathcal{C}}, \mathsf{coerc})$, with $\mathtt{ctxt} \in \mathsf{Trc}^{(1)}$
   Define $\mathsf{view}^{(1)} = (\mathsf{b}^{(1)}_{\mathcal{VD}}, \mathsf{b}^{(1)}_{\mathcal{V},\mathsf{priv}}, \mathsf{Trc}^{(1)}, \mathsf{rand}^{(1)}_{\mathcal{V}})$

5. $\mathsf{b} \leftarrow \mathcal{C}(\mathsf{view}^{(\beta)})$
6. If $\mathsf{b} = \beta$, return 1. Else, return 0.

Figure 6.1: Experiment for coercion-resistance for a computationally limited voter.

Definition 13 (Coercion-resistance for a computationally limited voter)**.** The protocol $\mathsf{Vote}_{\mathsf{Lim}}$ enjoys coercion-resistance (CR) if for any polynomial-time coercer $\mathcal{C}$, any coercion $(\mathsf{m}_{\mathcal{C}}, \mathsf{coerc})$ such that he actions of coerc are restricted to the class Cpb and any voting option m, there exists a polynomial-time simulator algorithm $\mathtt{Sim} \in \mathsf{Cpb}$ such that $\left|\Pr[1 \leftarrow \mathsf{Exp}^{\mathsf{CR}}_{\mathcal{C},\mathcal{V}^{\mathsf{OED}}}(\mathsf{pms}, \mathsf{m}_{\mathcal{C}}, \mathsf{m}, \mathsf{coerc})] - \frac{1}{2}\right| = \mathsf{negl}(\lambda)$ is a negligible function of the security parameter λ.

6.4 CR-CAI Solution For A Computationally Limited Voter

Consider a simple case when the voter $\mathcal{V}$ chooses one of the ℓ possible voting options $\{\mathsf{m}_1, \mathsf{m}_2, \ldots, \mathsf{m}_\ell\}$. The idea of the solution S_{Lim}, presented in Figure 6.2, is to use OR-proofs to ensure that the ballot contains the intended selection.

In our protocol, we need a public bulletin board $\mathcal{BB}$- an append-only, immutable database that offers everyone a consistent view of the received information. In other words, a new record can only be added at the end but never deleted, plus it is impossible to hide the published information from the viewers. It is a well-known cryptographic construct that simplifies the protocol construction [61].

We will use $\mathcal{X} : (\mathsf{y}) \mapsto \mathcal{BB}$ notation to say that an entity $\mathcal{X}$ published data y on the public bulletin board $\mathcal{BB}$. Similarly, if data y was taken from the public bulletin board $\mathcal{BB}$, we will use $(\mathsf{y}) \hookleftarrow \mathcal{BB}$ notation. To show that an official election device OED generated and published a nonce $\mathsf{nc}_{\mathcal{V}}$ on the public bulletin board $\mathcal{BB}$, we will use $\mathsf{OED} : \mathsf{nc}_{\mathcal{V}} \mapsto \mathcal{BB}$ notation.

Also, we require two collision-resistant hash functions $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_q$ and $\hat{\mathsf{H}} : \{0,1\}^* \to \mathcal{X}$, where $\mathcal{X}$ is the space of strings the voter can type and memorize yet it has enough entropy to avoid brute force attacks.

The solution S_{Lim} is given for ElGamal public encryption scheme. Recall it is instantiated over a cyclic group $\mathbb{G}_{\mathsf{q}}$ of order q with generator g, where the decisional Diffie-Hellman (DDH) problem is believed to be hard (see Section 3). The default spaces for the messages, ciphertext, and randomness are $\mathbb{G}$, $\mathbb{G}$, and $\mathbb{Z}_{\mathsf{q}}$ respectively: i.e., $\mathcal{M}_{\mathbb{E}} = \mathbb{G}, \mathcal{C}_{\mathbb{E}} = \mathbb{G}, \mathcal{RS}_{\mathbb{E}} = \mathbb{Z}_{\mathsf{q}}$.

Thus, the public paramaters are $\mathsf{pms} = (\mathbb{G}, \mathsf{pk}, \mathsf{g}, \mathbb{Z}_{\mathsf{q}})$.

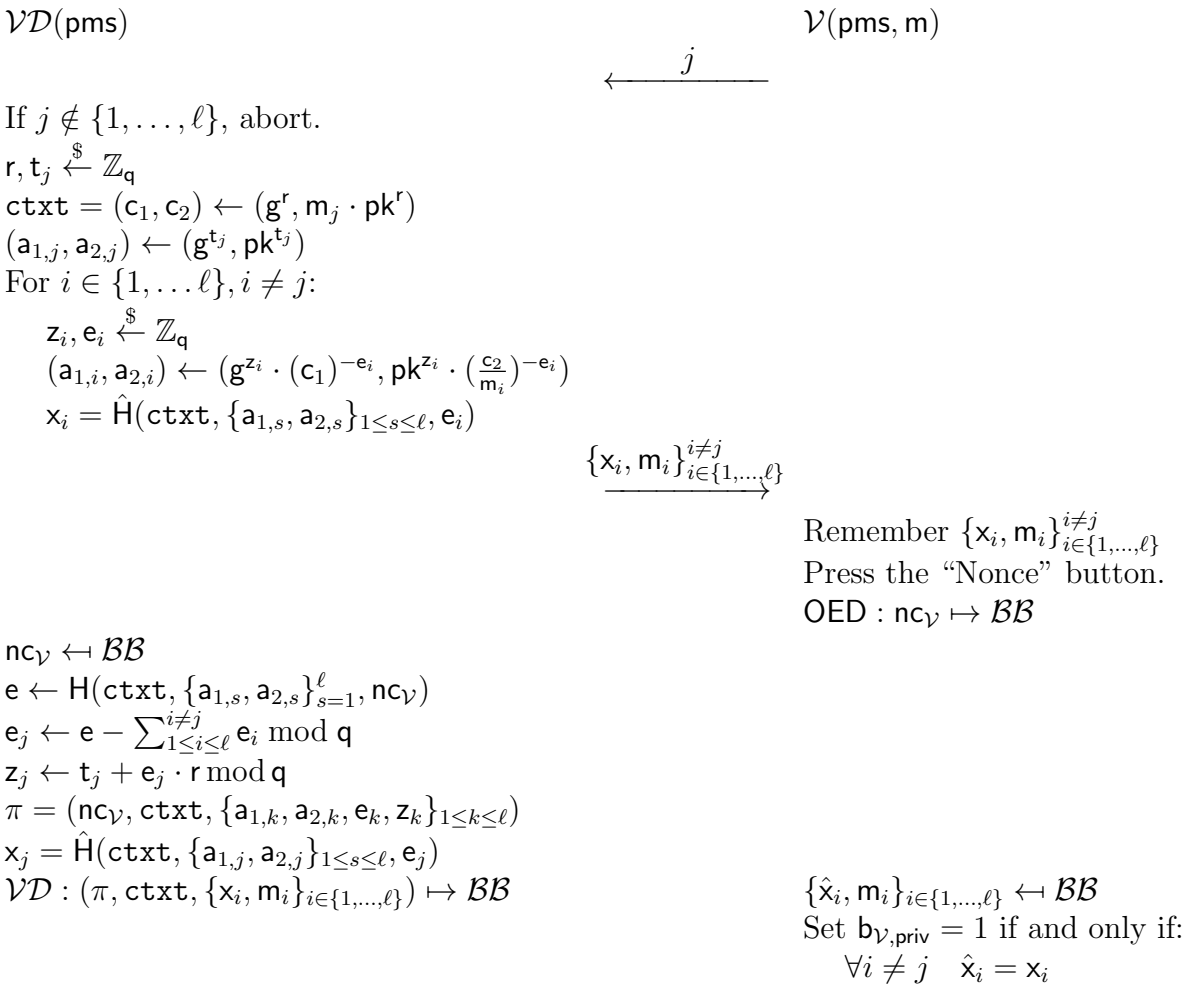


Figure 6.2: Solution S_{Lim}.

Following the notation of Section 6.1, we have: $\mathsf{Trc} = (\pi, \mathtt{ctxt}, \{\hat{\mathsf{x}}_i, \mathsf{m}_i\}_{i \in \{1,\ldots,\ell\}})$ and $\mathsf{rand}_{\mathcal{V}} = (j, \{\mathsf{x}_i, \mathsf{m}_i\}_{i \in \{1,\ldots,\ell\}}^{i \neq j})$.

Voter can verify the vote contains the intended selection j by running $\{\bot, \top\} \leftarrow \mathsf{ValidOption}(\mathsf{pms}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}})$, which works as shown in Figure 6.3.

For the public verification, anybody with enough computational resources can run $\{\bot, \top\} \leftarrow \mathsf{ValidProof}(\mathsf{pms}, \mathsf{Trc})$ as described in Figure 6.4 to ensure that the ballot is valid.

$\underline{\mathsf{ValidOption}(\mathsf{pms}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}})}$:

1. $(\pi, \mathtt{ctxt}, \{\hat{\mathsf{x}}_i, \mathsf{m}_i\}_{i\in\{1,\ldots,\ell\}}) \leftarrow \mathsf{Trc}$
2. $(j, \{\mathsf{x}_i, \mathsf{m}_i\}^{i\neq j}_{i\in\{1,\ldots,\ell\}}) \leftarrow \mathsf{rand}_{\mathcal{V}}$
3. For $i \in \{1, \ldots, \ell\}$ s.t. $i \neq j$:
   If $\hat{\mathsf{x}}_i \neq \mathsf{x}_i$, abort and return $\perp$.
4. If all verifications were successful, return $\top$

Figure 6.3: ValidOption verification.

$\underline{\mathsf{ValidProof}(\mathsf{pms}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}})}$:

1. $(\pi, \mathtt{ctxt}, \{\hat{\mathsf{x}}_i, \mathsf{m}_i\}_{i\in\{1,\ldots,\ell\}}) \leftarrow \mathsf{Trc}$
2. $\mathsf{e} \leftarrow \mathsf{H}(\mathtt{ctxt}, \{\mathsf{a}_{1,s}, \mathsf{a}_{2,s}\}^{\ell}_{s=1}, \mathsf{nc}_{\mathcal{V}})$
3. $\mathsf{e}' \leftarrow \sum_{i=1}^{\ell} \mathsf{e}_i \bmod \mathsf{q}$
4. If $\mathsf{e}' \neq \mathsf{e}$, abort and return $\perp$.
5. For $k \in \{1, \ldots, \ell\}$:
   If $\mathsf{g}^{\mathsf{z}_k} \neq \mathsf{a}_{1,k} \cdot (\mathsf{c}_1)^{\mathsf{e}_k}$, abort and return $\perp$.
   If $\mathsf{pk}^{\mathsf{z}_k} \neq \mathsf{a}_{2,k} \cdot \left(\frac{\mathsf{c}_2}{\mathsf{m}_k}\right)^{\mathsf{e}_k}$, abort and return $\perp$.
   If $\hat{\mathsf{x}}_k \neq \hat{\mathsf{H}}(\mathtt{ctxt}, \{(\mathsf{a}_{1,s}, \mathsf{a}_{2,s})\}_{1\leq s\leq \ell}, \mathsf{e}_k)$, abort and return $\perp$.
6. If all verifications were successful, return $\top$

Figure 6.4: ValidProof verification.

6.4.1 Security Analysis Of The Protocol

Consider the following OR-proof relation that we wish to prove:

$$\mathcal{R}_{\mathrm{OR}} = \{(\mathtt{ctxt}, \{\mathsf{m}_1, \ldots, \mathsf{m}_\ell\}) \mid \mathtt{ctxt} \text{ encrypts } \mathsf{m}_1 \text{ OR} \ldots \text{OR } \mathtt{ctxt} \text{ encrypts } \mathsf{m}_\ell\}$$

Theorem 6.4.1 (Solution S_{Lim} achieves CAI)**.** Assuming the special soundness of the Σ-protocol for $\mathcal{R}_{\mathrm{OR}}$, the protocol described in Figure 6.2 achieves the Cast-as-Intended (CAI) property for a computationally limited voter (Definition 12).

