PhD-FSTM-2023-053
The Faculty of Science, Technology and Medicine

DISSERTATION

Defence held on 22/06/2023 in Esch-sur-Alzette

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Sevdenur BALOGLU
Born on 1 March 1990 in Meram (Turkey)

FORMAL VERIFICATION OF VERIFIABILITY IN E-VOTING PROTOCOLS

Dissertation defence committee

Prof. Dr Sjouke MAUW, dissertation supervisor
Professor, Université du Luxembourg

Prof. Dr Kristian GJOSTEEN
Professor, Norwegian University of Science and Technology

Prof. Dr Jun PANG, Chairman
Assistant Professor, Université du Luxembourg

Dr Veronique CORTIER
Research Director CNRS, LORIA

Prof. Dr Peter Y. A. RYAN, Vice Chairman
Professor, Université du Luxembourg

UNIVERSITY OF LUXEMBOURG

DOCTORAL THESIS

Formal Verification of Verifiability in E-Voting Protocols

Author:
Sevdenur BALOGLU

Supervisor:
Prof. Dr. Sjouke MAUW

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Computer Science

in the

Security and Trust of Software Systems
Interdisciplinary Centre for Security, Reliability and Trust

July 24, 2023

Declaration of Authorship

I, Sevdenur BALOGLU, declare that this thesis titled, "Formal Verification of Verifiability in E-Voting Protocols" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed: Sevdenur

Date: 24/07/2023

UNIVERSITY OF LUXEMBOURG

Abstract

Interdisciplinary Centre for Security, Reliability and Trust

Doctor of Computer Science

Formal Verification of Verifiability in E-Voting Protocols

by Sevdenur BALOGLU

Election verifiability is one of the main security properties of e-voting protocols, referring to the ability of independent entities, such as voters or election observers, to validate the outcome of the voting process. It can be ensured by means of formal verification that applies mathematical logic to verify the considered protocols under well-defined assumptions, specifications, and corruption scenarios. Automated tools allow an efficient and accurate way to perform formal verification, enabling comprehensive analysis of all execution scenarios and eliminating the human errors in the manual verification. The existing formal verification frameworks that are suitable for automation are not general enough to cover a broad class of e-voting protocols. They do not cover revoting and cannot be tuned to weaker or stronger levels of security that may be achievable in practice. We therefore propose a general formal framework that allows automated verification of verifiability in e-voting protocols. Our framework is easily applicable to many protocols and corruption scenarios. It also allows refined specifications of election procedures, for example accounting for revote policies.

We apply our framework to the analysis of several real-world case studies, where we capture both known and new attacks, and provide new security guarantees. First, we consider Helios, a prominent web-based e-voting protocol, which aims to provide end-to-end verifiability. It is however vulnerable to ballot stuffing when the voting server is corrupt. Second, we consider Belenios, which builds upon Helios and aims to achieve stronger verifiability, preventing ballot stuffing by splitting the trust between a registrar and the server. Both of these systems have been used in many real-world elections. Our third case study is Selene, which aims to simplify the individual verification procedure for voters, providing them with trackers for verifying their votes in the clear at the end of election. Finally, we consider the Estonian e-voting protocol, that has been deployed for national elections since 2005. The protocol has continuously evolved to offer better verifiability guarantees but has no formal analysis. We apply our framework to realistic models of all these protocols, deriving the first automated formal analysis in each case. As a result, we find several new attacks, improve the corresponding protocols to address their weakness, and prove that verifiability holds for the new versions.

Acknowledgements

Completing this Ph.D. thesis has been a transformative journey, and I am deeply grateful for the unwavering support and encouragement I have received from numerous individuals throughout this endeavour. I would like to express my sincere appreciation to all those who have played a significant role in shaping this work and making it a reality.

First and foremost, I extend my deepest gratitude to my esteemed supervisor, Sjouke Mauw, for his invaluable guidance and unwavering belief in my abilities. I am grateful for the research opportunity he provided me four years ago that opened a new chapter in my life. I would also like to thank Jun Pang for his support and contributions to this research.

I wish to extend my heartfelt gratitude to Sergiu Bursuc, whose unwavering guidance, invaluable insights, and mentorship have been pivotal in shaping the direction and quality of this research. His expertise and dedication to my intellectual growth have been instrumental, and without his cooperation, this research would not have been possible.

I sincerely thank the esteemed jury members, Peter Y A Ryan, Kristian Gjøsteen, and Véronique Cortier, for their invaluable insights, thoughtful discussions during the defence, and feedback. I am truly grateful for their time and dedication, which have contributed to the academic excellence of this work.

My appreciation also extends to my fellow researchers and colleagues in my research group SaToSS, who have been supportive companions throughout this academic journey. Their camaraderie and exchange of ideas have enriched my research experience and provided a strong sense of belonging within the academic community. Especially, I thank my colleagues Alex, Zach, Aoran, Semyon, Reynaldo, and Ninghan for providing a friendly and supportive environment, enjoyable moments, and enriching discussions over the years.

I am deeply grateful to my family for their unwavering love and constant support throughout my academic pursuit. Their encouragement and belief in me have been a constant source of strength during challenging times.

Lastly, I acknowledge the support of the Luxembourg National Research Fund (FNR) and the Research Council of Norway for providing financial assistance for this research under the joint INTER project SURCVS, ref. 11747298.

In conclusion, this Ph.D. thesis represents the collective efforts of numerous individuals who have touched my life in various ways. Their contributions, whether big or small, have shaped the person and researcher I have become. I express my heartfelt gratitude to all of them for their support, encouragement, and inspiration.

Contents

List of Figures

List of Tables

To my father

Part I

Preliminaries and Foundations

Chapter 1

Introduction

The verifiability of e-voting protocols plays a fundamental role in establishing the credibility of election outcomes. Verifiability refers to the ability of independent entities, such as voters, election observers, or auditors, to validate the correctness and accuracy of the voting process. It allows stakeholders to gain confidence that the final results faithfully represent the will of the voters. While verifiability is crucial, its implementation in e-voting protocols presents complex technical and cryptographic challenges. Traditional paper-based voting systems often rely on physical security measures, such as sealed ballot boxes, to assure the verifiability of the election process. In contrast, e-voting protocols must employ cryptographic primitives and rigorous mathematical techniques to provide verifiability in a digital environment, where threats such as tampering, coercion, and implementation errors pose significant risks.

Formal verification techniques are essential in assessing the correctness of security of protocols in general, and e-voting protocols in particular. By leveraging mathematical logic, formal verification allows us to reason rigorously about the behaviour of e-voting protocols and of a potential adversary, detect potential vulnerabilities, and provide proofs of their security properties. This enhances the trustworthiness and reliability of e-voting systems by subjecting them to rigorous analysis and validation. Considering the complexity of e-voting protocols, i.e. the cryptographic primitives employed, the multiple parties, and the presence of a strong adversary, automated verification offers advantages in terms of efficiency, completeness, and precision when applied to these protocols. Automated verification explores all scenarios, and supports reproducibility, ensuring the correctness and security of e-voting protocols against adversarial attacks.

There are several works in the literature to evaluate the security of e-voting protocols within formal verification frameworks. The majority of them target formal verification of privacy. Compared to privacy, the state-of-the-art for formal verification of election verifiability is somehow poorer. There is no general definition that can be applied to a wide range of protocols and scenarios and allows automated verification simultaneously. Thus, challenges remain in achieving comprehensive and accurate analyses to assess verifiability, considering various aspects of existing protocols. This thesis addresses these challenges by proposing a general formal verification framework that allows to specify different protocols, election procedures, and revoting policies.

By providing a comprehensive verification framework, this thesis aims to contribute to the advancement of secure and verifiable e-voting protocols. The research conducted in this thesis bridges the gap between theoretical analysis and practical implementation by considering real-world e-voting protocols, such as Helios, Belenios, Selene, and the Estonian e-voting protocol, as case studies. The findings from the verifiability analyses of these protocols within the proposed framework will contribute to the body of knowledge in the field and inform the design and improvement of future e-voting systems.

1.1 State of the art of formal methods for verifiability

Verifiability [9, 10, 2, 37, 15] has emerged as one of the main requirements for the security of the e-voting protocols. At first, this notion aimed to ensure the election outcome corresponds to the votes cast by voters, and providing ballot casting assurance for voters, i.e. if the voters verify their vote, then the vote should be counted for them. These two notions of verifiability have been referred to as universal and individual verifiability, respectively. Subsequently, more general end-to-end notions have been introduced. Another notion of verifiability was introduced [38] to ensure the eligibility of the voters, i.e. votes to be counted in the result should have been cast by eligible voters, and there should be at most one vote per voter. Those three notions of verifiability entail end-to-end verifiability if the protocol enables voters to verify their votes and anyone to verify the votes in the outcome corresponds to the votes cast by eligible voters.

The first general symbolic definition [38] formalises individual, universal, and eligibility verifiability as a triple of boolean tests. The definition was applied to the protocols Helios [1], FOO [28], and Civitas [16], proving its generality. However, the definition requires all honest voters to verify their votes, which is not realistic. Also, the boolean tests cannot be expressed with the tools that allow automated verification, such as Tamarin and ProVerif. Another approach is proposed by the type-based symbolic definition [22], which formalises individual, universal, and end-to-end election verifiability as logical formulas to be checked by a type-checker. It also defines a formula to ensure that there are no clash attacks on the protocol and proves that these properties entail end-to-end verifiability. As a case study, Helios was modelled and proved secure with respect to end-to-end verifiability. However, the definition does not cover the notion of eligibility. It also does not capture revoting and is not suitable for automated verification tools like Tamarin and ProVerif.

We focus on the two recent formal definitions of election verifiability aimed to guarantee end-to-end verifiability: one from [20] is based on the classification of the multisets of votes in the outcome, and the other from [17] is a symbolic definition based on logical formulas (can be applied in Tamarin/ProVerif), and it implies the notion of multiset-based definition in [20]. According to the multiset-based verifiability definition, an e-voting protocol is verifiable if the votes in the outcome corresponds to the union of three multisets of votes: 1) the votes that have been cast and verified by honest voters; 2) a subset of votes that have been cast by honest voters but have not been verified; and 3) a number of votes bounded by the number of corrupt voters. On the other hand, the symbolic definition [17] is formulated corresponding to the notions of recorded-as-intended, individual verifiability, and eligibility and applied to BeleniosVS, a variant of Belenios. The definition in [17] varies according to the trust assumptions of Belenios-like protocols. Thus, two definitions were proved to imply end-to-end verifiability according to the multiset-based definition. These symbolic definitions allow automated verification. However, the approach in [17] is not generic due to the definition varying according to the trust assumptions. It also does not allow revoting or dynamic corruption of the voters. This means it cannot be directly applied to the protocols allowing revoting or having a different architecture, parties, and infrastructure components, i.e. it is not general.

Next, we present the existing work on formal verifiability of our case studies. Helios [1], the first web-based e-voting protocol providing end-to-end verifiability, has been extensively analysed in the literature; thus, it is representative of the state-of-the-art. Among those analyses, the three [38, 22, 40] focus on its verifiability in the symbolic model. The model in [22] was proven to provide end-to-end verifiability with type-checkers. However, none of those models enable revoting as a feature of Helios. Belenios [20, 18] has built upon Helios to provide stronger end-to-end verifiability with weaker trust assumptions. There are two verifiability analyses [20, 21] of Belenios in its computational model, i.e. the first provides a manual

proof, and the second does a machine-checked proof with EasyCrypt. Also, the model in [21] enables revoting. However, there is no verifiability analysis regarding its symbolic model. Instead, another symbolic analysis was performed for BeleniosVS, a variant of Belenios, in which BeleniosVS was proved to be secure with automated verification.

Selene [47] is an e-voting protocol that aims to provide both end-to-end verifiability and receipt-freeness; thus, it aims to achieve the ultimate goal of the e-voting protocols, contributing to the state-of-the-art. The analyses [36, 13] performed for Selene focus only on its privacy. Finally, the Estonian e-voting protocol has been deployed for the national elections since 2005. The first versions of the protocol did not enable individual verifiability for voters or universal verifiability for external auditors. Individual verifiability [32] was introduced in 2013, whereas universal verifiability [33] was presented to data auditors with a system design change in 2016. In 2022, two attacks [44, 43] were proposed against its verifiability and privacy. There is only one analysis [49] to the best of our knowledge, which evaluates the protocol's security in terms of practical aspects. However, the protocol has not yet been formally or symbolically analysed, making it a good target for us to apply our framework.

Summary of limitations:

- The first general symbolic definition [38] formalises verifiability with the boolean tests that cannot be expressed with the tools that allow automated verification, such as Tamarin and ProVerif. Moreover, it assumes all honest voters verify their votes, which is not realistic.
- The type-based symbolic definition [22] formalises verifiability as logical formulas corresponding to individual, universal, and end-to-end election verifiability, where it misses the notion of eligibility. It does not capture revoting and is not suitable for automated verification.
- The symbolic definition in [17] allows automated verification, but it is not general enough: it varies under two different trust assumptions, it does not allow revoting and dynamic corruption of the voters, and it is too specifically focusing on Belenios.
- There is no symbolic analysis of verifiability that allows revoting.
- There is no symbolic verifiability analyses for Belenios, Selene, and the Estonian e-voting protocol.
- There is no formal analysis for the stronger corruption scenarios in Helios, when the server is corrupt, or in Belenios when both the registrar and server are corrupt.

1.2 Contributions

In this thesis, our main contribution is providing a general formal verification framework for election verifiability that allows automated verification, accounts for revoting, covers a strong adversary with extended capabilities, captures verifiability attacks in the literature, e.g. clash attacks or ballot stuffing, and any unknown potential attack, e.g. arising from revoting. Our framework applies to a broad class of e-voting protocols, as we provide case studies from real-world e-voting protocols, such as Helios, Belenios, Selene, and the Estonian e-voting protocol. Thus, the second contribution is the verifiability analyses of those protocols with the framework we provide. We elaborate on our contributions as follows:

1. We extend the multiset-based election verifiability proposed by Cortier et al. [20] to obtain weaker or stronger notions of end-to-end verifiability according to what may be

possible in various scenarios or protocols. Their definition covers individual verifiability only for honest voters and limits the adversary to cast votes for only corrupt voters. Some protocols may guarantee that the votes from corrupt voters are counted in the tally if verified. For some protocols, the adversary may cast ballots also for the honest voters who did not verify their votes. Thus, we define four notions of end-to-end verifiability by combining two notions of weak or strong individual verifiability and result integrity.

2. We improve the symbolic election verifiability presented in [17] by Cortier et al. and propose a symbolic definition that is independent of trust assumptions, i.e. it does not vary according to the corruption scenarios as in [17], accounts for revoting, captures verifiability attacks, and allows automated verification. Thus, it is generic, general, and applicable to many protocols. It also allows verifying stronger and weaker notions of end-to-end verifiability. We prove that this definition is sound, i.e. it entails the multiset-based end-to-end verifiability definition.

3. We provide realistic symbolic models for case studies in which:

   (a) We consider a bulletin board that publishes the verifiable election data, as in many real-world protocols. We model the individual verification procedure of the voters using the information published on the bulletin board and any additional information they may receive from election parties.

   (b) We specify each election procedure according to the setup, voting, and tally phases, as described in the considered protocols, or in the variants that we propose.

   (c) We consider a stronger adversary capable of corrupting the communication network and any party in the protocol, i.e. voters, talliers, registrar, voting server, and voting platforms. We model corrupt voters as they can be corrupted anytime in the protocol, not at the beginning of the election, i.e. we allow dynamic corruption of voters. We allow other parties to be fully corrupt unless the information they provide is subjected to public verification. For example, we allow a corrupt server to accept and record any ballot on the bulletin board. However, since the validity of the ballot can be verified on the bulletin board, we ensure the ballots recorded on the bulletin board are valid with some restrictions.

   (d) If revoting is allowed by the protocol, we model it without any bound on the number of ballots. Typically, the last ballot cast is tallied, but more complex policies could also be expressed in our framework.

4. We perform a verifiability analysis for each case study considering a realistic model with the extended abilities of the adversary. Our analysis is based on the verification results obtained with Tamarin/ProVerif. Our findings for each case are as follows:

   (a) For Helios and Belenios, we consider four individual verification procedures, such as an individual verification procedure that allows voters to verify during the voting phase or another that allows it in the tally phase. We evaluate the verifiability of those protocols with respect to different individual verification procedures. We find that while they are secure if verification is allowed in the tally phase, they do not provide verifiability with the procedure allowing verification at any time during the voting phase, even if this is the procedure allowed by the protocol in practice.

   (b) In Helios and Belenios, when revoting is allowed, we capture new versions of clash attacks that are exploited only by the corrupt registrar. In the original scenario of clash attacks, the voting server and voting platforms should be corrupt in addition to the registrar.

(c) For Helios and Belenios, we provide the first automated proofs of verifiability when both the registrar and server are corrupt and individual verification is allowed in the tally phase. The proof is produced with respect to our notion of weak result integrity, i.e. the adversary is limited to casting ballots for corrupt voters and also for honest voters that did not verify their votes.

(d) Even assuming its prescribed trust assumptions, we find new attacks in Belenios when revoting is allowed and individual verification is performed during the voting phase. More precisely, our attacks exploit the corrupt network and/or the corrupt registrar while the server is honest. We propose solutions for Belenios in order to resist found attacks, and we introduce new variants of Belenios corresponding to those solutions. We prove that the variant so-called Belenios+ provides end-to-end verifiability with respect to the trust assumptions of Belenios.

(e) We analyse the verifiability of BeleniosRF, a receipt-free variant of Belenios, to evaluate the trade-off between receipt-freeness and verifiability. We find new verifiability attacks against BeleniosRF, implying that even if BeleniosRF strengthens the privacy of Belenios, it weakens its verifiability.

(f) We perform verifiability analysis for Selene, which is another voting protocol that provides both end-to-end verifiability and receipt-freeness. We prove that Selene provides end-to-end verifiability for honest voters if they verify their votes. On the other hand, we observe that Selene's receipt-freeness is not as strong as BeleniosRF; however, it can be modified to provide the same level of receipt-freeness with BeleniosRF in a variant so-called SeleneRF. We prove that SeleneRF provides the same verifiability guarantees as Selene. Thus, in the case of SeleneRF, receipt-freeness does not weaken end-to-end verifiability. However, we should note that usability of the receipt-freeness of Selene and SeleneRF is weaker, especially in presence of a vote buyer: while SeleneRF requires the voter to actively construct a fake receipt, this is not needed in BeleniosRF.

(g) Hyperion is a variant of Selene that simplifies some procedures, employing different cryptographic primitives. We prove that Hyperion provides similar end-to-end verifiability guarantees as Selene.

(h) In the analysis of the Estonian e-voting protocol, we capture the individual verifiability attack proposed by Pereira in [44] and new attacks. We also discover that the protocol is vulnerable to ballot copying attacks by checking standard notion of privacy in our model. Thus, we propose two solutions that prevent verifiability and ballot copying attacks. Our solutions require different procedures by the voters when they perform their individual verification. One of our solutions is more usable, and the other is more secure with respect to verifiability. Therefore, we show that there is a trade-off between usability and verifiability.

We note that even though there are security analyses in the literature for Helios and Belenios in the computational model, we present the first symbolic verifiability analysis of Belenios. Similarly, we provide the first formal symbolic analysis for BeleniosRF, Selene, Hyperion, and Estonian e-voting protocol.

Thesis structure: This thesis includes two parts, in which the first is dedicated to preliminaries and foundations for formal verification of election verifiability, and the second is to the case studies, i.e. Helios, Belenios, Selene, and the Estonian e-voting protocols. The first part has four chapters, including the introduction, whereas the second has the remaining five, including the conclusion. We summarise the chapters as follows:

- *Chapter 1:* First, we have introduced the general concept of e-voting protocols and the formal verification of election verifiability as one of the main security goals. Then, we

discussed the existing verifiability definitions and verifiability analyses in the literature. Finally, we have summarised our contributions provided in this thesis.

- *Chapter 2:* We delve into the verifiable e-voting protocols, i.e. we present their components in terms of cryptographic primitives used, the election parties and their roles, the election procedures and individual verification procedures followed in the protocol. Then, we discuss their security properties and present an example of a verifiable e-voting protocol.

- *Chapter 3:* We provide the formal verification preliminaries with respect to Tamarin and ProVerif. We also give the Tamarin and ProVerif specifications of the verifiable e-voting example presented in Chapter 2.

- *Chapter 4:* We discuss the informal notions and existing definitions of election verifiability, and their weaknesses. Then, we introduce formal and generic e-voting events, and use them to improve the existing definitions. The main result of this chapter is our symbolic definition of election verifiability. We also provide proof of soundness for our definition and illustrate its application to the verifiable e-voting example presented in Chapter 2. The results of this chapter are based on our two papers [6, 5].

  Case studies are given in Chapter 5, Chapter 6, Chapter 7, and Chapter 8.

- *Chapter 5:* We present the protocol structure, parties, and election procedures of Helios, including its individual verification procedure. Then, we provide its Tamarin specification, verification and our verifiability analysis, including security proofs and attacks found. Our analysis is present in [6].

- *Chapter 6:* We present the Belenios protocol, its Tamarin specification, verification, and analysis, as described for Helios above. Then, we improve Belenios with respect to the found attacks and propose a variant of Belenios that we call Belenios+. We prove that Belenios+ is resistant to all such attacks, and more generally we derive that it satisfies end-to-end verifiability, by combining the positive results of Tamarin with our soundness theorem. However, we should note that we performed an abstraction in order to make Tamarin terminate for Belenios+: we accept at most four ballots cast per voter. The main difficulty for Tamarin comes from the fact that Belenios+ subsequently links the ballots from the same voter to each other to protect against ballot reordering. Finally, we analyse the verifiability of BeleniosRF, a variant of Belenios providing receipt-freeness, and evaluate the trade-off between receipt-freeness and verifiability. The analyses that we performed regarding Belenios are published in [6, 7, 5].

- *Chapter 7:* We present the Selene, SeleneRF, and Hyperion protocols, their ProVerif specifications, verifications, and analyses. Then, we discuss their verification results, comparing their security and features. Our analyses are present in [5].

- *Chapter 8:* We present the Estonian e-voting protocol, its Tamarin specification, verification, and analysis. Then, we improve both the verifiability and privacy of the protocol with respect to the presented attacks, and we propose two variants of the Estonian e-voting protocol, where they differ in usability and practicality of the protocol.

- *Chapter 9:* We summarise our contributions and interpret our findings with respect to the case studies. Then, we pose open questions related to the future of e-voting and the practicality of the proposed solutions in the thesis.

Chapter 2

Verifiable E-Voting Protocols

E-voting protocols are complex security protocols based on several cryptographic primitives, including several protocol parties instead of two or three. Usually, the parties are categorised into two: the election authorities that organise and maintain the election, such as administrator, registrar, server and talliers, and the election attendees that are voters and their voting platforms. Also, the communication between those parties in the e-voting protocols can continue throughout the election, which may last from hours to a few days.

E-voting protocols are aimed to be used in real-world governmental elections because of their efficiency in collecting and tallying the votes, their availability to remote voters, and their numerous other advantages. However, achieving their security in an environment harbouring an adversary that intrudes on the communication or corrupts the protocol parties is really hard. Therefore, many countries, such as Germany and Norway, have given up using e-voting protocols for their governmental elections, while only a few countries, such as Estonia and Switzerland, still deploy and continuously improve them. On the other hand, e-voting protocols are widely deployed in low-stake elections, such as organisational elections to elect the organisation's president and vice president since those elections raise fewer security concerns than governmental elections about the outcome.

To extend the deployment of e-voting protocols widely for any election, they should be usable by people independent of their age and abilities, and their security should be theoretically and practically proved. Regarding e-voting, there is a consensus on two main security properties: privacy and verifiability. Privacy refers to the secrecy of the vote, i.e. no one knows how a voter voted in a particular way. On the other hand, verifiability refers to the ability of independent entities, such as voters, election observers, or auditors, to validate the election outcome. Throughout the chapter, the verifiable e-voting protocols will be detailed regarding their underlying cryptographic primitives, protocol parties, phases, and security properties.

Structure of the chapter. This chapter includes five sections. In Section 2.1, we present the cryptographic primitives employed by the e-voting protocols. In Section 2.2 and Section 2.3, we describe typical parties in an e-voting protocols along with their roles, and the election phases that separates the communication into three parts. In Section 2.4, we present the main security properties of the e-voting protocols. Finally, in Section 4.6, we provide examples for verifiable e-voting protocols.

2.1 Cryptographic Primitives

The way of communication between two distant parties has left its place from postal services to the internet, and thus, from envelopes carrying a message with sender-receiver information to encryptions and digital signatures. Similarly, the paper ballots being put in the envelopes in traditional voting have been replaced with digital ballots in e-voting that include the encryption of the vote and the digital signature of the voter who cast it. Other primitives like zero-knowledge proofs and commitments are used to ensure that the digital ballot has not

been altered until it is recorded on behalf of the voter. For the tally, sometimes, mixnets are deployed to anonymise the ballots against privacy violations. In this section, we briefly summarise the cryptographic primitives used in e-voting protocols.

2.1.1 Encryption Algorithms

For democratic elections, the voters should be able to vote with their will for their preferred choices, and this vote should be secret against vote buying and other types of coercion. Secrecy is directly provided for voters in traditional voting by preparing a ballot, i.e. covering the vote with an envelope, in a private voting booth, and then leaving the ballot into a ballot box, which anonymises the ballot among all other kinds and leaves no way to prove the content of the ballot. However, in e-voting, voters cast a digital ballot containing the vote that is digitally collected in a ballot box.

Digital environments are vulnerable to intrusions by the adversary, which may leave little evidence and be hard to discover. To provide security in digital environments, technical methods have been developed and deployed. The method being used for confidentiality is encryption. Encryptions hide data with a key and allow only those with the key to retrieve it. Thus, even if the adversary obtains the encrypted data with an intrusion, they cannot reveal the data without the key. In e-voting, the data being forwarded to the digital environment is the vote, which is only intended for the election authorities who count it, and thus, it requires confidentiality.

Encryptions have been used historically to hide a message content from unintended people. Ancient methods were using ciphers based on alphabet manipulations, yet modern methods use encryption algorithms based on complex mathematical operations on bits. The encryption algorithms are divided into two categories: symmetric and asymmetric encryption algorithms. Symmetric encryption requires to encrypt and decrypt data with one singular secret key. On the other hand, asymmetric encryption requires two keys: a private key is used for the decryption, and a public key used for the encryption.

Symmetric encryption: Let m be a message to be encrypted and k be the key provided for the encryption. A symmetric encryption algorithm generates a ciphertext c as an output, given the input of the message m and the key k. Assume the symmetric encryption algorithm utilises the function enc, then the ciphertext will be generated as:

$$\mathsf{c} = \mathsf{enc}(\mathsf{m}, \mathsf{k}).$$

In symmetric encryption, the same key, i.e. the encryption key k, is also used for the decryption. Assume the decryption algorithm uses the function dec, then the decryption will extract the original message as follows:

$$\mathsf{m} = \mathsf{dec}(\mathsf{c}, \mathsf{k}) = \mathsf{dec}(\mathsf{enc}(\mathsf{m}, \mathsf{k}), \mathsf{k}).$$

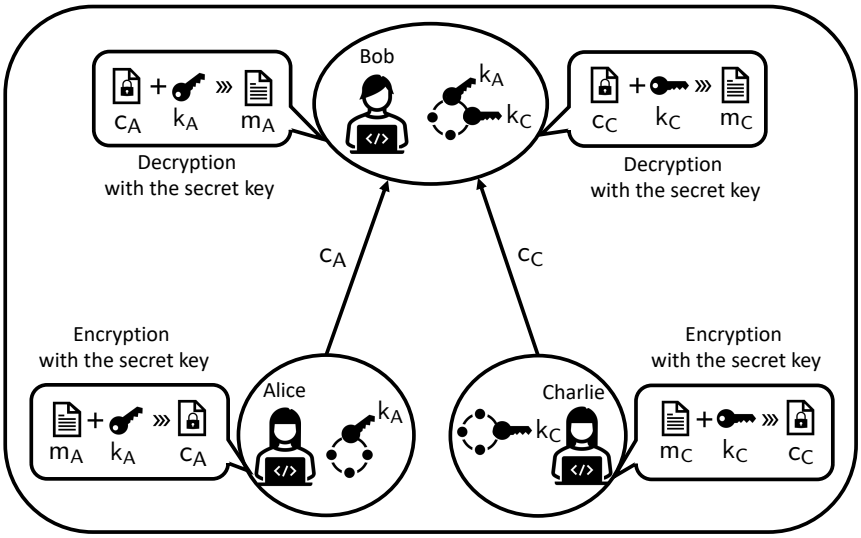
Asymmetric encryption: Let m be a message to be encrypted and $(\mathsf{k}, \mathsf{pk}(\mathsf{k}))$ be the key pair, where k is the private key and $\mathsf{pk}(\mathsf{k})$ is the public key generated from k. Then, any asymmetric encryption algorithm uses the public key $\mathsf{pk}(\mathsf{k})$ to generate a ciphertext c that encrypts the message m. Assume the algorithm utilises the function aenc for the encryption, then the ciphertext will be generated as:

$$\mathsf{c} = \mathsf{aenc}(\mathsf{m}, \mathsf{pk}(\mathsf{k})).$$

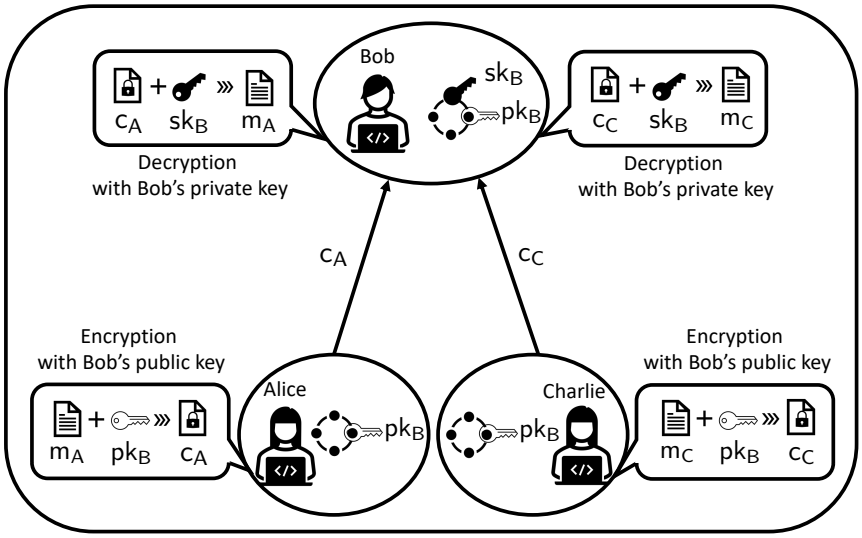
For the decryption of the message, the private key k is used. Assume the algorithm utilises the function adec with the key k. Then, the decryption will extract the original message as follows:

$$\mathsf{m} = \mathsf{adec}(\mathsf{c}, \mathsf{k}) = \mathsf{adec}(\mathsf{aenc}(\mathsf{m}, \mathsf{pk}(\mathsf{k})), \mathsf{k}).$$

The Figure 2.1 depicts the symmetric encryption/decryption on the left side whereas the asymmetric encryption/decryption on the right side. In both figures, the encryption is performed by Alice and Charlie, and Bob, upon receiving their encrypted messages, performs the decryption of the messages. In Figure 2.1a, Bob shares a secret key k_A with Alice and another k_C with Charlie. Then, Alice encrypts her message m_A with the secret key k_A, obtaining the ciphertext c_A. Similarly, Charlie uses k_C to encrypt her message m_C and obtains c_C. They both send their ciphertexts to Bob. After receiving, Bob decrypts them using the secret keys kept on the keychain. On the other hand, Bob generates a key pair (pk_B, sk_B), pk_B of which is published to be used for encryption in Figure 2.1b. Alice and Charlie adds pk_B to their keychains and use it to encrypt their messages m_A and m_C. Then, they send the obtained ciphertexts c_A and c_C to Bob so that he decrypts them using his private key sk_B to extract the original messages.



(A) Symmetric Encryption/Decryption.



(B) Asymmetric Encryption/Decryption.

FIGURE 2.1: Symmetric Encryption vs. Asymmetric Encryption.

As can be realised, the receiver has to manage a key for each sender in symmetric encryption, which is not efficient to be used in communication. Therefore, the first asymmetric encryption algorithm RSA was proposed in 1977 by Rivest, Shamir and Adleman, enabling an

efficient encryption algorithm for secure communication. The symmetric encryption, instead, is being used for encryption of large data, for which it is faster than asymmetric encryption algorithms, making it preferable. In e-voting, the encrypted data is transferred between voters and the election authorities. In this case, asymmetric encryption is much more convenient since one key pair can be used securely for all encryption and decryption operations. The most popular asymmetric encryption algorithms are ElGamal and Pallier, following RSA. In many e-voting protocols, ElGamal cryptosystem is preferred since it randomises the ciphertext with a randomness and allows homomorphic encryption, in which the ciphertexts generated by the voters are combined and transformed into one ciphertext, helping voters with their privacy in the protocol.

ElGamal encryption [27]: Let m be a message to be encrypted and $(\mathsf{k}, \mathsf{pk}(\mathsf{k}))$ be the key pair, where k is the private key and $\mathsf{pk}(\mathsf{k})$ is the public key generated from k. Then, the ElGamal encryption algorithm uses the public key $\mathsf{pk}(\mathsf{k})$ with a freshly generated randomness r to generate a ciphertext c for the input of the message m. Assume the function aenc denotes the encryption algorithm. Then, the ciphertext is obtained as follows:

$$\mathsf{c} = \mathsf{aenc}(\mathsf{m}, \mathsf{pk}(\mathsf{k}), \mathsf{r}).$$

Assume the function adec decrypts the ciphertext c with the private key k. Then, the message is revealed as follows:

$$\mathsf{m} = \mathsf{adec}(\mathsf{c}, \mathsf{k}) = \mathsf{adec}(\mathsf{aenc}(\mathsf{m}, \mathsf{pk}(\mathsf{k}), \mathsf{r}), \mathsf{k}).$$

Mathematically, the algorithm uses a generator g of a group of prime order q. The private key k is randomly chosen from $\mathbb{Z}_\mathsf{q}$, and the public key corresponds to $\mathsf{pk}(\mathsf{k}) = \mathsf{g}^\mathsf{k}$. For a randomness $\mathsf{r} \in \mathbb{Z}_\mathsf{q}$, the message m is encrypted as follows:

$$\mathsf{aenc}(\mathsf{m}, \mathsf{pk}(\mathsf{k}), \mathsf{r}) = (\mathsf{g}^\mathsf{r}, \mathsf{m} \cdot (\mathsf{g}^\mathsf{k})^\mathsf{r}).$$

Given the ciphertext $\mathsf{c} = (\mathsf{a}, \mathsf{b})$, the message m is revealed with the key k by computing $\mathsf{b}/\mathsf{a}^\mathsf{k}$, i.e.

$$\mathsf{m} = \frac{\mathsf{m} \cdot (\mathsf{g}^\mathsf{k})^\mathsf{r}}{(\mathsf{g}^\mathsf{r})^\mathsf{k}}.$$

ElGamal homomorphic encryption in e-voting: Let a vote v be encoded as an element in a small subset of $\mathbb{Z}_\mathsf{q}$. Then, the encryption of the vote v will be

$$\mathsf{c} = \mathsf{aenc}(\mathsf{v}, \mathsf{pk}(\mathsf{k}), \mathsf{r}) = (\mathsf{g}^\mathsf{r}, \mathsf{g}^\mathsf{v} \cdot (\mathsf{g}^\mathsf{k})^\mathsf{r}),$$

for some randomness $\mathsf{r} \in \mathbb{Z}_\mathsf{q}$. The ciphertext c is decrypted using the private key k as explained above, and the term g^v is revealed. To retrieve the vote v from g^v, the discrete logarithm algorithm is used in the small subset of $\mathbb{Z}_\mathsf{q}$. Thus, we have:

$$\mathsf{v} = \mathsf{adec}(\mathsf{c}, \mathsf{k}) = \mathsf{adec}(\mathsf{aenc}(\mathsf{v}, \mathsf{pk}(\mathsf{k}), \mathsf{r})).$$

Assume c_1 and c_2 are the two ciphertexts obtained encrypting the votes v_1 and v_2, i.e.

$$\mathsf{c}_1 = \mathsf{aenc}(\mathsf{v}_1, \mathsf{pk}(\mathsf{k}), \mathsf{r}_1) = (\mathsf{g}^{\mathsf{r}_1}, \mathsf{g}^{\mathsf{v}_1} \cdot (\mathsf{g}^\mathsf{k})^{\mathsf{r}_1}),$$
$$\mathsf{c}_2 = \mathsf{aenc}(\mathsf{v}_2, \mathsf{pk}(\mathsf{k}), \mathsf{r}_2) = (\mathsf{g}^{\mathsf{r}_2}, \mathsf{g}^{\mathsf{v}_2} \cdot (\mathsf{g}^\mathsf{k})^{\mathsf{r}_2}),$$

for some randomness r_1 and r_2. The ciphertexts c_1 and c_2 can be homomorphically multiplied to obtain a ciphertext c:

$$\begin{aligned} c = c_1 \cdot c_2 &= \mathsf{aenc}(v_1, \mathsf{pk}(k), r_1) \cdot \mathsf{aenc}(v_2, \mathsf{pk}(k), r_2) \\ &= (g^{r_1}, g^{v_1} \cdot (g^k)^{r_1}) \cdot (g^{r_2}, g^{v_2} \cdot (g^k)^{r_2}) \\ &= (g^{r_1+r_2}, g^{v_1+v_2} \cdot (g^k)^{r_1+r_2}) \\ &= \mathsf{aenc}(v_1 + v_2, \mathsf{pk}(k), r_1 + r_2), \end{aligned}$$

corresponding to the sum of votes $v_1 + v_2$:

$$v_1 + v_2 = \mathsf{adec}(c, k) = \mathsf{adec}(\mathsf{aenc}(v_1 + v_2, \mathsf{pk}(k), r_1 + r_2), k).$$

2.1.2 Digital Signatures

In traditional voting, the authorities in the polling stations lead voters to the voting booth after checking their eligibility and follow them until they cast their ballot. In this way, they can ensure that the ballot was cast by an eligible voter, and thus all the ballots in a ballot box. However, in e-voting, a digital ballot box collects the ballots from voters through the digital environment, e.g. through the internet, which gives rise to adversarial involvement. In this case, the ballot to be counted should be ensured that it comes from an eligible voter, and the integrity of the ballot should be satisfied, proving that the ballot has not been changed during the submission. For these guarantees, digital signatures may be attached to the ballots of the voters. Digital signatures provide authentication, non-repudiation and integrity in digital environments.

If the digital signatures are sometimes used in e-voting, each voter typically owns a valid signature key pair, which is similar to the key pair in asymmetric encryption, i.e. a private key for signing the message and a public key for the verification of the signature. The voters use their private key to sign the ciphertext corresponding to the encrypted vote so that the signature proves the ciphertext is cast by the voter who owns the private key. In this way, the voter cannot deny casting the ballot with the attached signature, providing non-repudiation. The election authorities that receive the ballot from the voter, verify the signature using the public key of the voter, which proves that the ballot has not changed after it is signed, and thus, provides integrity.

Digital signatures are often used in a Public Key Infrastructure (PKI) which means that each entity owning a signature key pair should be certified by a trusted Certification Authority (CA). Thus, the receiver of the signature can ensure that the public key of the sender is valid, not revoked or corrupted, with the certificate provided by CA. Otherwise, the signature key pair does not provide authentication, i.e. it could be generated by the adversary.

Digital signature: Let m be the message to be signed by a sender A who owns the signature key pair $(\mathsf{sk}_A, \mathsf{pk}_A)$, where the public key pk_A is generated from the private key sk_A. The sender A signs the message m using a function sign as follows:

$$s = \mathsf{sign}(m, \mathsf{sk}_A),$$

where s the signature of the message m. Then, A sends the tuple $\langle m, s \rangle$ to the receiver B who can verify the signature s with pk_A:

$$\mathsf{verify}(s, \mathsf{pk}_A) = \mathsf{verify}(\mathsf{sign}(m, \mathsf{sk}_A), \mathsf{pk}_A) = \mathsf{true}.$$

It may also check that pk_A has a valid certificate, or has been provided by a trusted authority.

2.1.3 Zero-Knowledge Proofs

Zero-knowledge proofs are used by a party to prove some knowledge or statement to another party without revealing any detail about the secret part of the knowledge. The prover develops/follows a strategy to prove their knowledge, making the verifier totally convinced about the knowledge since it is impossible for the prover to fake in the end of strategy. A zero-knowledge proof can be interactive or non-interactive. The interactive ones requires an interaction between the prover and the verifier, i.e. the strategy is followed by both prover and verifier in the manner of challenge and response. Interactive zero-knowledge proofs can be made non-interactive with Fiat-Shamir heuristic technique. With this technique, the prover assumes all the challenges and generates their corresponding responses without interacting with the verifier. Then, the verifier receives the proof together with all the challenges and responses and becomes convinced if they are all verified.

In e-voting, zero-knowledge proofs can be used to prove the validity of the vote with respect to election candidates, as in Helios [1], or the knowledge of the vote used in encryption, as in Selene [47] and Hyperion [46]. They may also be used to prove the correctness of the decryption or mixing if the mixnet is used.

Non-interactive zero-knowledge proof for validity of the vote: Let v_1 and v_2 be the two election candidates for some election with the public key pk_E. Assume the election requires encryption of the votes with an asymmetric key algorithm that randomises the ciphertext with a freshly generated randomness r, i.e. the voters encrypt their votes $\mathsf{v} \in \{\mathsf{v}_1, \mathsf{v}_2\}$ to obtain a ciphertext $\mathsf{c} = \mathsf{aenc}(\mathsf{v}, \mathsf{pk}_\mathsf{E}, \mathsf{r})$. The election also requires zero-knowledge proof for the validity of votes with respect to election candidates $\langle \mathsf{v}_1, \mathsf{v}_2 \rangle$. Therefore, the voters generate a zero-knowledge proof p, utilising the function zkp as follows:

$$\mathsf{p} = \mathsf{zkp}(\mathsf{c}, \mathsf{r}, \langle \mathsf{v}_1, \mathsf{v}_2 \rangle) = \mathsf{zkp}(\mathsf{aenc}(\mathsf{v}_\mathsf{i}, \mathsf{pk}_\mathsf{E}, \mathsf{r}), \mathsf{r}, \langle \mathsf{v}_1, \mathsf{v}_2 \rangle),$$

where $\mathsf{v} = \mathsf{v}_\mathsf{i}$ for some $\mathsf{i} \in \{1, 2\}$. The election authorities receive the ballot containing the ciphertext c and the zero-knowledge proof p. Then, they verify the validity of the vote inside the ballot, verifying the zero-knowledge proof p as follows:

$$\mathsf{ver}(\mathsf{p}, \mathsf{c}, \mathsf{pk}_\mathsf{E}, \langle \mathsf{v}_1, \mathsf{v}_2 \rangle) = \mathsf{ver}(\mathsf{zkp}(\mathsf{c}, \mathsf{r}, \langle \mathsf{v}_1, \mathsf{v}_2 \rangle), \mathsf{c}, \mathsf{pk}_\mathsf{E}, \langle \mathsf{v}_1, \mathsf{v}_2 \rangle) = \mathsf{true}.$$

Non-Interactive zero-knowledge proof of knowledge of the vote: Let r be the randomness freshly generated to be used in the encryption of the vote v. After generating the ciphertext for the vote, one can generate a proof p to prove the knowledge of v and of r. If the non-interactive zero-knowledge proof algorithm utilises a function zkp, then p will be generated as follows:

$$\mathsf{p} = \mathsf{zkp}(\mathsf{c}, \mathsf{v}, \mathsf{r}),$$

where c is the ciphertext corresponding to the vote v. Then, the verification algorithm uses a function ver to verify the proof p. Anyone can verify the proof with the ciphertext c and election public key pk_E:

$$\mathsf{ver}(\mathsf{p}, \mathsf{c}, \mathsf{pk}_\mathsf{E}) = \mathsf{ver}(\mathsf{zkp}(\mathsf{c}, \mathsf{v}, \mathsf{r}), \mathsf{c}, \mathsf{pk}_\mathsf{E}) = \mathsf{true}.$$

The proof described above is used to prove that the encryption is performed by the voter who generates the randomness, and thus, it helps to ensure no adversarial involvement to the generation of the ciphertext. However, it is not enough to prevent some attacks since it does not help to ensure the ciphertext cannot be used out of the intended context. The adversary could copy the ciphertext with the proof into another ballot, as in ballot copying attacks [19], which will be regarded as if the adversary generated the ciphertext after the verification of the

proof by election authorities. On the other hand, it will help adversary with learning the vote of the voter who generated the ciphertext. Therefore, e-voting protocols may require the use of zero-knowledge proof with the label of voter's public key, as proposed for Belenios [18]. When the voter's public key is added to the zero-knowledge proof, even if the adversary copies the proof, it would not be verified by election authorities. Thus, labeled zero-knowledge proof prevents the above-mentioned attack.

Non-interactive labeled zero-knowledge proof of knowledge of the vote: Assume each voter is provided with a signature key pair in an election, i.e. the voter with id has $(\mathsf{sk}_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}})$, and the election requires labeled zero-knowledge proof with the voter's public key $\mathsf{pk}_{\mathsf{id}}$, which is attached to the ciphertext generated by that voter. The ciphertext is obtained, encrypting the vote v with fresh randomness r, i.e. $\mathsf{c} = \mathsf{aenc}(\mathsf{v}, \mathsf{pk}_{\mathsf{E}}, \mathsf{r})$, where pk_{E} is the election public key. Then, the voter generates the labeled zero-knowledge proof p using their public key $\mathsf{pk}_{\mathsf{id}}$ as follows:

$$\mathsf{p} = \mathsf{zkp}(\mathsf{c}, \mathsf{v}, \mathsf{r}, \mathsf{pk}_{\mathsf{id}}) = \mathsf{zkp}(\mathsf{aenc}(\mathsf{v}, \mathsf{pk}_{\mathsf{E}}, \mathsf{r}), \mathsf{v}, \mathsf{r}, \mathsf{pk}_{\mathsf{id}}).$$

Upon receiving the ballot for the voter with id, election authorities verify the proof p with the voter's public key $\mathsf{pk}_{\mathsf{id}}$ in the following:

$$\mathsf{ver}(\mathsf{p}, \mathsf{c}, \mathsf{pk}_{\mathsf{E}}, \mathsf{pk}_{\mathsf{id}}) = \mathsf{ver}(\mathsf{zkp}(\mathsf{c}, \mathsf{v}, \mathsf{r}, \mathsf{pk}_{\mathsf{id}}), \mathsf{c}, \mathsf{pk}_{\mathsf{E}}, \mathsf{pk}_{\mathsf{id}}) = \mathsf{true}.$$

2.1.4 Commitments

Commitments are used to commit to a certain value with a secret, which hides the value until the secret is revealed. If a commitment is made, then the party who commits cannot deny or change the committed value by providing a different secret afterwards. In e-voting, commitments may be preferred to hide the votes of voters during the collection of votes, as in FOO [28]. Then, for the tally, the voters reveal them, providing the randomness used in the commitment that allows authorities to count their votes. Commitments may also be used for other purposes, such as generating election credentials, i.e. trackers, for voters, as in Selene [47]. In this case, election authorities commit to trackers at the beginning of the election, and they provide the commitment randomness with the corresponding voters at the end of the election, which allows voters to reveal their tracker and then verify their votes using the revealed trackers.

Commitment: Assume a party wants to commit to a message m, using a function commit with fresh randomness r:

$$\mathsf{com} = \mathsf{commit}(\mathsf{m}, \mathsf{r}),$$

and sends the commitment to another party that can open the commitment only if the randomness is provided, i.e.

$$\mathsf{open}(\mathsf{com}, \mathsf{r}) = \mathsf{open}(\mathsf{commit}(\mathsf{m}, \mathsf{r}), \mathsf{r}) = \mathsf{m}.$$

Commitment to the vote cast: Assume a voter who wants to cast a vote v commits to that vote during voting of the election, using a function commit with fresh randomness r:

$$\mathsf{com} = \mathsf{commit}(\mathsf{v}, \mathsf{r}),$$

and sends the commitment to election authorities after signing it. If the signature is valid, the authorities stores the commitment as a ballot of the voter. In the end of election, for the tally procedure, the voter sends the randomness r to the authorities via anonymous channel to

reveal the vote was cast, which allows authorities to open the commitment as follows:

$$\mathsf{open}(\mathsf{com}, \mathsf{r}) = \mathsf{open}(\mathsf{commit}(\mathsf{v}, \mathsf{r}), \mathsf{r}) = \mathsf{v}.$$

2.1.5 Mixnets

Mixnets, i.e. mix networks, are applied to a set of elements to shuffle them through several mixes. Each mix uses a permutation for shuffling and combines permutations with a mathematical operation, anonymising the output. In e-voting, mixnets provide anonymisation in the tally, shuffling the set of ciphertexts either re-encrypting or decrypting through a few mixes, which contributes to the privacy of the voters, breaking their link to the outcome. Each mix takes an input as a set of ciphertexts, re-encrypts/decrypts each element, permutes the resultant set using a random permutation, and provides a zero-knowledge proof of correct shuffling. In the end, the output of the last mix gives a set of ciphertexts, which cannot be publicly linked to the initial set of ciphertexts but still preserves the corresponding set of votes.

Re-encryption mixnet: Assume an election requires ElGamal encryption of the votes with the public key pk_E. Let $\{\mathsf{c}_1, ..., \mathsf{c}_\mathsf{n}\}$ be the set of ciphertexts of n voters at the end of the election such that any $\mathsf{c}_\mathsf{i} = \mathsf{aenc}(\mathsf{v}_\mathsf{i}, \mathsf{pk}_\mathsf{E}, \mathsf{r}_\mathsf{i})$ is the encryption of the vote v_i with fresh randomness r_i. ElGamal encryption allows re-encryption of the ciphertexts, i.e. re-encryption of c_i with fresh randomness r'_i is computed as:

$$\mathsf{c}'_\mathsf{i} = \mathsf{renc}(\mathsf{c}_\mathsf{i}, \mathsf{pk}_\mathsf{E}, \mathsf{r}'_\mathsf{i}) = \mathsf{aenc}(\mathsf{v}_\mathsf{i}, \mathsf{pk}_\mathsf{E}, \mathsf{r}_\mathsf{i} + \mathsf{r}'_\mathsf{i}).$$

Assume a re-encryption mixnet with k mixes, i.e. $\mathsf{M}_1, ..., \mathsf{M}_\mathsf{k}$, is applied to the set $\{\mathsf{c}_1, ..., \mathsf{c}_\mathsf{n}\}$ of ciphertexts for their anonymisation in the tally. The first mix M_1 takes the set $\{\mathsf{c}_1, ..., \mathsf{c}_\mathsf{n}\}$ as input, re-encrypts each element in the set, permutes the set using a random permutation ρ_1, and provides a zero-knowledge proof π_1 of correct shuffling. The resultant set of ciphertexts, i.e. $\{\mathsf{c}^1{}_{\rho_1(1)}, ..., \mathsf{c}^1{}_{\rho_1(\mathsf{n})}\}$ will be the input for the mix M_2, which will re-encrypt each element, permute the set with a random permutation ρ_2 and provide the proof π_2. After going through all the mixes, i.e. after applying M_k as last,

$$\begin{array}{ccccc}
\{\mathsf{c}_1, ..., \mathsf{c}_\mathsf{n}\} & \{\mathsf{c}^1{}_{\rho_1(1)}, ..., \mathsf{c}^1{}_{\rho_1(\mathsf{n})}\} & \dots & \{\mathsf{c}^{\mathsf{k}-1}{}_{\rho_{\mathsf{k}-1}(1)}, ..., \mathsf{c}^{\mathsf{k}-1}{}_{\rho_{\mathsf{k}-1}(\mathsf{n})}\} \\
\Downarrow & \Downarrow & & \Downarrow \\
\mathsf{M}_1 & \mathsf{M}_2 & \dots & \mathsf{M}_\mathsf{k} \\
\Downarrow & \Downarrow & & \Downarrow \\
\{\mathsf{c}^1{}_{\rho_1(1)}, ..., \mathsf{c}^1{}_{\rho_1(\mathsf{n})}\}, \pi_1 & \{\mathsf{c}^2{}_{\rho_2(1)}, ..., \mathsf{c}^2{}_{\rho_2(\mathsf{n})}\}, \pi_2 & \dots & \{\mathsf{c}^\mathsf{k}{}_{\rho_\mathsf{k}(1)}, ..., \mathsf{c}^\mathsf{k}{}_{\rho_\mathsf{k}(\mathsf{n})}\}, \pi_\mathsf{k}
\end{array}$$

the output set of the ciphertexts will be $\{\mathsf{c}^\mathsf{k}{}_{\rho_\mathsf{k}(1)}, ..., \mathsf{c}^\mathsf{k}{}_{\rho_\mathsf{k}(\mathsf{n})}\}$, where each $\mathsf{c}^\mathsf{k}{}_{\rho_\mathsf{k}(\mathsf{i})}$ corresponds to:

$$\mathsf{c}^\mathsf{k}{}_{\rho_\mathsf{k}(\mathsf{i})} = \mathsf{aenc}(\mathsf{v}_{\rho_1 \circ \dots \circ \rho_\mathsf{k}(\mathsf{i})}, \mathsf{pk}_\mathsf{E}, \mathsf{r}_\mathsf{i} + \mathsf{r}^1_\mathsf{i} + \dots + \mathsf{r}^\mathsf{k}_\mathsf{i}).$$

As can be seen, re-encryptions and permutations preserve the set of votes cast, while anonymising them compared to the initial set of ciphertexts.

2.2 E-Voting Protocol Parties

An e-voting protocol usually has multiple election authorities to set up and maintain the election, count the votes cast during the election, and publish the outcome. It also has a number of voters attending the election. Voters mainly use voting platforms to cast a ballot, but they

sometimes use additional vote verification platforms, any of which can be independently corrupt by the adversary. Therefore, we consider them as protocol parties. An election may also have auditors who audit any data relevant to the election's integrity.

2.2.1 Election Authorities

Election authorities are responsible for organising, maintaining, and finalising an election. The authorities can all be internal parties or may include some external parties helping with the election procedures. Sometimes external parties are added to the protocol to divide the trust among the election authorities so that they could help to detect the corruption of internal parties, or weaken the abilities of the adversary.

The authorities use a common bulletin board, denoted by BB, for the data utilised in the election, allowing them to monitor the recorded data during the election. It also helps auditing the election procedures with the data leading to the election outcome. Depending on the protocol, BB may be private or public. The election data is recorded on BB in portions. For example, the portion including the election key is denoted by BBkey.

- ***Administrator*** is responsible for organising the election, i.e. determines other election authorities and assigns their roles, prepares the list of candidates and the list of eligible voters, may determine other election parameters such as voter credentials to be used in the election (an identity or another public credential), revote policy determining which ballot will be tallied (first or last ballot if the voter revotes). Administrator records the election candidates on BBcand.

- ***Tallier*** is responsible for generating the election key pair according to the algorithm specified for the election. In general, a threshold key generation is preferred with multiple talliers, which distributes the trust between talliers and increases robustness in case of a failure of some talliers. In threshold key generation, assume k talliers share the election secret key, and at least t ($t \leq k$) of them have to cooperate to decrypt any ciphertext encrypted with the election public key. More specifically, each of the talliers generates some part of the election private key and its corresponding part of public key, then shares the public part of the key with others, while keeping the private part to themselves. The public part, when it is combined with others, constructs the election public key, which is recorded on BBkey and used by the voters to encrypt their votes. For the decryption of votes, each tallier decrypts the ciphertext output with their private part of the election key. In the end, all decryptions are combined to obtain the outcome of the election, which can be published on BBres.

- ***Voting Registrar:*** Some voting protocols require additional registrar for generating election credentials for voters, according to the voter list provided by the administrator. For example, the voting registrar may create a signature key pair for each voter to be used for signing the ciphertext of the vote, or it may generate an alias for each voter to cover their real identity. Because real identities, when they are attached to the ballots, may lead adversary to compromise the privacy of the voters. The ballots usually contain the encryptions of the votes, however, the encryption algorithm used can be deprecated in the future, breaking the security of the encryption. This would compromise all the votes attached to the identities, violating the privacy. If there is a voting registrar in the protocol, it records all the public components of the election credentials, e.g. the public keys of the signature pairs, on BBreg. Otherwise, the administrator records the voter identities on BBreg.

- ***Voting Server*** is responsible for collecting the digital ballots from eligible voters in the list of administrator, recording each ballot on its database, and publishing them

on BBcast. For accepting ballots in the digital environment, some protocols require authentication of the voters, for which voting server generates a password for each voter. Then, voters need to authenticate themselves entering correct password provided by the voting server, which allows them to submit their vote on behalf of them. Moreover, the voting server is responsible for providing the ballots to be tallied by the talliers. Thus, it records the ballots to be tallied on BBtally. If revoting is not allowed in the protocol, BBcast and BBtally contain the same batch of ballots. Otherwise, the voting server determines the ballot to be tallied for each voter according to the revote policy specified by the administrator.

2.2.2 Voters and Their Voting Platforms

Voters attend the elections by casting a ballot on the voting server. To prepare their digital ballots and submit them to the voting server, they use a voting platform, e.g. a website opened by a browser or a voting application on an electronic device. The voting server may require authentication before accepting a ballot generated by a voter. The authentication method may vary among the protocols. In general, the voting server provides a password to each voter before the election, and the voters must use this password to submit their ballot during the election.

Voting platforms perform mathematical operations to prepare a ballot for the voter's choice. The ballot may contain only the encryption of the vote, which is the ciphertext obtained by applying the encryption algorithm to the vote with the election public key published on the BBkey. It may contain an additional signature of the voter, which is obtained by signing the ciphertext with the public key of the voter. The signature proves that the ballot is prepared by the voter who submits it since the private key can only be known by them. The ballot may also contain a zero-knowledge proof proving that the ciphertext is created by the voter who submits the ballot, i.e. the voter did not copy it from elsewhere.

Depending on the protocol, voting platforms may display a receipt for the submitted ballot so voters can verify it on BB. If the voter verifies their ballot, it should be recorded as cast for them, and election auditors should ensure it will be tallied as recorded later. Either the same voting platform is used, or a separate verification platform is introduced to the voting protocol for verifying the ballot.

The e-voting protocol may allow revoting, which is not possible for traditional voting. This is an advantage for long-term elections since the voters may change their minds according to the election campaigns. Another advantage is that the voters can revote if coerced, assuming the coercer cannot follow the actions of the voters during the whole election period. In the case of revoting, the voting platform may submit many ballots to the voting server. Each is recorded on BBcast, but only one is selected to be tallied according to the revote policy and then recorded on BBtally.

2.2.3 Election Auditors

The election procedures and data obtained, processed, and published during the election should be audited by external parties called election auditors. This is needed to provide end-to-end verifiability and ensure security of the election. Therefore, the auditors should verify the eligibility of the voters who cast a ballot during the election, i.e. they match the lists of voter credentials published on BBreg, BBcast, and BBtally. They should check the validity of the ballots published on BBcast, i.e. check whether they contain a valid signature of the voter who cast the ballot, a valid zero-knowledge proof if provided. The auditors should also verify the proofs generated in the tally phase, i.e. they get the lists of BBtally and BBres and verify the proof of correct decryption.

2.3 E-Voting with respect to Election Phases

E-voting protocols usually consist of three phases of an election: a setup phase for determining the election parameters, the election key, voters, the candidates to be chosen as votes, etc.; a voting phase for collecting the ballots from voters, and a tally phase for deciding the ballots to be opened, decrypting the ciphertexts inside them, and announcing the election result. In general, e-voting protocols allow individual verification of the votes cast by the voters, which is a procedure convincing the voter about their vote that was received and collected correctly by the voting server. In the following, we describe a typical set of election procedures according to the phases and the individual verification procedures.

2.3.1 Setup Phase

Tallier generates a private key sk_E and computes the public key pk_E from sk_E, and then publishes the election public key on BBkey. It keeps sk_E secure for the decryption of votes later. The administrator determines the candidates to be elected, i.e. $\mathsf{v}_1, \ldots, \mathsf{v}_k$, and publishes them on BBcand. It also makes a list of eligible voters, i.e. $\mathsf{id}_1, ..., \mathsf{id}_n$. The voting registrar generates an election credential for each voter id and publishes the public part cr on BBreg. The voting registrar and server communicate to voter credentials required to authenticate and cast a ballot. At the end of the setup phase, BB will contain the following information generated by the election authorities:

$$\mathsf{BBkey} : \mathsf{pk}_E; \quad \mathsf{BBcand} : \mathsf{v}_1, ..., \mathsf{v}_k; \quad \mathsf{BBreg} : \mathsf{cr}_1, ..., \mathsf{cr}_n;$$

and the voters will have the credentials: $\mathsf{id}, \mathsf{cr}, \mathsf{pwd}$.

2.3.2 Voting Phase

This phase mainly represents the communication between voters and the voting server. Voters cast their ballots prepared by their voting platform. The voter chooses their vote v and the voting platform encrypts v with the election public key pk_E on BBkey and fresh randomness r, and obtains a ciphertext c. The voter signs the ciphertext, i.e. the signature s is obtained. Then, the voting platform produces a zero-knowledge proof p for the knowledge of the vote, leading the ballot structure to $\mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p} \rangle$, which is submitted to the voting server if it authenticates the voter id. The server performs some checks on the ballot. First, it checks the credential cr whether it is registered for the election, i.e. checks whether $\mathsf{cr} \in \mathsf{BBreg}$. Then, it verifies the signature s and the zero-knowledge proof p on the ballot. If verified, the voting server publishes $(\mathsf{cr}, \mathsf{b})$ on BBcast. At the end of the voting phase, BBcast will contain a ballot for each voter:

$$\mathsf{BBcast} : (\mathsf{cr}_1, \mathsf{b}_1), \ldots, (\mathsf{cr}_n, \mathsf{b}_n),$$

where b_i may be empty, i.e. $\mathsf{b}_i = \perp$, for some voters. If there is revoting in the protocol, BBcast may contain several ballots for each voter:

$$\mathsf{BBcast} : (\mathsf{cr}_1, \mathsf{b}_1^1, ..., \mathsf{b}_1^{\ell_1}), \ldots, (\mathsf{cr}_n, \mathsf{b}_n^1, ..., \mathsf{b}_n^{\ell_n}),$$

where ℓ_i may equal to 1 or b^{ℓ_i} may equal to $\perp$ for any i.

2.3.3 Tally Phase

After collecting all the ballots from voters and recording them on BBcast, the voting server selects a ballot for each voter in the tally phase, complying with the revote policy. All these ballots are published on $\mathsf{BBtally}$ so that they will be processed for the outcome. If there is no

revoting, BBcast and BBtally will have the same set of ballots. Otherwise, BBtally will have the following:

$$\mathsf{BBtally} : (\mathsf{cr}_1, \mathsf{b}_1^{\ell_1}), \ldots, (\mathsf{cr}_n, \mathsf{b}_n^{\ell_n}),$$

where ℓ_i could be 1 for each i representing the first ballot of the voter, or it could be the greatest one among all ballots of cr_i representing the last ballot of the voter depending on the revote policy.

The administrator gets the list of ballots to be tallied from BBtally, detaches all the credentials next to the ballots, and fetches the ciphertexts from ballots. It may either assign the tallier to use a mixnet for the election output or directly ask to decrypt the list of ciphertexts. Assume the list of ciphertexts $\mathsf{c}_1, \ldots, \mathsf{c}_n$ collected by the administrator. If a mixnet is used by the tallier, the ciphertexts will be mixed through re-encryptions and permutations, and thus, in the end, the list of ciphertexts will be $\mathsf{c}'_1, \ldots, \mathsf{c}'_n$. Otherwise, $\mathsf{c}_i = \mathsf{c}'_i$ for any i. Moreover, if the homomorphic encryption is utilised, then the set of ciphertexts will be combined into one ciphertext c corresponding to all the votes collected. Then, either the ciphertext c or the list of ciphertexts $\mathsf{c}'_1, \ldots, \mathsf{c}'_n$ is decrypted by the tallier using the private key of the election sk_E:

$$\mathsf{v} = \mathsf{dec}(\mathsf{c}, \mathsf{sk}_E) \quad \text{or} \quad \mathsf{v}_i = \mathsf{dec}(\mathsf{c}_i, \mathsf{sk}_E) \text{ for any i}$$

corresponding to the outcome $\mathsf{v}_1, \ldots, \mathsf{v}_n$. The result is published on BBres:

$$\mathsf{BBres} : \mathsf{v}_1, \ldots, \mathsf{v}_n.$$

Figure 2.2 displays election procedures followed by election authorities. The three election phases are divided by the horizontal lines for election authorities and BB.