Proof. Suppose a (dishonest) voting device $\mathcal{VD}$ can break cast-as-intended verifiability for a computationally limited voter. In other words, it makes the event $\mathsf{Cheat}_{\mathsf{Lim}}$ happen with a non-negligible probability, where $\mathsf{Cheat}_{\mathsf{Lim}}$ is defined as follows:

$$\mathsf{Cheat}_{\mathsf{Lim}} = \begin{cases} \mathsf{pms} \leftarrow \mathsf{Setup}(\lambda) \\ (\mathsf{b}_{\mathcal{V},\mathsf{priv}}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}}) \leftarrow \mathsf{Vote}^{\langle \mathcal{V}^{\mathsf{OED}}, \mathcal{VD}\rangle}_{\mathsf{Lim}}(\mathsf{pms}, \mathsf{m}, \emptyset) \\ \mathtt{ctxt} \in \mathsf{Trc} \\ \mathsf{b}_{\mathcal{V},\mathsf{pub}} = 1 \text{ i.e. } \top \leftarrow \mathsf{ValidProof}(\mathsf{pms}, \mathsf{Trc}) \\ \mathsf{b}_{\mathcal{V},\mathsf{priv}} = 1 \text{ i.e. } \top \leftarrow \mathsf{ValidOption}(\mathsf{pms}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}}) \\ \mathsf{m} \neq \mathsf{Dec}(\mathtt{ctxt}, \mathsf{sk}) // \text{ but } \mathtt{ctxt} \text{ does not contain the intent} \end{cases}$$

Then, we run $\mathsf{Vote}^{\mathsf{Lim}}$ protocol without changes until OED generated and published a nonce $\mathsf{nc}_{\mathcal{V}}$, where we make a fork and publish two different nonces $\mathsf{nc}_{\mathcal{V}} \neq \mathsf{nc}'_{\mathcal{V}}$ on $\mathcal{BB}$. Since we assumed that event $\mathsf{Cheat}_{\mathsf{Lim}}$ happens with a non-negligible probability, $\mathcal{VD}$ should be able to finish the two executions successfully and produce

accepting proofs π and π':

$$\pi = (\mathsf{nc}_{\mathcal{V}}, \mathtt{ctxt}, \{\mathsf{a}_{1,k}, \mathsf{a}_{2,k}, \mathsf{e}_k, \mathsf{z}_k\}_{1\leq k\leq \ell})$$

$$\pi' = (\mathsf{nc}'_{\mathcal{V}}, \mathtt{ctxt}', \{\mathsf{a}'_{1,k}, \mathsf{a}'_{2,k}, \mathsf{e}'_k, \mathsf{z}'_k\}_{1\leq k\leq \ell})$$

Since, up to publishing the nonce, all steps of both executions are identical, and the hash function $\hat{\mathsf{h}}$ is collision-resistant, we conclude that some values in π and π' are equal:

$$\mathtt{ctxt} = \mathtt{ctxt}'$$

$$\{(\mathsf{a}_{1,s}, \mathsf{a}_{2,s})\}_{1\leq s\leq \ell} = \{(\mathsf{a}'_{1,s}, \mathsf{a}'_{2,s})\}_{1\leq s\leq \ell}$$

$$\{\mathsf{e}_i\}_{i\in\{1,2,\ldots,\ell\}, i\neq j} = \{\mathsf{e}'_i\}_{i\in\{1,2,\ldots,\ell\}, i\neq j}$$

However, since $\mathsf{nc}_{\mathcal{V}} \neq \mathsf{nc}'_{\mathcal{V}}$, with overwhelming probability $\mathsf{e}_j \neq \mathsf{e}'_j$ as:

$$\mathsf{e} = \mathsf{H}(\mathtt{ctxt}, \{(\mathsf{a}_{1,k}, \mathsf{a}_{2,k})\}_{1\leq k\leq \ell}, \mathsf{nc}_{\mathcal{V}}) \neq \mathsf{H}(\mathtt{ctxt}, \{(\mathsf{a}_{1,s}, \mathsf{a}_{2,s})\}_{1\leq k\leq \ell}, \mathsf{nc}'_{\mathcal{V}}) = \mathsf{e}'$$

Now dividing the two satisfied equations $\mathsf{g}^{\mathsf{z}_j} = \mathsf{a}_{1,j} \cdot (\mathsf{c}_1)^{\mathsf{e}_j}$ and $\mathsf{g}^{\mathsf{z}'_j} = \mathsf{a}_{1,j} \cdot (\mathsf{c}_1)^{\mathsf{e}'_j}$ we get $\mathsf{c}_1 = \mathsf{g}^{\frac{\mathsf{z}_j - \mathsf{z}'_j}{\mathsf{e}_j - \mathsf{e}'_j}}$ on the one hand.

On the other hand, dividing the two satisfied equations $\mathsf{pk}^{\mathsf{z}_j} = \mathsf{a}_{2,j} \cdot \left(\frac{\mathsf{c}_2}{\mathsf{m}_j}\right)^{\mathsf{e}_j}$ and $\mathsf{pk}^{\mathsf{z}'_j} = \mathsf{a}_{1,j} \cdot \left(\frac{\mathsf{c}_2}{\mathsf{m}_j}\right)^{\mathsf{e}'_j}$ we get $\mathsf{c}_2 = \mathsf{m}_j \cdot \mathsf{pk}^{\frac{\mathsf{z}_j - \mathsf{z}'_j}{\mathsf{e}_j - \mathsf{e}'_j}}$.

Therefore, we have $\mathtt{ctxt} = (\mathsf{c}_1, \mathsf{c}_2) = (\mathsf{g}^{\mathsf{r}_j}, \mathsf{m}_j \cdot \mathsf{pk}^{\mathsf{r}_j})$ for $\mathsf{r}_j = \frac{\mathsf{z}_j - \mathsf{z}'_j}{\mathsf{e}_j - \mathsf{e}'_j} \bmod \mathsf{q}$, which means $\mathtt{ctxt}$ is an encryption of the voting option m_j. This contradicts the fact that event $\mathsf{Cheat}_{\mathsf{Lim}}$ was happening. □

Theorem 6.4.2 (Solution S_{Lim} achieves CR)**.** Assuming the zero-knowledge of the Σ-protocol for $\mathcal{R}_{\mathrm{OR}}$, the protocol described in Figure 6.2 achieves the Coercion-Resistance (CR) property for a computationally limited voter (Definition 13).

Proof. Recall that the voter's role in the protocol is limited - only choosing the intended voting option and pressing the "Nonce" button. Hence, the coercer $\mathcal{C}$ cannot enforce an elaborate voting strategy. The only possible instruction $\mathcal{C}$ can force voters to follow would be to vote for some option $\mathsf{m}_{\mathcal{C}}$.

Since the nonce is published directly on the public bulletin board, pressing the button multiple times or restarting the voting process (perhaps in hopes of obtaining a more suitable nonce) would be obvious. Hence, we exclude those coercion attacks as preventable - the system may simply prohibit multiple nonces or demand valid proof for the first nonce published.

Theoretically, a coercer can force the voter to press the button at the wrong moment or not press the button at all, but this would lead to a non-successful execution of the protocol, which our coercion-resistance definition does not take into account. Similarly, our notion of coercion-resistance does not cover coercion strategies that require the voter to repeat vote-casting multiple times until some arbitrary condition regarding the output is satisfied (e.g., the ciphertext starts with 42). However, our definition only considers coercers that favor a specific voting option, leaving all other coercion strategies open for future research.

Assume the coercer's goal is to make the voter vote for some option $\mathsf{m}_{\mathcal{C}} = \mathsf{m}_w$, for some $w \in \{1, \ldots, \ell\}$, likely different than the voting option $\mathsf{m} = \mathsf{m}_j$ that the voter wants to choose. But in this case, all that the voter (or strictly speaking, the simulator $\mathtt{Sim}$) has to do to deceive the coercer $\mathcal{C}$ is to say it received values $(\mathsf{m}_i, \mathsf{x}_i)$, for $i \neq w$ (instead of $i \neq j$) from $\mathcal{VD}$.

In terminology of Definition 13, the simulated the voter's set of random values $\mathsf{rand}_{\mathcal{V}}^{(1)}$ can be obtained from the real trace $\mathsf{rand}_{\mathcal{V}}$ by replacing $\mathsf{m} = \mathsf{m}_j$ with $\mathsf{m}_{\mathcal{C}} = \mathsf{m}_w$ and by replacing $\{(\mathsf{m}_i, \mathsf{x}_i)\}_{1 \leq i \leq \ell, i \neq j}$ with $\{(\mathsf{m}_i, \mathsf{x}_i)\}_{1 \leq i \leq \ell, i \neq w}$. Note that all required information is available to $\mathtt{Sim}$, which has the whole trace Trc and $\mathsf{m} = \mathsf{m}_w$ as inputs. Moreover, replacing operations performed by $\mathtt{Sim}$ clearly belong to the considered class Cpb, as required. The two views in the CR game from Definition 13 would be indistinguishable due to the zero-knowledge (or rather the witness indistinguishability) property of the Σ-protocol for $\mathcal{R}_{\mathrm{OR}}$. □

6.4.2 Comparison With Bingo Voting: On The Necessity Of OED

Our protocol is similar in spirit to Bingo voting [21]: both schemes consider computationally limited voters, and both solutions use a trusted external entity to generate a (pseudo-)random nonce. However, Bingo voting relies on adding dummy votes - votes that will not be counted - for each possible candidate each voter might choose. During the vote-casting, receipts for all dummy (unselected) and actual (selected) options are printed on the corresponding candidates. In the tally, all dummy votes will be discarded. To ensure it, the election authority commits to all dummy votes before the election begins. Then, during the voting, the voter uses a trusted random number generator to generate a random number for their intent. The voting device will randomly assign dummy votes to unselected options and print all numbers next to the corresponding candidates as a receipt. All printed numbers are indistinguishable for anyone but the voter; hence, no one can identify the voter's vote. After the voting, the election authority publishes all issued receipts so every voter can verify theirs is present. Finally, the election authority creates a non-interactive proof of correctness, which shows that all unopened commitments are assigned to receipts.

Our solution improves Bingo voting in two aspects. First, the pre-voting phase in the Bingo scheme is quite expensive as it requires choosing and committing to many dummy values. In our protocol, there is no pre-voting phase, only the publication of the public parameters of the election. Second, in the Bingo voting, a pseudo-random nonce must be sent to the voter and the voting device $\mathcal{VD}$ through a secure channel because the privacy of the voting phase would be lost if this value was leaked during that execution. In our solution, the random nonce $\mathsf{nc}_{\mathcal{V}} \in \mathcal{X}$ can be (and is) made public by the official election device OED right at the moment of its generation. Therefore, our solution does not need a secure channel between the external trusted entity and the voters.

At this point, one may wonder if the help of an external trusted entity (the official election device OED in our protocol) is necessary. All in all, its only task is to generate and publish a random nonce $\mathsf{nc}_{\mathcal{V}} \in \mathcal{X}$, something that even our limited voter $\mathcal{V}$ could do on its own: if we assume $\mathcal{V}$ can memorize and compare some strings

of numbers, then for sure it would be able to generate a random string of numbers, as well. But if we modify our protocol in such a way that the voter $\mathcal{V}$ samples the nonce instead of OED, the result is a protocol that is not coercion-resistant: a coercer who wants $\mathcal{V}$ to vote for an option $\mathsf{m}^\star = \mathsf{m}_j$ can ask the voter $\mathcal{V}$ to use as the nonce $\widehat{\mathsf{nc}}_{\mathcal{V}}$ a value which is computed deterministically from the value $\mathsf{com}_j = \mathsf{H}(\{\mathsf{x}_i, \mathsf{m}_i\}_{1 \leq s \leq \ell})$. For instance, to use as $\widehat{\mathsf{nc}}_{\mathcal{V}}$ the first digits of com_j. The voter $\mathcal{V}$ cannot vote for another option $i \neq j$ because, in such a case, $\mathcal{V}$ would get to know the value of x_j (needed to compute $\widehat{\mathsf{nc}}_{\mathcal{V}}$), which is only known after the nonce is sent.

Chapter 7

Stronger Coercion Settings: New Attacks And A New Construction

In this chapter, we present ongoing work done in collaboration with Peter Rønne. We discuss how new technologies, such as time-lock puzzles and delayed encryption, could help to coerce voters into voting in a specific way. We describe the strong coercion settings in Section 7.1 and show why the previous solution S_1 is vulnerable to new coercion attacks in Section 7.2. Then, in Section 7.3, we present a protocol S_3, which improves S_2 in the number of rounds, and show its security.

7.1 Stronger Coercion Settings

When we introduced coercion settings in Section 3.2, we mentioned that the coercer couldn't physically control the voter all the time. Otherwise, for all practical purposes, it is a voter impersonation, and nothing can be done to protect the freedom of the voter's will.

Yet, the coercer can instruct the voter to behave in a specific way or use some precomputed values. Those instructions vary from supplying only a voting option to sophisticated strategies the voter has to follow during vote-casting. But more importantly, the coercer might demand voters use pre-computed values without knowing how those values are generated or computed. Usually, the voter is not vulnerable to such attacks because blindly giving the coercer's data to the voting device would result in aborted execution. However, what if the coercer uses new technologies to ensure the voter can do certain computations without knowing the coercer's values?

For example, the coercer can use a Time-Lock Puzzle [95], delayed encryption [26], or similar technologies to ensure that the voter cannot obtain the value before time T. If the algorithm for faking the proof requires knowing the hidden value, then the voter cannot execute it before time T pass.

For simplicity, we would not go into internal details of the time-lock puzzle. Instead, we focus on high-level properties those technologies provide and how a coerced might use them. We say that time-delay encryption $\mathsf{Timed\text{-}Enc}$ is a tool that generates $\mathsf{Delay}(\mathsf{x})$ for any x, with the following properties:

- The voter nor the voting device can open $\mathsf{Delay}(\mathsf{x})$ and obtain x before time T.

- The opening of delayed encryption $\mathsf{Delay}(\mathsf{x})$ cannot be parallelized, facilitated by utilizing more powerful hardware, or distributed among several entities.
- Once the secret is opened, it is possible to prove that x is hidden inside $\mathsf{Delay}(\mathsf{x})$. However, no one can easily verify whether some value x' is the solution to the puzzle $\mathsf{Delay}(\mathsf{x})$ or not.

Another possibility is for the coercer to use a blockchain or a hardware device, which maintains an append-only chain of immutable commitments. Suppose the voter is instructed to commit to all values returned by the voting devices and (maybe) use the (pseudo-)random output of the blockchain or device instead of a random value. To fake obedience in such conditions would be extremely hard for the voter. We say that time-stamped commitment-chain Timed-CC is a tool that allows anyone to add a commitment for proving the events order and has the following properties:

- The voter nor the voting device can erase or modify commitments after they appear on Timed-CC.
- Each commit is time-stamped based on the internal clock of Timed-CC.
- The voter nor the voting device can modify the internal clock of Timed-CC.
- After each commit, Timed-CC outputs a pseudo-random value.
- The voter nor the voting device can predict the pseudo-random value Timed-CC will output.