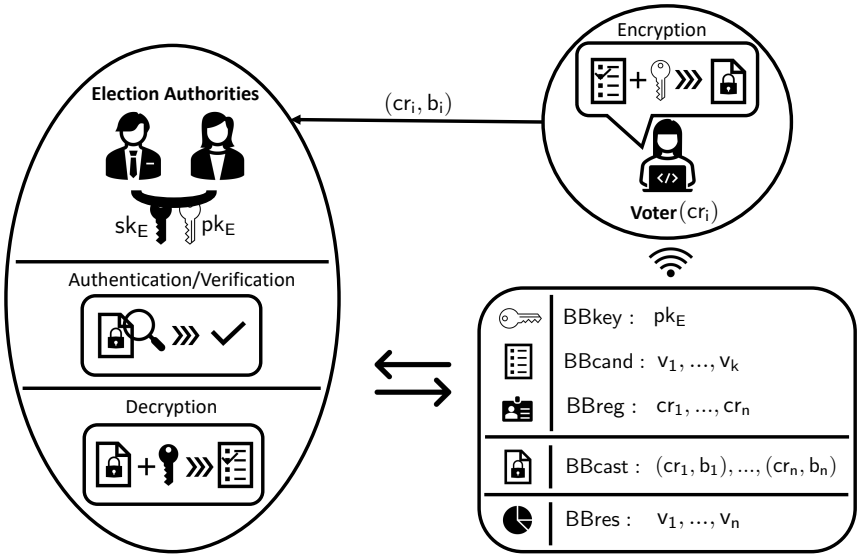


FIGURE 2.2: Election procedures with respect to phases.

2.3.4 Individual Verification

Most e-voting protocols allow individual verification by voters, which is the procedure that helps voters to ensure that their ballot is recorded as cast to the voting server. For the verification, the voter should match their vote or their ballot with the one recorded on BB. For directing the voter to the correct ballot, the protocol may provide a receipt for tracking. Usually, the verification procedure is independent of the voting procedure, and thus, the receipt helps to find the correct ballot among all other ballots on BB. Regarding the complexity of

the ballot, which is a long output of bits, matching the ballot cast with the one on BB could be hard for the voter. Therefore, the e-voting protocol may choose to publish a short version of the ballot for comparison, i.e. the hash of the ballot. Hashing shortens the ballot length in a way that the output does not collude with others since the input is different, i.e. the ballots are different for each other. Thus, publishing the hash of the ballots does not violate the security of the election.

The individual verification procedures may vary among the protocols. In the following, the various verification procedures will be discussed briefly:

- ***Individual verification of any ballot on*** BBcast***:*** Assuming revoting is enabled for the e-voting protocol, the voters are allowed to verify any of their ballot cast anytime during the voting phase on BBcast. This means that BBcast displays all ballots cast by each voter.

- ***Individual verification of last ballot on*** BBcast***:*** Similar to the above case, if revoting is allowed, the voters can verify only their last ballot cast on BBcast during the voting phase. If they revote, they lose the option to verify a former ballot. This means that BBcast hides former ballots cast from the voter if a new ballot is cast by them, i.e. BBcast only displays the last ballot for each voter on its database. If there is no revoting, BBcast just displays the ballot received from the voter. Otherwise, the ballot would be $\perp$.

- ***Individual verification of the ballot on*** BBtally***:*** In this case, the voters are allowed to verify only the ballot to be tallied for them. Thus, the verification is performed at the end of the election after the voting phase has ended. The ballot to be verified could also be $\perp$ so that the voter could verify that nobody used their credentials to cast a ballot.

- ***Individual verification of the vote on*** BBres***:*** In some e-voting protocols, the voters are provided with additional credentials, i.e. trackers, that allow voters to verify their vote in the clear on BBres. The trackers are used to provide privacy to the voters, while making the verification procedure much more easier. Instead of verification of the ballots by matching complex numbers, the voters just match their vote with a tracker provided by the election authorities. The trackers are generated uniquely for each voter so that any two voters' trackers will not collide.

2.4 Security Properties of E-Voting Protocols

The security needs for e-voting are different than the ones in traditional voting since the digital environment is more vulnerable to coercers and adversaries that can violate the privacy of the voters and easily manipulate the election even without being noticed by election authorities. To prevent any attempt making the election fail, several security properties, such as privacy, verifiability, integrity, robustness, etc., were proposed in the literature. Nevertheless, privacy and verifiability are the two mainly sought properties for e-voting protocols.

Privacy is based on hiding the link of the voter to the clear vote, and therefore, removing any means in the protocol that will disclose to the adversary the voter's vote. On the other hand, verifiability is based on transparency of the procedures followed by the protocol. For example, all the votes cast should be tracked to eligible voters to ensure that there is no adversarial involvement. Thus, one property requiring secrecy naturally conflicts with another requiring transparency. The eventual aim in e-voting protocols is to achieve a good balance between privacy and verifiability without sacrificing one to another. In addition to them, usability is another property which is essential among all e-voting protocols. The protocol should be simple and usable by any person independent of their age and abilities.

2.4.1 Privacy

In democratic elections, anyone should express their will by voting for the candidate they desire without being suppressed or coerced. Traditional voting prevents coercion to a great extent since the vote is covered by an envelope in a private voting booth, and the ballot (the vote covered by an envelope) becomes anonymous in the ballot box among all others, which also restricts the abilities of voters to prove how they voted. The coercer may attempt vote buying if the voter can provide a proof, such as a photograph taken in the voting booth, or the coercer may prevent the voter from voting, i.e. making them absentee.

On the other hand, e-voting leads to coercion on a broader scale. Digital ballots are mostly accompanied with election credentials of the voters, which is needed for verifiability. Then, the ballots cannot contain clear votes not to violate privacy of the voters, i.e. the ballots contain encoded votes of the voters. Therefore, the voters should verify their ballot to ensure that it contains their votes. Many e-voting protocols allow voters to verify their ballot providing them with some receipt. This receipt can be requested by the coercer to verify the vote, which makes voter vulnerable to coercion. The receipt may also be used for vote buying.

The e-voting protocols, in general, should preserve their strength regarding coercion with some devised techniques allowing individual verification. Fundamentally, the protocol should provide ballot privacy, i.e. the vote should not be linked to the voter by any means. Then, a higher level of privacy requires receipt-freeness, i.e. the voter cannot provide any receipt to the coercer for verifying the vote. Lastly, the highest level of privacy requires coercion resistance, i.e. even though the voter cooperates with the coercer, there is no way for them to prove how they voted. Achieving coercion resistance is really challenging since, in many e-voting protocols, all the ballots received by the authorities are published, and in that case, the coercer may notice that the voter voted even if they coerced the voter to be absentee.

- ***Ballot privacy:*** Nobody should learn how the voter voted, i.e. the vote cast by the voter should be anonymised among others.
- ***Receipt-freeness:*** The voter cannot provide a receipt to prove how they voted.
- ***Coercion resistance:*** The voter cannot prove how they voted or whether they voted in any way, even though they cooperate with the coercer.

Among our case studies, we will have Helios and Belenios that only satisfy ballot privacy, and BeleniosRF, Selene, Hyperion, and the Estonian e-voting protocol that aim to satisfy receipt-freeness.

2.4.2 Verifiability

Verifiability basically tracks all the election data to their origin and ensures that the procedures held by the election authorities are followed in the right way. This is needed to ensure that there is no adversarial involvement in the election, i.e. the adversary cannot fake an election credential, cast adversarial ballots for fake credentials, change the ballots cast by eligible voters nor change the election outcome.

One aspect of verifiability involves voters verifying that their ballot is correctly collected by the election authorities. The verification procedure varies among e-voting protocols. In some protocols, the voter's vote is verified in the clear, whereas in others, a receipt is provided to the voter for verification of the ballot encoding the vote. The other aspects of verifiability involve anyone verifying that all ballots collected by the election authorities were cast by eligible voters and that all the votes in the outcome come from the ballots collected by the authorities. In contrast to traditional voting, in e-voting, the voters can cast multiple ballots during the election if the protocol allows revoting. In that case, the protocol specifies a revote

policy for selecting the ballot to be tallied, i.e. the vote to be counted for each voter. In general, the revote policy selects the last ballot cast by each voter for the tally. Thus, the selection of the correct ballot, among others, should also be verified. A protocol that is verifiable with respect to all election procedures is called end-to-end verifiable. End-to-end verifiable protocols give security guarantees about the election outcome that it reflects the intention of the eligible voters.

- ***Individual verifiability:*** Any voter should be able to verify that their ballot was collected as it was cast.
- ***Eligibility verifiability:*** Anybody should be able to verify that all the votes counted by election authorities were cast by eligible voters.
- ***Universal verifiability:*** Anybody should be able to verify that the election outcome corresponds to the set of ballots collected by election authorities, complying with the revote policy.

We will discuss more about verifiability in Chapter 4.

2.4.3 Usability

Usability is an essential property of e-voting protocols. Security can be defined with different measures, leading the protocol to deploy very complex cryptographic primitives that are combined with complicated election procedures. For many people, those procedures are hard to follow and barely understood. Thus, this ambiguity affects people's trust in the protocol being deployed and decreases the voters' participation in the elections, which is not desired for democracy. Therefore, the election procedures should be very simple and understandable to be followed by anyone regardless of their educational background, age, and abilities.

2.5 Verifiable E-Voting Protocols with Examples

In this section, we give three examples of verifiable e-voting protocols, focusing on the individual verification procedures of the voters. In each example, the verification credentials differ from those in others. The protocol in Example 1 uses a mechanism similar to Helios [1], where the verification credentials are the voter identities. Example 2 is similar to the Belenios [18] protocol, where the public keys of the voters are the verification credentials, and the protocol in Example 3 is similar to Selene [47] since it allows voters to verify their votes using trackers. After providing examples, we present a diagram in Figure 2.3 to illustrate the third example of verifiable e-voting protocols.

Example 1. *Let the election credentials of the voters be their identities. Then, for any eligible voter holding the identity* id*, the* id *is published on* BBreg*. In the voting phase, the voter prepares a ballot* b *containing the vote* v *and attaches their identity to* b *for submission to election authorities, i.e.* $(\mathsf{id}, \mathsf{b})$ *is submitted to the voting server and published on* BBcast*. For individual verification, the voter matches* b *with the ballot* b' *posted next to their* id *on* BBcast*.*

In Example 1, the identities are clearly published on BBcast with the ballots cast by them. Assuming the ballot contains the encryption of the vote, the privacy of the identity depends on the security of encryption algorithms. Most of the encryption algorithms used today are secure only for today. Anytime in the future, they can be broken or require longer key lengths. This means that the adversary can possess the election secret key used for the decryption to learn the votes of the identities in the future, which will violate their privacy. Therefore, many

protocols do not prefer to use identities as election credentials. Instead, they generate or utilise other credentials, such as aliases or public keys of the signature key pairs.

Example 2. *Let the election credentials of the voters be their public keys. Then, for any eligible voter with* id*, the voting registrar generates a signature key pair, where the public key of the pair, denoted by* cr*, is published on* BBreg*. In the voting phase, the voter prepares a ballot* b *containing the vote* v *and the signature* s*, then attaches their credential to* b *for submission to election authorities, i.e.* (cr, b) *is submitted to the voting server. The voting server verifies the signature in the ballot and publishes* (cr, b) *on* BBcast*. For individual verification, the voter matches* b *with the ballot* b′ *posted next to their credential* cr *on* BBcast*.*

In Example 2, the crucial point is the uniqueness of the election credentials. A credential generated by the voting registrar should not be assigned to more than one voter. Otherwise, the votes of those voters will be counted as one in the tally phase. Encryption of the vote with fresh randomness makes the ballot unique. Therefore, even if many voters share a credential, their ballots will be different. Thus, if they attempt to verify their ballot on BB, only one of those will achieve it, making others detect the attack. However, if revoting is allowed by the protocol, the ballot on BB will be replaced each time the credential submits a ballot. In this case, catching the attack by individual verification will be harder. The voters may have to perform individual verification many times to ensure no voter sharing the same credential overwrites their ballot, i.e. their ballot is received and will be counted.

Example 3. *Assume the protocol provides voters with two election credentials; one is their public credentials, and the other is their verification credentials. Let the public credentials be the public keys of the voters. As in Example 2, the voting registrar generates a signature key pair for each voter, i.e. a public credential* cr *is assigned to the voter with* id*, which is also published on* BBreg*. Election authorities also generate a verification credential allowing the voter to track their vote, which is called a tracker and denoted by* tr*. Similar to Example 2, the voter submits* (cr, b)*, and the voting server publishes it on* BBcast*. All the ballots are opened in the tally phase, and the votes obtained are published on* BBres *next to the trackers. Receiving the tracker from authorities, the voter verifies their vote on* BBres*, i.e. the voter finds* (tr, v′) *on* BBres *and matches* v *with* v′*.*

In the example above, the votes are published in the clear on BBres at the end of the election, making the verification procedure easier for voters. For the verification of the ballots, the voters need to match long, complex strings, constituting the ciphertexts or their hashes, which is complicated for many voters. The simplest way to perform an individual verification is to match the clear votes. The Figure 2.3 displays a diagram for Example 3.

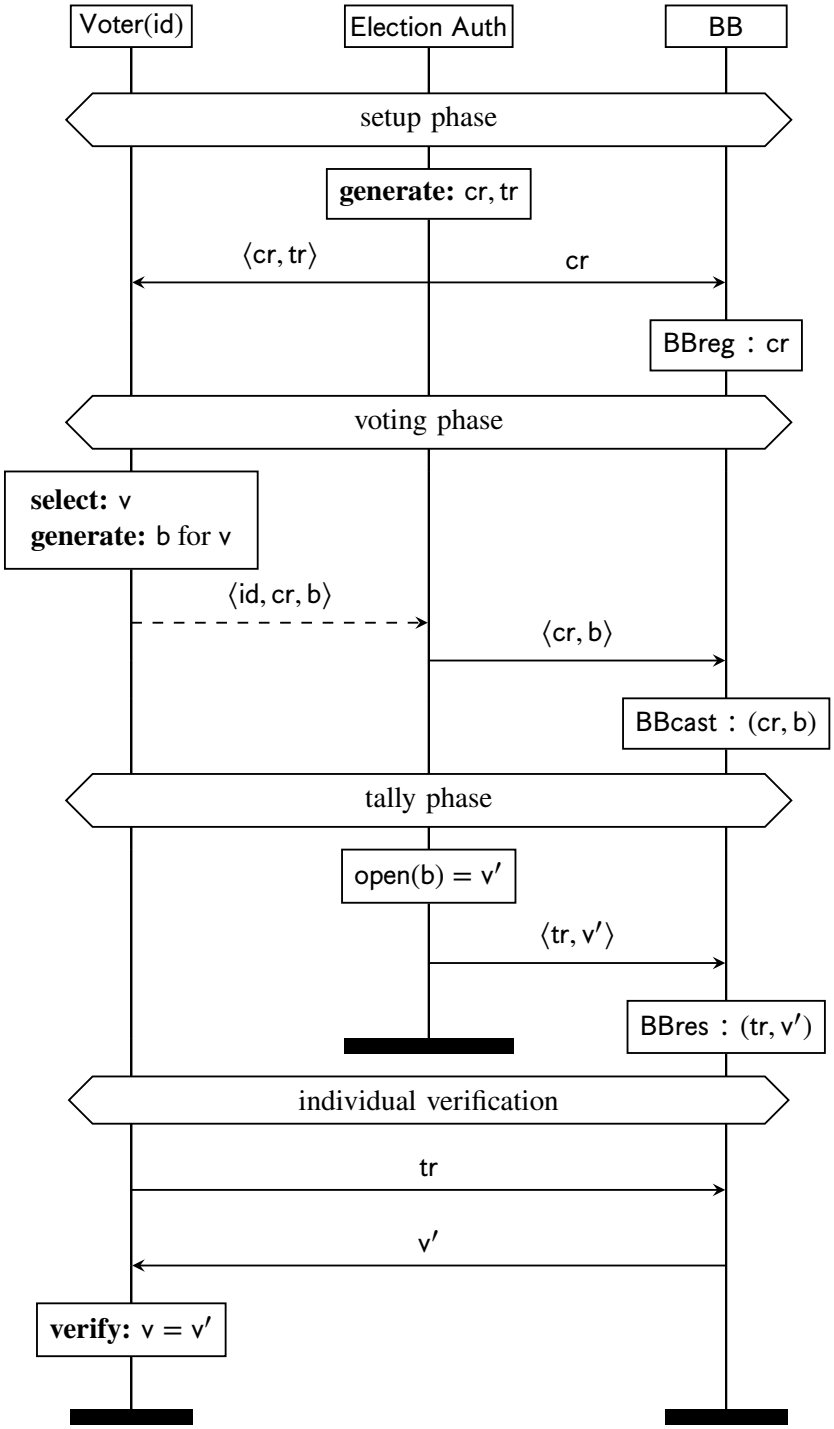


FIGURE 2.3: Verifiable e-voting protocol example.

Chapter 3

Formal Verification Preliminaries

Protocols are the building blocks of today's communication through the internet. They organise the communication flow between parties so that any party can successfully communicate with another via a private or public channel. The protocols are designed for a specific purpose, requiring some properties to be achieved. For example, Diffie-Hellman key exchange protocol [25] allows two parties to establish a shared secret key via a public channel. Each party in the protocol generates a key pair (private-public keys), sends the public key to the other party using a public channel, and then combines the received public key with their private key to compute the shared key. This protocol satisfies the secrecy of the shared key against any eavesdropper since the eavesdropper cannot compute the shared key from the information on the public channel, i.e. from the public keys of the two parties. In general, protocols can be much more complex than this, especially the e-voting protocols.

To ensure the protocols satisfy the required properties, they should be analysed rigorously, considering systematically all possible scenarios. A failure of any property concerning a scenario may point to a flaw in the design of a protocol. If it is the failure of some security property, it can lead to security breaches that definitely will not be desired for communication. Considering complex protocols that are held through the internet, security flaws may not be easily discovered. To capture any fault in the protocol or ensure there is no flaw, formal methods can be applied to the protocol to obtain an abstract mathematical model. Then, the formal verification of the specified model with respect to its properties will either show the fault or provide proof, ensuring those properties through the model. In case of a failure of any property, the protocol design can be improved to prevent it.

Formal verification of a protocol specification ensures the required properties providing manual proof, machine-assisted proof, or fully automated proof. Manual proofs are error-prone due to many tedious details that can be missed easily. Machine-assisted proofs require interaction with a user to generate them and are typically used for computational models. Fully automated proofs are generated on symbolic models by the tools in an automated way, giving more assurance than manual proofs. Symbolic models require a language for specifying the protocol and a logic for specifying the properties.

Tamarin [51, 41, 48] and ProVerif [45, 11] are the two promising tools that allow automated verification of the symbolic models of the protocols. In these tools, the protocol is symbolically modelled, specifying the actions of the parties, i.e. the generation and exchange of messages through public or private channels, and the actions of the adversary that controls the public channels and the messages exchanged, e.g. it can also corrupt the protocol parties using its abilities. Cryptography is assumed to be perfect; the adversary can perform cryptographic operations only if it possesses all the required terms. Then, the automated verification is performed, exhausting all the traces of the protocol specification and checking whether the security properties that are specified as trace properties hold.

Structure of the chapter: This chapter includes three sections for which Section 3.1 is dedicated to Tamarin, Section 3.2 is to ProVerif, and Section 3.3 presents the generic protocol specifications and properties, regarding Tamarin and ProVerif.

3.1 Formal Verification with Tamarin

Tamarin is an automated verification tool [51, 41, 48] that has an expressive language to specify protocols and adversaries as labeled transition systems based on multiset rewriting rules. The rules are constructed using the symbolic representations of the protocol's states, fresh information to generate the messages, and the messages received/sent from/to a public channel also representing the adversary's knowledge. When executed, each rule consumes some of the mentioned protocol states and messages and generates new states and messages, recording actions as events in the traces of the transition system. Security properties are specified as trace properties that are checked against all possible traces of the transition system or specified as equivalence properties that are checked in terms of the observational equivalence of two transition systems.

3.1.1 Specifying Protocols and Properties in Tamarin

In Tamarin, messages (or terms) are built from a set of function symbols and properties of cryptographic primitives are modelled by a set of equations. Cryptography is assumed perfect, thus, there is no other way to derive messages other than applying function symbols and equations. Let $\mathcal{F}$ be a set of function symbols. Then, messages are built by applying functions symbols from $\mathcal{F}$ to variables from an infinite set $\mathcal{X}$, constants from $\mathcal{F}$, names from an infinite set $\mathcal{N}$ or, iteratively, to other messages built in this way. Certain names may be specified to be fresh and private in an execution trace. The prefix \$ refers to a public name within the adversary's knowledge. The function symbols can be endowed with a set of equations $\mathcal{E}$, specifying term equalities according to the properties of functions. Equalities between terms are implicitly interpreted as being modulo $\mathcal{E}$.

Example 4. *Let* $\mathcal{F} = \{\mathsf{pk}, \mathsf{enc}, \mathsf{dec}, \mathsf{sign}, \mathsf{verify}\}$ *and* $\mathcal{E}$ *be the set of following equations:*

$$(1)\quad \mathsf{dec}(\mathsf{enc}(x, \mathsf{pk}(y), z), y) = x,$$
$$(2)\quad \mathsf{verify}(\mathsf{sign}(x, y), x, \mathsf{pk}(y)) = \mathsf{true}.$$

A term $\mathsf{enc}(m, \mathsf{pk}(k), r)$ *represents a ciphertext where* m *is the plaintext of a message,* $\mathsf{pk}(k)$ *is the public key corresponding to the private key* k*, and* r *is the randomness used by the encryption algorithm. Equations (1) and (2) specify the standard properties of asymmetric encryption and digital signatures. The ciphertext* $\mathsf{enc}(m, \mathsf{pk}(k), r)$ *can be decrypted only if the private key* k *is provided. Similarly, the signature* $\mathsf{sign}(m, k')$ *for the message* m *can be verified only with the public key* $\mathsf{pk}(k')$ *corresponding to the private key* k'*.*

The set of symbols is extended with fact symbols to represent protocol state information and adversarial knowledge. A fact is represented by $\mathsf{F}(m_1, \ldots, m_k)$, where F is a fact symbol and $m_1, \ldots, m_k$ are messages. There are four types of special fact symbols built in Tamarin: In and Out - for a message received from and send to the public channel controlled by the adversary, respectively, K for the adversary's knowledge and Fr for a fresh (random) information. Other symbols may be added as required for representing the protocol state. Facts can be *persistent*, i.e. used as premise any number of times, or *linear*, i.e. used at most once. A fact is preceded by ! to denote that it is persistent, e.g. $!\mathsf{F}(m_1, \ldots, m_k)$.

A multiset rewriting rule is defined by $[L]\text{—}[\ M\ ]\!\rightarrow[N]$, in which a set of *premise facts* L allows to derive a set of *conclusion facts* N, while recording certain events in *action facts* M. To ease protocol specification, the syntax of multiset rewriting rules can be extended with variable assignments and equality constraints, i.e. for a rule of the form $[L]\text{—}[\ E, M\ ]\!\rightarrow[N]$, L may contain expressions $x = t$ to define local variables and E may contain a set of equations of the form $u \equiv v$. The expressions are specified with let … in notation in Tamarin.

For two multisets of facts F_0, F_1 and rule R : $[L]$—[E, M]→$[N]$, F_1 can be obtained from F_0 by applying the rule R, instantiated with a substitution θ if every equality in $E\theta$ is true and every fact in $L\theta$ is included in F_0. Both of these statements are evaluated according to the equational theory. Then, F_1 is obtained from F_0 by removing linear facts included in $L\theta$ and adding all facts from $N\theta$.

There are three special classes of rules in Tamarin: *network deduction rules* specify that the adversary obtains protocol's outputs, provides protocol's inputs, knows public information and does not know freshly generated information; *intruder deduction rules* allow the adversary to apply functions and exploit their equational properties (a function can be declared private if the adversary is not supposed to use it); *protocol rules* allow to specify the actions of honest parties.

Example 5. *Consider the set $\mathcal{Q}_{\mathsf{ses}}$ of the following rules modelling session key exchange protocol between server and client:*

$\mathsf{R}_{\mathsf{key}}$: [Fr(k)]—[ ]→[!Sk(k), !Pk(pk(k)), Out(pk(k))]

$\mathsf{R}_{\mathsf{enc}}$: let c = enc(s, x, r) in
[In(x), Fr(s), Fr(r)]—[Key(s), Enc(c)]→[!SessionKeyC(s), Out(c)]

$\mathsf{R}_{\mathsf{dec}}$: [In(y), !Sk(k)]—[Dec(y)]→[!SessionKeyS(dec(y, k))]

The rule $\mathsf{R}_{\mathsf{key}}$ models the generation of the server's key pair that will be used to exchange the session keys with clients. The server generates a fresh secret key k*, which is stored for later use in* Sk*, while the corresponding public key is also stored in* Pk *and output to the network. The rule $\mathsf{R}_{\mathsf{enc}}$ specifies a client's actions for generating a fresh session key* s *that will be used for the encryption of the session with the server, receiving the public key of the server from the network, i.e. the input* x*, encrypting the session key* s *with* x *using fresh randomness* r *and outputting the ciphertext* c *to the server. In the rule $\mathsf{R}_{\mathsf{dec}}$, the server receives the encryption of the session key, i.e. the input* y*, and decrypts it using its secret key* k *stored in* Sk*. The session key is stored in* SessionKeyS *on the server's side, whereas in* SessionKeyC *on the client's side, to encrypt and decrypt the messages exchanged during the session. The action facts* Key(s)*,* Enc(c)*, and* Dec(y) *record the respective events in the execution trace.*

Trace Properties. For a set of rules P, an *execution trace* τ is defined by a sequence of multisets of facts $F_0, \ldots, F_n$ and a sequence of rules $R_1, \ldots, R_n \in P$ such that, for every $i \in \{1, \ldots, n\}$, F_i can be obtained from F_{i-1} by applying R_i instantiated with a substitution θ_i. Let $Act(R)$ be the action facts of a rule R. For a trace τ as above and any $i \in \{1, \ldots, n\}$, we define $Facts(\tau, i)$ to be the set of action facts occurred in the trace τ at the timepoint i.

- $Facts(\tau, i) = Act(R_i)\theta_i$ if R_i is a protocol or network deduction rule. This represents the fact that certain actions took place at timepoint i.
- $Facts(\tau, i) = \{\mathsf{K}(v\theta_i)\}$ if R_i is an intruder deduction rule with conclusion $\{\mathsf{K}(v)\}$. This represents the fact that the adversary knows $v\theta_i$ at timepoint i.

Temporal variables are used to specify the timepoints when actions occurred in a trace. If an action fact F should occur at a timepoint i, then it is denoted by $\mathsf{F}@i$. A *trace property* is a first-order logic formula over action facts and timepoints. Thus, the formula is obtained from universal and existential quantifications over term and temporal variables, logical connectives such as implication, conjunction, disjunction, and negation, action constraints such as $\mathsf{F}@i$, temporal ordering $i \prec j$ or equality $i = j$ for temporal variables i and j, and message equalities $\mathsf{m}_1 = \mathsf{m}_2$ for the messages m_1 and m_2. Similarly, a *restriction* is a logical formula that constrains the execution traces with respect to the action facts and timepoints.

The satisfaction relation $\tau \vDash \Phi$, for a trace τ and a trace formula Φ is defined recursively following the usual semantics for logical connectives, quantifiers, and time ordering constraints. We have the following notable case for action facts: $\tau \vDash \mathsf{F}@i$ if and only if $\mathsf{F} \in \mathit{Facts}(\tau, i)$. Note that term variables are interpreted over terms and temporal variables over timepoints. For a set of rules P, let $\mathsf{tr}(P)$ be the set of traces of P. Then, for a trace formula Φ, $P \vDash \Phi$ if and only if any trace in $\mathsf{tr}(P)$ satisfies the formula Φ. Moreover, let $(P; \Psi)$ be the specification of P with a restriction formula Ψ. Then, $(P; \Psi) \vDash \Phi$ if and only if any trace in $\mathsf{tr}(P)$ satisfying the restriction Ψ also satisfies the trace formula Φ. That is:

$$\begin{aligned} P \vDash \Phi &\iff \forall \tau \in \mathsf{tr}(P).\ \tau \vDash \Phi, \\ (P; \Psi) \vDash \Phi &\iff \forall \tau \in \mathsf{tr}(P).\ \tau \vDash \Psi \Rightarrow \Phi. \end{aligned}$$

Example 6. *Continuing Example 5, the formula* $\Phi_{\mathsf{sec}} : \forall \mathsf{x}, i.\ \mathsf{Key}(\mathsf{x})@i \Rightarrow \neg(\exists j.\ \mathsf{K}(\mathsf{x})@j)$ *states that if a term* x *is a secret session key recorded in* Key *generated by the second rule, then there is no timepoint at which the intruder knows it. We have* $Q_{\mathsf{ses}} \nvDash \Phi_{\mathsf{sec}}$ *since the adversary in the middle may replace the server's public key with its public key to learn the session key.*

3.1.2 Case Study: Verifiable E-Voting Example

We present the Tamarin specification of Example 3 from Section 2.5 in the following. Recall that digital ballots contain the encryption of the vote and the voter's signature on the encryption such that encryption is computed using an asymmetric encryption algorithm that randomises the ciphertext with fresh randomness, and the signature is generated using a digital signature algorithm compatible with the encryption algorithm. Therefore, we use the two equations defined in Example 4 to model the properties of those algorithms with the function symbols $\mathsf{pk}, \mathsf{enc}, \mathsf{dec}, \mathsf{sign}, \mathsf{verify}$:

$$\begin{aligned} &(1) \quad \mathsf{dec}(\mathsf{enc}(\mathsf{x}, \mathsf{pk}(\mathsf{y}), \mathsf{z}), \mathsf{y}) = \mathsf{x}, \\ &(2) \quad \mathsf{verify}(\mathsf{sign}(\mathsf{x}, \mathsf{y}), \mathsf{x}, \mathsf{pk}(\mathsf{y})) = \mathsf{true}. \end{aligned}$$

Next, we provide the specification $(P; \Psi)$, where P is the set of rules specifying the actions of the parties in the protocol, and Ψ is the conjunction of restrictions. There are three parties: Voter, Election Auth and BB, as represented in Figure 2.3, corresponding to Example 3, denoted by V, EA and BB in the specification, respectively. The actions of BB are limited to publishing the received messages, and EA privately sends those messages. Therefore, the party BB is omitted in the specification; instead, the messages published on BB are recorded as action facts in the rules of EA. Then, the set P will be the union of the sets of rules specifying the actions of V and EA such that $P = P_{\mathsf{V}} \cup P_{\mathsf{EA}}$. Any rule in P_{A} for $\mathsf{A} \in \{\mathsf{V}, \mathsf{EA}\}$ will then be denoted by $\mathsf{R}^{\mathsf{A}}_{\mathsf{n}}$, where n is a name describing the action of A in the rule.

Considering that there are three phases of an election in the considered protocol, the rules specify the phases in which the actions took place, i.e. they record the following action facts: $\mathsf{Phase}('\mathsf{setup}')$, $\mathsf{Phase}('\mathsf{voting}')$, and $\mathsf{Phase}('\mathsf{tally}')$. In the setup phase, EA generates an election key pair $(\mathsf{sk}_{\mathsf{E}}, \mathsf{pk}_{\mathsf{E}})$ and publishes pk_{E} on BBkey, which is modelled by the following rule:

$$\mathsf{R}^{\mathsf{EA}}_{\mathsf{key}}: \quad \mathsf{let}\ \ \mathsf{pk}_{\mathsf{E}} = \mathsf{pk}(\mathsf{sk}_{\mathsf{E}})\ \ \mathsf{in} \\ [\ \mathsf{Fr}(\mathsf{sk}_{\mathsf{E}})\ ] — [\ \mathsf{BBkey}(\mathsf{pk}_{\mathsf{E}}), \mathsf{Phase}('\mathsf{setup}')\ ] \rightarrowtail [\ !\mathsf{SkE}(\mathsf{sk}_{\mathsf{E}}), !\mathsf{PkE}(\mathsf{pk}_{\mathsf{E}})\ ]$$

In the rule $\mathsf{R}^{\mathsf{EA}}_{\mathsf{key}}$, the election's private key sk_{E} is freshly generated, and the corresponding public key pk_{E} computed by applying the function pk to sk_{E}. Then, the persistent facts SkE

and PkE record the respective keys for later use. The public key published on BB is recorded in the action fact $\mathsf{BBkey}(\mathsf{pk_E})$.

EA continues their actions for the setup, receiving election candidates from the network and publishing them on BBcand. Similarly, EA receives the voter id from the network and registers it, generating a signature key pair $(\mathsf{sk_{id}}, \mathsf{pk_{id}})$ and a tracker tr for the voter and publishing the voter's public key as a public credential cr on BBreg. The mentioned EA actions are modelled as follows:

$$\begin{aligned}
\mathsf{R}^{\mathsf{EA}}_{\mathsf{cand}} :\ & [\ \mathsf{In}(\langle \mathsf{v}_1, \mathsf{v}_2 \rangle)\ ] -\![\ \mathsf{BBcand}(\mathsf{v}_1), \mathsf{BBcand}(\mathsf{v}_2), \mathsf{Phase}('\mathsf{setup}')\ ]\!\rightarrow \\
& [\ !\mathsf{Cand}(\mathsf{v}_1), !\mathsf{Cand}(\mathsf{v}_2)\ ] \\
\mathsf{R}^{\mathsf{EA}}_{\mathsf{cred}} :\ & \mathsf{let}\ \ \mathsf{pk_{id}} = \mathsf{pk}(\mathsf{sk_{id}});\ \ \mathsf{cr} = \mathsf{pk_{id}}\ \ \mathsf{in} \\
& [\ \mathsf{In}(\mathsf{id}), \mathsf{Fr}(\mathsf{sk_{id}}), \mathsf{Fr}(\mathsf{tr})\ ] -\![\ \mathsf{BBreg}(\mathsf{cr}), \mathsf{Phase}('\mathsf{setup}')\ ]\!\rightarrow \\
& [\ !\mathsf{Cred}(\mathsf{id}, \mathsf{sk_{id}}, \mathsf{cr}, \mathsf{tr}), !\mathsf{Voter}(\mathsf{id}, \mathsf{cr}, \mathsf{tr})\ ]
\end{aligned}$$

In the above rules, two election candidates $\mathsf{v}_1, \mathsf{v}_2$ and the voter id received from the network are modelled with the fact In. For the voter id, the credentials $\mathsf{sk_{id}}$ and tr are freshly generated, cr is computed from $\mathsf{sk_{id}}$, and all the credentials are assigned to the voter being recorded in Cred. The fact Cred, generated by EA and used by V, represents the communication of credentials to the corresponding voter through a private channel.

In the voting phase, V interacts with EA to cast a ballot b, which contains the encryption of their vote v and the signature generated for the encryption. EA checks whether cr was registered for the election, verifies the signature inside the ballot, and then publishes the ballot with the voter's public credential. The following two rules model the V's action for casting a ballot and the EA's action for accepting and publishing it:

$$\begin{aligned}
\mathsf{R}^{\mathsf{V}}_{\mathsf{vote}} :\ & \mathsf{let}\ \ \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk_E}, \mathsf{r});\ \ \mathsf{s} = \mathsf{sign}(\mathsf{c}, \mathsf{sk_{id}});\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{s} \rangle\ \ \mathsf{in} \\
& [\ !\mathsf{Cred}(\mathsf{id}, \mathsf{sk_{id}}, \mathsf{cr}, \mathsf{tr}), !\mathsf{Cand}(\mathsf{v}), !\mathsf{PkE}(\mathsf{pk_E}), \mathsf{Fr}(\mathsf{r})\ ] \\
& -\![\ \mathsf{Vote}(\mathsf{id}, \mathsf{v}), \mathsf{Phase}('\mathsf{voting}')\ ]\!\rightarrow [\ !\mathsf{Voted}(\mathsf{id}, \mathsf{v}), \mathsf{Out}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{b} \rangle)\ ] \\
\mathsf{R}^{\mathsf{EA}}_{\mathsf{cast}} :\ & \mathsf{let}\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{s} \rangle\ \ \mathsf{in} \\
& [\ \mathsf{In}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{b} \rangle), !\mathsf{Voter}(\mathsf{id}, \mathsf{cr}, \mathsf{tr})\ ] \\
& -\![\ \mathsf{verify}(\mathsf{s}, \mathsf{c}, \mathsf{cr}) \equiv \mathsf{true}, \mathsf{BBcast}(\mathsf{cr}, \mathsf{b}), \mathsf{Phase}('\mathsf{voting}')\ ]\!\rightarrow [\ !\mathsf{Cast}(\mathsf{cr}, \mathsf{b})\ ]
\end{aligned}$$

The rule $\mathsf{R}^{\mathsf{V}}_{\mathsf{vote}}$ records the voting action of V with the fact Vote(id, v), outputting the respective ballot b with their credentials to the network. On the other hand, the rule $\mathsf{R}^{\mathsf{EA}}_{\mathsf{cast}}$ checks the voter's eligibility, matching the received credentials with the ones in Voter and verifies the signature matching the equality in the middle part of the rule before recording the action of EA for publishing the ballot with the fact BBcast(cr, b).

In the tally phase, EA retrieves the V's ballot from Cast, decrypts the ballot with the election's private key recorded in SkE, and publishes the output with the tracker for V on BBres. Then, V verifies their vote by matching it with the one next to their tracker on BBres. The EA's action for tally and V's for vote verification are modelled with the following rules:

$$\begin{aligned}
\mathsf{R}^{\mathsf{EA}}_{\mathsf{tally}} :\ & \mathsf{let}\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{s} \rangle;\ \ \mathsf{v}' = \mathsf{dec}(\mathsf{c}, \mathsf{sk_E})\ \ \mathsf{in} \\
& [\ !\mathsf{Cast}(\mathsf{cr}, \mathsf{b}), !\mathsf{SkE}(\mathsf{sk_E}), !\mathsf{Voter}(\mathsf{id}, \mathsf{cr}, \mathsf{tr})\ ] \\
& -\![\ \mathsf{BBres}(\mathsf{tr}, \mathsf{v}'), \mathsf{Phase}('\mathsf{tally}')\ ]\!\rightarrow [\ !\mathsf{Res}(\mathsf{tr}, \mathsf{v}')\ ] \\
\mathsf{R}^{\mathsf{V}}_{\mathsf{verify}} :\ & [\ !\mathsf{Cred}(\mathsf{id}, \mathsf{sk_{id}}, \mathsf{cr}, \mathsf{tr}), !\mathsf{Voted}(\mathsf{id}, \mathsf{v}), !\mathsf{Res}(\mathsf{tr}, \mathsf{v}')\ ] \\
& -\![\ \mathsf{v} \equiv \mathsf{v}', \mathsf{Verified}(\mathsf{id}, \mathsf{v}), \mathsf{Phase}('\mathsf{tally}')\ ]\!\rightarrow [\ ]
\end{aligned}$$

The rule $\mathsf{R}^{\mathsf{V}}_{\mathsf{verify}}$ records the action of verification with the fact Verified(id, v) after verifying the vote equality.

The specification of the actions V and EA has been completed with the following set of rules described above:

$$\begin{aligned} P_{\mathsf{V}} &= \{\mathsf{R}^{\mathsf{V}}_{\mathsf{vote}}, \mathsf{R}^{\mathsf{V}}_{\mathsf{verify}}\}, \\ P_{\mathsf{EA}} &= \{\mathsf{R}^{\mathsf{EA}}_{\mathsf{key}}, \mathsf{R}^{\mathsf{EA}}_{\mathsf{cand}}, \mathsf{R}^{\mathsf{EA}}_{\mathsf{cred}}, \mathsf{R}^{\mathsf{EA}}_{\mathsf{cast}}, \mathsf{R}^{\mathsf{EA}}_{\mathsf{tally}}\}. \end{aligned}$$

Now, we specify the restrictions for the ordering of the phases and the uniqueness of the election key as follows:

$$\begin{aligned} \Psi_{\mathsf{order}_1} &: \quad \forall i, j.\ \mathsf{Phase}('\mathsf{setup}')\ @i \ \wedge\ \mathsf{Phase}('\mathsf{voting}')\ @j \Rightarrow i \prec j \\ \Psi_{\mathsf{order}_2} &: \quad \forall i, j.\ \mathsf{Phase}('\mathsf{voting}')\ @i \ \wedge\ \mathsf{Phase}('\mathsf{tally}')\ @j \Rightarrow i \prec j \\ \Psi_{\mathsf{key}} &: \quad \forall \mathsf{x}, \mathsf{y}, i, j.\ \mathsf{BBkey}(\mathsf{x})\ @i \ \wedge\ \mathsf{BBkey}(\mathsf{y})\ @j \Rightarrow \mathsf{x} = \mathsf{y} \end{aligned}$$

The restrictions Ψ_{order_1} and Ψ_{order_2} constrain the execution traces with respect to the expected phase ordering, and Ψ_{key} unifies all the election keys generated during the election, i.e. there can be at most one. Then, the restriction Ψ for P will be the conjunction of the restrictions above, i.e. $\Psi = \Psi_{\mathsf{order}_1} \wedge \Psi_{\mathsf{order}_2} \wedge \Psi_{\mathsf{key}}$. Thus, the specification of the protocol described in Example 3 will be the following:

$$(P; \Psi) = (P_{\mathsf{V}} \cup P_{\mathsf{EA}}; \Psi_{\mathsf{order}_1} \wedge \Psi_{\mathsf{order}_2} \wedge \Psi_{\mathsf{key}}).$$

Assume we want to verify a trace property Φ_{valid} that ensures the validity of the ballot published on BBcast. Specifically, we want to check whether the ballot contains an encryption of a valid vote on BBcand with the expected election public key, recorded on BBkey. The property Φ_{valid} can be formulated as follows:

$$\begin{aligned} \Phi_{\mathsf{valid}} &: \quad \forall \mathsf{cr}, \mathsf{c}, \mathsf{s}, i.\ \mathsf{BBcast}(\mathsf{cr}, \langle \mathsf{c}, \mathsf{s} \rangle)\ @i \\ &\qquad \Rightarrow \exists \mathsf{k}, \mathsf{v}, \mathsf{r}, j, l.\ \mathsf{BBkey}(\mathsf{k})\ @j \ \wedge\ \mathsf{BBcand}(\mathsf{v})\ @l \ \wedge\ \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{k}, \mathsf{r}). \end{aligned}$$

For the verification of the property Φ_{valid} with respect to the protocol specification $(P; \Psi)$, Tamarin checks all the traces of $(P; \Psi)$ and concludes $(P; \Psi) \vDash \Phi_{\mathsf{valid}}$. Here, the ballot on BBcast is received from the network, leaving a possibility for the adversary's intrusion. However, the signature on the ballot with respect to a valid credential protects the ballot against any alteration by the adversary. Moreover, the model specifies only honest voters preparing a ballot selecting a candidate from BBcand, and encrypting it using the public key from BBkey. Therefore, the property Φ_{valid} holds as expected, ensuring the ballot validity. The ballot validity ensures that the election outcome will correspond to legitimate candidates.

3.2 Formal Verification with ProVerif

ProVerif is an automated verification tool [45, 11] whose language, a variant of the applied pi calculus, allows to specify protocols as concurrent processes specifying the actions of the parties that generate messages and exchange them via channels while recording events to represent those actions. Security properties are specified as trace properties that are checked against all possible traces of the transition system, or specified as equivalence properties that are checked in terms of the observational equivalence of two transition systems. Given a symbolic model and properties, ProVerif automatically constructs a proof or presents a counterexample representing an attack for a property specified within the protocol, instantiating processes for an unbounded number of times in parallel.

3.2.1 Specifying Protocols and Properties in ProVerif

In ProVerif, messages are represented in an equational theory where terms are built from a set of types $\mathcal{T}$, a set of names $\mathcal{N}$, a set of constants, a set C of constructors endowed with a set D of destructors. Types are not essential for describing the protocol specifications in this thesis, therefore, they will be skipped. A tuple is a term of the form $(M_1, ..., M_n)$, where $M_1, ..., M_n$ are terms. A constructor is a function symbol f used to build a term $f(M_1, ..., M_k)$, from terms $M_1, ..., M_k$. A destructor is a symbol with associated rewrite rules modelling the properties of cryptographic primitives. Rewrite rules are equations $u = v$ that should be read from left to right. For example, a destructor $proj_i$ can be defined to fetch the arguments of f: $proj_i(f(M_1, ..., M_k)) = M_i$. If a term contains a destructor for which no rewrite rule can be applied, it is denoted by $t = \bot$, i.e. the evaluation of the term fails. A public communication channel is represented by a special name $pub \in \mathcal{N}$ in this thesis. Unless declared private, names and functions are known by the adversary. A term M can be stored in a table, e.g. Tb(M) for a table named Tb. Tables are based on private constructors and are not accessible by the adversary. Terms can be checked with equality or disequality conditions, and such conditions may be combined with logical connectors, see Figure 3.1.

Example 7. *The set of constructors* $C = \{pk, enc, sign\}$ *can model cryptographic primitives along with the corresponding destructors defined by:*

$$(1)\quad dec(enc(x, pk(y), z), y) = x,$$
$$(2)\quad verify(sign(x, y), x, pk(y)) = true.$$

Equations (1) and (2) specify the standard properties of asymmetric encryption and digital signatures, as also specified in Example 4. Cryptography is assumed perfect: there is no way to derive messages other than by applying function symbols and equations.

Processes are denoted by P, Q, …, and generated with the grammar from Figure 3.1. Recursively, 0 represents the null process, P | Q the parallel execution of P and Q, and !P an arbitrary number of parallel executions of P. The restriction new n ; P generates the name n, which can represent fresh randomness in the process P. The process in(pub, x) ; P models a message received from channel pub and stored in x, while the process out(pub, M) ; P models a process sending the message M on channel pub and continuing as P. The conditional if M then P else Q runs P if M evaluates to true, runs Q if M = false. If a term M contains a destructor where no rewrite rule can be applied, M fails. The term evaluation let x = M in P else Q, bounds x to M and executes P, when M does not fail. If M fails, it directly executes the process Q. The table insertion insert Tb(M) ; P stores the term M in Tb. The table extraction get Tb(x) suchthat M in P else Q selects an entry from Tb that evaluates to M. If there is no such entry, it executes Q.

Example 8. *The processes in Figure 3.2 model the session key exchange protocol between server and client corresponding to the Tamarin specification described in Example 5. The facts in Tamarin are modelled by tables in ProVerif. For example, the server's secret key is stored in the table* Sk*, while its public key is in* Pk*. The fresh terms are generated by* new*, e.g.* k, s, r*. The protocol is executed on the public channel. The server sends its public key to client, who generates a session key* s *and uses the public key received from the channel to encrypt it, while recording events* Key *and* Enc *for the session key and the encryption of it. Then, the client sends the encryption to the server, which retrieves its secret key* k *from the table* Sk *and decrypts the ciphertext recording an event* Dec*. They both store the session key on the tables* SessionKeyC *and* SessionKeyS*.*

Security Properties. ProVerif can prove reachability properties, correspondence assertions, and observational equivalence. *Reachability queries* can be stated by e(M) for a term M and

M, N :=	terms
a, b, c, ...	names
x, y, z, ...	variables
$(M_1, ..., M_k)$	tuples
$f(M_1, ..., M_k)$	functions
M = N	term equality
M <> N	term disequality
M && N	conjunction
M || N	disjunction
not(M)	negation
P, Q :=	processes
0	null process
P | Q	parallel composition
!P	replication
new n ; P	name restriction
in(pub, x) ; P	message input
out(pub, M) ; P	message output
if M then P else Q	conditional
let x = M in P else Q	term evaluation
insert Tb(M) ; P	table insertion
get Tb(x) suchthat M in P else Q	table extraction
event $e(M_1, ..., M_k)$; P	event

FIGURE 3.1: Terms and processes in ProVerif grammar.

proved by searching all the traces of a process P for the event's occurrence. A reachability query can be stated for the adversarial knowledge of a term such that attacker(M), in which ProVerif tests the secrecy of the term. *Correspondence assertions* capture the relations between events. For example, the formula $e(M) \Rightarrow e'(M')$ states that if the event e(M) is executed in the trace, then the event $e'(M')$ has been executed before in that trace for the terms M and M'. Queries can be more complex, combining events with relations between terms inside the events, e.g. $M = M'$ can be added on the right-hand side of the previous query. A query may also use temporal variables to specify the timepoints where events occurred in a trace. The event $e(M)@i$ represents the event occurring at the timepoint i. A trace property is a first-order logic formula over events and timepoints. Thus, the formula is obtained from logical connectives such as implication, conjunction, disjunction, and negation, event constraints such as $e(M)@i$, temporal ordering $i \leq j$ or equality $i = j$ for temporal variables i and j, and term equalities $M = M'$ for the terms M and M'. The timepoint can be omitted from the event if it is not relevant for the property in the following. Similarly, a restriction is a logical formula that constrains the execution traces with respect to the events. *Observational equivalence* refers to the indistinguishability of the two processes, i.e. the adversary cannot distinguish a process P from another process Q. The observational equivalence of the two processes P and Q is denoted by $P \approx Q$.

Example 9. *Considering the processes from Figure 3.2, the secrecy of the session key can be queried by* attacker(s). *Moreover, the correspondence assertion for the events* Enc(x) *and* Dec(x) *can be formulated as follows:* Dec(x) ⇒ Enc(x). *This formula checks whether for each* x *occurred in the event* Dec *there is a corresponding event* Enc *for the same term that occurred before in the same trace.*

```
let ServerKey =
  new k;
  insert Sk(k);
  insert Pk(pk(k));
  out(pub, (pk(k))).

let ServerDec =
  get Sk(k) in
  in(pub, y);
  event Dec(y);
  insert SessionKeyS(dec(y, k)).
```

```
let ClientEnc =
  in(pub, x);
  new s;
  event Key(s);
  insert SessionKeyC(s);
  new r;
  let c = enc(s, x, r) in
  event Enc(x);
  out(pub, c).
```

FIGURE 3.2: Session key exchange between client and server.

An execution trace τ of a process P is defined as a sequence of events $\mathsf{e}_1, ..., \mathsf{e}_n$ that occurred during the execution of P. The satisfaction relation $\tau \vDash \Phi$, for a trace τ and a trace formula Φ, is defined recursively applying the usual semantics of logical and ordering connectives. We have the following notable case for events: $\tau \vDash \mathsf{e}(\mathsf{M})@i$ if and only if $\mathsf{e}(\mathsf{M})$ occurs on the i-th step of the trace. For a process P, let $\mathsf{tr}(\mathsf{P})$ be the set of traces of P. Then, for a trace formula Φ, $\mathsf{P} \vDash \Phi$ if and only if any trace in $\mathsf{tr}(\mathsf{P})$ satisfies the formula Φ. Moreover, let $(\mathsf{P}; \Psi)$ be the specification of P with a restriction formula Ψ. Then, $(\mathsf{P}; \Psi) \vDash \Phi$ if and only if any trace in $\mathsf{tr}(\mathsf{P})$ satisfying the restriction Ψ also satisfies the trace formula Φ. That is:

$$\begin{aligned} \mathsf{P} \vDash \Phi &\iff \forall \tau \in \mathsf{tr}(\mathsf{P}).\ \tau \vDash \Phi, \\ (\mathsf{P}; \Psi) \vDash \Phi &\iff \forall \tau \in \mathsf{tr}(\mathsf{P}).\ \tau \vDash \Psi \Rightarrow \Phi. \end{aligned}$$

3.2.2 Case Study: Verifiable E-Voting Example

We present the ProVerif specification of the verifiable e-voting Example 3 from Section 2.5 in the following. Similar to its Tamarin specification, we use the two equations defined in Example 7 to represent its encryption and digital signature algorithms with the constructors $\mathsf{pk}, \mathsf{enc}, \mathsf{sign}$ and the destructors $\mathsf{dec}, \mathsf{verify}$:

$$\begin{aligned} &(1) \quad \mathsf{dec}(\mathsf{enc}(\mathsf{x}, \mathsf{pk}(\mathsf{y}), \mathsf{z}), \mathsf{y}) = \mathsf{x}, \\ &(2) \quad \mathsf{verify}(\mathsf{sign}(\mathsf{x}, \mathsf{y}), \mathsf{x}, \mathsf{pk}(\mathsf{y})) = \mathsf{true}. \end{aligned}$$

The ProVerif specification $(\mathsf{P}; \Psi)$ of Example 3 is similar to its Tamarin specification, but here P denotes the process specifying the actions of the parties in the protocol. The parties Voter and Election Auth are similarly denoted by V and EA, and the party BB is omitted in the specification. The messages published on BB will be recorded as events in the processes of EA. Then, the process P is defined by the parallel executions of the subprocesses specifying the actions of V and EA such that $\mathsf{P} = \mathsf{P}_\mathsf{V} \,||\, \mathsf{P}_{\mathsf{EA}}$. Any process in P_A for $\mathsf{A} \in \{\mathsf{V}, \mathsf{EA}\}$ will then be denoted by An, where A is the abbreviation of the party, and n is a name describing the action of A within the process.

Considering the three phases of an election, the processes modelling the actions of EA in the setup phase are described in Figure 3.3. In the setup phase, EA generates an election key pair $(\mathsf{sk}_\mathsf{E}, \mathsf{pk}_\mathsf{E})$ and stores the election's private key and public key in the tables SkE and PkE, respectively. Then, EA publishes pk_E on BB, which is recorded in the event BBkey. The generation of the election key pair and the related actions are modelled in the process EAkey. Setting the candidates for the election is modelled in the process EAcand, in which

EA receives the election candidates from the public channel, stores them in the table Cand and publishes them on BBcand, recording the events BBcand for the respective candidates.

```
let EAkey =
  new sk_E;
  let pk_E = pk(sk_E) in
  event BBkey(pk_E);
  insert SkE(sk_E);
  insert PkE(pk_E).

let EAcand =
  in(pub, (v_1, v_2));
  event BBcand(v_1);
  event BBcand(v_2);
  insert Cand(v_1);
  insert Cand(v_2).

let EAcred =
  in(pub, id);
  new sk_id;
  let pk_id = pk(sk_id) in
  let cr = pk_id in
  new tr;
  event BBreg(cr);
  insert Cred(id, sk_id, cr, tr);
  insert Voter(id, cr, tr).
```

FIGURE 3.3: Processes specifying the setup phase of Example 3.

The voter registration and the election credentials generation for the respective voter are modelled in the process EAcred. EA receives the voter id from the public channel, generates a signature key pair (sk_{id}, pk_{id}) and a tracker tr for the voter, and publishes the voter's public key as a public credential cr on BB, recording the event BBreg. EA also stores all the credentials of the voter id in the table Cred that will be extracted by V later, which represents the communication of credentials to the voter via a private channel.

The processes modelling the actions of V and EA in the voting phase are described in the processes Vvote and EAcast, respectively, in Figure 3.4. In the voting phase, V extracts the voter credentials, the vote, and the election's public key from the tables Cred, Cand, and PkE, respectively. Then, V uses the election's public key to generate a ciphertext of the vote, and the signing key sk_{id} to sign the ciphertext, i.e. generates a ballot in the form $b = (c, s)$. The action for voting is recorded in the event Vote, and the vote v is stored in the table Voted for the voter id. The ballot b is sent to EA with the voter credentials attached to it. EA checks voter's eligibility, i.e. matches id and cr with the ones stored in Voter. Then, EA verifies the signature inside the ballot and publishes the ballot with the voter's public credential on BB, recording the event BBcast. The table Cast stores the ballot b for the voter id.

```
let Vvote =
  get Cred(id, sk_id, cr, tr) in
  get Cand(v) in
  get PkE(pk_E) in
  new r;
  let c = enc(v, pk_E, r) in
  let s = sign(c, sk_id) in
  let b = (c, s) in
  event Vote(id, v);
  insert Voted(id, v);
  out(pub, (id, cr, b)).

let EOcast =
  get Voter(id, cr, tr) in
  in(pub, (= id, = cr, b));
  let b = (c, s) in
  if verify(s, c, cr) = true then
  event BBcast(cr, b);
  insert Cast(cr, b).
```

FIGURE 3.4: Processes specifying the voting phase of Example 3.

The processes modelling the actions of EA and V in the tally phase are described in the processes EAtally and Vverify, respectively, in Figure 3.5. In the tally phase, EA retrieves the election's secret key, the voter credentials, and the voter's ballot from the tables SkE, Voter, and Cast, respectively. Then, EA decrypts the ciphertext inside the ballot with the election's secret key and publishes the output with the voter's tracker on BB, recording the event BBres and storing them in the table Res. V verifies their vote stored in Voted by matching it with the one next to their tracker stored in Res. The action for the verification of the vote is recorded in the event Verified.

```
let EOtally =
  get SkE(sk_E) in
  get Voter(id, cr, tr) in
  get Cast(= cr, b) in
  let b = (c, s) in
  let v' = dec(c, sk_E) in
  event BBres(tr, v');
  insert Res(tr, v').

let Vverify =
  get Cred(id, sk_id, cr, tr) in
  get Voted(id, v) in
  get Res(tr, v') in
  if v = v' then
  event Verified(id, v).
```

FIGURE 3.5: Processes specifying the tally phase of Example 3.

The specification of the actions V and EA will be the parallel executions of the respective subprocesses described above:

$$\begin{aligned} P_V &= \mathsf{Vvote} \mid\mid \mathsf{Vverify}, \\ P_{EA} &= \mathsf{EAkey} \mid\mid \mathsf{EAcand} \mid\mid \mathsf{EAcred} \mid\mid \mathsf{EAcast} \mid\mid \mathsf{EAtally}. \end{aligned}$$

The restriction $\Psi_{key} : \mathsf{BBkey}(x) \wedge \mathsf{BBkey}(y) \Rightarrow x = y$ is specified to ensure the uniqueness of the election key. Thus, the ProVerif specification of Example 3 is:

$$(P; \Psi) = (P_V \mid\mid P_{EA}; \Psi_{key}).$$

We can similarly verify the property $\Phi_{valid} : \mathsf{BBcast}(cr, (c, s)) \Rightarrow \mathsf{BBkey}(k) \wedge \mathsf{BBcand}(v) \wedge c = \mathsf{enc}(v, k, r)$ to ensure the validity of the ballot published on BBcast. For the verification of the property Φ_{valid} with respect to the protocol specification $(P; \Psi)$, ProVerif checks all the traces of $(P; \Psi)$ and concludes $(P; \Psi) \vDash \Phi_{valid}$.

3.3 Generic Protocol Specifications and Properties

We have seen that both Tamarin and ProVerif rely on a protocol specification $(P; \Psi)$, where P is a set of rules or processes recording the action facts or events in the execution traces of P, respectively, and Ψ is the restriction that constraints the execution traces with respect to those action facts or events and timepoints. The properties to be verified are first-order logic formulas over those facts or events and the timepoints where they occur. For a property Φ, $(P; \Psi) \vDash \Phi$ if and only if any trace in the set $\mathsf{tr}(P)$ of all traces of P satisfies the formula Φ, i.e.

$$(P; \Psi) \vDash \Phi \iff \forall \tau \in \mathsf{tr}(P).\ \tau \vDash \Psi \Rightarrow \Phi.$$

Therefore, in the rest of the thesis, we will consider generic protocol specifications $S = (P; \Psi)$, whose executions generate a set of traces with sets of action facts or events associated to each timepoint. Properties Φ can then be evaluated over the sets of traces of such a specification.

For notation, we omit the quantifiers $\exists, \forall$, the prefix $@$, and the declaration of variables from the properties and restrictions, unless stated otherwise. In general, a formula will be of the form $\Phi_1 \Rightarrow \Phi_2$, where all variables in Φ_1 are quantified universally, and all variables in Φ_2, which are not in Φ_1, are quantified existentially. Thus, a simplified formula $\mathsf{F}_1(\mathsf{x}_1, \mathsf{x}_2) \Rightarrow \mathsf{F}_2(\mathsf{y}_1, \mathsf{y}_2)$ represents $\forall \mathsf{x}_1, \mathsf{x}_2, i.\ \mathsf{F}_1(\mathsf{x}_1, \mathsf{x}_2)@i \Rightarrow \exists \mathsf{y}_1, \mathsf{y}_2, j.\ \mathsf{F}_2(\mathsf{y}_1, \mathsf{y}_2)@j$. Moreover, we represent the logical connectives used in the trace formulas by the same symbols for Tamarin and ProVerif, i.e. conjunction by $\wedge$, disjunction by $\vee$, the negation by $\neg$, and term disequality by $\mathsf{M} \neq \mathsf{M}'$ for the terms M and M'. For example, the property

$$\begin{aligned}\Phi_{\mathsf{valid}} :\ & \forall \mathsf{cr}, \mathsf{c}, \mathsf{s}, i.\ \mathsf{BBcast}(\mathsf{cr}, \langle \mathsf{c}, \mathsf{s} \rangle)\ @i \\ & \Rightarrow \exists \mathsf{k}, \mathsf{v}, \mathsf{r}, j, l.\ \mathsf{BBkey}(\mathsf{k})\ @j\ \wedge\ \mathsf{BBcand}(\mathsf{v})\ @l\ \wedge\ \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{k}, \mathsf{r})\end{aligned}$$

specified for the verifiable e-voting example is simplified as:

$$\Phi_{\mathsf{valid}} : \mathsf{BBcast}(\mathsf{cr}, \langle \mathsf{c}, \mathsf{s} \rangle) \Rightarrow \mathsf{BBkey}(\mathsf{k})\ \wedge\ \mathsf{BBcand}(\mathsf{v})\ \wedge\ \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{k}, \mathsf{r}).$$

Chapter 4

Formal Definition of Election Verifiability

Election verifiability has emerged as a requirement for the security of the e-voting protocols, aiming to ensure the election outcome corresponds to the votes recorded in the digital ballot box or on the bulletin board. Later, it evolved to cover ballot casting assurance for honest voters and prohibit ballot stuffing for non-eligible voters. For ballot casting assurance, the protocol provides voters with some means of individual verification of their votes on the bulletin board, which will assure them their ballot will be tallied, i.e. their vote will be counted on their behalf. On the other hand, to prevent ballot stuffing, only the ballots from eligible credentials, i.e. the ones registered at the beginning of the election, are recorded on the bulletin board, and one vote per such credential is counted in the election result.

In this chapter, we first present the existing notions and definitions of election verifiability in the literature, corresponding to the evolution of the verifiability property mentioned above. In particular, we focus on the verifiability definition in [20] that we call the multiset-based definition since it is based on counting the votes in the election outcome as multisets of votes and the symbolic definition in [17] since it is the most recent in the literature and allows automated verification with Tamarin/ProVerif.

The multiset-based definition in [20] partitions the votes in the outcome into the votes from honest voters who verified their votes, the votes from honest voters who did not verify, and the votes from corrupt voters that may be cast by the adversary. Therefore, it ensures that the votes from honest voters, if verified, are counted in the result, and the adversary can cast ballots only for corrupt voters, which entails end-to-end verifiability. This definition can be extended in a few ways. Indeed, in some cases, corrupt voters may verify their votes, unaware of the corruption of their credentials. Their votes should also be counted in the result. For some protocols or adversarial models, the adversary could be stronger to cast ballots even for honest voters, if they did not perform the individual verification procedure. Accounting for stronger individual verifiability and a stronger adversary, we extend the multiset-based definition to cover stronger or weaker notions of end-to-end verifiability.

The symbolic definition in [17] is too specific to a variant of Belenios, which does not allow revoting and requires different trust assumptions based on two main corruption scenarios: a corrupt registrar and a corrupt server, where both are not corrupt at the same time. Therefore, the symbolic definition, i.e. the conditions that achieve end-to-end verifiability, vary according to the corruption case, which results in two different proofs. This means it cannot be directly applied to the protocols allowing revoting or having a different architecture, parties, and infrastructure components. To cover a broad class of e-voting protocols, we improve the symbolic definition in [17] and propose a general symbolic election verifiability that accounts for revoting, allows automated verification, and is independent of the corruption scenarios and the protocol. Our symbolic definition is sound and can be instantiated according to the level of end-to-end verifiability expected to hold in the protocol, i.e. it guarantees

the stronger and/or weaker notions of end-to-end verifiability, implying the extended multiset-based verifiability definition we propose in this chapter. Also, it applies to various e-voting protocols, such as Helios, Belenios, Selene, and Estonian e-voting protocols, as we show in the second part of the thesis.

Structure of the chapter: Section 4.1 provides the existing notions and definitions of election verifiability. In Section 4.2, we define the general e-voting events that are recorded in the execution traces of an e-voting protocol. The e-voting events help us to propose the extended version of the multiset-based verifiability definition in Section 4.3 and the symbolic verifiability definition in Section 4.4. Section 4.5 provides the soundness proof of the symbolic verifiability definition, i.e. it implies the multiset-based election verifiability. Finally, in Section 4.5, we apply our symbolic verifiability definition to an example of verifiable e-voting protocols.

4.1 Existing Notions and Definitions

In this section, we will go through the existing notions of election verifiability and the multiset-based definition resulting from all these notions. Then, we will focus on the symbolic verifiability definitions in the literature and discuss their limitations, which motivate us to propose our definition in the following sections.