The natural question is whether our two solutions for coercion-resistant cast-as-intended verification (see Chapter 4) still work in strong settings where the coercer can use Timed-Enc and Timed-CC tools. Unfortunately, as we see in the next section, our first protocol S_1 does not withstand the new coercion attacks. As for the second construction S_2, we are still working on identifying whether it can resist new threats.

7.2 Why S_1 Is Vulnerable To Strong Coercion?

Recall the solution S_1 from Chapter 4, the simplified version of which is given in Figure 7.1. Essentially the idea is to commit to the challenge before starting the Σ-proof of the ciphertext correctness. This way (potentially coerced) voters cannot sabotage the simulatability of the transcript by selecting the challenge based on the values the voting device shows.

The coercion-resistant cast-as-intended verification works because any voter can simulate the protocol using a different or the same e. The critical observation is that, in the actual interaction, the first move of the encryption correctness proof com is computed before e is known. On the contrary, fake proof requires knowing e before generating the first move com.

However, using the new tool Timed-Enc, the coercer can ensure that the voter cannot simulate the proof. For that, it gives the voter a pre-computed commitment

cmt, a voting option m and a value Delay(e) ← Timed-Enc, which ensures that the voter cannot learn e until time T pass. Then the coercer instructs the voter to use cmt in the first round and enter displayed ciphertext ctxt and the first move of the proof com return by the voting device in the second round into Timed-CC before time T.

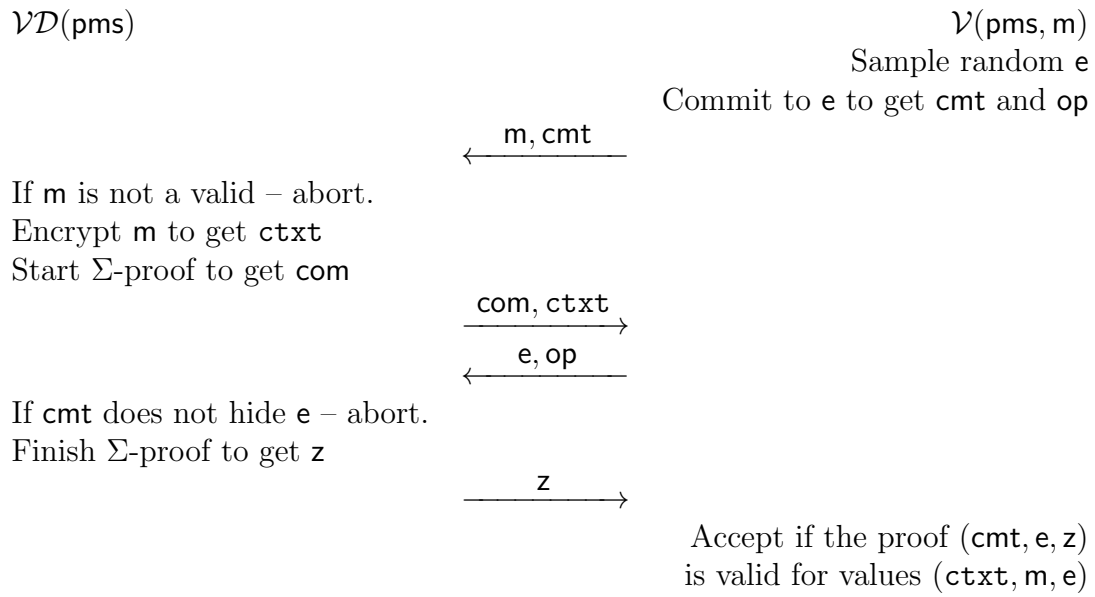


Figure 7.1: The simplified version of the solution S_1 from Chapter 4.

Recall that the voter can fake a proof for a different voting option m^* from the one encrypted in ctxt by running Sim for Σ-protocol for $(\mathtt{ctxt}, \mathsf{m}^*)$ using the same e. This will result in cmt^* and z^*, which are different from the actual cmt and z. Yet now e is unknown until after time T, which means the voter cannot run the simulator and obtain cmt^*. Hence, assuming the delay function Delay cannot be broken before T, the voter cannot disobey without being caught.

7.3 Solution S_3: Random Group Generator

At some point during our research, we doubted that S_2 could be secure in front of extreme coercion attacks[1] based on blockchains and homomorphic delay functions. This motivated us to search for new solutions, resistant to those strong coercion attacks. As a result, we obtained a three-round solution S_3 that we present and analyze now. Since (in Section 3.5) we eliminated all two-round protocols, a three-round solution S_3 is optimal in the round complexity.

Contrary to solutions S_1 and S_2 in Chapter 4, the new solution S_3 is not generic: it works for the specific setting of cyclic groups $\mathbb{G}$ of prime order q where the Discrete Logarithm problem is considered to be hard. The voter generates a non-interactive proof of the trapdoor x knowledge for the group generator g_1 randomly selected by the voting device. This way, the coercer cannot give the voter pre-computed non-interactive proof without disclosing the witness x. Once the voter successfully

[1] In the end, we concluded that such attacks were not possible.

proves x knowledge, the voting device replies with a non-interactive OR proof to show that the ciphertext encrypts the correct voting option or knows x.

$\mathcal{VD}(\mathsf{pms})$ | $\mathcal{V}(\mathsf{pms}, \mathsf{m})$

$\mathsf{g}_1 \xleftarrow{\$} \mathbb{G}$

$\xrightarrow{\mathsf{g}_1}$

If $\mathsf{g}_1 \notin \mathbb{G}$, set $\mathsf{b}_{\mathcal{V}} = 0$ and abort.

$\mathsf{x}, \mathsf{s} \xleftarrow{\$} \mathbb{Z}_{\mathsf{q}}$

$\mathsf{y} \leftarrow \mathsf{g}_1^{\mathsf{x}}$

$\mathsf{com} \leftarrow \mathsf{g}_1^{\mathsf{s}}$

$\mathsf{c} = \mathsf{H}_1(\mathsf{g}_1, \mathsf{y}, \mathsf{com})$

$\mathsf{z} \leftarrow \mathsf{s} + \mathsf{c} \cdot \mathsf{x}$

$\xleftarrow{\mathsf{m}, \mathsf{y}, \mathsf{c}, \mathsf{z}}$

If $\mathsf{m} \notin \mathbb{G}$, set $\mathsf{b}_{\mathcal{VD}} = 0$ and abort.

$\mathsf{c}' = \mathsf{H}_1(\mathsf{g}_1, \mathsf{y}, \mathsf{g}_1^{\mathsf{z}} \mathsf{y}^{-\mathsf{c}})$

If $\mathsf{c}' \neq \mathsf{c}$, set $\mathsf{b}_{\mathcal{VD}} = 0$ and abort.

$\mathsf{r}, \mathsf{t}_0, \mathsf{z}_1, \mathsf{e}_1 \xleftarrow{\$} \mathbb{Z}_{\mathsf{q}}$

$\mathtt{ctxt} = (\mathsf{c}_1, \mathsf{c}_2) \leftarrow (\mathsf{g}^{\mathsf{r}}, \mathsf{m} \cdot \mathsf{pk}^{\mathsf{r}})$

$\mathsf{a}_0 = (\mathsf{a}_{01}, \mathsf{a}_{02}) \leftarrow (\mathsf{g}^{\mathsf{t}_0}, \mathsf{pk}^{\mathsf{t}_0})$

$\mathsf{a}_1 \leftarrow \mathsf{g}^{\mathsf{z}_1} \cdot \mathsf{y}^{-\mathsf{e}_1}$

$\mathsf{e} = \mathsf{H}_2(\mathtt{ctxt}, \mathsf{a}_0, \mathsf{a}_1)$

$\mathsf{e}_0 = \mathsf{e} \oplus \mathsf{e}_1 \bmod \mathsf{q}$

$\mathsf{z}_0 = \mathsf{t}_0 + \mathsf{e}_0 \cdot \mathsf{r}$

$\mathsf{b}_{\mathcal{VD}} = 1$

$\xrightarrow{\mathsf{z}_0, \mathsf{z}_1, \mathsf{e}_0, \mathsf{e}_1}$

$\mathsf{a}_0 = (\mathsf{g}^{\mathsf{z}_0} \cdot \mathsf{c}_1^{-\mathsf{e}_0}, \mathsf{pk}^{\mathsf{z}_0} \cdot \left(\frac{\mathsf{c}_2}{\mathsf{m}}\right)^{-\mathsf{e}_0})$

$\mathsf{a}_1 = \mathsf{g}^{\mathsf{z}_1} \cdot \mathsf{y}^{-\mathsf{e}_1}$

Set $\mathsf{b}_{\mathcal{V}} = 1$ if and only if:

$\mathsf{e}_1 \oplus \mathsf{e}_2 = \mathsf{H}_2(\mathtt{ctxt}, \mathsf{a}_0, \mathsf{a}_1)$

Figure 7.2: The solution S_3. The hash function is defined as $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_{\mathsf{q}}$

7.3.1 Security Analysis Of The Protocol

Theorem 7.3.1 (Solution S_3 achieves CAI)**.** Assuming the Discrete Logarithm problem in group $\mathbb{G}$ is hard, the protocol described in Figure 7.2 achieves the Cast-as-Intended (CAI) property as per Definition 7 in the Random Oracle Model.

Proof. Remember that the CAI property is satisfied if the probability of Cheat is negligible in the security parameter λ, where Cheat is defined as follows:

$$\mathsf{Cheat} = \begin{cases} \mathsf{pms} \leftarrow \mathsf{Setup}(\lambda) \\ (\mathsf{b}_{\mathcal{V}}, \mathsf{b}_{\mathcal{VD}}, \mathsf{Trc}, \mathsf{rand}_{\mathcal{V}}) \leftarrow \mathsf{Vote}^{\langle \mathcal{V}, \mathcal{VD} \rangle}(\mathsf{pms}, \mathsf{m}, \emptyset) \\ \mathtt{ctxt} \in \mathsf{Trc} \\ \mathsf{b}_{\mathcal{V}} = 1 \text{ (voter accepts the proof)} \\ \mathsf{m} \neq \mathsf{Dec}(\mathtt{ctxt}, \mathsf{sk}) \text{ (but } \mathtt{ctxt} \text{ does not contain the intent)} \end{cases}$$

Suppose a malicious $\mathcal{VD}$ with non-negligible probability can output a valid proof, meaning that for H_2 output e it can return values $(\mathsf{z}_0, \mathsf{z}_1, \mathsf{e}_0, \mathsf{e}_1)$ that pass verification run by $\mathcal{V}$. We rewind $\mathcal{VD}$ and program H_2 to output $\mathsf{e}' \neq \mathsf{e}$. Given that $\mathcal{VD}$ succeeds with non-negligible probability, we get two valid executions for the same query $\mathsf{H}_2(\mathtt{ctxt}, \mathsf{a}_0, \mathsf{a}_1)$ but two different answers $\mathsf{e} \neq \mathsf{e}'$.

Suppose it is $\mathsf{e}_1 \neq \mathsf{e}'_1$, then we could use such a $\mathcal{VD}$ to break the discrete logarithm problem w.r.t. basis g_1 selected at random by $\mathcal{VD}$ as follows: given g_1 and a random element y, we can program the Random Oracle H_1 to choose $\mathsf{c}, \mathsf{z} \in \mathbb{Z}_\mathsf{q}$, compute $\mathsf{com} = (\mathsf{g}_1)^\mathsf{z} \cdot \mathsf{y}^{-\mathsf{c}}$ and define $\mathsf{H}_1(\mathsf{g}_1, \mathsf{y}, \mathsf{com}) = \mathsf{c}$. The $\mathcal{VD}$ would successfully verify the proof from $\mathcal{V}$ and output an OR-proof. Then, given that we assumed $\mathcal{VD}$ computes the OR proof without knowing r, from the two executions $\mathsf{e}_1 \neq \mathsf{e}'_1$, we could extract discrete logarithm of y w.r.t. basis g_1.

Thus, it is impossible, and our assumption that a malicious $\mathcal{VD}$ can output a valid proof without knowing the randomness r in Step 3 with non-negligible probability is wrong.

This leads to a contradiction, so it must be $\mathsf{e}_0 \neq \mathsf{e}'_0$, but in this case, we can extract from the two valid transcripts the randomness r s.t. $\mathtt{ctxt} = \mathbb{E}.\mathsf{Enc}(\mathsf{m}; \mathsf{r})$ can be extracted, which means CAI is satisfied.

□

Theorem 7.3.2 (Solution S_3 achieves CR in the weak settings (without tools Timed-CC and Timed-Enc))**.** Assuming DDH assumption in group $\mathbb{G}$ holds, and the instructions instr that $\mathcal{C}$ give to the voter consist of algebraic algorithms in $\mathbb{G}$ only, the protocol described in Figure 7.2 achieves the Coercion-Resistance (CR) property as per Definition 8.

Proof. In the algebraic group model [53], the coercer $\mathcal{C}$ gives the Voter $\mathcal{V}$ instructions instr consisting of an algebraic algorithm $\mathcal{B}$ and group elements $(G_1, G_2, \ldots, G_\ell)$, where $G_i \in \mathbb{G} \quad \forall i \in (1, \ell)$, where $\mathcal{B}$ outputs a group element accompanied by a representation of that group element in terms of all the group elements that $\mathcal{B}$ has seen so far [55]. In other words, if $(\mathsf{y}_1, \ldots, \mathsf{y}_k)$ are elements output or seen previously, the next output y is accompanied by its representation $(\mathsf{x}_1, \ldots, \mathsf{x}_k)$ such that $\mathsf{y} = \prod_{i=1}^{k} \mathsf{y}_i^{\mathsf{x}_i}$. The algebraic group model captures the majority of strategies and algorithms that can take place in the setting of a cyclic group $\mathbb{G}$, especially the most natural ones.