4.1.1 Informal Notions and Multiset-based Definition

The security of e-voting protocols is based on cryptographic primitives deployed by the protocol as well as the behaviours of the election authorities. Cryptographic primitives provide, if they are correctly deployed, the correctness of all the operations. They can be considered functions; given an input, they generate the corresponding output. However, if one is able to make some changes to the input, such as a corrupt election authority, then the protocol security may be affected even if the cryptography deployed is correct. Therefore, besides the security of cryptographic primitives, the correct behaviours of the election authorities should be ensured. This is the point where the election verifiability emerges.

Election verifiability ensures that all the election operations and procedures achieve their goal, even though some parties in the protocol are corrupt. In other words, the corrupt parties cannot misbehave in their particular actions if the election procedures are verifiable. In a typical election procedure of an e-voting protocol, the voters generate a ballot encrypting their vote using a voting platform, which sends their ballots to the election authorities that will record them on the bulletin board. At the end of the election, the set of ballots on the bulletin board is tallied into a set of votes, which is also recorded on the bulletin board as the election result. Considering these procedures, the correctness of the tally should be ensured. Furthermore, if a voter casts a ballot, the voter should be able to ensure that the ballot is delivered to the authorities and recorded on the bulletin board.

Correctness of the tally and the guarantees provided to the voters for their vote cast are referred to in the literature [37, 15, 38] as the following two notions of election verifiability:

- ***Individual verifiability:*** a voter is able to verify that the voting server correctly records their ballot, i.e. their vote.

- ***Universal verifiability:*** anyone is able to verify that the outcome corresponds to the ballots recorded on the bulletin board.

Kremer et al. added another notion of verifiability [38] to the notions above as follows:

- ***Eligibility verifiability:*** anyone is able to verify that any vote in the outcome was cast by an eligible voter, i.e. a registered voter, and there is at most one vote per voter.

Eligibility ensures that all the ballots recorded on the bulletin board were legitimately cast, i.e. the ballots from non-eligible voters are not recorded on the bulletin board. Thus, it limits ballots stuffing, restricting the adversary's abilities to cast ballots only for eligible corrupt voters.

The voter's ability to verify their ballot was correctly cast and recorded on the bulletin board, and anyone's ability to verify that all the votes in the outcome corresponds to ballots cast by eligible voters, i.e. verifiability provided in all election procedures from ballot casting to the tally procedure is called ***end-to-end verifiability***. In any end-to-end verifiable election, the eligible voters can be divided into honest and corrupt. In [38], all honest voters are assumed to verify their votes, which is not realistic. Some honest voters verify their votes, and others do not. In general, end-to-end verifiability guarantees that the votes from honest voters who successfully verified their votes will be counted on their behalf. However, it does not provide any guarantee for honest voters who did not verify their votes. Therefore, the adversary may drop the ballots from some of those voters. Considering these aspects, Cortier et al. improved the end-to-end verifiability definition from [38] and proposed a multiset-based definition of election verifiability in [20]. According to this definition, any e-voting protocol is verifiable if the votes in the election outcome can be partitioned into the three multisets of votes as follows:

1. the votes cast by honest voters that verified their vote was cast correctly;
2. a subset of the votes cast by honest voters that did not verify their vote was cast correctly;
3. at most n valid votes where n is the number of corrupt voters.

For generality, we will show that it is useful to consider also different partitions. For example, the votes from corrupt voters, if verified, could be counted in the result, and this can be ensured by some protocols and scenarios. This may also be desirable. Assume the voter has leaked their credentials unwittingly and thus becomes corrupt according to the formal definition. In this case, if the voter verifies their vote successfully, their vote should also be counted. On the other hand, some scenarios may allow a stronger adversary to cast ballots for the voters who did not verify their votes. In this sense, the adversary would not be limited to casting ballots only for corrupt voters. Thus, we will extend the multiset-based definition in Section 4.3, allowing us to specify different levels of individual verifiability and ballot stuffing restrictions for the adversary.

4.1.2 Existing Symbolic Definitions

There are two approaches developed to analyse protocols and prove their security. The analysis can be performed either on the computational model, focusing on the cryptographic primitives deployed by the protocol, or on the symbolic model, abstracting the cryptographic primitives and focusing on the logical flow of the protocol. In this thesis, we adopt the approach of symbolic modelling, which also allows us to perform automated verification using the tools Tamarin or ProVerif. Thus, we go through the existing symbolic definitions of election verifiability in the literature, specifically the ones from [38, 22, 17].

The first general symbolic definition, defined in [38], formalises individual, universal, and eligibility verifiability as a triple of boolean tests. A boolean test corresponds to the conjunctions and disjunctions of term equalities in applied pi-calculus. The symbolic protocol models are subjected to boolean tests to ensure verifiability. The definition is applied to the protocols Helios [1], FOO [28], and Civitas [16]. This proves the generality of the definition.

However, the definition requires all honest voters to verify their votes, which is not realistic. Also, the boolean tests cannot be expressed with the tools that allow automated verification, such as Tamarin and ProVerif.

The type-based symbolic definition in [22] formalises individual, universal, and end-to-end election verifiability as logical formulas to be checked by a type-checker (F*). It also defines a formula to ensure no clash attacks on the protocol and proves that if no clash formula holds, individual and universal verifiability entail end-to-end verifiability. As a case study, Helios is modelled in two versions; one version with mixnets and the other with the homomorphic tally, and both proved secure with respect to end-to-end verifiability. The analysis is also made with respect to privacy for the version with the homomorphic tally. Thus, the paper [22] proposes the first automated verification of Helios with the homomorphic tally. However, the definition does not account for ballot stuffing, allowing non-eligible voters to cast ballots. It also does not capture revoting, and is not suitable for automated verification tools like Tamarin and ProVerif. In the following, we give more details about clash attacks and the property proposed in [22] that ensures no clash attacks.

Clash attacks [40]: These are the attacks against the verifiability of e-voting protocols, causing a clash on the public credentials of the voters and their respective ballots. In this attack, the adversary targets a number of voters who will vote for the same candidate. Then, it corrupts the registrar responsible for generating public credentials to distribute the same credential to those voters. Moreover, the adversary corrupts the voting platforms of those voters so that those platforms use the same randomness to generate a ciphertext of the vote, i.e. they generate the same ballot. Furthermore, the adversary corrupts the voting server to record only a single ballot on the BB for the public credential shared by the voters. Thus, the votes of targeted voters are counted as a single vote, where the adversary is able to cast votes for different credentials as many as the number of targeted voters (minus 1). This attack will not be detected by observing the public bulletin board nor by the voters who share the same credential since they are able to verify their ballot on the bulletin board. The following property was described in [22] to ensure no clash on the ballots of the two voters:

$$\Phi_{\mathsf{cl}} : \mathsf{MyBallot}(\mathsf{id}_1, \mathsf{v}_1, \mathsf{b}) \ \wedge \ \mathsf{MyBallot}(\mathsf{id}_2, \mathsf{v}_2, \mathsf{b}) \Rightarrow \mathsf{id}_1 = \mathsf{id}_2 \ \wedge \ \mathsf{v}_1 = \mathsf{v}_2$$

However, this property is not sufficient to capture new versions of clash attacks with revoting. For example, when revoting is allowed, the adversary does not need to corrupt the voting platforms of the targeted voters or the voting server. Just with a corrupt registrar, a similar attack can be mounted without requiring the clash on the ballots. For example, assume that the individual verification is allowed anytime during the voting phase. The voters who receive the same public credential from the registrar may cast their individual ballots and verify them successfully on the BB. The procedure of a voter for casting a ballot and verifying it on the BB can be followed by the procedure of another voter. This will look like revoting from the public view of BB. However, in the election result, one vote will be counted for the voters who share the same credential, even though each verifies their ballot on BB. In this attack, the clash is not on the ballots but on the public credentials of the voters who verify. Therefore, the formula Φ_{cl} should be improved regarding a general class of clash attacks.

The symbolic definition from [17] is closest to our goal, being the most recent in the literature and allowing automated verification. Thus, we present the definition in detail and discuss its limitations in the following.

Symbolic definition from [17]: Cortier et al. formalise end-to-end election verifiability as logical formulas corresponding to the notions of recorded-as-intended, individual, and eligibility verifiability in [17], proving that these formulas entail multiset-based end-to-end verifiability

similar to the one in [20]. The definition is formulated for Belenios-like protocols, where the registrar and the server are not simultaneously corrupt. Belenios-like protocols require a registrar to distribute election credentials to the voters in addition to the login credentials provided by the voting server. In the case of a corrupt registrar, the voting server is assumed to be honest. It can be trusted to ensure some consistency properties between voter identities and cast ballots, i.e. it does not accept two ballots for the same identity. This gives rise to a variant of the verifiability definition that is called identity-based. On the contrary, in the case of a corrupt voting server, the registrar is assumed to be honest and trusted to ensure the consistency between identities and election credentials of the voters. Since revoting is not allowed, the corrupt server cannot accept two ballots for the same credential. Therefore, the credentials are reliable in this case to define credential-based verifiability. Thus, their definition requires two different sets of formulas to achieve verifiability for the two main corruption scenarios of a corrupt registrar and a corrupt server, respectively.

The definition from [17] considers the following events of a protocol:

- $\mathsf{Voter}(\mathsf{id}, \mathsf{cr}, \ell)$: A voter id has been registered with the credential cr and labelled with ℓ, where $\ell \in \{\mathsf{H}, \mathsf{D}\}$, representing an honest voter or a dishonest (corrupt) voter. A dishonest voter leaks all his credentials to the adversary.
- $\mathsf{Vote}(\mathsf{id}, \mathsf{v})$: The voter id has cast a vote v.
- $\mathsf{GoingToTally}(\mathsf{id}, \mathsf{cr}, \mathsf{b})$: The voting server has recorded the ballot b on the bulletin board after associating it with id and cr.
- $\mathsf{Verified}(\mathsf{id}, \mathsf{v})$: The voter id has verified their vote v.

It defines three notions forming end-to-end verifiability: recorded-as-intended, eligibility verifiability and individual verifiability. Recorded-as-intended property, i.e. a short form of recorded-as-cast and cast-as-intended together, ensures that the ballots from honest voters, recorded on the bulletin board, correctly encode the votes cast by them. Eligibility ensures that any valid ballot on the bulletin board has been cast by a registered voter: either honest or corrupt. Therefore, it allows the adversary to cast a ballot only with the credentials of a corrupt voter. Individual verifiability ensures that if a voter verifies a vote, and their ballot is on the bulletin board, then the ballot really encodes the vote verified.

These notions are formulated for identity-based verifiability as follows:

$$
\begin{aligned}
\Phi^{\mathsf{id}}_{\mathsf{rec}} &: \mathsf{GoingToTally}(\mathsf{id}, \mathsf{cr}, \mathsf{b}) \wedge \mathsf{Voter}(\mathsf{id}, \mathsf{cr}', \mathsf{H}) \Rightarrow \mathsf{Voted}(\mathsf{id}, \mathsf{v}) \wedge \mathsf{v} = \mathsf{open}(\mathsf{b}) \\
\Phi^{\mathsf{id}}_{\mathsf{eli}} &: \mathsf{GoingToTally}(\mathsf{id}, \mathsf{cr}, \mathsf{b}) \wedge \mathsf{valid}(\mathsf{b}) \Rightarrow \mathsf{Voter}(\mathsf{id}, \mathsf{cr}', \ell) \wedge \mathsf{open}(\mathsf{b}) \in \mathcal{V} \\
\Phi^{\mathsf{id}}_{\mathsf{iv}} &: \mathsf{Verified}(\mathsf{id}, \mathsf{v}) \Rightarrow \mathsf{GoingToTally}(\mathsf{id}, \mathsf{cr}, \mathsf{b}) \wedge \mathsf{valid}(\mathsf{b}) \wedge \mathsf{v} = \mathsf{open}(\mathsf{b}) \\
\Phi^{\mathsf{id}}_{\mathsf{cons}} &: \mathsf{GoingToTally}(\mathsf{id}, \mathsf{cr}, \mathsf{b}) \wedge \mathsf{GoingToTally}(\mathsf{id}, \mathsf{cr}', \mathsf{b}') \Rightarrow \mathsf{b} = \mathsf{b}'
\end{aligned}
$$

where open is a function that models opening a ballot and revealing the vote inside. It is usually a decryption function but may also be defined differently. The last formula, $\Phi^{\mathsf{id}}_{\mathsf{cons}}$, ensures consistency between identities and ballots, i.e. there is at most one ballot recorded for a voter identity. The formulas for credential-based verifiability are similar:

$$
\begin{aligned}
\Phi^{\mathsf{cr}}_{\mathsf{rec}} &: \mathsf{GoingToTally}(\mathsf{id}, \mathsf{cr}, \mathsf{b}) \wedge \mathsf{Voter}(\mathsf{id}', \mathsf{cr}, \mathsf{H}) \Rightarrow \mathsf{Voted}(\mathsf{id}', \mathsf{v}) \wedge \mathsf{v} = \mathsf{open}(\mathsf{b}) \\
\Phi^{\mathsf{cr}}_{\mathsf{eli}} &: \mathsf{GoingToTally}(\mathsf{id}, \mathsf{cr}, \mathsf{b}) \wedge \mathsf{valid}(\mathsf{b}) \Rightarrow \mathsf{Voter}(\mathsf{id}', \mathsf{cr}, \ell) \wedge \mathsf{open}(\mathsf{b}) \in \mathcal{V} \\
\Phi^{\mathsf{cr}}_{\mathsf{iv}} &: \mathsf{Verified}(\mathsf{id}, \mathsf{v}) \\
&\quad \Rightarrow \mathsf{Voter}(\mathsf{id}, \mathsf{cr}, \mathsf{H}) \wedge \mathsf{GoingToTally}(\mathsf{id}', \mathsf{cr}, \mathsf{b}) \wedge \mathsf{valid}(\mathsf{b}) \wedge \mathsf{v} = \mathsf{open}(\mathsf{b}) \\
\Phi^{\mathsf{cr}}_{\mathsf{cons1}} &: \mathsf{GoingToTally}(\mathsf{id}, \mathsf{cr}, \mathsf{b}) \wedge \mathsf{GoingToTally}(\mathsf{id}', \mathsf{cr}, \mathsf{b}') \Rightarrow \mathsf{b} = \mathsf{b}' \\
\Phi^{\mathsf{cr}}_{\mathsf{cons2}} &: \mathsf{Voter}(\mathsf{id}, \mathsf{cr}, \ell) \wedge \mathsf{Voter}(\mathsf{id}', \mathsf{cr}', \ell') \\
&\quad \Rightarrow (\mathsf{id} = \mathsf{id}' \wedge \mathsf{cr} = \mathsf{cr}') \vee (\mathsf{id} \neq \mathsf{id}' \wedge \mathsf{cr} \neq \mathsf{cr}')
\end{aligned}
$$

The formula Φ^{cr}_{iv} above has an additional Voter event on the right to make a connection between the credential associated with the ballot on the bulletin board and the credential of the honest voter who performs the verification. In this case, the voting server is assumed to be corrupt, and the registrar is honest. Therefore, each voter should have been registered with one credential, and each two voters should have been registered with two different credentials, which is ensured with an additional consistency formula Φ^{cr}_{cons2}. Note also that the corresponding formulas in the two cases differ in whether they put a constraint for identities or credentials, respectively.

The symbolic definition above implies end-to-end verifiability under a few assumptions:

a) The voters are labelled at the registration, as being honest or corrupt. A voter cannot be registered with both labels.

b) If a voter performs an individual verification for a vote, then the voter should have cast that vote before. Furthermore, only votes verified by honest voters are guaranteed to be counted in the result.

c) There is no revoting, i.e. the voters can cast a single ballot.

d) There are no duplicate ballots on the bulletin board associated with any identity and credential.

Considering all, the definition from [17] has the following limitations:

1. The definition is based on two different trust assumptions depending on the corruption scenario of a corrupt registrar or server, respectively. Therefore, the definition varies according to the corruption case, i.e. identity-based verifiability when the registrar is corrupt and credential-based verifiability when the server is corrupt. Thus, the formulas required to obtain identity-based verifiability differ from the ones for credential-based verifiability. Consequently, two different proofs are required to show that these symbolic definitions are sound concerning multiset-based verifiability. However, for generality, the definition should be independent of the trust assumptions; therefore, it should apply to corruption scenarios other than the ones considered in this definition. Furthermore, the definition should be protocol independent and as generic as possible to be applicable to a broader range of protocols.

2. The assumptions of the definition makes it very specific to the protocols, in which (i) the voters are labelled as honest or corrupt at the beginning of the election, (ii) there is no revoting. Therefore, it is not suitable for many protocols, even for Belenios itself, where revoting is allowed. The approach of labelling the voters at the beginning of the election does not create a problem since the voters can cast a single vote. However, when there is revoting, the voters may become corrupt at any moment in the election.

3. The event GoingToTally(id, cr, b) plays a key role in the definition, representing the ballot recorded on the bulletin board. Thus, it is used in all three formulas for end-to-end verifiability. However, it specifically models the action of the voting server that records the ballot b on the bulletin board after associating it with a voter identity id and a credential cr. Considering that the voting server is assumed to be corrupt in one of the two cases, the definition should not rely on the information provided by the voting server; instead, it should rely on the one on the bulletin board, which is verifiable by anybody. Therefore, there should be an event for the information on the bulletin board, recording only (cr, b).

4. Clash attacks exploit the verifiability of an e-voting protocol since they allow the adversary to manipulate the election outcome. Therefore, any verifiability definition should consider the case of clash attacks and ensure their nonexistence. The definition in [17] does not include an explicit notion to ensure no clash attacks. Instead, it relies on distinct assumptions and formulas to prevent this in each case. These assumptions would not hold in more general scenarios, e.g. when revoting is allowed or both registrar and server are corrupt. The classic clash attacks [40] hold when both the registrar and server are corrupt, and this is by definition out of the scope of [17]. As we show later, even when the server is honest and revoting is allowed, there can be clash attacks, and these would also not be captured by [17]. In this case, the adversary can exploit another version of clash attacks that only relies on a corrupt registrar: the voter id_1 and the voter id_2 receive the same credential cr; the voter id_1 casts the ballot b_1 and verifies it on the bulletin board; the voter id_2 casts the ballot b_2 verifies it on the bulletin board; only one ballot b_2 is tallied for the credential cr.

The definition in [17] can be generalised to cover protocols that allow revoting, and at the same time, it can be improved with respect to the limitations discussed above. Adapting the definition to the case of revoting requires the following points to be considered:

- The voters may cast several votes, and thus, may verify any of them.
- There should be a revote policy to determine which votes to be tallied. Furthermore, individual verifiability formula should ensure that the verified vote complying with the revote policy goes to tally.
- The individual verification can be performed anytime, during, and after the voting phase.
- The voters may become corrupt anytime during the election, i.e. they can be corrupted after they cast a vote or even after they verify a vote.
- New versions of clash attacks [40] become possible only with a corrupt registrar depending on the individual verification procedure.

Accounting for all the points stated above and improving the symbolic definition from [17], we propose a general symbolic end-to-end verifiability definition in Section 4.4, which covers a broad class of e-voting protocols and allows automated verification.

4.2 Election Verifiability Events

Certain events are recorded in an execution trace of a protocol, representing the particular actions of the parties and publicly observable information. Then, the recorded events allow us to verify if certain properties hold with respect to the execution of the protocol. In this section, we define the events related to an execution trace of an e-voting specification that will allow us to define *multiset-based verifiability* in Section 4.3 and *symbolic verifiability* in Section 4.4. We start with an informal description of events before gathering them together in Definition 1, which specifies the class of verifiable e-voting specifications.

4.2.1 Public Events

Verifiable e-voting protocols rely on a bulletin board, denoted by BB, to record all the public election data. The data on BB can be classified as eligible candidates, eligible voters (represented with their public credentials), the ballots cast by the voters, the votes in the outcome,

and any publicly verifiable zero-knowledge proofs. As explained in the following, we will denote parts of BB with specific names.

At the beginning of an election, all the candidates eligible to be elected, i.e. $\mathsf{v}_1, \ldots, \mathsf{v}_k$, are determined and recorded on BBcand. Then, all the identities eligible to vote, i.e. $\mathsf{id}_1, \ldots, \mathsf{id}_n$ are determined, and their public credentials $\mathsf{cr}_1, \ldots, \mathsf{cr}_n$ are recorded on BBreg. In many e-voting protocols, the public credentials are chosen to be different than the identities since using identities may violate the privacy of the voters.

In the voting phase, the voters cast their ballots which are recorded on BBcast, where the tuple $(\mathsf{cr}, \mathsf{b})$ on BBcast denotes the ballot b cast by the eligible credential cr. For generality, assume revoting is allowed. In this case, the existing ballot on BBcast may be overwritten by a new ballot cast for a credential. Even so, all the ballots received could be logged for audits in the protocol. Therefore, assume BBcast contains all the ballots cast by the voters. Then, at the end of the voting phase, BBcast may contain several ballots for a credential, i.e. $(\mathsf{cr}, \mathsf{b}^1, \ldots, \mathsf{b}^\ell) \in \mathsf{BBcast}$. Among them, one is selected according to a revote policy, usually the last one (b^ℓ), and goes to the tally. The ballot to be tallied for each credential is recorded on $\mathsf{BBtally}$.

For the tally, all the ballots recorded on $\mathsf{BBtally}$ are opened, and the votes inside are revealed. This is achieved in a publicly verifiable way relying on re-encryption mixnets or homomorphic tally, whose goal is to compute the result correctly while hiding the link between each credential and corresponding vote. In our symbolic models, we will assume the tally procedure is perfect, as in [17], and the election outcome consists in $\mathsf{open}(\mathsf{b}_1), \ldots, \mathsf{open}(\mathsf{b}_n)$, where the function open is used to retrieve the votes from ballots, i.e. $\mathsf{open}(\mathsf{b}) = \mathsf{v}$. Even if the result is published as a set of votes on BB, we associate each vote to the corresponding credential and symbolically represent it by an event BBres, i.e. $\mathsf{BBres}(\mathsf{cr}, \mathsf{v}) \equiv \mathsf{BBtally}(\mathsf{cr}, \mathsf{b}) \wedge \mathsf{v} = \mathsf{open}(\mathsf{b})$. Moreover, a zero-knowledge proof π_{res} is produced to prove the correctness of the tally, i.e. it proves that the votes recorded on BBres are obtained from the ballots recorded on $\mathsf{BBtally}$ applying the function open. Hence, in addition to the candidates published on BBcand, the information on BB of a verifiable e-voting protocol will be as follows:

Setup:	$\mathsf{BBreg}:$	cr_1	$\ldots$	cr_n	
Voting:	$\mathsf{BBcast}:$	$(\mathsf{cr}_1, \mathsf{b}_1^1, \ldots, \mathsf{b}_1^{\ell_1})$	$\ldots$	$(\mathsf{cr}_n, \mathsf{b}_1^1, \ldots, \mathsf{b}_n^{\ell_n})$	
Tally:	$\mathsf{BBtally}:$	$(\mathsf{cr}_1, \mathsf{b}_1^{\ell_1})$	$\ldots$	$(\mathsf{cr}_n, \mathsf{b}_n^{\ell_n})$	
Result:	$\mathsf{BBres}:$	$(\mathsf{cr}_1, \mathsf{v}_1)$	$\ldots$	$(\mathsf{cr}_n, \mathsf{v}_n)$	π_{res}

In some protocols like Selene, after revealed, the votes are recorded next to the trackers, i.e. $\mathsf{tr}_1, \ldots, \mathsf{tr}_n$, that are verification credentials different from public credentials. The trackers are provided to the associated voters, which allows them to verify their votes anonymously. Thus, for such protocols, we have the following as the election result:

Result:	$\mathsf{BBres}:$	$(\mathsf{tr}_1, \mathsf{v}_1)$	$\ldots$	$(\mathsf{tr}_n, \mathsf{v}_n)$	π_{res}

From the information on BB, we can ensure that all ballots on BBcast have been cast by eligible credentials recorded on BBreg, all ballots on $\mathsf{BBtally}$ correspond to the ballots from BBcast for which the revote policy is applied, and similarly, all the votes on BBres correspond to the ballots on $\mathsf{BBtally}$. The last one is ensured by verifying the zero-knowledge proof π_{res}. While all these parts of BB may be necessary to enforce the specification of the protocol, to ensure end-to-end verifiability, we just need the election result and the list of eligible public

credentials. Therefore, the following public information on BB is required:

Setup:	BBreg :	cr_1	...	cr_n
Result:	BBres :	$(\mathsf{tr}_1, \mathsf{v}_1)$	...	$(\mathsf{tr}_n, \mathsf{v}_n)$

Thus, we assume the executions of verifiable e-voting protocols record the public events $\mathsf{BBreg}(\mathsf{cr})$ and $\mathsf{BBres}(\mathsf{tr}, \mathsf{v})$ to denote the public eligible credentials and the votes recorded next to trackers in the election outcome. Note that for generality, we assume that the votes are associated with the trackers on a symbolic BBres. For the protocols that do not use trackers, we have $\mathsf{tr} = \mathsf{cr}$, and BBres is obtained from BBtally, as explained above.

4.2.2 Events for consistency

From the public information on BB, we can ensure that all the ballots to be tallied have been cast by eligible credentials. However, for eligibility, any vote in the outcome should correspond to a registered voter, and there should be at most one vote per such voter. Therefore, there should be a one-to-one correspondence between voter identities $\mathsf{id}_1, \ldots, \mathsf{id}_n$ and public credentials $\mathsf{cr}_1, \ldots, \mathsf{cr}_n$ recorded on BBreg, i.e. each voter identity should hold at most one credential, and each credential should be given at most one voter identity. To ensure that each voter identity is associated with a public credential, we assume the executions of verifiable e-voting protocols record an event $\mathsf{Reg}(\mathsf{id}, \mathsf{cr})$ whenever an election authority associates a voter identity with a public credential. To ensure the consistency between voter identities and public credentials, the association should be made by an honest authority. Therefore, the placement of the event $\mathsf{Reg}(\mathsf{id}, \mathsf{cr})$ can change according to the specification of the protocol with respect to the corruption scenarios. For example, in Belenios-like protocols, the registrar, when honest, can associate voter identities with public credentials since it generates the public credentials and communicates them to voters. Otherwise, the honest voting server can associate identities with public credentials when it receives ballots attached to those public credentials via a login operation. If both are corrupt, the event $\mathsf{Reg}(\mathsf{id}, \mathsf{cr})$ can be recorded in the specification of voters.

Similarly, we can ensure that all the votes in the result correspond to the ballots recorded on BBtally if the zero-knowledge proof π_{res} of the correct tally is verified. However, for the protocols that use trackers, i.e. when $\mathsf{tr} \neq \mathsf{cr}$, we need to ensure further the one-to-one correspondence between public credentials and trackers. In reality, the authorities who assign the trackers to the public credentials provide zero-knowledge proof, which should imply, in particular, a unique association between public credentials and trackers. Thus, we assume the protocol execution records the events $\mathsf{Link}(\mathsf{cr}, \mathsf{tr})$ as soon as the public information, e.g. the zero-knowledge proofs, allows one to ensure the link between a public credential and a tracker. In some protocols, this association can also be made by a trusted party.

As a consequence, we can ensure that the voter identities are associated with public credentials through the events $\mathsf{Reg}(\mathsf{id}, \mathsf{cr})$, and similarly, the public credentials are linked to the

trackers through the events $\mathsf{Link}(\mathsf{cr}, \mathsf{tr})$ as follows:

	id_1	…	id_n
	$\mathsf{Reg}(\mathsf{id}_1, \mathsf{cr}_1)$	…	$\mathsf{Reg}(\mathsf{id}_n, \mathsf{cr}_n)$
$\mathsf{BBreg}:$	cr_1	…	cr_n
	$\mathsf{Link}(\mathsf{cr}_1, \mathsf{tr}_1)$	…	$\mathsf{Link}(\mathsf{cr}_n, \mathsf{tr}_n)$
$\mathsf{BBres}:$	$(\mathsf{tr}_1, \mathsf{v}_1)$	…	$(\mathsf{tr}_n, \mathsf{v}_n)$

4.2.3 Voter Events

To partition the votes in the election outcome according to the multiset-based verifiability, first, we should make a distinction between the votes cast by either honest voters or corrupt voters. The voters cast their ballots using their voting platforms, where we assume the event $\mathsf{Vote}(\mathsf{id}, \mathsf{v})$ is recorded for this action, representing the voter id casts a ballot for the vote v. On the other hand, if the voter is corrupt, the adversary can generate a ballot and cast it on behalf of the corrupt voter through the public channel. This means that for the ballot recorded on BBcast, there may not be a corresponding $\mathsf{Vote}(\mathsf{id}, \mathsf{v})$ event. Therefore, we assume the corrupt voter identities are recorded in the events $\mathsf{Corr}(\mathsf{id})$ when they leak their credentials. Then, it can be ensured that for any ballot on BBcast, i.e. for any vote on BBres, either the event $\mathsf{Vote}(\mathsf{id}, \mathsf{v})$ or the event $\mathsf{Corr}(\mathsf{id})$ is recorded. Thus, we can directly determine the set of corrupt voter identities whose votes are recorded on BBres. On the other hand, we can obtain the set of honest voter identities by excluding corrupt voter identities recorded in $\mathsf{Corr}(\mathsf{id})$ from those recorded in the events $\mathsf{Vote}(\mathsf{id}, \mathsf{v})$. Next, we determine the voters who verified their votes with verification events.

4.2.4 Verification Events

Verification events allow us to determine the votes that have been verified in the outcome and, thus, the voters who perform individual verification for those votes. Assume the event $\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{t})$ is recorded in the execution trace when the voter id verifies for a credential cr the vote v that has been cast at time t. If the e-voting protocol satisfies individual verifiability and does not allow revoting, then the vote v recorded in the event $\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{t})$ gives us the vote in the outcome that is verified by the voter id. However, if revoting is allowed, the recorded vote in $\mathsf{Verified}$ may not be tallied for the voter id. In the case of revoting, a revote policy determines the ballot to be tallied, i.e. the vote to be counted. Then, individual verifiability is satisfied, provided that the revote policy selects the vote verified. Consider that a voter has verified many votes during the voting phase but not the one to be counted. In that case, there is no guarantee for individual verifiability. Therefore, we represent the revote policy with a formula $\Omega(\mathsf{id}, \mathsf{v}, \mathsf{t})$ which specifies the vote v that has to be tallied among all the votes cast by the voter id, depending on the time t that it has been cast. For example, $\Omega(\mathsf{id}, \mathsf{v}, \mathsf{t})$ may select the last vote cast by the voter id, ordering the times and selecting the vote v for the last occurrence of t. Then, we can ensure that a vote v recorded on BBres is also verified if $\Omega(\mathsf{id}, \mathsf{v}, \mathsf{t})$ has selected the vote v for the time t and there is an event $\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{t})$ recorded in the execution trace.

See the following example for the association of the time t recorded when the vote is cast with the one recorded when the vote is verified. The example is given as a snippet from a Tamarin specification.

Example 10. *Assume the Tamarin specification of an e-voting protocol specifies the rules* $\mathsf{R_{vote}}$ *and* $\mathsf{R_{verify}}$ *for ballot casting procedure and the individual verification procedure of the voter, respectively, as follows:*

$$\begin{aligned}\mathsf{R_{vote}}: \quad & \mathsf{let}\ \ \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk_E}, \mathsf{r});\ \ \mathsf{s} = \mathsf{sign}(\mathsf{c}, \mathsf{sk_{id}});\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}\rangle\ \ \mathsf{in} \\ & [\ !\mathsf{Cred}(\mathsf{id}, \mathsf{cr}, \mathsf{sk_{id}}), !\mathsf{BBcand}(\mathsf{v}), !\mathsf{BBkey}(\mathsf{pk_E}), \mathsf{Fr}(\mathsf{r}), \mathsf{Fr}(\mathsf{t})\ ] \\ & -[\ \mathsf{Vote}(\mathsf{id}, \mathsf{v}), \mathsf{VoteTime}(\mathsf{id}, \mathsf{v}, \mathsf{t})\ ]\rightarrow[\ \mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{b}, \mathsf{t}), \mathsf{Out}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{b}\rangle)\ ]\end{aligned}$$

$$\mathsf{R_{verify}}: \quad [\ \mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{b}, \mathsf{t}), \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]-[\ \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{t})\ ]\rightarrow[\ ]$$

In the rule $\mathsf{R_{vote}}$*, a ballot* $\mathsf{b} = \langle \mathsf{c}, \mathsf{s}\rangle$ *is generated as a tuple of ciphertext* c *of the chosen vote* v *and the respective signature* s*. The vote* v *is chosen from* BBcand *and encrypted with the public key on* BBkey *and fresh randomness* r*. The signature is generated with the signing key* $\mathsf{sk_{id}}$ *recorded in* Cred*. Then, the ballot* b *is cast through the communication network. The rule, when executed, records the events (action facts)* Vote(id, v) *and* VoteTime(id, v, t)*, representing the voter* id *casts a vote* v*. The event* VoteTime *also records the fresh term* t*, representing the time the voter casts the vote* v*. All the information is recorded in* Voted *to be used for individual verification. The rule* $\mathsf{R_{verify}}$ *uses the facts* Voted *and* BBcast *for the verification. Whenever the ballots in both match, the rule records an event* Verified*, representing the voter* id *verified the vote* v *cast at time* t *for the credential* cr*.*

Assume an e-voting specification $\mathcal{S}$ records the public, consistency, voter and verification events corresponding to the information recorded on BB, the association of voter identities with public credentials, as well as public credentials with trackers, votes cast or verified by the voter identities, the voter identities corrupted by the adversary, as discussed above. Then, we can verify whether $\mathcal{S}$ satisfies end-to-end verifiability, either by counting the votes in the outcome according to the classification given in [20] or by checking correspondence assertions between these events as done in symbolic definitions suitable for automated verification.