Suppose the coercer $\mathcal{C}$ gives the voter $\mathcal{V}$ as instructions a list of group $\mathbb{G}$ elements $(G_1, G_2, \ldots, G_\ell)$ and an algebraic algorithm $\mathcal{B}$. According to our model, the voter must compute a valid answer in Step 2, meaning a valid proof $(\mathsf{y}, \mathsf{com}, \mathsf{z})$ after receiving element g_1 in Step 1. Without loss of generality, we assume that, in the instructions, the element y is computed before com. It means that, other than y, the voter obtains representations $a_0, a_1, \ldots, a_\ell, \delta, b_0, b_1, \ldots, b_\ell \in \mathbb{Z}_\mathsf{q}$ such that:

$$\mathsf{y} = (\mathsf{g}_1)^{a_0} \cdot \prod_{1 \leq i \leq \ell} G_i^{a_i} \ ,$$

$$\mathsf{com} = \mathsf{y}^{\delta} \cdot (\mathsf{g}_1)^{b_0} \cdot \prod_{1 \leq i \leq \ell} G_i^{b_i}$$

If $\prod_{1\leq i\leq \ell} G_i^{a_i} = 1$, then the voter knows the discrete logarithm of y w.r.t. g_1 and can avoid coercion by simulating the OR proof in Step 3, for a ciphertext that does not encrypt the option $\mathsf{m}_{\mathcal{C}}$ imposed by $\mathcal{C}$.

Thus, we can assume $\prod_{1\leq i\leq \ell} G_i^{a_i} \neq 1$, which means that at least one $a_i \neq 0$. Since y, com and z are accepted by the $\mathcal{VD}$, then it holds that $\mathsf{g}_1^{\mathsf{z}} = \mathsf{com} \cdot \mathsf{y}^{\mathsf{c}}$, where $\mathsf{c} = \mathsf{H}_1(\mathsf{g}_1, \mathsf{y}, \mathsf{com})$ is considered to be random the Random Oracle Model. In turn, it implies the following:

$$(\mathsf{g}_1)^{\mathsf{z}} = \left((\mathsf{g}_1)^{a_0} \cdot \prod_{1\leq i\leq \ell} G_i^{a_i}\right)^{\delta} \cdot (\mathsf{g}_1)^{b_0} \cdot \prod_{1\leq i\leq \ell} G_i^{b_i} \cdot \left((\mathsf{g}_1)^{a_0} \cdot \prod_{1\leq i\leq \ell} G_i^{a_i}\right)^{\mathsf{c}}$$

If we denote $\alpha_0 = (\delta+\mathsf{c})a_0+b_0 \bmod \mathsf{q}$ and $\alpha_i = (\delta+\mathsf{c})a_i+b_i \bmod \mathsf{q}$ for $i = 1, \ldots, \ell$, we can write the previous equality as follows:

$$(\mathsf{g}_1)^{\mathsf{z}-\alpha_0} = \prod_{1\leq i\leq \ell} G_i^{\alpha_i}$$

Since the value c is random and at least one $a_i \neq 0$, we have that: (i) $\mathsf{z} - \alpha_0$ is random, and thus it is different from 0 with overwhelming probability, (ii) at least one of the exponents α_i is random.

Now we prove that the existence of such a coercion instruction given by $\mathcal{C}$ would imply that the discrete logarithm problem in $\mathbb{G}$ can be solved. Let (g, Y) be a given instance of this problem for a generator g and random $Y \in \langle \mathsf{g} \rangle = \mathbb{G}$. Then we can run the protocol with such a coercion $\ell + 1$ times using $\mathsf{g}_1^{(j)} = \mathsf{g}^{u_j} \cdot Y^{v_j}$ as inputs, where $u_j, v_j \in \mathbb{Z}_{\mathsf{q}}$ are random and independent for $j = (1, \ldots, \ell + 1)$. For each of these executions, we would end with the following equality:

$$\left(\mathsf{g}_1^{(j)}\right)^{\mathsf{z}^{(j)}-\alpha_0^{(j)}} = \prod_{1\leq i\leq \ell} G_i^{\alpha_i^{(j)}} \tag{7.1}$$

Now the $\ell + 1$ vectors $\vec{\alpha}^{(j)} = (\alpha_1^{(j)}, \ldots, \alpha_\ell^{(j)})$ must be linearly dependent in $(\mathbb{Z}_{\mathsf{q}})^{\ell}$, which means that there exist $\ell + 1$ coefficients $(\mu_1, \ldots, \mu_\ell, \mu_{\ell+1})$ such that $\sum_{1\leq j\leq \ell+1} \mu_j \vec{\alpha}^{(j)} = \vec{0} \bmod \mathsf{q}$. Raising each Equation 7.1 to μ_j and multiplying the resulting $\ell + 1$ equations would result in:

$$\prod_{1\leq j\leq \ell+1} \left(\mathsf{g}_1^{(j)}\right)^{\left(\mathsf{z}^{(j)}-\alpha_0^{(j)}\right)\mu_j} = 1$$

Given that $\mathsf{g}_1^{(j)} = \mathsf{g}^{u_j} \cdot Y^{v_j}$, the last equality can be re-written as $\mathsf{g}^E \cdot Y^F = 1$, where $E = \sum_{1\leq j\leq \ell+1} u_j\mu_j \left(\mathsf{z}^{(j)} - \alpha_0^{(j)}\right) \bmod \mathsf{q}$ and $F = \sum_{1\leq j\leq \ell+1} v_j\mu_j \left(\mathsf{z}^{(j)} - \alpha_0^{(j)}\right) \bmod \mathsf{q}$. Also, since values v_j and $\mathsf{z}^{(j)} - \alpha_0^{(j)}$ are random, with overwhelming probability $F \neq 0 \bmod \mathsf{q}$. Therefore, with overwhelming probability, we can output the value $-E \cdot F^{-1} \bmod \mathsf{q}$ as the discrete logarithm of y for the generator g.

Hence, the only possible instructions that $\mathcal{C}$ can impose on the voter $\mathcal{V}$ imply that the voter knows the value x such that $\mathsf{y} = (\mathsf{g}_1)^{\mathsf{x}}$. However, then the voter can use the knowledge of x to simulate a valid and indistinguishable OR proof for any ciphertext $\mathtt{ctxt}$ and plaintext m in Step 3, which includes plaintext $\mathsf{m}_{\mathcal{C}}$ chosen by $\mathcal{C}$ and a ciphertext that encrypts a different voting option m. □

7.4 Future Research

This chapter contains ongoing research that has not been finalized yet. In particular, some lines of research remain open:

1. Formally prove security of S_2 and S_3 in front of strong coercions. While, intuitively, both S_2 and S_3 solutions seem to withstand strong coercion, we must formally prove it before making any claims. However, such proof is not trivial to construct in the generic case, where we cannot make assumptions regarding the underlying mathematical group.

2. Generalize S_3. In contrast to S_2, the solution S_3 was proven to provide coercion-resistance and cast-as-intended verification for the specific settings: cyclic groups $\mathbb{G}$ of prime order q where the DDH problem is hard. It would be good to generalize it to remove the dependency on the DDH assumption.

3. Find a post-quantum secure version of S_3. It remains an open problem to instantiate S_3 with lattice-based primitives or, alternatively, find a three-rounds protocol with CAI-CR with post-quantum security.

4. Explore strong coercion settings more. We believe that with more research, more coercion attacks based on newly appeared primitives can be identified. Also, it seems there are settings where it is impossible to achieve both coercion-resistance and cast-as-intended properties simultaneously. Finally, through this research, we focused only on guaranteed verification, while one might also consider probabilistic verification.

Chapter 8

Conclusion

This thesis has focused on the two contradictory properties of electronic voting: cast-as-intended verification and coercion-resistance. We can distinguish three main parts of the research done in this work.

In the first part, after analyzing the current state of the art, we put forward two formal definitions for achieving coercion-resistant cast-as-intended verification in settings without pre-exchanged data. After that, we presented two practical constructions, which are provably secure under the proposed definitions. Also, we implemented those constructions and gave the benchmarks to prove the practicality.

In the second part, we considered post-quantum settings and noticed the usability issues rooted in the lattice-based math affecting both proposed solutions. As a solution, we proposed a generic transformation, which turns a multi-round interactive zero-knowledge system (with a possibility of re-runs) into a 3-move zero-knowledge system (always without re-runs). To demonstrate practicality, we applied the construction to a 5-round protocol of Bootle *et al.* [23] for proving the knowledge of a ternary solution to a linear equation over $\mathbb{Z}_{\mathsf{q}}$.

In the final part, we focused on the practical aspects of the coercion-resistant cast-as-intended verification. First, we studied the case of a computationally limited voter, which we consider the most realistic. We proved that even such a voter enjoys coercion-resistant cast-as-intended verification if given the help of a simple aid device for a nonce generation. After that, we focused on cases of extreme coercion relying on delay encryption and blockchain and discovered that one of the previous solutions is vulnerable. Finally, during our ongoing research on advanced coercion threats, we found a protocol with three rounds of interactions (which is optimal) and gave the intuition of its security.

To summarize, we looked at the problem of combining coercion-resistance with cast-as-intended verification from different angles: formal definition, practical constructions, post-quantum security and usability, limited voter, and extreme coercion. As a result of our research, we obtained two formal definitions and four different constructions for addressing coercion threats in different situations.

This Ph.D. thesis ends here, but we leave several doors open for future research. It remains an open problem to generalize and instantiate in post-quantum settings solution S_3. Also, we believe diving deeper into the extreme coercion based on newly appeared primitives will allow identify new potential attack vectors and pre-

pare for them beforehand. Additionally, it would be interesting to find settings where the contradiction between coercion-resistance and cast-as-intended properties would lead to the impossibility of having both properties simultaneously. Finally, through this research, we focused only on guaranteed verification, while one might also consider probabilistic verification.

The last conclusion we want to share, which we learned from our practical experience working in a company specializing in secure electronic voting solutions, is that security is not enough. Not only must any protocol (such as voting, verification, anonymization, etc.) be safe, but it must also be usable and compliant with legislation. Otherwise, voters would be unable to use even the best (from a cryptographic point of view) solutions.

Appendix A

Brief Description Of Mentioned E-Voting Schemes

In this Appendix, we briefly describe some of the e-voting schemes discussed in the Introduction of this thesis. We do not aim to give a detailed specification for every single existing solution, rather we want to help readers to understand our references and conclusions.

A.1 Helios Voting System

Helios is probably one of the most famous electronic voting schemes. Its derivatives were used to run the election of the president of the University of Louvain-La-Neuve and the election of 2010, 2011, and 2012 new board directors of the International Association for Cryptographic Research (IACR) [36]. It has inspired numerous usability studies, resulted in countless scheme variations, and become one of the central comparison points. The derivatives of Helios vary from enabling voting by proxy [73] to extending the scheme to account for new properties[74].

Therefore, the Helios voting system is more of a family combining different versions under the same umbrella rather than one particular design. There were numerous adaptations, upgrades, and slight changes, which resulted in multiple constructions that could be called Helios. However, since the core idea - Benaloh's challenge - remains roughly the same for all Helios variations, we would focus on the voting part of the protocol.

Helios protocol: The setup phase mainly consists of establishing the election key, plus voters' signing and authentication keys. Some articles specify that a group of Trustees generates the election key. However, there are no details on how all voters receive their keys securely. Therefore we assume all voters' keys are created and distributed by the election authority.

After a voter enters the choice, the voting device creates a ballot and correctness proof. Then it displays a hash of the ballot and gives the voter a chance to either audit or cast it. The voter must record or remember the hash value since it will serve as a commitment for verification or receipt of a sent vote. If the voter chooses to audit, the voting device outputs the encryption randomness so the voter can

reconstruct the ciphertext on a verification device and check that it contains an intended vote. The voter can perform audits multiple times before deciding to cast the ballot, which will be sent directly without verification. The entire vote-casting and vote-verification process is detailed in Figure A.1.

Depending on the Helios version, ballot correctness proofs vary. Helios 1.0 provides no details, even though it is essential for preventing vote copying (replay attack). A custom version of Helios [25] adds ballot validity proof but switches from ElGamal to a modification of the TDH2[101] encryption scheme that preserves homomorphic properties. Helios 2.0[42], patched Helios 3.0[47] and latest Helios 4.0[4] with fixed Fiat-Shamir transformation and Helios-C[36] use disjunctive-or NIZKs to guarantee that selection is appropriate.