Definition 1. *A protocol specification* $\mathcal{S}$ *is a* ***verifiable*** *e-voting specification if it relies on fact symbols*

$$\mathsf{BBreg}, \mathsf{BBres}, \mathsf{Reg}, \mathsf{Link}, \mathsf{Vote}, \mathsf{Verified}, \mathsf{Corr}$$

to record the following events:

- BBreg(cr)*: the public credential* cr *has been recorded on* BB *as eligible;*
- BBres(tr, v)*: the vote* v *has been recorded on* BB *next to the tracker* tr *as counted in the result;*
- Reg(id, cr)*: the voter* id *has been registered with the public credential* cr*;*
- Link(cr, tr)*: the public credential* cr *has been linked to the tracker* tr*;*
- Vote(id, v)*: the voter* id *has cast a vote* v*;*
- Verified(id, cr, v, t)*: the voter* id *with public credential* cr *has successfully performed the verification procedure related to the vote* v *that was cast at time* t*;*
- Corr(id)*: the voter* id *has leaked all their credentials to the adversary.*

In Definition 1, when tr = cr, we may have BBres(cr, v) and omit Link(cr, tr), which is trivial in this case.

4.2.5 Revote Policy

A revote policy specifies the vote to be counted among the votes cast by a voter, i.e. if the voter casts several votes $\mathsf{v}^1 \ldots, \mathsf{v}^\ell$, only one of these votes, say v^i, will be counted in the outcome, according to the revote policy in place. For an e-voting protocol to guarantee individual verifiability, the vote v_i (selected by the revote policy and counted in the result) should have been verified by the voter. For the protocols that allow individual verification of the votes in the tally phase, if the vote v_i is verified, that vote will be counted on the voter's behalf. However, if the protocol allows individual verification at any time during the election, even if the voter verifies the vote v_j as to be counted, another vote v_i could be selected for the tally.

For example, assume the revote policy selects the last vote cast by the voter, and the protocol allows individual verification at any time during the election. Moreover, assume the protocol records an event $\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v})$ whenever the voter id verifies the vote v for a credential cr. Then, consider the following scenario: The voter id casts two different votes v_1 and v_2, and verifies the vote v_2 cast as last, recording the event $\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v}_2)$ in the execution trace of the protocol. However, the adversary in the network reorders the votes, and after the voter verifies v_2, it casts the vote v_1 on the voter's behalf. Thus, according to its database, the voting server selects the last vote v_1 for the tally. This scenario shows that the event $\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v})$ recorded in the execution trace is not sufficient to ensure the corresponding vote v will be counted for the voter id. Therefore, the formal definition of verifiability requires additional constraints to be satisfied for the verified vote to be counted.

We can consider a parameterised formula $\Omega(\mathsf{id}, \mathsf{v})$, i.e. a policy, to add the respective constraints for a given triple $(\mathsf{id}, \mathsf{cr}, \mathsf{v})$. For example, the policy

$$\Omega^{\mathsf{last}}(\mathsf{id}, \mathsf{v}) \,:\, \mathsf{Vote}(\mathsf{id}, \mathsf{v})\ @i \Rightarrow (\mathsf{Vote}(\mathsf{id}, \mathsf{v}')\ @j \Rightarrow j \prec i \ \vee\ \mathsf{v} = \mathsf{v}')$$

specifies the last vote v cast by the voter id, based on the event $\mathsf{Vote}(\mathsf{id}, \mathsf{v})$. Then, $\Omega^{\mathsf{last}}(\mathsf{id}, \mathsf{v}) \wedge \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v})$ restricts the verified vote v to be the last vote cast by the voter id. However, the constraint on the event $\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v})$ only matches v and, thus, the last vote v cast by the voter id could have been verified as the first vote cast. For instance, the events $\mathsf{Vote}(\mathsf{id}, \mathsf{v}) - \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v}) - \mathsf{Vote}(\mathsf{id}, \mathsf{v}') - \mathsf{Vote}(\mathsf{id}, \mathsf{v}) - \mathsf{Link}(\mathsf{cr}, \mathsf{tr}) - \mathsf{BBres}(\mathsf{tr}, \mathsf{v}')$ could be recorded in the execution trace. That is, the voter id casts a vote v and verifies it. Then, the voter revotes for the vote v' and again for the vote v, but does not verify any of them. The adversary drops the last vote of the voter. Therefore, the voting server tallies the vote v' for the voter id, linking their credential cr to the tracker tr. In this execution trace, even though the voter did not verify their last vote cast, $\Omega^{\mathsf{last}}(\mathsf{id}, \mathsf{v}) \wedge \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v})$ implies it as verified and, thus, it will cause a false individual verification attack. Thus, the policy does not correctly capture what we want to achieve.

Consider the following policy:

$$\Omega^{\mathsf{last}}(\mathsf{id}, \mathsf{v}, \mathsf{t}) \,:\, \mathsf{VoteTime}(\mathsf{id}, \mathsf{v}, \mathsf{t})\ @i \Rightarrow (\mathsf{VoteTime}(\mathsf{id}, \mathsf{v}', \mathsf{t}')\ @j \Rightarrow j \prec i \ \vee\ \mathsf{t} = \mathsf{t}')$$

where the event $\mathsf{Votetime}$ records a fresh term t to model the voting time whenever id casts a vote v, as in Example 10. The policy $\Omega^{\mathsf{last}}(\mathsf{id}, \mathsf{v}, \mathsf{t})$ specifies the last time t recorded for the voter id, i.e. the last vote cast by the voter id. Thus, even though the voter casts the same vote several times, the last one will be specified with the time t recorded last for that voter. In this case, the term t should be inherited in the specification and thus be a parameter of the event $\mathsf{Verified}$, as in Example 10. Only then, $\Omega^{\mathsf{last}}(\mathsf{id}, \mathsf{v}, \mathsf{t}) \wedge \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{t})$ will point out the verified vote as the last vote cast. If the protocol does not allow revoting, then the verifiability definition considers a trivial policy $\Omega^{\mathsf{no}}(\mathsf{id}, \mathsf{v}, \mathsf{t})$, which is set to be always true, and requires any verified vote to be counted.

Definition 2 introduces a revote policy to a verifiable e-voting protocol specification, which either specifies a vote among all cast votes with respect to the times they were cast or is trivial when revoting is not allowed.

Definition 2. *A verifiable e-voting protocol specification $\mathcal{S}$ admits a revote policy Ω if*

(1) (a) every $\mathsf{Vote}(\mathsf{id},\mathsf{v})$ *event is accompanied by* $\mathsf{VoteTime}(\mathsf{id},\mathsf{v},\mathsf{t})$ *event, where* t *is a fresh randomness, and*

(b) $\Omega(\mathsf{id},\mathsf{v},\mathsf{t})$ *is a formula of the form* $\mathsf{VoteTime}(\mathsf{id},\mathsf{v},\mathsf{t})\ @i \Rightarrow \Omega'(\mathsf{id},\mathsf{v},\mathsf{t},i)$ *for some formula* $\Omega'(\mathsf{id},\mathsf{v},\mathsf{t},i)$*,*

(2) $\Omega(\mathsf{id},\mathsf{v},\mathsf{t}) = \mathsf{true}$ *if there is no revoting (or no revote policy).*

4.3 Multiset-based Election Verifiability

In this section, we recall the multiset-based definition of election verifiability in [20] and extend it to have a general notion that can cover stronger or weaker guarantees of election verifiability depending on considered voting protocols or trust assumptions. We also introduce some notation that is useful for both the multiset-based definition and the symbolic definition of election verifiability.

The definition of election verifiability from [20] covers the notion of end-to-end verifiability, counting the votes in the final result as partitioning them into three multisets of votes, i.e. V_1, V_2, V_3, where

1. V_1 are votes cast by honest voters who verified their votes;
2. V_2 is a subset of the votes from honest voters who did not verify their votes; it contains at most one vote for each such voter, but some votes may be dropped by the adversary;
3. V_3 represents the votes cast by the adversary, and its size should be bounded by the number of corrupt voters.

In this definition, individual verifiability guarantees at point 1) are provided only for honest voters. However, corrupt voters may also cast and verify their votes. In some cases, it may be desirable and possible to provide individual verifiability guarantees even for corrupt voters. Therefore, the point 1) can be strengthened by requiring that all verified votes should be present in the outcome, not only those of honest voters but also those of corrupt voters. On the other hand, in some protocols, we may not be able to prevent ballot stuffing for honest voters that have not verified their votes. For example, this is the case in Helios when the voting server is corrupt. To also provide end-to-end verifiability guarantees in that case, the point 3) can be weakened by allowing the adversary to cast votes for honest voters who did not verify their vote. Thus, the existing multiset-based end-to-end election verifiability definition can be improved as we show in the following.

Consider the following multisets/sets associated with a trace, defined informally for now:

- $\mathsf{Ver}^\bullet$ - the multiset of all verified votes complying with the revote policy;
- Ver° - as above, but only including votes verified by honest voters;
- $\mathsf{Adv}^\bullet$ - the set of corrupt voter identities;
- Adv° - the set of voter identities that includes corrupt voter identities and voter identities who did not perform any verification procedure for their cast votes.

- $\mathsf{Vote}^{\nabla,\diamond}$ - the set containing all subsets of the votes from non-adversarial voter identities who have not verified their votes, i.e. the voter identities who are not in $\mathsf{Adv}^{\diamond}$ and have not verified their votes; the definition of this set varies according to the scenario $(\mathsf{Ver}^{\nabla}_{\mathsf{id}}, \mathsf{Adv}^{\diamond})$ for $\nabla, \diamond \in \{\bullet, \circ\}$, where $\mathsf{Ver}^{\nabla}_{\mathsf{id}}$ represents the set of voter identities corresponding to Ver^{∇}.

- Res - the multiset of votes in the result of the election.

The definitions of the multisets $\mathsf{Ver}^{\bullet}$ and Ver° require verified votes complying with the revote policy. If there is no revoting, the revote policy will be trivially true, and the votes in those sets will be verified votes among all cast votes. In the case of revoting, the multisets $\mathsf{Ver}^{\bullet}$ and Ver° will be the subset of all verified votes. Consider that a voter has verified many votes during the voting phase but not the one complying with the revote policy. Then, the definition of the multisets $\mathsf{Ver}^{\bullet}$ and Ver° require this vote not to be included. The set Adv° includes corrupt voter identities from the set $\mathsf{Adv}^{\bullet}$ and also the ones who did not perform any verification procedure. Thus, it represents the voter identities for which the adversary may cast a ballot if the protocol is not strong enough. For illustration, consider a few scenarios where a vote from a particular voter may not be counted: (i) the voter does not vote; (ii) the adversary drops the cast vote; (iii) revoting is allowed, and the adversary replaces the cast vote. In all these cases, the voter can perform the verification procedure for their choice, i.e. for the abstention in (i), and for the cast vote in (ii) and (iii). $\mathsf{Adv}^{\circ}(\mathsf{id})$ says that the adversary can cast a vote for the respective voter only if the voter did not perform the verification procedure. $\mathsf{Adv}^{\bullet}(\mathsf{id})$ only allows this when id is corrupt. For the set $\mathsf{Vote}^{\nabla,\diamond}$, assume s honest voters voted during the election but did not verify their votes. Moreover, assume that the adversary did not cast any vote for them, i.e. their identities are not in Adv°. If the adversary does not drop the vote of any such voters, the set of votes in the outcome will be $\{\!|\, \mathsf{v}_1, \ldots, \mathsf{v}_s \,|\!\}$. If the adversary drops only the vote of the voter id_s, then the multiset in the outcome will be $\{\!|\, \mathsf{v}_1, \ldots, \mathsf{v}_{s-1} \,|\!\}$. More generally, any subset of $\{\!|\, \mathsf{v}_1, \ldots, \mathsf{v}_s \,|\!\}$ may be dropped. Considering each such multiset of votes for s voters, i.e. the multiset of votes in the outcome, the set of multisets define $\mathsf{Vote}^{\nabla,\diamond}$. For example, for the two voters id_1 and id_2 who voted for v_1 and v_2, respectively, but did not verify, $\mathsf{Vote}^{\nabla,\diamond} = \{\{\!|\, \mathsf{v}_1 \,|\!\}, \{\!|\, \mathsf{v}_2 \,|\!\}, \{\!|\, \mathsf{v}_1, \mathsf{v}_2 \,|\!\}\}$. Finally, the set Res is defined by the votes in the election outcome.

The following definition formally defines the sets described above, relating them to the events of an e-voting specification.

Definition 3. *Let τ be a trace produced by a verifiable e-voting specification $\mathcal{S}$ having Ω as revote policy. Assuming the following formulas:*

$$\begin{aligned}
\mathsf{Ver}^{\bullet}(\mathsf{id},\mathsf{cr},\mathsf{v}) &\equiv \exists \mathsf{t}.\ \mathsf{Verified}(\mathsf{id},\mathsf{cr},\mathsf{v},\mathsf{t}) \wedge \Omega(\mathsf{id},\mathsf{v},\mathsf{t}) \\
\mathsf{Ver}^{\circ}(\mathsf{id},\mathsf{cr},\mathsf{v}) &\equiv \exists \mathsf{t}.\ \mathsf{Verified}(\mathsf{id},\mathsf{cr},\mathsf{v},\mathsf{t}) \wedge \Omega(\mathsf{id},\mathsf{v},\mathsf{t}) \wedge \neg\mathsf{Corr}(\mathsf{id}) \\
\mathsf{Adv}^{\bullet}(\mathsf{id}) &\equiv \mathsf{Corr}(\mathsf{id}) \\
\mathsf{Adv}^{\circ}(\mathsf{id}) &\equiv \mathsf{Corr}(\mathsf{id}) \vee \neg\exists \mathsf{cr}, \mathsf{v}'.\ \mathsf{Ver}^{\circ}(\mathsf{id},\mathsf{cr},\mathsf{v}')
\end{aligned}$$

we define the following sets for $\nabla, \diamond \in \{\circ, \bullet\}$:

$$\begin{aligned}
\mathsf{Ver}^{\nabla}(\tau) &= \{\!|\, \mathsf{v} \mid \exists \mathsf{id},\mathsf{cr}.\ \tau \vDash \mathsf{Ver}^{\nabla}(\mathsf{id},\mathsf{cr},\mathsf{v}) \wedge \mathsf{v} \neq \bot \,|\!\} \\
\mathsf{Ver}^{\nabla}_{\mathsf{id}}(\tau) &= \{\mathsf{id} \mid \exists \mathsf{cr},\mathsf{v}.\ \tau \vDash \mathsf{Ver}^{\nabla}(\mathsf{id},\mathsf{cr},\mathsf{v}) \wedge \mathsf{Reg}(\mathsf{id},\mathsf{cr}) \wedge \mathsf{v} \neq \bot\} \\
\mathsf{Adv}^{\diamond}(\tau) &= \{\mathsf{id} \mid \tau \vDash \mathsf{Adv}^{\diamond}(\mathsf{id})\} \\
\mathsf{Res}(\tau) &= \{\!|\, \mathsf{v} \mid \exists \mathsf{tr}.\ \tau \vDash \mathsf{BBres}(\mathsf{tr},\mathsf{v}) \wedge \mathsf{v} \neq \bot \,|\!\}
\end{aligned}$$

Additionally, let $\mathsf{Vote}^{\triangledown,\diamond}(\tau)$ *be the set of multisets* $\{|\, \mathsf{v}_1, ..., \mathsf{v}_s \,|\}$ *such that there are distinct* $\mathsf{id}_1, ..., \mathsf{id}_s$ *and for all* $\mathsf{i} \in \{1, \ldots, \mathsf{s}\}$,

$$\tau \vDash \mathsf{Vote}(\mathsf{id}_\mathsf{i}, \mathsf{v}_\mathsf{i}) \ \wedge \ \mathsf{id}_\mathsf{i} \notin (\, \mathsf{Ver}^{\triangledown}_{\mathsf{id}}(\tau) \cup \mathsf{Adv}^{\diamond}(\tau)\,).$$

Considering the sets formally defined above, we define the extended notion of end-to-end election verifiability as follows.

Definition 4. *A trace* τ *of an e-voting specification* S *having* Ω *as revote policy satisfies* $[\mathsf{iv}_\triangledown, \mathsf{res}_\diamond]$-***election verifiability*** *for* $\triangledown, \diamond \in \{\circ, \bullet\}$ *if and only if there exist multisets* V_1, V_2, V_3 *of votes such that* $\mathsf{Res}(\tau) = V_1 \uplus V_2 \uplus V_3$ *and*

$$\begin{aligned} &(1) \quad V_1 = \mathsf{Ver}^{\triangledown}(\tau), \\ &(2) \quad V_2 \in \mathsf{Vote}^{\triangledown,\diamond}(\tau), \\ &(3) \quad |V_3| \leq |\mathsf{Adv}^{\diamond}(\tau) \setminus \mathsf{Ver}^{\triangledown}_{\mathsf{id}}(\tau)|, \end{aligned}$$

which is denoted by $\tau \Vdash \mathsf{E2E}[\mathsf{iv}_\triangledown, \mathsf{res}_\diamond]$. *If all traces of* S *satisfy it, then* $S \Vdash \mathsf{E2E}[\mathsf{iv}_\triangledown, \mathsf{res}_\diamond]$.

Definition 4 gives four different notions of verifiability:

- $\mathsf{E2E}[\mathsf{iv}_\bullet, \mathsf{res}_\bullet]$ is the strongest notion, ensuring that verified votes of corrupt voters are also included in the final tally (this is strong individual verifiability).
- $\mathsf{E2E}[\mathsf{iv}_\circ, \mathsf{res}_\bullet]$ corresponds to the definition from [20, 21, 17], providing individual verifiability only for honest voters.
- $\mathsf{E2E}[\mathsf{iv}_\bullet, \mathsf{res}_\circ]$: in addition to corrupt voters, this notion allows the adversary to cast votes for voters who have not performed the verification procedure (this is sometimes called ballot stuffing). However, for corrupt voters who successfully verified their votes, it can be ensured that these votes are correctly counted.
- $\mathsf{E2E}[\mathsf{iv}_\circ, \mathsf{res}_\circ]$ is the weakest notion, only allowing individual verifiability for honest voters and allowing ballot stuffing for honest voters who did not verify their votes.

4.4 Symbolic Election Verifiability

In this section, improving the symbolic definition in [17], we propose a more general end-to-end verifiability definition that covers a broad class of e-voting protocols. Our definition is similarly structured as a conjunction of formulas, corresponding to the individual verifiability ($\Phi^{\triangledown}_{\mathsf{iv1}}, \Phi^{\triangledown}_{\mathsf{iv2}}, \Phi^{\triangledown}_{\mathsf{iv3}}$), eligibility ($\Phi_{\mathsf{eli}}$), result integrity ($\Phi^{\diamond}_{\mathsf{res}}$), and the consistency ($\Phi_{\mathsf{reg1}}, \Phi_{\mathsf{reg2}}$, $\Phi_{\mathsf{link1}}, \Phi_{\mathsf{link2}}, \Phi_{\mathsf{one}}$). However, our formulas are more generic, relying on the events introduced in Section 4.2. They also account for revoting and a more general class of clash attacks.

Definition 5. *Consider the formulas from Figure 4.1. For* $\triangledown, \diamond \in \{\circ, \bullet\}$, *let*

$$\begin{aligned} \Phi^{\triangledown}_{\mathsf{iv}} &= \Phi^{\triangledown}_{\mathsf{iv1}} \wedge \Phi^{\triangledown}_{\mathsf{iv2}} \wedge \Phi^{\triangledown}_{\mathsf{iv3}} && \textit{(individual verifiability)} \\ \Phi^{\bullet}_{\mathsf{cons}} &= \Phi_{\mathsf{reg1}} \wedge \Phi_{\mathsf{reg2}} \wedge \Phi_{\mathsf{link1}} \wedge \Phi_{\mathsf{link2}} \wedge \Phi_{\mathsf{one}} && \textit{(consistency)} \\ \Phi^{\circ}_{\mathsf{cons}} &= \Phi^{\circ}_{\mathsf{reg2}} \wedge \Phi_{\mathsf{link1}} \wedge \Phi_{\mathsf{link2}} \wedge \Phi_{\mathsf{one}} && \textit{(weak consistency)} \end{aligned}$$

and

$$\begin{aligned} (a) \quad \mathsf{SE2E}[\mathsf{iv}_\triangledown, \mathsf{res}_\diamond] &= \Phi^{\triangledown}_{\mathsf{iv}} \wedge \Phi_{\mathsf{eli}} \wedge \Phi^{\diamond}_{\mathsf{res}} \wedge \Phi^{\bullet}_{\mathsf{cons}} \\ (b) \quad \mathsf{SWE2E}[\mathsf{iv}_\circ, \mathsf{res}_\circ] &= \Phi^{\circ}_{\mathsf{iv}} \wedge \Phi_{\mathsf{eli}} \wedge \Phi^{\circ}_{\mathsf{res}} \wedge \Phi^{\circ}_{\mathsf{cons}} \end{aligned}$$

An e-voting specification $\mathcal{S}$ satisfies

(a) $[\mathsf{iv}_\triangledown, \mathsf{res}_\diamond]$-***symbolic election verifiability*** *if and only if* $\mathcal{S} \vDash \mathsf{SE2E}[\mathsf{iv}_\triangledown, \mathsf{res}_\diamond]$,

(b) $[\mathsf{iv}_\circ, \mathsf{res}_\circ]$-***symbolic weak election verifiability*** *if and only if* $\mathcal{S} \vDash \mathsf{SWE2E}[\mathsf{iv}_\circ, \mathsf{res}_\circ]$.

Basic properties defining symbolic E2E verifiability

$\Phi^{\triangledown}_{\mathsf{iv1}} : \mathsf{Ver}^{\triangledown}(\mathsf{id},\mathsf{cr},\mathsf{v}) \wedge \mathsf{Link}(\mathsf{cr},\mathsf{tr}) \wedge \mathsf{BBres}(\mathsf{tr},\mathsf{v}') \Rightarrow \mathsf{v} = \mathsf{v}'$ ***(individual verifiability)***

$\Phi^{\triangledown}_{\mathsf{iv2}} : \mathsf{Ver}^{\triangledown}(\mathsf{id},\mathsf{cr},\mathsf{v}) \wedge \mathsf{Ver}^{\triangledown}(\mathsf{id}',\mathsf{cr},\mathsf{v}') \Rightarrow \mathsf{id} = \mathsf{id}'$ ***(no clash)***

$\Phi^{\triangledown}_{\mathsf{iv3}} : \mathsf{Ver}^{\triangledown}(\mathsf{id},\mathsf{cr},\mathsf{v}) \Rightarrow \mathsf{Link}(\mathsf{cr},\mathsf{tr}) \wedge \mathsf{BBreg}(\mathsf{cr})$ ***(eligibility)***

$\Phi_{\mathsf{eli}} : \mathsf{BBres}(\mathsf{tr},\mathsf{v}) \Rightarrow \mathsf{Link}(\mathsf{cr},\mathsf{tr}) \wedge \mathsf{BBreg}(\mathsf{cr}) \wedge (\mathsf{Reg}(\mathsf{id},\mathsf{cr}) \vee \mathsf{v} = \bot)$ ***(eligibility)***

$\Phi^{\diamond}_{\mathsf{res}} : \mathsf{BBres}(\mathsf{tr},\mathsf{v}) \wedge \mathsf{Link}(\mathsf{cr},\mathsf{tr}) \wedge \mathsf{Reg}(\mathsf{id},\mathsf{cr}) \Rightarrow \mathsf{Vote}(\mathsf{id},\mathsf{v}) \vee \mathsf{Adv}^{\diamond}(\mathsf{id}) \vee \mathsf{v} = \bot$ ***(result integrity)***

Consistency properties

$\Phi_{\mathsf{reg1}} : \mathsf{Reg}(\mathsf{id},\mathsf{cr}) \wedge \mathsf{Reg}(\mathsf{id}',\mathsf{cr}) \Rightarrow \mathsf{id} = \mathsf{id}'$

$\Phi_{\mathsf{reg2}} : \mathsf{Reg}(\mathsf{id},\mathsf{cr}) \wedge \mathsf{Reg}(\mathsf{id},\mathsf{cr}') \Rightarrow \mathsf{cr} = \mathsf{cr}'$

$\Phi_{\mathsf{link1}} : \mathsf{Link}(\mathsf{cr},\mathsf{tr}) \wedge \mathsf{Link}(\mathsf{cr}',\mathsf{tr}) \Rightarrow \mathsf{cr} = \mathsf{cr}'$

$\Phi_{\mathsf{link2}} : \mathsf{Link}(\mathsf{cr},\mathsf{tr}) \wedge \mathsf{Link}(\mathsf{cr},\mathsf{tr}') \Rightarrow \mathsf{tr} = \mathsf{tr}'$

$\Phi_{\mathsf{one}} : \mathsf{BBres}(\mathsf{tr},\mathsf{v})\ @i \wedge \mathsf{BBres}(\mathsf{tr},\mathsf{v}')\ @j \Rightarrow i = j$

$\Phi_{\mathsf{cand}} : \mathsf{BBres}(\mathsf{tr},\mathsf{v}) \Rightarrow \mathsf{BBcand}(\mathsf{v}) \vee \mathsf{v} = \bot$

$\Phi^{\circ}_{\mathsf{reg2}} : \neg\mathsf{Adv}^{\circ}(\mathsf{id}) \Rightarrow \Phi_{\mathsf{reg2}}$

Two possible cases for $\mathsf{Ver}^{\triangledown}(\mathsf{id},\mathsf{cr},\mathsf{v})$

$\mathsf{Ver}^{\bullet}(\mathsf{id},\mathsf{cr},\mathsf{v}) : \exists \mathsf{t}.\ \mathsf{Verified}(\mathsf{id},\mathsf{cr},\mathsf{v},\mathsf{t}) \wedge \Omega(\mathsf{id},\mathsf{v},\mathsf{t})$

$\mathsf{Ver}^{\circ}(\mathsf{id},\mathsf{cr},\mathsf{v}) : \exists \mathsf{t}.\ \mathsf{Verified}(\mathsf{id},\mathsf{cr},\mathsf{v},\mathsf{t}) \wedge \Omega(\mathsf{id},\mathsf{v},\mathsf{t}) \wedge \neg\mathsf{Corr}(\mathsf{id})$

Two possible cases for $\mathsf{Adv}^{\diamond}(\mathsf{id})$

$\mathsf{Adv}^{\bullet}(\mathsf{id}) : \mathsf{Corr}(\mathsf{id})$

$\mathsf{Adv}^{\circ}(\mathsf{id}) : \mathsf{Corr}(\mathsf{id}) \vee \neg\exists \mathsf{cr}', \mathsf{v}'.\ \mathsf{Ver}^{\circ}(\mathsf{id},\mathsf{cr},\mathsf{v}')$

Examples of revote policies Ω

$\Omega^{\mathsf{no}}(\mathsf{id},\mathsf{v},\mathsf{t}) : \mathsf{true}$

$\Omega^{\mathsf{last}}(\mathsf{id},\mathsf{v},\mathsf{t}) : \mathsf{VoteTime}(\mathsf{id},\mathsf{v},\mathsf{t})\ @i \Rightarrow (\mathsf{VoteTime}(\mathsf{id},\mathsf{v}',\mathsf{t}')\ @j \Rightarrow j \prec i \vee \mathsf{t} = \mathsf{t}')$

FIGURE 4.1: Formulas for symbolic election verifiability.

The formulas used in Definition 5 can be explained informally as follows:

- The property $\Phi^{\triangledown}_{\mathsf{iv}} = \Phi^{\triangledown}_{\mathsf{iv1}} \wedge \Phi^{\triangledown}_{\mathsf{iv2}} \wedge \Phi^{\triangledown}_{\mathsf{iv3}}$ corresponds to *individual verifiability*, where $\Phi^{\triangledown}_{\mathsf{iv1}}$ represents the intuitive notion of individual verifiability, $\Phi^{\triangledown}_{\mathsf{iv2}}$ ensures no clash, and $\Phi^{\triangledown}_{\mathsf{iv3}}$ guarantees eligibility of the voters who perform verification. The formula $\Phi^{\triangledown}_{\mathsf{iv1}}$ is therefore complemented with the other two to obtain formal guarantees.
  - $\Phi^{\triangledown}_{\mathsf{iv1}}$ specifies that if a voter id successfully verified a vote v complying with the revote policy, then the vote v is counted for that voter id in the election result, which is ensured when the voter's public credential cr is linked to a tracker tr and

the voter's vote v is recorded next to that tracker in the outcome. Note that both events $\mathsf{Ver}^{\triangledown}$ and BBres are placed on the left in the formula. For instance, the event BBres can happen after the event $\mathsf{Ver}^{\triangledown}$, and when it happens, the desired property is ensured. Thus, it covers the protocols that allow individual verification during or after the voting phase, in the tally phase, or even after the result is announced.

  - $\Phi^{\triangledown}_{\mathsf{iv2}}$ specifies that no clash should occur on the public credentials: two distinct voters who successfully verified their votes should have different public credentials. Note that the formula $\Phi^{\triangledown}_{\mathsf{iv2}}$ is more general than the no clash property described in [22], which considers the clash only on the credentials, not the ballots. Therefore, our formula covers a more general class of clash attacks.
  - $\Phi^{\triangledown}_{\mathsf{iv3}}$ ensures that, for every voter, successful verification implies that the corresponding public credential is registered on the bulletin board and linked to a tracker tr.

- Φ_{eli} corresponds to the *eligibility* of the trackers in the outcome, ensuring that each tracker is linked to a registered public credential. If there is a vote next to the tracker, i.e. $\mathsf{v} \neq \bot$, it further ensures that the credential is also linked to a voter identity. As discussed before, the party which records the event $\mathsf{Reg}(\mathsf{id}, \mathsf{cr})$ varies according to the specification: it could be that not all voters are registered with a public credential, i.e. the credential may not be linked to the voter identity if the voter did not vote during the election. In this case, the tracker points to $\bot$. Consequently, we have $\mathsf{Reg}(\mathsf{id}, \mathsf{cr})$ or $\mathsf{v} = \bot$.

- $\Phi^{\diamond}_{\mathsf{res}}$ corresponds to *result integrity*, specifying that any vote in the outcome, recorded with a tracker that is linked to a registered public credential associated with a voter identity, should have been cast either by that voter identity or by the adversary who corrupts that voter identity, or the vote is empty, i.e. $\mathsf{v} = \bot$. Relying on $\mathsf{Adv}^{\diamond}(\mathsf{id})$ as a parameter, $\Phi^{\diamond}_{\mathsf{res}}$ further circumscribes adversarial influence on the final result: the adversary is able to cast votes for corrupt voters when $\diamond = \bullet$, the adversary is able to cast votes for corrupt voters and also for the voters who did not verify their votes when $\diamond = \circ$.

- *Consistency properties* Φ_{reg1} and Φ_{reg2} ensure that each voter identity is uniquely associated with a public credential. Conversely, each public credential is uniquely associated with a voter identity.

- *Consistency properties* Φ_{link1} and Φ_{link2} ensure that each public credential is uniquely associated with a tracker. Conversely, each tracker is uniquely associated with a public credential.

- *The consistency property* Φ_{one} ensures that there is at most one vote counted for each tracker.

- *The consistency property* Φ_{cand} ensures that all the votes in the outcome are valid, as being recorded on BBcand or empty, i.e. $\mathsf{v} = \bot$. For the protocols that require a homomorphic tally of the ballots, this property prevents the adversary from casting multiple votes within a single ballot. For others requiring a tally based on mixnets, each ballot is decrypted individually; thus, invalid votes can be removed directly from the outcome.

Definition 5 provides two symbolic definitions, where the weak one $\mathsf{SWE2E}[\mathsf{iv}_{\circ}, \mathsf{res}_{\circ}]$ differs from the other $\mathsf{SE2E}[\mathsf{iv}_{\triangledown}, \mathsf{res}_{\diamond}]$ with one missing formula Φ_{reg1} and the weaker notions of

individual verifiability, result integrity, and the consistency, i.e. $\Phi^{\circ}_{\mathsf{iv}}$, $\Phi^{\circ}_{\mathsf{res}}$, and $\Phi^{\circ}_{\mathsf{cons}}$, respectively. Typically, if the party that registers the voter identities with credentials is honest, the consistency between registered voter identities and the public credentials is ensured, and thus the formulas Φ_{reg1} and Φ_{reg2} hold. However, suppose there is no honest party in the protocol to register voter identities, e.g. in Helios, when the server is corrupt, or in Belenios, when both registrar and server are corrupt. In that case, we cannot ensure this consistency, but still, the protocol may provide the weaker notion of end-to-end verifiability $\mathsf{E2E}[\mathsf{iv}_{\circ}, \mathsf{res}_{\circ}]$ and we can rely on $\mathsf{SWE2E}[\mathsf{iv}_{\circ}, \mathsf{res}_{\circ}]$ to prove it symbolically.

The formula Φ_{reg1} ensures that no two voter identities are registered with the same public credential, and the formula Φ_{reg2} similarly ensures that no two public credentials are registered for a single voter identity. If the registration party is corrupt, non-adversarial voter identities, i.e. honest voter identities that verify their votes, can ensure that they do not own more than one credential, i.e. the formula $\Phi^{\circ}_{\mathsf{reg2}}$ holds. However, if Φ_{reg1} fails, they will not know whether they were registered with a credential that is shared with another voter identity. Nevertheless, due to the no clash property $\Phi^{\circ}_{\mathsf{iv2}}$, no two voter identities verified a ballot for the same credential, i.e. all the credentials used for a successful individual verification correspond to different voter identities, i.e. honest voter identities. This will allow us to ensure that all verified votes are in the outcome and also to avoid ballot stuffing for these voters, which is why we can obtain $\Phi^{\circ}_{\mathsf{res}}$ and deduce $\mathsf{E2E}[\mathsf{iv}_{\circ}, \mathsf{res}_{\circ}]$. We will show the formal implication in our soundness proof.

4.5 Soundness Proof of Symbolic Election Verifiability

In this section, we provide soundness proof for the symbolic election verifiability proposed in Section 4.4. Definition 5 is sound with respect to multiset-based election verifiability based on counting the number of votes in the outcome. Informally, the multiset-based definition states that: 1) the final outcome should include all of the votes that have been successfully verified; 2) the number of votes in the final outcome coming from the adversary should be bounded by the number of adversarial voter identities; 3) all other votes in the outcome correspond to a vote from a non-adversarial voter identity - the adversary is allowed to drop, but not change such votes.

Recall the general flow of electronic voting protocols: a set of public credentials is determined at registration and recorded on BBreg; ballots from corresponding voters are collected, and the ballot to be tallied for each credential is recorded on $\mathsf{BBtally}$; the final result is computed from the ballots in $\mathsf{BBtally}$. Assuming there are n public credentials, the information recorded on BB is as follows:

Setup:	$\mathsf{BBreg}:$	cr_1	$\ldots$	cr_n
Tally:	$\mathsf{BBtally}:$	$(\mathsf{cr}_1, \mathsf{b}_1)$	$\ldots$	$(\mathsf{cr}_n, \mathsf{b}_n)$
Result:	$\mathsf{BBres}:$	$(\mathsf{tr}_1, \mathsf{v}_1)$	$\ldots$	$(\mathsf{tr}_n, \mathsf{v}_n)$

where $\mathsf{b}_i = \perp$ and $\mathsf{v}_i = \perp$ in case of abstention, invalid ballot casting, or ballot blocking by the adversary. Recall that in some e-voting protocols like Helios and Belenios, BBres is defined implicitly from $\mathsf{BBtally}$ taking $\mathsf{tr}_i = \mathsf{cr}_i$ and $\mathsf{v}_i = \mathsf{open}(\mathsf{b}_i)$. To prove general end-to-end verifiability, one additional assumption is required about the execution trace of an e-voting specification, namely that it contains the final state of the bulletin board. The following formula can formalise a trace containing the final state of the bulletin board:

$$\Psi_{\mathsf{E2E}}: \quad |\{\mathsf{cr} \mid \tau \vDash \mathsf{BBreg}(\mathsf{cr})\}| = |\{\mathsf{tr} \mid \exists \mathsf{v}.\ \tau \vDash \mathsf{BBres}(\mathsf{tr}, \mathsf{v})\}|$$

stating that the number of trackers recorded on the final result bulletin board is equal to the number of registered credentials. A trace that satisfies Ψ_{E2E} is called an *end-to-end e-voting trace*.

To prove general end-to-end verifiability for symbolic weak election verifiability, another assumption is required about the end-to-end e-voting trace of an e-voting specification. That is, the number of voter identities is equal to the number of public credentials. Before formalising the assumption, we define the following sets:

$$\begin{aligned} \mathsf{Vote}_{\mathsf{id}}(\tau) &= \{\mathsf{id} \mid \exists \mathsf{v}.\ \tau \vDash \mathsf{Vote}(\mathsf{id}, \mathsf{v})\} \\ \mathsf{Verified}_{\mathsf{id}}(\tau) &= \{\mathsf{id} \mid \exists \mathsf{cr}, \mathsf{v}.\ \tau \vDash \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v})\} \\ \mathsf{Corr}(\tau) &= \{\mathsf{id} \mid \tau \vDash \mathsf{Corr}(\mathsf{id})\} \end{aligned}$$

which allows us to define the set of identities occur in the execution trace as follows:

$$\mathsf{Id}(\tau) = \mathsf{Vote}_{\mathsf{id}}(\tau) \cup \mathsf{Verified}_{\mathsf{id}}(\tau) \cup \mathsf{Corr}(\tau)$$

Then, we can define the following formula to formalise the assumption:

$$\Psi_{\mathsf{E2E}'}: \quad |\{\mathsf{id} \mid \tau \vDash \mathsf{Id}(\mathsf{id})\}| = |\{\mathsf{cr} \mid \tau \vDash \mathsf{BBreg}(\mathsf{cr})\}|$$

stating that the number of voter identities is equal to the number of registered credentials.

Theorem 1. *For every e-voting specification $\mathcal{S}$, symbolic election verifiability implies multiset-based election verifiability for end-to-end e-voting traces. That is, for any trace τ of $\mathcal{S}$,*

(a) $\tau \vDash \mathsf{SE2E}[\mathsf{iv}_{\triangledown}, \mathsf{res}_{\diamond}] \wedge \tau \vDash \Psi_{\mathsf{E2E}} \implies \tau \Vdash \mathsf{E2E}[\mathsf{iv}_{\triangledown}, \mathsf{res}_{\diamond}]$,

(b) $\tau \vDash \mathsf{SWE2E}[\mathsf{iv}_{\circ}, \mathsf{res}_{\circ}] \wedge \tau \vDash \Psi_{\mathsf{E2E}} \wedge \Psi_{\mathsf{E2E}'} \implies \tau \Vdash \mathsf{E2E}[\mathsf{iv}_{\circ}, \mathsf{res}_{\circ}]$.

Proof. We prove the two cases simultaneously, showing how case ***(b)*** is handled differently due to the missing formulas. The proof strategy is as follows: ***(1)*** we link the trackers in the outcome to the registered credentials, ***(2)*** we associate registered public credentials with voter identities, ***(3)*** we associate each vote in the outcome with a registered voter identity, ***(4)*** we partition the set defined containing the identity-vote pairs at ***(3)*** and show that the partition satisfies the requirements in Definition 4.

Assume

(a) $\tau \vDash \mathsf{SE2E}[\mathsf{iv}_{\triangledown}, \mathsf{res}_{\diamond}] \wedge \tau \vDash \Psi_{\mathsf{E2E}}$, ***(b)*** $\tau \vDash \mathsf{SWE2E}[\mathsf{iv}_{\circ}, \mathsf{res}_{\circ}] \wedge \tau \vDash \Psi_{\mathsf{E2E}} \wedge \Psi_{\mathsf{E2E}'}$,

for any end-to-end e-voting trace τ of an e-voting specification.

(1) Let $\{\mathsf{cr} \mid \tau \vDash \mathsf{BBreg}(\mathsf{cr})\}$ be the set of all registered public credentials. Assume there are n such credentials in total, i.e. $\mathsf{cr}_1, ..., \mathsf{cr}_\mathsf{n}$. By $\tau \vDash \Psi_{\mathsf{E2E}}$, we have $\mathsf{n} = |\{\mathsf{tr} \mid \exists \mathsf{v}.\ \tau \vDash \mathsf{BBres}(\mathsf{tr}, \mathsf{v})\}|$, implying that there are precisely n distinct trackers $\{\mathsf{tr}_1, ..., \mathsf{tr}_\mathsf{n}\}$ for which $\tau \vDash \mathsf{BBres}(\mathsf{tr}, \mathsf{v})$ is true for some v. By $\tau \vDash \Phi_{\mathsf{eli}}$, any tracker $\mathsf{tr}_\mathsf{i} \in \{\mathsf{tr}_1, ..., \mathsf{tr}_\mathsf{n}\}$ is linked to a credential $\mathsf{cr}_\mathsf{i} \in \mathsf{BBreg}$, i.e. we have $\tau \vDash \mathsf{Link}(\mathsf{cr}_\mathsf{i}, \mathsf{tr}_\mathsf{i})$. From the consistency properties Φ_{link1} and Φ_{link2}, i.e. $\tau \vDash \Phi_{\mathsf{link1}} \wedge \Phi_{\mathsf{link2}}$, for any $\tau \vDash \mathsf{Link}(\mathsf{cr}_\mathsf{i}, \mathsf{tr}_\mathsf{i}) \wedge \mathsf{Link}(\mathsf{cr}_\mathsf{j}, \mathsf{tr}_\mathsf{j})$, we have $\mathsf{tr}_\mathsf{i} \neq \mathsf{tr}_\mathsf{j}$ and $\mathsf{cr}_\mathsf{i} \neq \mathsf{cr}_\mathsf{j}$ for any $\mathsf{i} \neq \mathsf{j}$. Furthermore, by $\tau \vDash \Phi_{\mathsf{one}}$, there is at most one occurrence of the event $\mathsf{BBres}(\mathsf{tr}, \mathsf{v})$ for each tracker. Therefore, we can deduce that there is a one-to-one correspondence between the trackers in the outcome, i.e. $\mathsf{tr}_1, ..., \mathsf{tr}_\mathsf{n}$, and the credentials that are linked to those, i.e. $\mathsf{cr}_1, ..., \mathsf{cr}_\mathsf{n}$.

(2) Without loss of generality, let $\mathsf{tr}_1, ..., \mathsf{tr}_\mathsf{k}$ be all the trackers in the outcome for which $\mathsf{v} \neq \bot$. By $\tau \vDash \Phi_{\mathsf{eli}}$, for any tracker tr_i in this subset, we have $\tau \vDash \mathsf{Link}(\mathsf{cr}_\mathsf{i}, \mathsf{tr}_\mathsf{i}) \wedge \mathsf{Reg}(\mathsf{id}_\mathsf{i}, \mathsf{cr}_\mathsf{i})$,

which implies that each such tracker tr_i has been linked to a credential cr_i registered with a voter identity id_i. By ***(1)***, there are exactly k distinct credentials, i.e. $\mathsf{cr}_1, ..., \mathsf{cr}_k$, corresponding to $\mathsf{tr}_1, ..., \mathsf{tr}_k$.

(a) From the consistency properties Φ_{reg1} and Φ_{reg2}, i.e. $\tau \vDash \Phi_{\mathsf{reg1}} \wedge \Phi_{\mathsf{reg2}}$, for any $\tau \vDash \mathsf{Reg}(\mathsf{id}_i, \mathsf{cr}_i) \wedge \mathsf{Reg}(\mathsf{id}_j, \mathsf{cr}_j)$, we have $\mathsf{id}_i \neq \mathsf{id}_j$ and $\mathsf{cr}_i \neq \mathsf{cr}_j$ for any $i \neq j$. This implies that each credential cr_i has been registered with a distinct voter identity id_i. Therefore, there are exactly k distinct voter identities $\mathsf{id}_1, ..., \mathsf{id}_k$ corresponding to the credentials $\mathsf{cr}_1, .., \mathsf{cr}_k$ for which we have $\tau \vDash \mathsf{Reg}(\mathsf{id}_i, \mathsf{cr}_i)$. By ***(1)***, we can conclude that there is a one-to-one correspondence between the distinct voter identities $\mathsf{id}_1, ..., \mathsf{id}_k$ and the trackers $\mathsf{tr}_1, ..., \mathsf{tr}_k$ for which $\mathsf{v} \neq \perp$ on BBres.

(b) From the weak consistency property $\Phi^{\circ}_{\mathsf{reg2}}$, i.e. $\tau \vDash \Phi^{\circ}_{\mathsf{reg2}}$, for any $\tau \vDash \mathsf{Reg}(\mathsf{id}, \mathsf{cr}) \wedge \mathsf{Reg}(\mathsf{id}, \mathsf{cr}')$, we have $\mathsf{cr} \neq \mathsf{cr}'$ for any non-adversarial voter identity id, i.e. id is honest and verified their vote. Assume there are s such voter identities $\mathsf{id}_1, \ldots, \mathsf{id}_s$ for which $\tau \vDash \mathsf{Ver}^{\circ}(\mathsf{id}_i, \mathsf{cr}_i, \mathsf{v}_i)$. By $\tau \vDash \Phi^{\circ}_{\mathsf{iv2}}$, for any $\mathsf{id}_i, \mathsf{id}_j \in \{\mathsf{id}_1, \ldots, \mathsf{id}_s\}$, $\tau \vDash \mathsf{Ver}^{\circ}(\mathsf{id}_i, \mathsf{cr}_i, \mathsf{v}_i) \wedge \mathsf{Ver}^{\circ}(\mathsf{id}_j, \mathsf{cr}_j, \mathsf{v}_j)$ implies that $\mathsf{cr}_i \neq \mathsf{cr}_j$ for some $i \neq j$. Thus, there are exactly s mutually distinct public credentials $\mathsf{cr}_1, \ldots, \mathsf{cr}_s$ corresponding to the voter identities $\mathsf{id}_1, \ldots, \mathsf{id}_s$ for which we have $\tau \vDash \mathsf{Reg}(\mathsf{id}_i, \mathsf{cr}_i)$. Thus, we establish a one-to-one correspondence between registered non-adversarial voter identities $\mathsf{id}_1, \ldots, \mathsf{id}_s$ and their public credentials $\mathsf{cr}_1, \ldots, \mathsf{cr}_s$.

On the other hand, by $\tau \vDash \Psi_{\mathsf{E2E}'}$, there are precisely n distinct voter identities $\{\mathsf{id}_1, ..., \mathsf{id}_n\}$ since $\mathsf{n} = |\{\mathsf{cr} \mid \tau \vDash \mathsf{BBreg}(\mathsf{cr})\}|$. Among those voter identities, s of them, i.e. $\mathsf{id}_1, \ldots, \mathsf{id}_s$, are honest and verified their votes. Thus, the remaining identities, i.e. $\mathsf{id}_{s+1}, \ldots, \mathsf{id}_n$, are the adversarial voter identities, i.e. $\mathsf{id}_i \in \mathsf{Adv}^{\circ}(\mathsf{id})$. Moreover, from the public credentials $\mathsf{cr}_1, \ldots, \mathsf{cr}_k$ associated with the trackers in the outcome for which $\mathsf{v} \neq \perp$, s of them, i.e. $\mathsf{cr}_1, \ldots, \mathsf{cr}_s$, are associated with the voter identities $\mathsf{id}_1, \ldots, \mathsf{id}_s$. Thus, the remaining public credentials, i.e. $\mathsf{cr}_{s+1}, \ldots, \mathsf{cr}_k$, are associated with the adversarial voter identities. There are $\mathsf{k} - \mathsf{s}$ such public credentials. Since $\mathsf{k} - \mathsf{s} \leq \mathsf{n} - \mathsf{s}$, we can choose for each credential $\mathsf{cr}_i \in \{\mathsf{cr}_{s+1}, \ldots, \mathsf{cr}_k\}$ a distinct voter identity $\mathsf{id}_j \in \{\mathsf{id}_{s+1}, \ldots, \mathsf{id}_n\}$. Thus, we establish a one-to-one correspondence between $\mathsf{k} - \mathsf{s}$ mutually distinct adversarial voter identities and the public credentials $\mathsf{cr}_{s+1}, \ldots, \mathsf{cr}_k$.

Overall, we establish a one-to-one correspondence between distinct voter identities $\mathsf{id}_1, ..., \mathsf{id}_k$ and public credentials $\mathsf{cr}_1, .., \mathsf{cr}_k$. Hence, by ***(1)***, we can conclude that there is a one-to-one correspondence between the distinct voter identities $\mathsf{id}_1, ..., \mathsf{id}_k$ and the trackers $\mathsf{tr}_1, ..., \mathsf{tr}_k$ for which $\mathsf{v} \neq \perp$ on BBres.

For the rest of the proof, consider $\diamond, \triangledown \in \{\circ, \bullet\}$ for ***(a)*** and $\diamond, \triangledown = \circ$ for ***(b)***.

(3) In order to associate each vote $\mathsf{v} \neq \perp$ in the outcome with a registered voter identity, we define the following set:

$$\mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau) = \{(\mathsf{id}, \mathsf{v}) \mid \tau \vDash \mathsf{BBres}(\mathsf{tr}, \mathsf{v}) \wedge \mathsf{Link}(\mathsf{cr}, \mathsf{tr}) \wedge \mathsf{Reg}(\mathsf{id}, \mathsf{cr}) \wedge \mathsf{v} \neq \perp\}.$$

According to the one-to-one correspondence between trackers and voter identities established at point ***(2)***, we have:

$$|\mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau)| = |\{(\mathsf{tr}, \mathsf{v}) \mid \tau \vDash \mathsf{BBres}(\mathsf{tr}, \mathsf{v}) \wedge \mathsf{v} \neq \perp\}| = \mathsf{k}.$$

On the other hand, by Definition 3, we have:

$$\mathsf{Res}(\tau) = \{| \mathsf{v} \mid \exists \mathsf{tr}.\ \tau \vDash \mathsf{BBres}(\mathsf{tr}, \mathsf{v}) \wedge \mathsf{v} \neq \perp |\}.$$

Thus, $\mathsf{Res}(\tau) = \{|\, \mathsf{v} \mid (\mathsf{id}, \mathsf{v}) \in \mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau) \,|\}$.

(4) By $\tau \vDash \Phi^{\diamond}_{\mathsf{res}}$, for any $(\mathsf{id}, \mathsf{v}) \in \mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau)$, either $\tau \vDash \mathsf{Vote}(\mathsf{id}, \mathsf{v})$ or $\tau \vDash \mathsf{Adv}^{\diamond}(\mathsf{id})$, and these events may not be mutually exclusive. Let us define the following multisets:

$$\begin{aligned} \mathsf{Vote}_{\mathsf{ID},\mathsf{V}}(\tau) &= \{|\, (\mathsf{id}, \mathsf{v}) \mid \tau \vDash \mathsf{Vote}(\mathsf{id}, \mathsf{v}) \,|\} \\ \mathsf{Adv}_{\mathsf{ID},\mathsf{V}}(\tau) &= \{|\, (\mathsf{id}, \mathsf{v}) \mid \mathsf{id} \in \mathsf{Adv}^{\diamond}(\tau) \,|\} \end{aligned}$$

Considering the multisets above, we can define the following sets:

$$\begin{aligned} R'_2 &= (\, \mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau) \cap \mathsf{Vote}_{\mathsf{ID},\mathsf{V}}(\tau) \,) \setminus \mathsf{Adv}_{\mathsf{ID},\mathsf{V}}(\tau), \\ R'_3 &= \mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau) \cap \mathsf{Adv}_{\mathsf{ID},\mathsf{V}}(\tau). \end{aligned}$$

Then, any $(\mathsf{id}, \mathsf{v}) \in \mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau)$ is either in R'_2 or in R'_3, where $R'_2 \cap R'_3 = \emptyset$. Intuitively, R'_2 and R'_3 represent the partition of the final result according to honest or adversarial votes, i.e. $\mathsf{Res}(\tau) = R'_2 \uplus R'_3$. Now, let us define:

$$\mathsf{Ver}_{\mathsf{ID},\mathsf{V}}(\tau) = \{(\mathsf{id}, \mathsf{v}) \mid \exists \mathsf{cr}.\ \tau \vDash \mathsf{Ver}^{\triangledown}(\mathsf{id}, \mathsf{cr}, \mathsf{v}) \ \wedge\ \mathsf{Reg}(\mathsf{id}, \mathsf{cr})\}$$

and

$$\begin{aligned} R_1 &= \mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau) \cap \mathsf{Ver}_{\mathsf{ID},\mathsf{V}}(\tau), \\ R_2 &= R'_2 \setminus \mathsf{Ver}_{\mathsf{ID},\mathsf{V}}(\tau), \\ R_3 &= R'_3 \setminus \mathsf{Ver}_{\mathsf{ID},\mathsf{V}}(\tau). \end{aligned}$$

By definition, $R_1 \cap R_2 = R_1 \cap R_3 = R_2 \cap R_3 = \emptyset$. Thus, we can conclude $\mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau) = R_1 \uplus R_2 \uplus R_3$, i.e. they form a partition. It follows that $\mathsf{Res}(\tau) = V_1 \uplus V_2 \uplus V_3$, where $V_i = \{|\, \mathsf{v} \mid \exists \mathsf{id}.\ (\mathsf{id}, \mathsf{v}) \in R_i \,|\}$ for $i = 1, 2, 3$. Next, we show that the multisets V_1, V_2, V_3 satisfy the requirements of Definition 4, respectively, i.e.

$$\begin{aligned} &(1)\ V_1 = \mathsf{Ver}^{\triangledown}(\tau), \\ &(2)\ V_2 \in \mathsf{Vote}^{\triangledown,\diamond}(\tau), \\ &(3)\ |V_3| \leq |\mathsf{Adv}^{\diamond}(\tau) \setminus \mathsf{Ver}^{\triangledown}_{\mathsf{id}}(\tau)|. \end{aligned}$$

(4.1) By definition, $V_1 \subseteq \mathsf{Ver}^{\triangledown}(\tau)$. Thus, we show $\mathsf{Ver}^{\triangledown}(\tau) \subseteq V_1$. Let $\mathsf{id}_1, \ldots, \mathsf{id}_q$ be the set of (mutually distinct) voter identities who verified their votes (different from $\perp$) for some credential. This means that there are non-empty sets of votes $A_1, \ldots, A_q$ such that

$$\forall \mathsf{i} \in \{1, \ldots, q\}, \forall \mathsf{v} \in A_i, \exists \mathsf{cr}.\ \tau \vDash \mathsf{Ver}^{\triangledown}(\mathsf{id}_i, \mathsf{cr}, \mathsf{v}).$$

Intuitively, for each id_i, A_i represents the set of votes that are verified by that voter and which satisfy the revote policy. By definition, we have $\mathsf{Ver}^{\triangledown}(\tau) = A_1 \uplus \ldots \uplus A_q$.

For each $\mathsf{i} \in \{1, \ldots, q\}$, let C_i be the set of credentials for which $\tau \vDash \mathsf{Ver}^{\triangledown}(\mathsf{id}_i, \mathsf{cr}, \mathsf{v})$. From no clash property, i.e. $\tau \vDash \Phi^{\triangledown}_{\mathsf{iv2}}$, we deduce that $C_i \cap C_j = \emptyset$ for any $\mathsf{i} \neq \mathsf{j}$. Furthermore, assume $C_i = \{\mathsf{cr}_1, \ldots, \mathsf{cr}_s\}$ for some $\mathsf{i} \in \{1, \ldots, q\}$. By $\tau \vDash \Phi^{\triangledown}_{\mathsf{iv3}} \wedge \Phi_{\mathsf{link2}}$,

$$\begin{aligned} \tau &\vDash \mathsf{Link}(\mathsf{cr}_1, \mathsf{tr}_1) \wedge \ldots \wedge \mathsf{Link}(\mathsf{cr}_s, \mathsf{tr}_s), \\ \tau &\vDash \mathsf{BBreg}(\mathsf{cr}_1) \wedge \ldots \wedge \mathsf{BBreg}(\mathsf{cr}_s), \end{aligned}$$

for mutually distinct $\mathsf{tr}_1, \ldots, \mathsf{tr}_s$. Let T_i be the set of trackers linked to the credentials in C_i, i.e. $T_i = \{\mathsf{tr}_1, \ldots, \mathsf{tr}_s\}$. Then, by $\tau \vDash \Phi_{\mathsf{link1}}$ and $C_i \cap C_j = \emptyset$, we conclude $T_i \cap T_j = \emptyset$ for any $\mathsf{i} \neq \mathsf{j}$. Moreover, there is a one-to-one correspondence between C_i and T_i.

On the other hand, as we showed at point ***(1)***, each cr on BBreg is linked to a unique tr on BBres with $\tau \vDash \mathsf{Link}(\mathsf{cr}, \mathsf{tr})$. From the consistency properties Φ_{link1} and Φ_{link2}, the same credential cannot be linked to two different trackers. Then, we can deduce that each set T_i of trackers, that are linked to the credentials in C_i, are included in the final result. Moreover, by $\tau \vDash \Phi_{\mathsf{one}}$, there is at most one vote per tracker in the result. Therefore, we have

$$\tau \vDash \mathsf{BBres}(\mathsf{tr}_1, \mathsf{v}'_1) \wedge \ldots \wedge \mathsf{BBres}(\mathsf{tr}_\mathsf{s}, \mathsf{v}'_\mathsf{s}),$$

for some votes $\mathsf{v}'_1, \ldots, \mathsf{v}'_\mathsf{s}$.

For each $\mathsf{i} \in \{1, \ldots, \mathsf{q}\}$, let A'_i be the set of votes in the result that correspond to the trackers in T_i. From one-to-one correspondence between C_i (associated with the set A_i) and T_i (associated with the set A'_i), there is also a one-to-one correspondence between the sets A_i and A'_i. We show that these sets are actually equal. Indeed, for any $\mathsf{cr} \in C_\mathsf{i}$, $\mathsf{v} \in A_\mathsf{i}$ and associated $\mathsf{tr} \in T_\mathsf{i}$, $\mathsf{v}' \in A'_\mathsf{i}$, we have

$$\tau \vDash \mathsf{Ver}^{\triangledown}(\mathsf{id}_\mathsf{i}, \mathsf{cr}, \mathsf{v}) \wedge \mathsf{Link}(\mathsf{cr}, \mathsf{tr}) \wedge \mathsf{BBres}(\mathsf{tr}, \mathsf{v}').$$

By $\tau \vDash \Phi^{\triangledown}_{\mathsf{iv1}}$, we deduce $\mathsf{v} = \mathsf{v}'$, and we can conclude that $A_\mathsf{i} = A'_\mathsf{i}$. Therefore, we obtain:

$$\mathsf{Ver}^{\triangledown}(\tau) = A'_1 \uplus \ldots \uplus A'_\mathsf{q} = \{\mathsf{v}'_1, ..., \mathsf{v}'_\mathsf{p}\} \qquad (\star)$$

for some votes $\mathsf{v}'_1, ..., \mathsf{v}'_\mathsf{p}$ and some $\mathsf{p} \geq \mathsf{q}$ (every voter verified at least one vote).

On the other hand, by definition of the set R_1, it consists of all the pairs $(\mathsf{id}, \mathsf{v}) \in \mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau)$ such that $\mathsf{v} \neq \perp$ and some id' verified v. Thus, we deduce

$$\{(\mathsf{id}_1, \mathsf{v}'_1), \ldots, (\mathsf{id}_\mathsf{p}, \mathsf{v}'_\mathsf{p})\} \subseteq R_1, \text{ and therefore}$$
$$\{\mathsf{v}'_1, \ldots, \mathsf{v}'_\mathsf{p}\} \subseteq V_1 \qquad (\star\star)$$

From $(\star)$ and $(\star\star)$, we deduce $\mathsf{Ver}^{\triangledown}(\tau) \subseteq V_1$ and we can conclude $V_1 = \mathsf{Ver}^{\triangledown}(\tau)$ as required. Furthermore, since $\mathsf{p} \geq \mathsf{q}$, we have also established

$$\mid \mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau) \cap \mathsf{Ver}_{\mathsf{ID},\mathsf{V}}(\tau) \mid \geq \mid \mathsf{Ver}^{\triangledown}_{\mathsf{id}}(\tau) \mid \qquad (*)$$

i.e. the number of verified votes in the final result is greater than or equal to the number of voters for whom their votes are verified.

(4.2) We show that $V_2 \in \mathsf{Vote}^{\triangledown,\diamond}(\tau)$. By definition, we have

$$R_2 = (\mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau) \cap \mathsf{Vote}_{\mathsf{ID},\mathsf{V}}(\tau)) \setminus (\mathsf{Ver}_{\mathsf{ID},\mathsf{V}}(\tau) \cup \mathsf{Adv}_{\mathsf{ID},\mathsf{V}}(\tau))$$

and $V_2 = \{\!| \mathsf{v} \mid \exists \mathsf{id}.\ (\mathsf{id}, \mathsf{v}) \in R_2 |\!\}$. Furthermore, by ***(2)***, there can be no two distinct identities associated to each vote in the result:

$$R_2 = \{(\mathsf{id}_1, \mathsf{v}_1), \ldots, (\mathsf{id}_\mathsf{s}, \mathsf{v}_\mathsf{s})\} \text{ and } V_2 = \{\mathsf{v}_1, \ldots, \mathsf{v}_\mathsf{s}\}$$

for some mutually distinct $\mathsf{id}_1, \ldots, \mathsf{id}_\mathsf{s}$. Thus, for all $\mathsf{i} \in \{1, \ldots, \mathsf{s}\}$ we have:

$$\tau \vDash \mathsf{Vote}(\mathsf{id}_\mathsf{i}, \mathsf{v}_\mathsf{i}) \ \wedge \ \mathsf{id}_\mathsf{i} \notin (\ \mathsf{Ver}^{\triangledown}_{\mathsf{id}}(\tau) \cup \mathsf{Adv}^{\diamond}(\tau)\ )$$

Therefore, using the definition of $\mathsf{Vote}^{\triangledown,\diamond}(\tau)$, we can conclude that $V_2 \in \mathsf{Vote}^{\triangledown,\diamond}(\tau)$.

(4.3) We show that $|V_3| \leq |\mathsf{Adv}^{\diamond}(\tau) \setminus \mathsf{Ver}^{\triangledown}_{\mathsf{id}}(\tau)|$.

(a) By definition, we have

$$R_3 = (\mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau) \cap \mathsf{Adv}_{\mathsf{ID},\mathsf{V}}(\tau)) \setminus \mathsf{Ver}_{\mathsf{ID},\mathsf{V}}(\tau)$$

and $V_3 = \{\!|\, \mathsf{v} \mid \exists \mathsf{id}.\ (\mathsf{id}, \mathsf{v}) \in R_3 \,|\!\}$. Furthermore, by ***(2)***,

$$\mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau) \cap \mathsf{Adv}_{\mathsf{ID},\mathsf{V}}(\tau) = \{(\mathsf{id}_1, \mathsf{v}_1), \ldots, (\mathsf{id}_\mathsf{t}, \mathsf{v}_\mathsf{t})\}$$

for some mutually distinct $\mathsf{id}_1, \ldots, \mathsf{id}_\mathsf{t}$. We can also deduce that

$$|\mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau) \cap \mathsf{Adv}_{\mathsf{ID},\mathsf{V}}(\tau)| \leq |\mathsf{Adv}^{\diamond}(\tau)| \qquad (\dagger)$$

By $(*)$ and $\tau \vDash \Phi_{\mathsf{one}}$, there is one and only one vote in the result for each tracker; therefore for each voter id by ***(3)***, we deduce

$$|\ \mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau) \cap \mathsf{Ver}_{\mathsf{ID},\mathsf{V}} \cap \mathsf{Adv}_{\mathsf{ID},\mathsf{V}}(\tau)\ | \geq |\ \mathsf{Ver}^{\triangledown}_{\mathsf{id}}(\tau) \cap \mathsf{Adv}^{\diamond}(\tau)\ | \quad (\dagger\dagger)$$

Thus, we can conclude that

$$\begin{array}{rl} |R_3| = & |\ (\mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau) \cap \mathsf{Adv}_{\mathsf{ID},\mathsf{V}}(\tau)) \setminus \mathsf{Ver}_{\mathsf{ID},\mathsf{V}}(\tau)\ | \\ \leq & |\ (\mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau) \cap \mathsf{Adv}_{\mathsf{ID},\mathsf{V}}(\tau)) \setminus (\mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau) \cap \mathsf{Ver}_{\mathsf{ID},\mathsf{V}}(\tau))\ | \\ \leq & |\ \mathsf{Adv}^{\diamond}(\tau) \setminus \mathsf{Ver}^{\triangledown}_{\mathsf{id}}(\tau)\ | \end{array}$$

where, to deduce the second inequality, we rely on $(\dagger), (\dagger\dagger)$ and the general set theory fact that for all sets A_1, A_2, B_1, B_2:

$$\begin{array}{c} |A_1| \leq |A_2| \ \wedge\ |A_1 \cap B_1| \geq |A_2 \cap B_2| \\ \Longrightarrow\ |A_1 \setminus B_1| \leq |A_2 \setminus B_2| \end{array}$$

Hence, $|V_3| = |R_3| \leq |\mathsf{Adv}^{\diamond}(\tau) \setminus \mathsf{Ver}^{\triangledown}_{\mathsf{id}}(\tau)|$, as required.