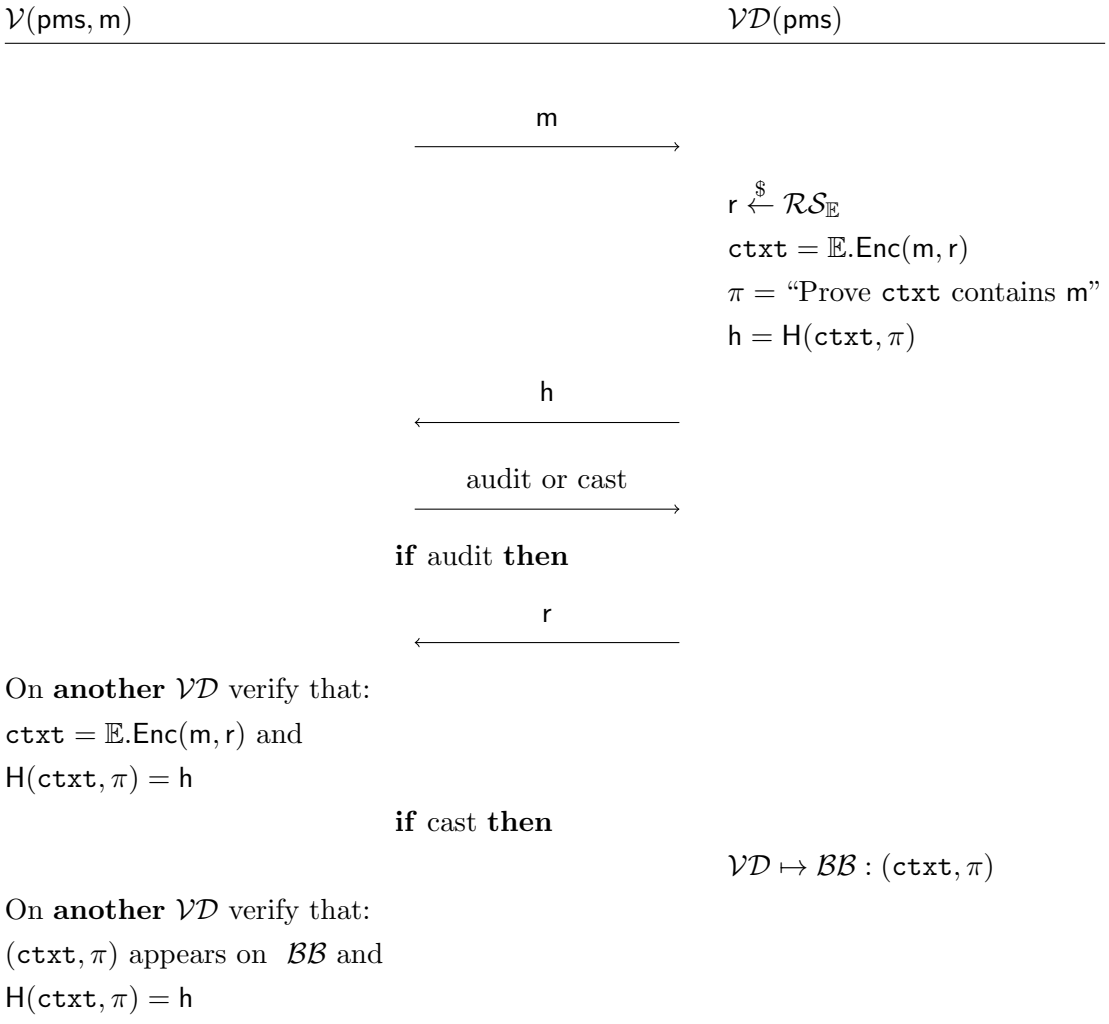


Figure A.1: A ballot verification based on Benaloh's challenge.

The tally is typically done homomorphically, yet some versions offer compatibility with a verifiable shuffle [5, 25]. We focus on Helios 4.0, which only supports homomorphic counting and ensures result correctness by publishing decryption NIZK proofs.

Overall, the cast-or-challenge mechanism is an easy verification technique. However, since the cast ballot is never verified, technically, it does not guarantee that the vote contains the intended option. A malicious voting device can guess when the voter would stop the audit and encrypt another voting option. An additional challenge comes from the fact that voters must remember the hash of the shown ballot before deciding whether to cast or verify it. Otherwise, an adversary can commit to one vote but cast or give another for verification. Finally, the audit must always happen on another voting device; otherwise, the voting device can always claim everything was correct.

A.2 Belenios Voting System

The Belenios system is another family of voting schemes. It started with Helios-C[36] offering partial verification, later referred to as Belenios[34]. Then the solution was enhanced with a re-encrypting server to provide receipt-freeness at the cost of castes-intended verification resulting in BeleniosRF [29]. The latest version - BeleniosVS [35] - is based on pre-delivered QR codes containing encrypted ballots.

Belenios protocol: The setup phase is similar to Helios and consists of establishing the election key and voters' signing keys.

During the voting phase, the voter enters the preferred vote m - an integer between 0 and $k-1$, where k is the total number of voting choices. The voting device encrypts the selection m, then samples a random value s_0, computes $\mathsf{s}_1 = (\mathsf{m} - \mathsf{s}_0) \bmod k$ and encrypts s_0 and s_1 under the election public key pk. After that, the voting device creates a zero-knowledge proof showing that encryption for m contains the same plaintext as the product of ciphertexts encrypting s_0 and s_1. Finally, the voting device signs the ballot and publishes ciphertexts, the proof, and the signature on the bulletin board.

For the cast-as-intended verification, the voting device shows the ballot (or a hash of the ballot) and s_0 and s_1 to the voter. The voter checks that $\mathsf{s}_0 + \mathsf{s}_1 \bmod k = \mathsf{m}$ and the ballot is published, then selects a bit $\alpha = \{0, 1\}$ at random and sends it to the voting device. If $\alpha = 0$, then the voting device publicly reveals $\mathsf{s}_0, \mathsf{r}_0$ and $\mathsf{s}_1, \mathsf{r}_1$ otherwise. To complete the verification, the voter must access the bulletin board and ensure that the published s_α is correct. The entire vote-casting and vote-verification process is detailed in Figure A.2.

During the tally, the votes are anonymized and decrypted. Only the encrypted voting option m enters the decryption stage - other ciphertexts are removed during the anonymization.

Belenios allows partial audit of the ballot - only one of two additional ciphertexts $\mathtt{ctxt}_0$ or $\mathtt{ctxt}_1$ are opened. Therefore, a malicious voting device still has a 50% probability of undetectably modifying the voter's choice. Moreover, the voter would need an additional device to verify that the published on the bulletin board ballot is identical to the one voter saw during the protocol. Otherwise, a malicious voting device can assign the same vote to multiple voters and successfully pass all checks.

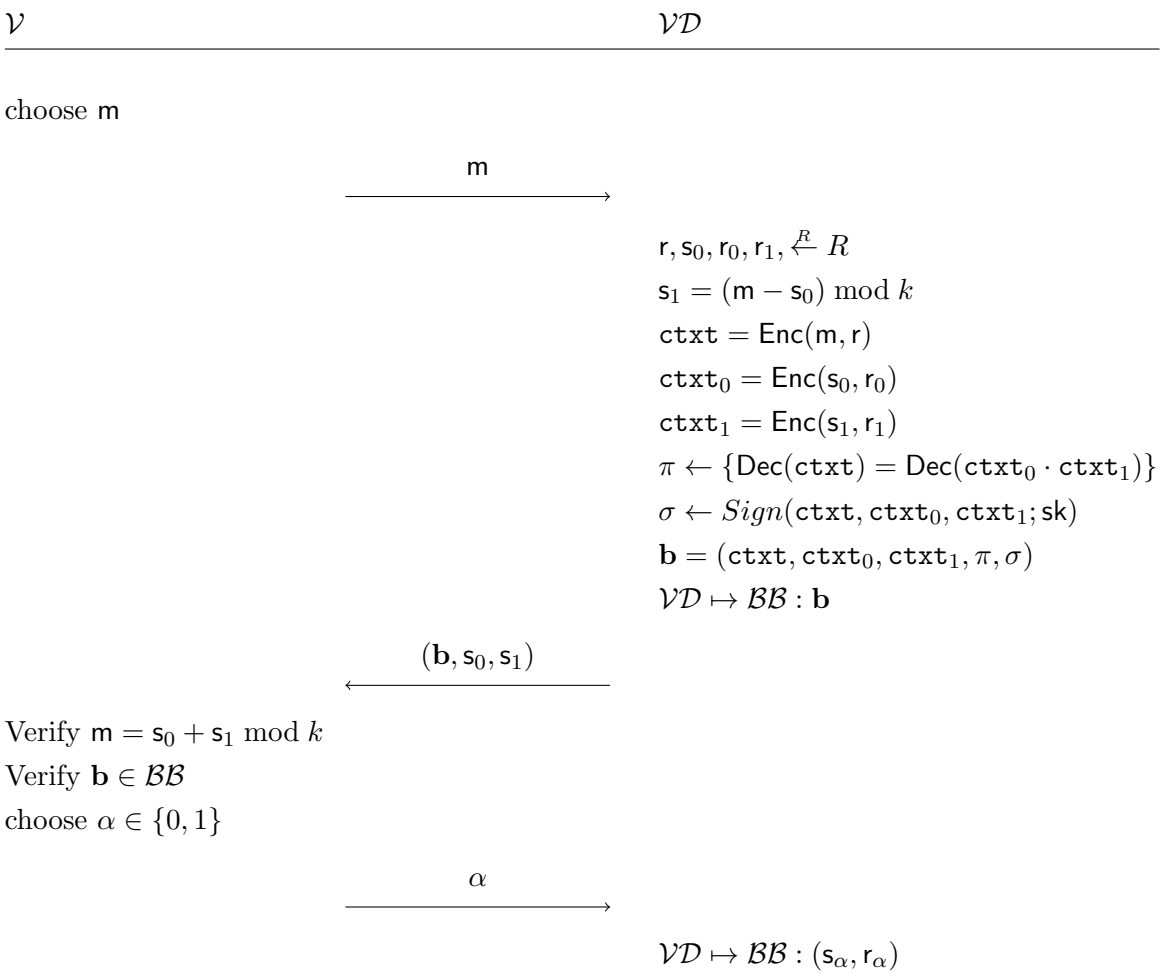


Figure A.2: A ballot verification from Belenios protocol.

A.3 Demos Voting System

Demos [71] is a code-based voting system that does not put any trust in the election authority for verifiability.

During the setup, the election authority prints a two-sided voting card with options (in random order) and corresponding vote codes for each voter. For example, for a simple yes/no choice, the card might look as follows:

Side A		**Side B**	
yes	1234	no	7764
no	5611	yes	9877
voter ID: 421111		voter ID: 421111	

The election authority encodes each j-th voting option as N^{j-1}, where N is the total number of voters plus one, and commits to all encodings and vote codes. Finally, it starts a Σ-protocol to show that commitments correspond to an unknown

permutation of the encoded candidates. The first move of the Σ-proof is publicly available on a bulletin board before an election, but the proof will be finalized after the voting phase is over.

The vote-casting is simple and does not require any encryption, see Figure A.3 for details. A voter randomly selects one of the ballot sides, authenticates using voter ID, and sends the preferred vote code, e.g. (id: 421111, code: 1234, side: 'A'). After voting, the voter should cut off and destroy the used side of the voting card. The unused side and the selected code will serve as a receipt.

$\mathcal{V}(\mathsf{pms}, \mathsf{m}, \text{two-sided voting card})$		$\mathcal{VD}(\mathsf{pms})$
Select ballot side at random Find code that corresponds to m		
	code, voter ID $\longrightarrow$	
		$\mathcal{VD} \mapsto \mathcal{BB}$: (code, voter ID)
On **another** $\mathcal{VD}$ verify that: The right code appears on $\mathcal{BB}$		

Figure A.3: The vote casting protocol of Demos voting system.

During the tally, the election authority opens all commitments to vote codes from both ballots' sides and encoded options from unused sides to prove consistency with the printed ballots. Then, it uses randomness extracted from the voter's ballot-sides selections as a challenge to finalize the Σ-protocol. Finally, the election authority homomorphically tallies selected voting options and decrypts the result. All proofs and openings are available on a bulletin board.

A major practical drawback of the Demos verification is expectations regarding voters' behavior. Voters must select a ballot side at random, destroy the used side of the ballot and perform receipt verification after the voting phase is over. While there are no studies regarding ballot destruction, experiments show that about 85% of voters tend to select side A [69]. Finally, while coercion is not covered, the soundness of the Σ-proof might be affected when the coercer forces enough voters to use a specific ballot side.

A.4 Demos-2

Demos-2 [70] is a scalable encryption-based variation of the Demos scheme that achieves similar performance to Helios.

During the setup, the election authority generates and publishes a master common reference string[1] (crs). The crs can be either perfectly sound or perfectly simu-

[1]Common reference string is a model where a public string is generated in a trusted manner, and each party has access to it.

latable. The former results in perfectly sound NIZK proofs; the latter makes every NIZK proof simulatable due to the use of a trapdoor. Therefore, the election authority starts a Σ-protocol to show that this crs is of a perfectly sound type. The first move of the Σ-proof is publicly available on a bulletin board before an election, but the proof will be finalized after the voting phase is over.

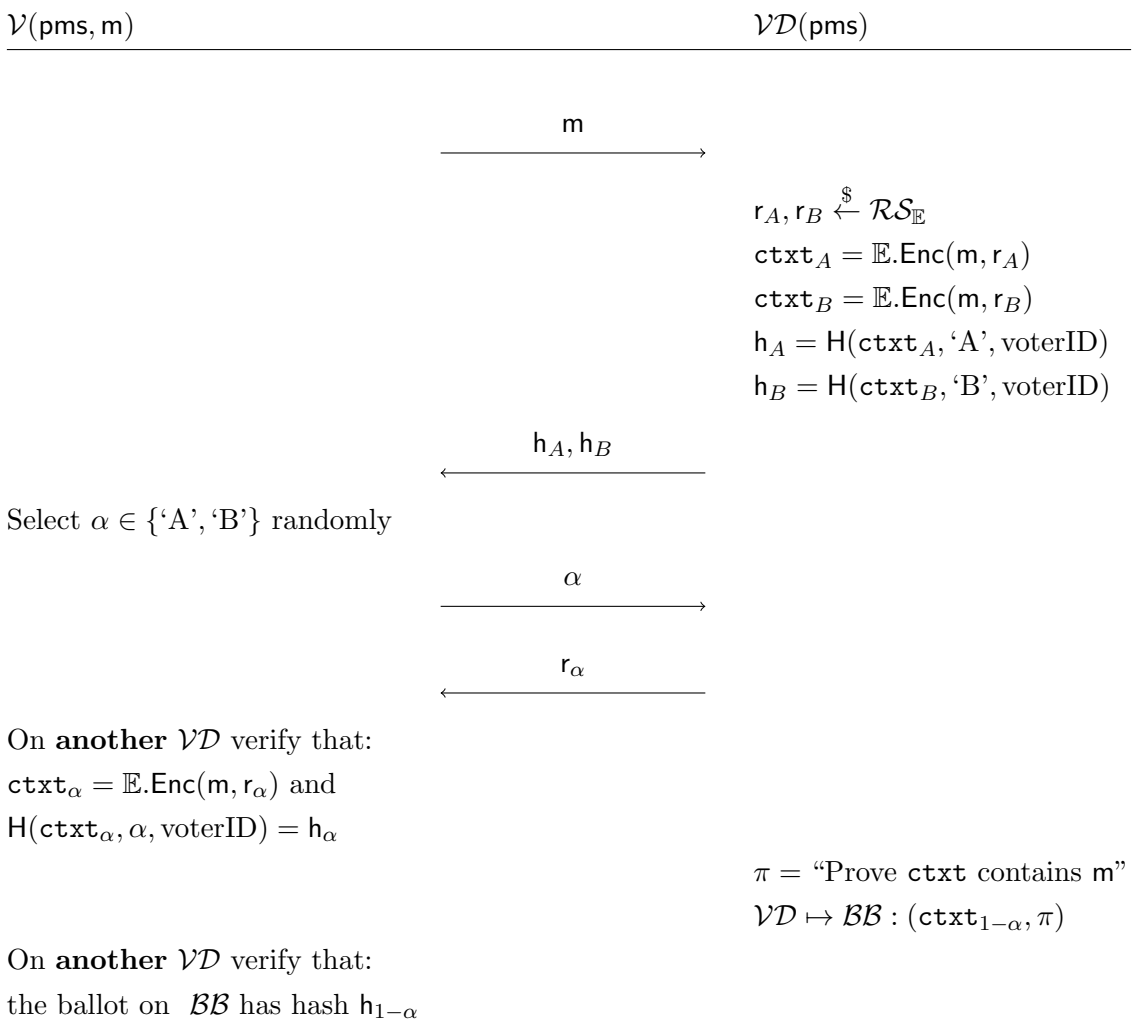


Figure A.4: The vote-casting and voter-verification of the Demos-2 voting system.

To cast a vote, the voter types the choice, which will be encoded as a binary vector. The voting device encrypts each bit independently twice and obtains two ballots; the hash of each is shown to the voter. The voter chooses which ballot to verify on a separate device; the remaining one will be cast. For the cast ballot, the voting device computes a ballot correctness proof: i.e. encrypts a binary vector. The proof soundness heavily relies on the fact that the master crs is perfectly sound. The entire vote-casting and vote-verification process is detailed in Figure A.4.

During the tally, the electoral authority uses voters' ballot side selections as a challenge to finalize the Σ-protocol to prove crs is of perfectly sound type. Then, it

homomorphically tallies selected voting options and decrypts the result. All proofs and openings are available on a bulletin board.

Like its precursor demos, Demos-2 still expects voters randomly choose between two ballots. Also, the scheme requires two separate devices for vote-casting and vote-verification. It is not very convenient, as it might prevent people with only one device from verification. Additionally, such a verification mechanism requires public education to ensure that no one mistakenly uses the same machine for voting and verification.

A.5 Swiss Post Voting System

Swiss Post voting system is a code-based voting scheme that uses return codes to provide individual verifiability and requires a mandatory vote confirmation. The first version with distributed server functionalities appeared in 2018 [41]. Later, in 2020, the scheme was slightly upgraded to remove reliance on auditors for individual verifiability. The rest of the section is based on the latest system specification, i.e. version 1.2.0, available online [92].

During the setup phase, the setup component generates voters' credentials, return codes, start voting keys and ballot casting keys. Then, jointly with four online control components, it prepares a mapping table, which stores the encrypted correspondence between the voting options (and confirmation messages) and a set of return codes. The setup component sends the return codes and keys to the printing component, which prints the voting cards and securely distributes them among voters. For example, for a simple yes/no choice, the voting card might look as follows (abstention option is always present):

Voter ID	
Start voting key:	ti6v-kvtc-yiju-hh5h-pjsk
Question 1:	
No	1225
Yes	7096
Empty	8790
Ballot casting key:	8147-1584-8
Vote cast return code:	0742-5185

Finally, the setup component, control components, and the electoral board together generate the election key pair. The fifth control component - the tally control component- does not participate in the setup phase.

To vote, the voter enters their private Start Voting Key (SVK) (printed on their voting card) and the selected voting options. The voting client creates an encrypted ballot with two parts: one contains the encrypted product of selections, and the other the encrypted pre-return codes. Each pre-return code is the exponentiation of each voter's choice to the voter's private key. The zero-knowledge proofs link both ballot parts and ensure they contain the same options.

A voting server forwards the ballot to the control components that verify its correctness and eligibility. The control components jointly decrypt the pre-return

code, then check it is inside the valid list of pre-return codes and, if so, exponentiate the hash of it to a control component's secret key associated with the voter. The voting server uses those exponentiated values to extract the Choice Return Codes from the mapping table and sends them back to the voting client.

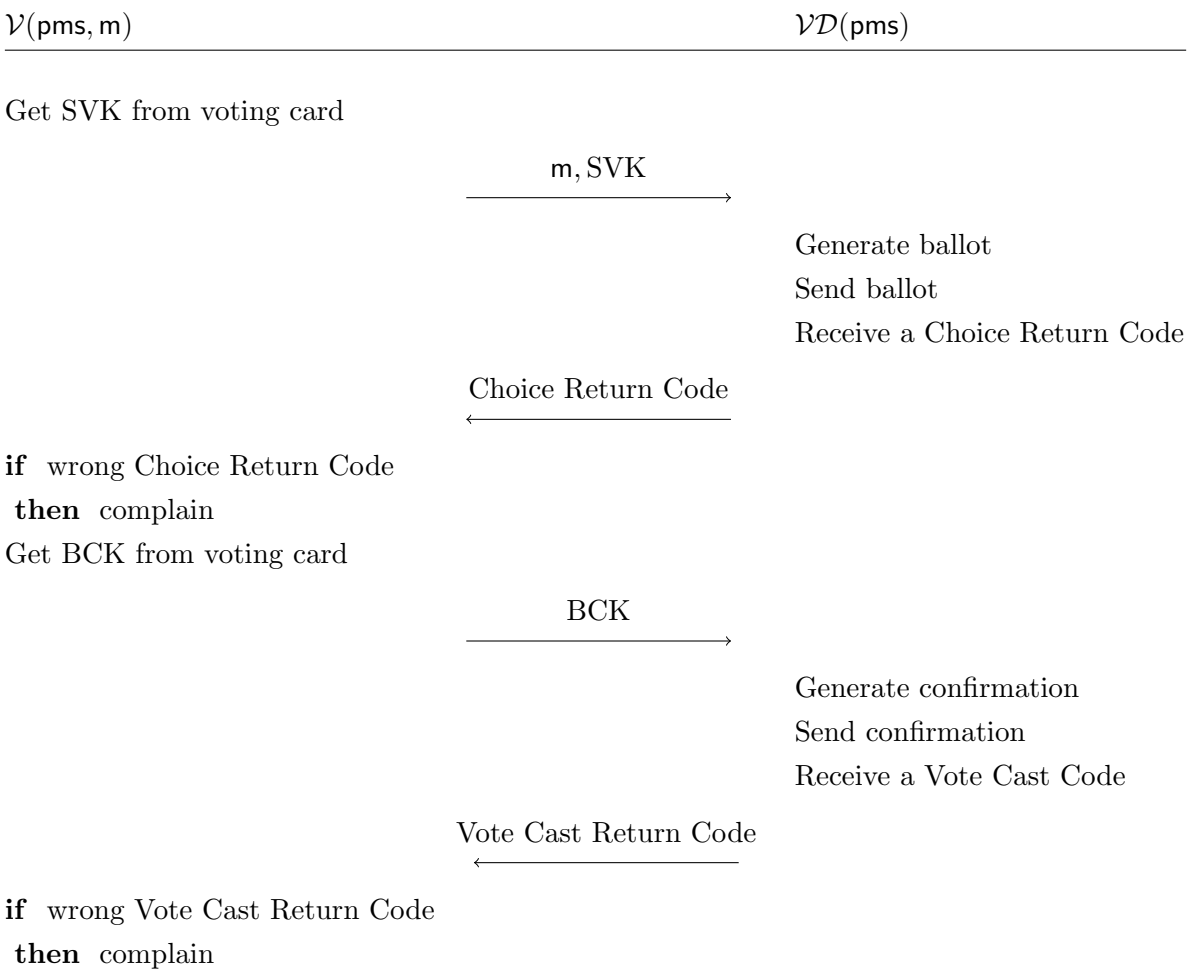


Figure A.5: The vote-casting and voter-verification of the Swiss Post voting system.

The voting client shows the received codes to the voter, who cross-checks them with the codes printed on their voting card. If all codes match, the voter confirms the vote by entering their private Ballot Casting Key (BCK) printed on their voting card.

The vote-confirmation part is similar to the vote-casting procedure. The voting client derives the confirmation message from that key and sends it to the voting server, which forwards it further. Each control component performs exponentiation of the confirmation message and outputs the hash of the result. However, the control component does not reveal the result until it receives hash values from other control components and ensures all hash values are in the allowed list. Any vote that has hashes in the allowed list is marked as confirmed and outputs the exponentiation

result. As before, the voting server uses those exponentiated values to extract the Vote Cast Return Codes from the mapping table. If the voter does not receive the code or it is incorrect, they complain to the authorities.

The exact procedure required for extracting the return codes from the mapping table is a bit complicated due to the involvement of four control components. However, from the voter's point of view, the verification and confirmation processes are extremely simple - see Figure A.5 for details.

During the tally phase, control components select the confirmed votes and conduct their verification, perform verification, verifiably shuffle, and partially decrypt confirmed votes. Finally, the electoral board reveals its passwords to the tally control component, which does the final audit, shuffle, decryption, and decoding of the result.

From a practical view, the Swiss Post Voting scheme inherits all limitations of return-code-based solutions: the need for a secure printing facility, mandatory secure channels for voting card delivery, and a limited number of candidates. However, it enforces mandatory vote-verification as there is a vote-confirmation stage.

A.6 Estonian Internet Voting Scheme

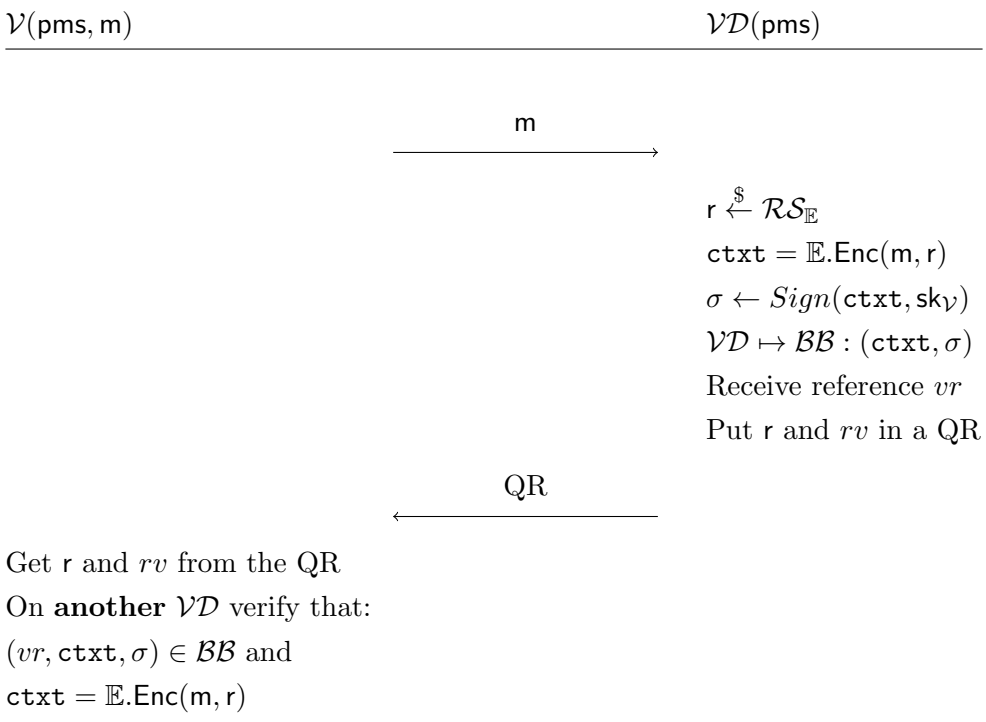


Figure A.6: The vote-casting and voter-verification of Estonian voting system.

The first version of the Estonian Internet voting scheme [62] was developed in the early 2000s and used between 2005-2016 for several elections in Estonia. It was based on RSA encryption, and no mixing was performed during the counting. Lately, in

2017, appeared a new IVXV version ES2017 [2], which is based on ElGamal and verifiable mixing.

During the setup, trustees jointly generate an election key pair and store their private shares on smart cards.

To vote, voters authenticate in a voting application installed on their devices using their national ID card, digital identity document (Digi-ID), or Mobile-ID. After authentication, the system verifies the eligibility of the voter. Then, the voter selects the desired voting options, and the voting device generates a random seed r and uses it to encrypt selections. The resulting vote is signed with the national ID card (we refer to it as $\mathsf{sk}_{\mathcal{V}}$ secret key) and sent to the voting server. The voting server verifies the signature, stores the ballot, and returns a reference vr to the vote. Finally, the voter obtains a QR code with vr and r for verification purposes.

The system provides cast-as-intended verifiability based on the cast-and-decrypt approach via the QR code given to the voter, see Figure A.6 for details. The check is done by scanning the QR code with the verification device, which is a smartphone in this case. The verification device uses the vote reference vr to request the vote from the voting server and the list of candidates. Once received, the verification device decrypts the ballot using the randomness r and shows the result to the voter, who checks if it matches their intention.

When the election finishes, the ballots are validated and anonymized.

A.7 Hyperion Voting System

Hyperion [99] is an enhanced version of a Selene system [98] that aims to provide intuitively understandable verification for voters by assigning each voter a private tracking number. While the original Selene posts tracking numbers with each ballot, Hyperion reveals tracking numbers privately. Yet, both schemes allow voters to identify and check their votes after all votes are decrypted.

As a precondition, the scheme assumes that each voter already generated trapdoor keys $(\mathsf{x}_i, \mathsf{h}_i = \mathsf{g}^{\mathsf{x}_i})$ via the voter's app and registered the public part h_i. Also, voters are expected to have a signing key pair. It is not specified which entity generates those keys. For simplicity, we assume that an election authority is in charge of this task.

During the setup phase, the tellers jointly generate a tellers' public key, which is posted on a bulletin board. It is not specified who generates the election key pair. For simplicity, we assume a group of Trustees distributedly generates election keys.

To cast a vote, the voter chooses the intended selections. Then the voting device encrypts them, proving the plaintext knowledge and signs. The resulting ballot is published on the bulletin board by the voting server. After that, each teller j exponentiates the voter's public key $\mathsf{h}_i^{\mathsf{r}_{ij}}$ to a random r_{ij}, publishes the result, and keeps the corresponding r_{ij} secret. The product of results from all tellers is called the beta term and denoted as $\beta = \mathsf{h}_i^{\mathsf{r}_i}$.

After the voting phase, votes are stripped of signatures and the corresponding zero-knowledge proofs. Then Trustees pair the stripped ciphertexts with beta terms and perform verifiable shuffle and decryption. The decrypted votes and corresponding re-randomized beta terms $\mathsf{h}_i^{\mathsf{r}_i \cdot \mathsf{s}_i}$ are posted on the bulletin board. It is

not specified how, but all tellers obtain the value s_i.

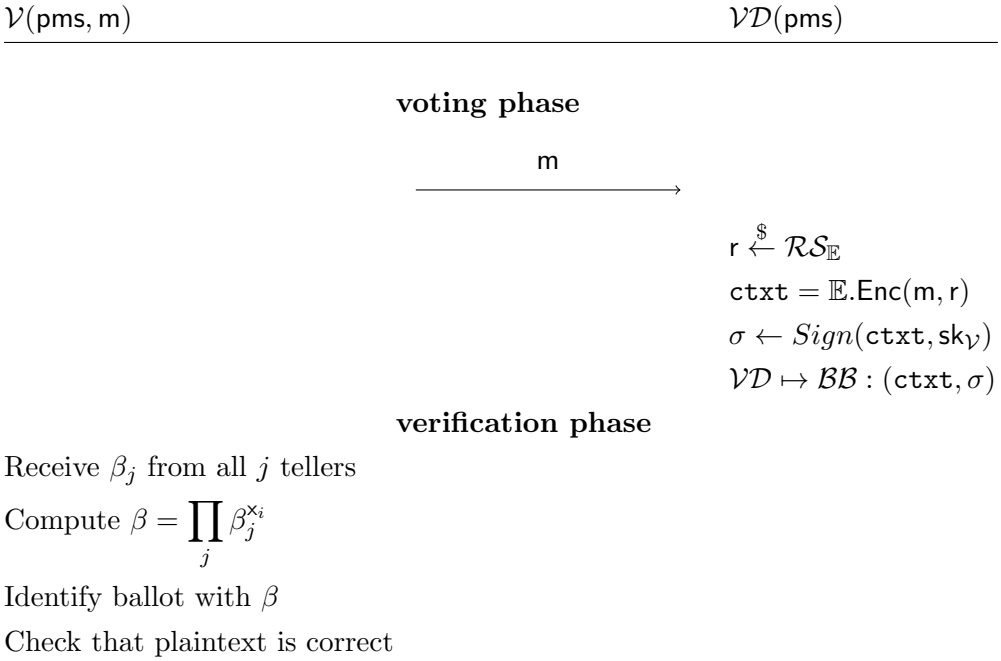


Figure A.7: The vote-casting and voter-verification of Hyperion voting system.

The voter can engage in cast-as-intended verification only after the results are published. For that, each teller j sends $\beta_j = \mathsf{g}^{\mathsf{r}_{ij} \cdot \mathsf{s}_i}$ over a private channel in a deniable fashion. Voters use their trapdoor key x_i to compute their beta term $(\prod_j \mathsf{g}^{\mathsf{r}_{ij} \cdot \mathsf{s}_i})^{\mathsf{x}_i}$ and identify their vote. To avoid coercion, the voter can fake some value $\mathsf{g}^{\mathsf{r}_{ij} \cdot \mathsf{s}}$ and make it open to any beta term it wishes.

To further strengthen coercion-resistance and prevent the coercer from claiming that some beta term belongs to him, Hyperion offers individual views of Bulletin Board BB for each voter. For that, mixed and decrypted ballots are grouped by the candidate, then for each voter, votes are again permuted and re-randomized within candidate groups. Therefore, each voter will receive their unique view, which makes it difficult for the coercer to recognize their beta term.

Overall, the cast-as-intended and recorded-as-cast verifiability rely on beta terms published along with decrypted votes. Therefore, the voter must be able to reconstruct their beta term. For that, *all* tellers should send their parts $\mathsf{g}^{\mathsf{r}_{ij}}$ to the voter; in other words, no teller is malicious and withholds the share. Moreover, the verification device should be honest. Otherwise, it can use knowledge of the trapdoor key to show a fake beta term for another selection.

Also, the adversary should not know the voter's trapdoor key. Otherwise, he can disguise himself as a teller and send a fake $\mathsf{g}^{\mathsf{r}^*_{ij}}$ term. Since the terms arrive through anonymous channels, the voter would not know the difference and will use a bogus value that will lead to a different beta term. Therefore, the voter's app should be trustworthy. The voter's app trustworthiness also ensures that all beta terms would be unique as public keys h_i would be random.

Bibliography

[1] D-demos: A distributed, end-to-end verifiable, internet voting system. In *Proceedings - 2016 IEEE 36th International Conference on Distributed Computing Systems, ICDCS 2016*, Proceedings - International Conference on Distributed Computing Systems, pages 711–720. Institute of Electrical and Electronics Engineers Inc., August 2016. Publisher Copyright: © 2016 IEEE.; 36th IEEE International Conference on Distributed Computing Systems, ICDCS 2016 ; Conference date: 27-06-2016 Through 30-06-2016.

[2] General framework of electronic voting and implementation thereof at national elections in estonia. Technical report, State Electoral Office of Estonia, 2017.

[3] Claudia Z. Acemyan, Philip Kortum, Michael D. Byrne, and Dan S. Wallach. Usability of voter verifiable, end-to-end voting systems: Baseline data for helios, prêt à voter, and scantegrity II. In *2014 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 14)*, 2014.

[4] Ben Adida. Helios v4 verification specs. In *Helios documentation*. Accessed: 2022-09-5.

[5] Ben Adida. Helios: Web-based open-audit voting. In *Proceedings of the 17th USENIX Security Symposium*, pages 335–348, 2008.

[6] Ben Adida and C. Andrew Neff. Ballot casting assurance. In *2006 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT 06)*, Vancouver, B.C., August 2006. USENIX Association.

[7] Ben Adida and C. Andrew Neff. Efficient receipt-free ballot casting resistant to covert channels. Cryptology ePrint Archive, Paper 2008/207, 2008. `https://eprint.iacr.org/2008/207`.

[8] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 99–108, New York, NY, USA, 1996. Association for Computing Machinery.

[9] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, page 284–293, New York, NY, USA, 1997. Association for Computing Machinery.

[10] Nabil Alkeilani Alkadri, Rachid El Bansarkhani, and Johannes Buchmann. On lattice-based interactive protocols: An approach with less or no aborts. In *Information Security and Privacy*, pages 41–61, Cham, 2020. Springer International Publishing.

[11] Joël Alwen, Rafail Ostrovsky, Hong-Sheng Zhou, and Vassilis Zikas. Incoercible multi-party computation and universally composable receipt-free voting. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 763–780. Springer, 2015.

[12] Roberto Araujo, Narjes Ben Rajeb, Riadh Robbana, Jacques Traoré, and Souheib Yousfi. Towards practical and secure coercion-resistant electronic elections. pages 278–297, 12 2010.

[13] Xavier Arnal, Abraham Cano, Tamara Finogina, and Javier Herranz. How to avoid repetitions in lattice-based deniable zero-knowledge proofs. Cryptology ePrint Archive, Paper 2022/803, 2022. `https://eprint.iacr.org/2022/803`.

[14] Thomas Attema and Serge Fehr. Parallel repetition of $(k_1, \ldots, k_\mu)$-special-sound multi-round interactive proofs. Cryptology ePrint Archive, Paper 2021/1259, 2021. `https://eprint.iacr.org/2021/1259`.

[15] Thomas Attema, Serge Fehr, and Michael Klooß. Fiat-shamir transformation of multi-round interactive proofs. *IACR Cryptol. ePrint Arch.*, page 1377, 2021.

[16] Rouzbeh Behnia, Yilei Chen, and Daniel Masny. On removing rejection conditions in practical lattice-based signatures. In *Post-Quantum Cryptography*, pages 380–398, Cham, 2021. Springer International Publishing.

[17] Josh Benaloh. Verifiable secret-ballot elections. PhD thesis, Yale University, 1987ß.

[18] Josh Benaloh. Ballot casting assurance via voter-initiated poll station auditing. In *USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'07*, 2007.

[19] Josh Benaloh. Rethinking voter coercion: The realities imposed by technology. In *Presented as part of the 2013 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections*, Washington, D.C., 2013. USENIX.

[20] David Bernhard, Oksana Kulyk, and Melanie Volkamer. Security proofs for participation privacy, receipt-freeness and ballot privacy for the helios voting scheme. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, page 1–10, 2017.

[21] Jens-Matthias Bohli, Jörn Müller-Quade, and Stefan Röhrich. Bingo voting: Secure and coercion-free voting using a trusted random number generator. In

Ammar Alkassar and Melanie Volkamer, editors, *E-Voting and Identity, First International Conference, VOTE-ID 2007, Bochum, Germany, October 4-5, 2007, Revised Selected Papers*, volume 4896 of *Lecture Notes in Computer Science*, pages 111–124. Springer, 2007.

[22] Jonathan Bootle, Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. More efficient amortization of exact zero-knowledge proofs for LWE. In *ESORICS 202*, volume 12973 of *Lecture Notes in Computer Science*, pages 608–627. Springer, 2021.

[23] Jonathan Bootle, Vadim Lyubashevsky, and Gregor Seiler. Algebraic techniques for short(er) exact lattice-based zero-knowledge proofs. In *Advances in Cryptology - CRYPTO 2019*, volume 11692 of *Lecture Notes in Computer Science*, pages 176–202. Springer, 2019.

[24] Ahto Buldas and Triinu Mägi. Practical security analysis of e-voting systems. In Atsuko Miyaji, Hiroaki Kikuchi, and Kai Rannenberg, editors, *Advances in Information and Computer Security*, pages 320–335, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[25] Philippe Bulens, Damien Giry, and Olivier Pereira. Running mixnet-based elections with helios. In *Proceedings of the 2011 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, EVT/WOTE'11, page 6, USA, 2011. USENIX Association.

[26] Jeffrey Burdges and Luca De Feo. Delay encryption. In *Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I*, pages 302–326. Springer, 2021.

[27] R Canetti and R Gennaro. Incoercible multiparty computation. pages 504 – 513, Novermber 1996.

[28] Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. Beleniosrf: A non-interactive receipt-free electronic voting scheme. Cryptology ePrint Archive, Report 2015/629, 2015. `https://eprint.iacr.org/2015/629`.

[29] Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. BeleniosRF: A non-interactive receipt-free electronic voting scheme. In *Proceedings of ACM SIGSAC Conference*, pages 1614–1625, 2016.

[30] David Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy Magazine*, 2:38–47, 2004.

[31] Jeremy Clark and Urs Hengartner. Selections: Internet voting with over-the-shoulder coercion-resistance. In *Proceedings of the 15th International Conference on Financial Cryptography and Data Security*, pages 47–61, 2011.

[32] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a secure voting system. In *2008 IEEE Symposium on Security and Privacy (S&P 2008)*, pages 354–368, 2008.

[33] Véronique Cortier, Jannik Dreier, Pierrick Gaudry, and Mathieu Turuani. A simple alternative to Benaloh challenge for the cast-as-intended property in Helios/Belenios. working paper or preprint, 2019.

[34] Véronique Cortier, Jannik Dreier, Pierrick Gaudry, and Mathieu Turuani. A simple alternative to Benaloh challenge for the cast-as-intended property in Helios/Belenios. working paper or preprint, 2019.

[35] Véronique Cortier, Alicia Filipiak, and Joseph Lallemand. BeleniosVS: Secrecy and verifiability against a corrupted voting device. In *32nd IEEE Computer Security Foundations Symposium, Hoboken, NJ, USA, June 25-28, 2019*.

[36] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Election verifiability for helios under weaker trust assumptions. In Mirosław Kutyłowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014*, pages 327–344, Cham, 2014. Springer International Publishing.

[37] Véronique Cortier, Pierrick Gaudry, and Quentin Yang. Is the JCJ voting system really coercion-resistant? working paper or preprint, April 2022.

[38] Véronique Cortier, Alicia Filipiak, and Joseph Lallemand. Beleniosvs: Secrecy and verifiability against a corrupted voting device. In *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, pages 367–36714, 2019.

[39] Jordi Cucurull, Adrià Rodríguez-Pérez, Tamara Finogina, and Jordi Puiggalí. Blockchain-based internet voting: Systems' compliance with international standards. In Witold Abramowicz and Adrian Paschke, editors, *Business Information Systems Workshops*, pages 300–312, Cham, 2019. Springer International Publishing.

[40] Ivan Damgård. On σ-protocols. `https://www.cs.au.dk/~ivan/Sigma.pdf/`, 2002.

[41] Scytl R&S David Galindo. sVote with Control Components Voting Protocol - Computational Proof of Complete Veriability. 2018.

[42] Olivier de Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of helios. In *Proceedings of EVT/WOTE Workshop*, 2009.

[43] S. Delaune, S. Kremer, and M. Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *19th IEEE Computer Security Foundations Workshop (CSFW'06)*, pages 12–42, 2006.

[44] Stéphanie Delaune, Steve Kremer, and Mark Ryan. *Verifying Privacy-Type Properties of Electronic Voting Protocols: A Taster*, pages 289–309. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[45] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the fiat-shamir transformation in the quantum random-oracle model. In *Advances in Cryptology - CRYPTO 2019*, volume 11693 of *Lecture Notes in Computer Science*, pages 356–383. Springer, 2019.

[46] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In George Robert Blakley and David Chaum, editors, *Advances in Cryptology*, pages 10–18, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

[47] Saghar Estehghari and Yvo Desmedt. Exploiting the client vulnerabilities in internet e-voting systems: Hacking helios 2.0 as an example. In *EVT/WOTE Workshop*, pages 1–9, 2010.

[48] Tamara Finogina. Designated cast-as-intended verification and universal proof of vote correctness from chameleon hashes. In *XVI RECSI*, volume 28, pages 101–106, 2021.

[49] Tamara Finogina and Javier Herranz. On remote electronic voting with both coercion resistance and cast-as-intended verifiability. *Journal of Information Security and Applications*, 76:103554, 2023.

[50] Tamara Finogina, Javier Herranz, and Enrique" Larraia. How (not) to achieve both coercion resistance and cast as intended verifiability in remote evoting. In *Cryptology and Network Security*, pages 483–491, 2021.

[51] Tamara Finogina, Adrià Rodríguez-Pérez, and Jordi Puiggalí. Dubious security practices in e-voting schemes. between tech and legal standards. 10 2022.

[52] Ashley Fraser, Elizabeth A. Quaglia, and Ben Smyth. A critique of game-based definitions of receipt-freeness for voting. In *Provable Security*, page 189–205, Berlin, Heidelberg. Springer-Verlag.

[53] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 33–62, Cham, 2018. Springer International Publishing.

[54] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory*, 31(4):469–472, 1985.

[55] Chaya Ganesh, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Fiat–shamir bulletproofs are non-malleable (in the algebraic group model). In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022*, pages 397–426, Cham, 2022. Springer International Publishing.

[56] Ryan W. Gardner, Sujata Garera, and Aviel D. Rubin. Coercion resistant end-to-end voting. In Roger Dingledine and Philippe Golle, editors, *Financial*

Cryptography and Data Security, pages 344–361, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[57] Oded Goldreich. *Foundations of Cryptography*, volume 1. Cambridge University Press, 2001.

[58] Sandra Guasch and Paz Morillo. How to challenge and cast your e-vote. In Jens Grossklags and Bart Preneel, editors, *Financial Cryptography and Data Security*, pages 130–145, 2017.

[59] Thomas Haines and Ben Smyth. Surveying definitions of coercion resistance. Cryptology ePrint Archive, Report 2019/822, 2019. `https://ia.cr/2019/822`.

[60] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols: Techniques and Constructions*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.

[61] James Heather and David Lundin. The append-only web bulletin board. In Pierpaolo Degano, Joshua Guttman, and Fabio Martinelli, editors, *Formal Aspects in Security and Trust*, pages 242–256, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[62] Sven Heiberg, Peeter Laud, and Jan Willemson. The application of i-voting for estonian parliamentary elections of 2011. In Aggelos Kiayias and Helger Lipmaa, editors, *E-Voting and Identity*, pages 208–223. Springer Berlin Heidelberg, 2012.

[63] Vincenzo Iovino, Alfredo Rial, Peter B. Rønne, and Peter Y. A. Ryan. Using selene to verify your vote in jcj. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y.A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson, editors, *Financial Cryptography and Data Security*, pages 385–403, Cham, 2017. Springer International Publishing.

[64] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In *Advances in Cryptology - EUROCRYPT '96*, pages 143–154. Springer, 1996.

[65] Wojciech Jamroga, Damian Kurpiewski, Vadim Malvone, Thomas Groß, and Luca Viganò. How to measure usable security: Natural strategies in voting protocols1. *J. Comput. Secur.*, 30(3):381–409, jan 2022.

[66] Rui Joaquim, Carlos Ribeiro, and Paulo Ferreira. Veryvote: A voter verifiable code voting system. In Peter Y. A. Ryan and Berry Schoenmakers, editors, *E-Voting and Identity*, pages 106–121, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[67] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Towards Trustworthy Elections, New Directions in Electronic Voting*, pages 37–63, 2010.

[68] Shuichi Katsumata. A new simple technique to bootstrap various lattice zero-knowledge proofs to QROM secure nizks. In *Advances in Cryptology - CRYPTO 2021*, volume 12826 of *Lecture Notes in Computer Science*, pages 580–610. Springer, 2021.

[69] A. Kiayias, T. Zacharias, and B. Zhang. An efficient e2e verifiable e-voting system without setup assumptions. *IEEE Security Privacy*, 15(3):14–23, 2017.

[70] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. Demos-2: Scalable e2e verifiable elections without random oracles. CCS '15, page 352–363, New York, NY, USA, 2015. Association for Computing Machinery.

[71] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. End-to-end verifiable elections in the standard model. *IACR Cryptology ePrint Archive*, 2015:346, 2015.

[72] Kristjan Krips and Jan Willemson. On practical aspects of coercion-resistant remote voting systems. 2019.

[73] Oksana Kulyk, Karola Marky, Stephan Neumann, and Melanie Volkamer. Introducing proxy voting to helios. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*, pages 98–106, 2016.

[74] Oksana Kulyk, Vanessa Teague, and Melanie Volkamer. Extending helios towards private eligibility verifiability. In Rolf Haenni, Reto E. Koenig, and Douglas Wikström, editors, *E-Voting and Identity*, pages 57–73, Cham, 2015. Springer International Publishing.

[75] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. A game-based definition of coercion-resistance and its applications. In *2010 23rd IEEE Computer Security Foundations Symposium*, pages 122–136, 2010.

[76] Enrique Larraia, Tamara Finogina, and Nuria Costa. svote with control components voting protocol. computational proof of complete verifiability and privacy. Cryptology ePrint Archive, Paper 2022/1509, 2022. `https://eprint.iacr.org/2022/1509`.

[77] Peter Hyun-Jeen Lee and Siamak F. Shahandashti. Theoretical attacks on e2e voting systems. Cryptology ePrint Archive, Report 2016/447, 2016. `https://ia.cr/2016/447`.

[78] Vadim Lyubashevsky. Basic lattice cryptography: Encryption and fiat-shamir signatures. `https://drive.google.com/file/d/1JTdW5ryznp-dUBBjN12QbvWz9R41NDGU/view`. Accessed: 2023-03-20.

[79] Vadim Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In *Advances in Cryptology – ASIACRYPT 2009*, pages 598–616, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

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

[81] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43:1–43:35, 2013.

[82] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-lwe cryptography. Cryptology ePrint Archive, Paper 2013/293, 2013. `https://eprint.iacr.org/2013/293`.

[83] Emmanouil Magkos, Mike Burmester, and Vassilios Chrissikopoulos. Receipt-freeness in large-scale elections without untappable channels. In *Towards The E-Society: E-Commerce, E-Business, and E-Government, The First IFIP Conference on E-Commerce, E-Business, E-Government (I3E 2001), October 3-5, Zürich, Switzerland*, volume 202 of *IFIP Conference Proceedings*, pages 683–693. Kluwer, 2001.

[84] Karola Marky, Oksana Kulyk, Karen Renaud, and Melanie Volkamer. What did i really vote for? on the usability of verifiable e-voting schemes. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2018.

[85] Karola Marky, Marie Laure Zollinger, Peter Roenne, Peter Y.A. Ryan, Tim Grube, and Kai Kunze. Investigating usability and user experience of individually verifiable internet voting schemes. *ACM Transactions on Computer-Human Interaction*, 28(5), October 2021.

[86] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006*, pages 373–392, 2006.

[87] C. Andrew Neff. Practical high certainty intent verification for encrypted votes, 2004.

[88] Tatsuaki Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Security Protocols Workshop*, 1997.

[89] Alisa Pankova and Jan Willemson. Relations between privacy, verifiability, accountability and coercion-resistance in voting protocols. Cryptology ePrint Archive, Report 2021/1501, 2021. `https://ia.cr/2021/1501`.

[90] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology - CRYPTO '91*, pages 129–140, 1991.

[91] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *J. Cryptol.*, 13(3):361–396, 2000.

[92] Swiss Post. Swiss post voting system.

[93] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), sep 2009.

[94] Stanislas Riou, Oksana Kulyk, and David Marcos del Blanco. A formal approach to coercion resistance and its application to e-voting. *Mathematics*, 10, 02 2022.

[95] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.

[96] Peter B. Rønne, Arash Atashpendar, Kristian Gjøsteen, and Peter Y. A. Ryan. Coercion-resistant voting in linear time via fully homomorphic encryption: Towards a quantum-safe scheme. *CoRR*, abs/1901.02560, 2019.

[97] Peter B. Rønne, Peter Y. A. Ryan, and Ben Smyth. Cast-as-intended: A formal definition and case studies. In *Financial Cryptography and Data Security. FC 2021 International Workshops - CoDecFin, DeFi, VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers*, volume 12676 of *Lecture Notes in Computer Science*, pages 251–262. Springer, 2021.

[98] Peter Ryan, Peter Rønne, and Vincenzo Iovino. Selene: Voting with transparent verifiability and coercion-mitigation. volume 9604, pages 176–192, 02 2016.

[99] Peter Y. A. Ryan, Simon Rastikian, and Peter B. Rønne. Hyperion: An enhanced version of the selene end-to-end verifiable voting scheme. In *Proceedings of E-Vote-ID*, pages 285–287. University of Tartu, 2021.

[100] Peter Y. A. Ryan, Peter B. Roenne, Dimiter Ostrev, Fatima-Ezzahra El Orche, Najmeh Soroush, and Philip B. Stark. Who was that masked voter? the tally won't tell! In Robert Krimmer, Melanie Volkamer, David Duenas-Cid, Oksana Kulyk, Peter Rønne, Mihkel Solvak, and Micha Germann, editors, *Electronic Voting*, pages 106–123, Cham, 2021. Springer International Publishing.

[101] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In Kaisa Nyberg, editor, *Advances in Cryptology — EUROCRYPT'98*, pages 1–16, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[102] Warren D. Smith. New cryptographic election protocol with best-known theoretical properties. Workshop on Frontiers in Electronic Election, 2005. `https://users.encs.concordia.ca/~clark/biblio/coercion/Smith%202005-1.pdf`.

[103] Ben Smyth. Athena: A verifiable, coercion-resistant voting system with linear complexity. Cryptology ePrint Archive, Report 2019/761, 2019. `https://eprint.iacr.org/2019/761`.

[104] Jacques Stern. A new identification scheme based on syndrome decoding. In Douglas R. Stinson, editor, *Advances in Cryptology — CRYPTO' 93*, pages 13–21, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[105] Swiss Post. Protocol of the Swiss Post Voting System. Computational Proof of Complete Verifiability and Privacy. Version 1.1.0. `https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/tree/master/Protocol`, 2022.

[106] Masoud Tabatabaei, Wojciech Jamroga, and Peter Ryan. Preventing coercion in e-voting: Be open and commit. 01 2013.

[107] Alexander H. Trechsel and Kristjan Vassil. Internet voting in estonia : a comparative analysis of four elections since 2005 : report for the council of europe. 2010.

[108] Dwight Tuinstra and Josh Benaloh. Receipt-free secret-ballot elections. In *STOC '94 Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, pages 544–553. Association for Computing Machinery, Inc., May 1994.

[109] Dominique Unruh. Post-quantum security of fiat-shamir. In *Advances in Cryptology - ASIACRYPT 2017*, volume 10624 of *Lecture Notes in Computer Science*, pages 65–95. Springer, 2017.

[110] Dominique Unruh and Jörn Müller-Quade. Universally composable incoercibility. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, pages 411–428, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[111] Rupeng Yang, Man Ho Au, Zhenfei Zhang, Qiuliang Xu, Zuoxia Yu, and William Whyte. Efficient lattice-based zero-knowledge arguments with standard soundness: Construction and applications. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, pages 147–175, 2019.