(b) By definition, we have

$$R_3 = \mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau) \cap \mathsf{Adv}_{\mathsf{ID},\mathsf{V}}(\tau)$$

and $V_3 = \{\!|\, \mathsf{v} \mid \exists \mathsf{id}.\ (\mathsf{id}, \mathsf{v}) \in R_3 \,|\!\}$. Furthermore, as shown at point ***(2)***,

$$\mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau) \cap \mathsf{Adv}_{\mathsf{ID},\mathsf{V}}(\tau) = \{(\mathsf{id}_1, \mathsf{v}_1), \ldots, (\mathsf{id}_{\mathsf{k}-\mathsf{s}}, \mathsf{v}_{\mathsf{k}-\mathsf{s}})\}$$

for some mutually distinct $\mathsf{id}_1, \ldots, \mathsf{id}_{\mathsf{k}-\mathsf{s}}$, where $|\mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau)| = \mathsf{k}$ and s is the number of pairs in $\mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau)$ corresponding to the non-adversarial voter identities.
By $\tau \vDash \Psi_{\mathsf{E2E}'}$ and ***(2)***, there are precisely n distinct voter identities and $\mathsf{n} - \mathsf{s}$ of them are the adversarial voter identities, i.e. $|\mathsf{Adv}^{\circ}(\tau)| = \mathsf{n} - \mathsf{s}$. Thus, since $\mathsf{k} \leq \mathsf{n}$, we have $\mathsf{k} - \mathsf{s} \leq \mathsf{n} - \mathsf{s}$. Therefore, we can deduce that

$$|\mathsf{Res}_{\mathsf{ID},\mathsf{V}}(\tau) \cap \mathsf{Adv}_{\mathsf{ID},\mathsf{V}}(\tau)| \leq |\mathsf{Adv}^{\circ}(\tau)|$$

Hence, $|V_3| = |R_3| \leq |\mathsf{Adv}^{\circ}(\tau)|$, as required.

□

4.6 Case Study: Verifiable E-Voting Example

Assume the verifiable e-voting example, Example 3, in Section 2.5 records the events specified in Figure 4.2. For the voter id, election authorities generate a public credential cr and a tracker

tr, recording the event $\mathsf{Reg}(\mathsf{id}, \mathsf{cr})$ for the association between id and cr, the event $\mathsf{BBreg}(\mathsf{cr})$ for the eligibility of cr, and the event $\mathsf{Link}(\mathsf{cr}, \mathsf{tr})$ for linking cr to tr. Then, the authorities send the credential and tracker to the voter via a private channel. In the voting phase, the voter id votes for v, generating a ballot b for the vote v and recording the event $\mathsf{Vote}(\mathsf{id}, \mathsf{v})$, and then the voter sends the ballot with their public credential to the authorities via a public channel. Authorities publish the ballot on BB with the public credential who cast it. In the tally phase, election authorities open the ballot b for the credential cr and obtain the vote v'. Then, they extract the tracker tr which is linked to the credential cr, and publish the vote v' together with the tracker tr, recording the event $\mathsf{BBres}(\mathsf{tr}, \mathsf{v}')$. The voter id uses their tracker tr received from the election authorities in the setup phase to verify their vote v on BBres. As soon as their vote v matches with the one v' recorded with their tracker tr, i.e. $\mathsf{v} = \mathsf{v}'$, the voter completes the individual verification, recording the event $\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v})$.

In this example, individual verification procedures are performed at the end of the election via trackers. Assuming there is no revoting, i.e. $\Omega(\mathsf{id}, \mathsf{v}, \mathsf{t}) = \mathsf{true}$, and all the voters are honest, $\mathsf{Ver}^{\nabla}(\mathsf{id}, \mathsf{cr}, \mathsf{v}) = \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v})$. Since $\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v})$ is recorded as soon as the voter id matches the vote v with the one recorded with their tracker tr on BBres, and there is an event $\mathsf{Link}(\mathsf{cr}, \mathsf{tr})$ recorded when election authorities link the voter's public credential cr to tr,

$$\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v}) \ \wedge \ \mathsf{Link}(\mathsf{cr}, \mathsf{tr}) \ \wedge \ \mathsf{BBres}(\mathsf{tr}, \mathsf{v}') \Rightarrow \mathsf{v} = \mathsf{v}',$$

i.e. Φ_{iv1} holds for the specification $\mathcal{S}_{\mathsf{ex}}$ corresponding to Figure 4.2. Assuming election authorities are all honest, each public credential is freshly generated for each voter id, i.e.

$$\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v}) \ \wedge \ \mathsf{Verified}(\mathsf{id}', \mathsf{cr}, \mathsf{v}') \Rightarrow \mathsf{id} = \mathsf{id}'.$$

Therefore, $\mathcal{S}_{\mathsf{ex}} \vDash \Phi_{\mathsf{iv2}}$. Moreover, each public credential is privately sent to the corresponding voter id and linked to a freshly generated tracker. Therefore, if a voter id successfully performs a verification procedure, their public credential should be registered on BBreg and linked to tr, i.e.

$$\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v}) \Rightarrow \mathsf{Link}(\mathsf{cr}, \mathsf{tr}) \ \wedge \ \mathsf{BBreg}(\mathsf{cr}).$$

Thus, $\mathcal{S}_{\mathsf{ex}} \vDash \Phi_{\mathsf{iv3}}$.

The election result consists of all the votes published together with trackers on BBres. Each $(\mathsf{tr}, \mathsf{v})$ pair in the outcome is obtained from a pair $(\mathsf{cr}, \mathsf{b})$, where $\mathsf{open}(\mathsf{b}) = \mathsf{v}$ and cr is linked to tr, recording the event $\mathsf{Link}(\mathsf{cr}, \mathsf{tr})$. Furthermore, each cr is generated for a voter id, i.e. associated with a voter id, recording the event $\mathsf{Reg}(\mathsf{id}, \mathsf{cr})$. Therefore,

$$\mathsf{BBres}(\mathsf{tr}, \mathsf{v}) \Rightarrow \mathsf{Link}(\mathsf{cr}, \mathsf{tr}) \ \wedge \ \mathsf{BBreg}(\mathsf{cr}) \ \wedge \ \mathsf{Reg}(\mathsf{id}, \mathsf{cr}),$$

i.e. $\mathcal{S}_{\mathsf{ex}} \vDash \Phi_{\mathsf{eli}}$. Since all the voters are honest, for each pair $(\mathsf{tr}, \mathsf{v})$ in the outcome, either $\mathsf{v} = \bot$ or the vote v has been cast by a voter id, recording the event $\mathsf{Vote}(\mathsf{id}, \mathsf{v})$. This follows that

$$\mathsf{BBres}(\mathsf{tr}, \mathsf{v}) \ \wedge \ \mathsf{Link}(\mathsf{cr}, \mathsf{tr}) \ \wedge \ \mathsf{BBreg}(\mathsf{cr}) \Rightarrow \mathsf{Vote}(\mathsf{id}, \mathsf{v}) \ \vee \ \mathsf{v} = \bot,$$

i.e. $\mathcal{S}_{\mathsf{ex}} \vDash \Phi_{\mathsf{res}}$.

Finally, since all public credentials and trackers are freshly generated for each voter, the consistency properties Φ_{reg1}, Φ_{reg2}, Φ_{link1}, and Φ_{link2} hold for $\mathcal{S}_{\mathsf{ex}}$. Hence, $\mathcal{S}_{\mathsf{ex}}$ satisfies end-to-end verifiability.

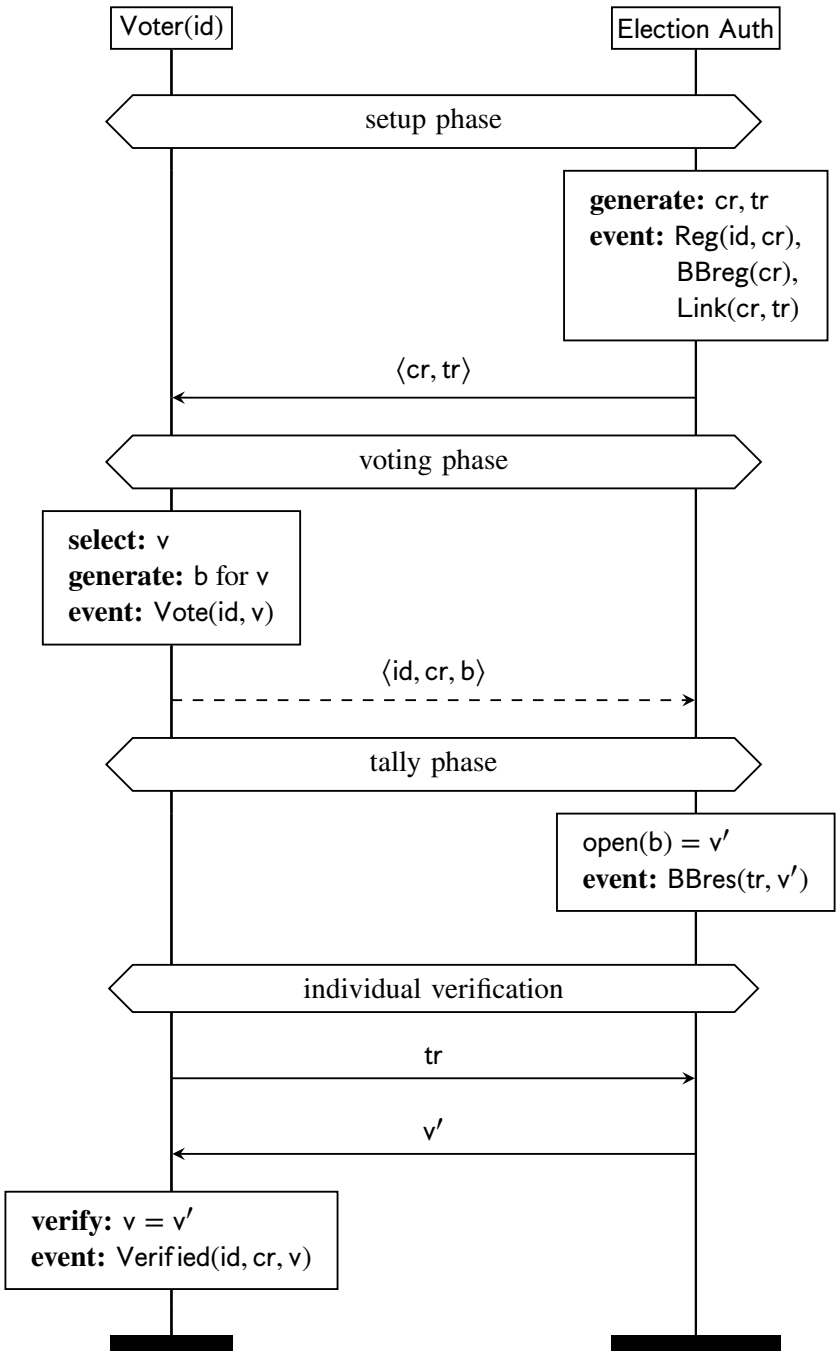


FIGURE 4.2: Verifiable e-voting protocol example with events.

Part II

Case Studies

Chapter 5

Helios

Helios [1, 34] is the first web-based e-voting protocol that provides end-to-end verifiability guarantees for voters. It aims to obtain verifiability notions defined in the literature in a practical protocol that allows voters to audit their ballot generation, verify their ballot on a public bulletin board, and the election outcome that comes from the ballots collected on the bulletin board. Helios is not suitable for government elections, where coercion is a concern. It aims to be used in low-stakes elections that require ballot secrecy, such as local club elections and university elections. Thus, it has been used in many elections, including the one to elect the Catholic University of Louvain president in 2009 and the ones to elect new board directors of the International Association for Cryptographic Research (IACR) since 2010 [35].

The first variant of Helios [1], designed in 2008, deploys the Helios server to generate the election key pair, register voters, manage a public bulletin board that displays all ballots cast, and tally the ballots using a mixnet at the end of the election. The second variant of Helios [3] upgrades the first to be used in the presidential election of the Catholic University of Louvain in 2009, which considers an additional registration server to register voters with the credentials not known by the Helios server, and multiple talliers to generate the election's public key in a distributed fashion, where the private key is not known to a single tallier. However, none is secure against ballot stuffing attacks by a corrupt server. The corrupt server in [1] or the corrupt registration server in [3] can cast ballots on behalf of the voters, and public checks on the bulletin board may not detect this.

Helios has been extensively analysed in the literature [38, 22, 40]. The first symbolic model of Helios for its second variant, presented in [38], considered a corrupt server to publish the election data and proved individual and universal verifiability based on boolean tests. Later, extending the adversary's abilities to corrupt the registrar and voting platform, [40] discovered clash attacks against the verifiability of Helios. Another symbolic model of Helios, presented in [22], considered an approach to evaluate privacy and verifiability with typecheckers and provided the first automated verification of Helios with homomorphic encryption. The approach in [22] included no clash property but did not consider eligibility verifiability. Considering all, the approaches presented so far are limited, and none of those analyses has modelled revoting as a feature of Helios.

This chapter provides a formal analysis of Helios based on the end-to-end election verifiability definition from Chapter 4, using the automated verification tool Tamarin. In this analysis, revoting is accounted for, the effects of different individual verification procedures on verifiability are explored, and the verifiability is checked automatically with Tamarin on several corruption scenarios for the protocol parties. As a result, the known attacks against Helios; ballot stuffing and clash attacks, are captured. In addition, new versions of clash attacks are found, accounting for revoting and individual verification at any time during the election.

Structure of the chapter: In Section 5.1, we describe the protocol structure and the election procedures of Helios. Then, in Section 5.2, we present the Tamarin specification of Helios with respect to different individual verification procedures and several corruption scenarios.

Finally, in Section 5.3, we provide the verifiability analysis of Helios, i.e. the security proofs and attacks found with automated verification for the concerned corruption scenarios.

5.1 Helios Protocol Structure

Helios has the regular parties of an e-voting protocol as described in Section 2.2:

- Administrator A is responsible for the election configuration, i.e. it determines the candidates and voters eligible for the election, and talliers if the Helios server is not deployed to generate the election key pair.
- Talliers T generate an election key pair: a private key and its corresponding public key, and tally all the ballots at the end of the election. A single tallier, the Helios server itself, may generate the election key pair, or multiple talliers generate the public key in a distributed way, where no tallier knows the private part of the key generated by the other tallier. For the tally, T either shuffles all the encrypted ballots through a mixnet and decrypts each randomised ballot or decrypts the single ciphertext corresponding to the homomorphic tally of the ballots.
- Registrar VR registers public credentials of the voters for the election. VR may generate a public credential for each voter, i.e. an alias, or it may just register the voter identities as public credentials.
- Voting server VS generates a login credential, i.e. a password, for each registered voter, accepts ballots from the voters if they are authenticated with the login credential, and then publishes them on the bulletin board.
- Voting platform VP allows voters to generate a ballot, i.e. an encryption of their choice with the election public key, and cast their ballot to the voting server via a login operation. It also allows voters to audit their ballot, revealing the encryption randomness.
- Voters V registered to the election as eligible may cast several ballots using their voting platform during the election or abstain from voting.
- Election auditors EA audit the public information recorded on the bulletin board. They ensure that all ballots have been cast by eligible public credentials, all the ballots recorded on the bulletin board are valid, the ballots to be tallied are the ones recorded last for the credentials on the bulletin board, and the election result corresponds to the tallied ballots. To ensure the correctness of the tally, for example, they verify the zero-knowledge proofs produced by T.

In Helios, VR and VS are usually subsumed under the same party, i.e. the Helios server. Helios relies on an append-only public bulletin board, denoted by BB, to display the public election data: the election's public key, the election candidates, the public credentials of the registered voters, the cast ballots, the tallied ballots, the produced zero-knowledge proofs and the election outcome. It allows voters to verify their ballots and election auditors to verify the public election data on BB. The data on BB can be displayed in portions; for example, the portion of BBkey displays the election's public key.

The election procedures and the individual verification procedure of Helios are described as follows:

Setup phase. A determines the list of candidates $\mathsf{v}_1, \ldots, \mathsf{v}_k$ and voters $\mathsf{id}_1, \ldots, \mathsf{id}_n$ that are eligible for the election, delegates T to generate the election key pair $(\mathsf{sk}_E, \mathsf{pk}_E)$, VR to generate public credentials $\mathsf{cr}_1, \ldots, \mathsf{cr}_n$, where each cr_i corresponds to the voter id_i, and VS to generate

a login credential pwd_i for each voter id_i. The public information displayed on the portions of BB is as follows:

$$\mathsf{BBkey} : \mathsf{pk}; \quad \mathsf{BBcand} : \mathsf{v}_1, \ldots, \mathsf{v}_k; \quad \mathsf{BBreg} : \mathsf{cr}_1, \ldots, \mathsf{cr}_n$$

In this phase, each voter id obtains a public credential cr and a password pwd.

Helios with identities. The administrator A in Helios may decide to use voter identities as public credentials; in this case, $\mathsf{cr} = \mathsf{id}$. Then, the cast ballot is displayed on the bulletin board next to the voter identity who cast it. However, using voter identities in the clear may violate privacy if T is corrupt and leaks the secret key. It could also be the case that the secret key itself can be broken in the future. Therefore, in general, it is not preferred.

Voting phase. V interacts with VP to construct a ballot b:

$$\begin{aligned}
\mathsf{VP}: &\ \text{downloads } \mathsf{pk}_E \in \mathsf{BBkey} \text{ and } \mathsf{v}_1, \ldots, \mathsf{v}_k \in \mathsf{BBcand},\\
\mathsf{V}: &\ \text{selects } \mathsf{v} \in \mathsf{BBcand},\\
\mathsf{VP}: &\ \text{encrypts } \mathsf{v} \text{ with } \mathsf{pk}_E \text{ and a randomness } \mathsf{r} : \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk}_E, \mathsf{r}),\\
&\ \text{produces a proof} : \mathsf{p} = \mathsf{pr}_R(\mathsf{c}, \mathsf{r}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle).
\end{aligned}$$

The zero-knowledge proof p is produced if a homomorphic tally is chosen for the election, which proves the encrypted vote is within a valid range $\langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle$. Thus, VP constructs the ballot $\mathsf{b} = \langle \mathsf{c}, \mathsf{p} \rangle$, and V decides to cast or audit it. In the case of auditing, VP reveals the vote v and the randomness r so that V can encrypt v with the same randomness and match the ciphertexts. Otherwise, VP asks for login credentials of V to cast the ballot b. V provides their id and pwd obtained in the setup phase, which prompts a connection to VS. If VS authenticates V, it receives the ballot b on behalf of id, verifies the proof p, and then records it on BBcast next to the public credential cr of id.

Tally phase. At the end of the voting phase, VS selects the *last* ballot recorded on BBcast for each credential and publishes it on BBtally. The final version of BBtally is:

$$\mathsf{BBtally} : (\mathsf{cr}_1, \mathsf{b}_1), \ldots, (\mathsf{cr}_n, \mathsf{b}_n),$$

where $\mathsf{b} = \perp$ if no ballot was cast for cr. T tallies the ciphertexts corresponding to non-empty ballots on BBtally and announces the final result along with a zero-knowledge proof for the result, showing that it corresponds to the input, i.e. the set of ciphertexts. More specifically, if the protocol requires a tally with mixnets, there will be proofs for correct shuffling and decryption of the ciphertexts. On the other hand, if it requires a homomorphic tally, then the proof will be for the correctness of the homomorphic tally.

Individual verification. In Helios, V can verify that the ballot generated by VP correctly encodes their chosen vote with an audit option. VP does not cast the ballot audited; instead, it generates another ballot for casting. However, since each ballot is generated with a possibility of an audit, the corrupt VP cannot cheat on the ballot generation or cannot take the risk of being caught. Thus, the voter ensures that the ballot is cast as intended. Moreover, V can verify that VS correctly captures and records their ballot on BB. For that, Helios provides V with the hash of their ballot and presents the same hash next to their credential on BB. Then, V has to match the two hashes to ensure the ballot is recorded as cast.

Helios allows revoting, and each time a ballot is received for a credential, the old ballot is replaced with the new one on BB. Thus, V cannot verify an older ballot on BB. Helios publishes all the ballots to be tallied on BB. If V verifies their last ballot cast on BBtally, then V can ensure that their ballot will be tallied and their vote will be counted on their behalf. The tally procedure of Helios is verifiable by anyone, i.e. the public data on BB allows one to verify the lists of tallied ballots and counted votes with produced zero-knowledge proofs.

5.2 Tamarin Specification of Helios

This section presents the Tamarin specification $\mathcal{S}_{\mathcal{H}}$ of the Helios protocol. To distinguish different individual verification procedures and adversary models, the specification $\mathcal{S}_{\mathcal{H}}$ is separated into three components, i.e. $\mathcal{S}_{\mathcal{H}} = (\mathcal{P}_{\mathcal{H}}, \mathcal{V}, \mathcal{A})$, where

- $\mathcal{P}_{\mathcal{H}}$ models the actions of the honest parties,
- $\mathcal{V}$ models individual verification procedures,
- $\mathcal{A}$ models adversarial capabilities.

Each component above has its own rules and restrictions that are intended for the execution of its rules.

The protocol specification $\mathcal{P}_{\mathcal{H}}$. Helios uses the ElGamal encryption algorithm to encrypt the votes, and a non-interactive zero-knowledge protocol to prove that the encrypted votes are in a valid range of eligible candidates. Thus, the specification requires the following equations defined for $\mathcal{E}_{\mathcal{H}}$:

(1) $\mathsf{dec}(\mathsf{enc}(x, \mathsf{pk}(y), z), y) = x$,
(2) $(\forall i)\ \mathsf{ver}_R(\mathsf{pr}_R(\mathsf{enc}(x_i, y, z), z, \langle x_1, \ldots, x_k\rangle), \mathsf{enc}(x_i, y, z), y, \langle x_1, \ldots, x_k\rangle) = \mathsf{true}$.

We have described the protocol parties of Helios in Section 5.1, as abbreviated by A, T, VR, VS, V, EA. Now, we model the actions of those parties with the rules in $\mathcal{P}_{\mathcal{H}}$. Any rule in $\mathcal{P}_{\mathcal{H}}$ is denoted by $\mathsf{R}^{\mathsf{a}}_{\mathsf{n}}$, where a is the party's abbreviation, and n is the name describing the action in the rule. For simplicity, we remove the symbol ! from the facts, representing persistent facts, since we consider all protocol facts except the special ones (In, Out, and Fr) persistent. We model Helios, considering the three phases of an election.

Setup phase. T generates an election key pair $(\mathsf{sk}_E, \mathsf{pk}_E)$ and publishes pk_E on BBkey, whereas A determines the lists of eligible candidates and voters, which we model with the following rules:

$$\mathsf{R}^{\mathsf{T}}_{\mathsf{key}}: \ [\ \mathsf{Fr}(\mathsf{sk}_E)\ ] -[\ \mathsf{BBkey}(\mathsf{pk}(\mathsf{sk}_E))\ ]\to [\ \mathsf{SkE}(\mathsf{sk}_E), \mathsf{BBkey}(\mathsf{pk}(\mathsf{sk}_E)), \mathsf{Out}(\mathsf{pk}(\mathsf{sk}_E))\ ]$$

$$\mathsf{R}^{\mathsf{A}}_{\mathsf{cand}}: \ \mathsf{let}\ \ \mathsf{vlist} = \langle v_1, \ldots, v_k\rangle\ \ \mathsf{in}$$
$$[\ \mathsf{In}(\mathsf{vlist})\ ] -[\ \mathsf{BBcand}(v_1), \ldots, \mathsf{BBcand}(v_k), \mathsf{Vlist}(\mathsf{vlist})\ ]\to$$
$$[\ \mathsf{BBcand}(v_1), \ldots, \mathsf{BBcand}(v_k), \mathsf{Vlist}(\mathsf{vlist})\ ]$$

$$\mathsf{R}^{\mathsf{A}}_{\mathsf{id}}: \ [\ \mathsf{In}(\mathsf{id})\ ] -[\ ]\to [\ \mathsf{Id}(\mathsf{id})\ ]$$

In the rule $\mathsf{R}^{\mathsf{T}}_{\mathsf{key}}$, the election's public key $\mathsf{pk}(\mathsf{sk}_E)$ is made available to the adversary with the fact Out. In the rules $\mathsf{R}^{\mathsf{A}}_{\mathsf{cand}}$ and $\mathsf{R}^{\mathsf{A}}_{\mathsf{id}}$, the eligible candidates and voters are received from the network with the fact In, implying that the adversary chooses, and thus, knows them. The fact Vlist records the list of candidates to produce and verify the zero-knowledge proofs in each ballot. Note that the candidates recorded in Vlist correspond to all the candidates on BBcand.

Continuing the setup phase, VR/VS generates a public credential for each id recorded in Id, whereas VS generates a password for each such id. VS also prepares BBcast, recording $\perp$ for each registered credential that will be filled with a ballot if the voter with that credential casts one, as follows:

$$\mathsf{R}^{\mathsf{VR/VS}}_{\mathsf{reg}}: \ [\ \mathsf{Id}(\mathsf{id}), \mathsf{Fr}(\mathsf{cr})\ ] -[\ \mathsf{Reg}(\mathsf{id}, \mathsf{cr}), \mathsf{BBreg}(\mathsf{cr})\ ]\to [\ \mathsf{Voter}(\mathsf{id}, \mathsf{cr}), \mathsf{BBreg}(\mathsf{cr}), \mathsf{Out}(\mathsf{cr})\ ]$$

$$\mathsf{R}^{\mathsf{VS}}_{\mathsf{pwd}}: \ [\ \mathsf{Id}(\mathsf{id}), \mathsf{Fr}(\mathsf{pwd})\ ] -[\ ]\to [\ \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd})\ ]$$

$$\mathsf{R}^{\mathsf{VS}}_{\mathsf{bb}}: \ [\ \mathsf{BBreg}(\mathsf{cr})\ ] -[\ \mathsf{BBcast}(\mathsf{cr}, \perp)\ ]\to [\ \mathsf{BBcast}(\mathsf{cr}, \perp)\ ]$$

In the rule $\mathsf{R}^{\mathsf{VR/VS}}_{\mathsf{reg}}$, VR/VS associates id with a public credential cr, which is recorded in the action fact Reg. A voter id obtains their credentials as recorded in the facts Voter and Pwd. The action facts BBreg and BBcast record the related information displayed on BB.

Voting phase. VP generates a ballot b for the voter id, encrypting the vote recorded on BBcand and producing a zero-knowledge proof using the vote list recorded in Vlist. The voter's authentication via a login credential is abstracted with the help of a hash function h. Hash functions are generally used to ensure data integrity. In this case, they ensure that the ballot is sent by the voter id who shares the secret, i.e. pwd, with VS. This is modelled by computing an authentication message $\mathsf{a} = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{b}\rangle)$ that hides pwd inside the hash h on VP's side, then computing a similar authentication message a' with the knowledge of pwd on VS's side and checking whether they match. If the authentication messages match and the proof p is inside the ballot is verified, VS records the ballot next to the voter's credential cr on BBcast. The actions of VP and VS are modelled in the following two rules:

$$\begin{aligned}
\mathsf{R}^{\mathsf{VP}}_{\mathsf{vote}}: \quad & \mathsf{let}\ \ \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk}_{\mathsf{E}}, \mathsf{r});\ \ \mathsf{p} = \mathsf{pr}_{\mathsf{R}}(\mathsf{c}, \mathsf{r}, \mathsf{vlist});\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{p}\rangle;\ \ \mathsf{a} = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{b}\rangle)\ \ \mathsf{in} \\
& [\ \mathsf{Voter}(\mathsf{id}, \mathsf{cr}), \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd}), \mathsf{BBcand}(\mathsf{v}), \mathsf{Vlist}(\mathsf{vlist}), \mathsf{BBkey}(\mathsf{pk}_{\mathsf{E}}), \mathsf{Fr}(\mathsf{r}), \mathsf{Fr}(\mathsf{t})\ ] \\
& -[\ \mathsf{Vote}(\mathsf{id}, \mathsf{v}), \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b}), \mathsf{VoteTime}(\mathsf{id}, \mathsf{v}, \mathsf{t})\ ]\rightarrow \\
& [\ \mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{b}, \mathsf{t}), \mathsf{Out}(\langle \mathsf{id}, \mathsf{b}, \mathsf{a}\rangle)\ ] \\
\mathsf{R}^{\mathsf{VS}}_{\mathsf{cast}}: \quad & \mathsf{let}\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{p}\rangle;\ \ \mathsf{a}' = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{b}\rangle)\ \ \mathsf{in} \\
& [\ \mathsf{In}(\langle \mathsf{id}, \mathsf{b}, \mathsf{a}\rangle), \mathsf{Voter}(\mathsf{id}, \mathsf{cr}), \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd}), \mathsf{BBkey}(\mathsf{pk}_{\mathsf{E}}), \mathsf{Vlist}(\mathsf{vlist})\ ] \\
& -[\ \mathsf{a}' \equiv \mathsf{a}, \mathsf{ver}_{\mathsf{R}}(\mathsf{p}, \mathsf{c}, \mathsf{pk}_{\mathsf{E}}, \mathsf{vlist}) \equiv \mathsf{true}, \mathsf{VScast}(\mathsf{id}, \mathsf{b}), \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]\rightarrow \\
& [\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ] \\
\Psi^{\mathsf{VS}}_{\mathsf{order}}: \quad & \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ @i\ \wedge\ \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b}')\ @j\ \wedge \\
& \mathsf{VScast}(\mathsf{id}, \mathsf{b})\ @k\ \wedge\ \mathsf{VScast}(\mathsf{id}, \mathsf{b}')\ @l\ \wedge\ i \prec j \Rightarrow k \prec l \\
\Psi^{\mathsf{VS/EA}}_{\mathsf{cast}}: \quad & \mathsf{BBcast}(\mathsf{cr}, \mathsf{b}) \Rightarrow \mathsf{BBreg}(\mathsf{cr})\ \wedge\ (\ \mathsf{b} \neq \bot \Rightarrow \\
& \mathsf{BBkey}(\mathsf{pk}_{\mathsf{E}})\ \wedge\ \mathsf{Vlist}(\mathsf{vlist})\ \wedge\ \mathsf{b} = \langle \mathsf{c}, \mathsf{p}\rangle\ \wedge\ \mathsf{ver}_{\mathsf{R}}(\mathsf{p}, \mathsf{c}, \mathsf{pk}_{\mathsf{E}}, \mathsf{vlist}) = \mathsf{true}\ )
\end{aligned}$$

The rule $\mathsf{R}^{\mathsf{VP}}_{\mathsf{vote}}$ can be executed many times for a voter, recording the event Vote, either for the same vote v or for a different vote v', which models revoting. The action fact VoteB records the ballot cast on VP, whereas VScast records it on VS. The fact Voted records the data necessary for individual verification. In reality, the voter knows what they voted for and obtains the ballot hash after casting one. The fact VoteTime is required for the revote policy. It records a fresh term t whenever the voter casts a vote v, which allows the revote policy to select the last vote cast among all cast votes of the voter id with a constraint on t.

On the other hand, the restrictions $\Psi^{\mathsf{VS}}_{\mathsf{order}}$ and $\Psi^{\mathsf{VS/EA}}_{\mathsf{cast}}$ ensure that VS processes the ballots cast by VP in the order they have been sent, any ballot published on BBcast corresponds to a registered credential, and if the ballot is not $\bot$, the proof p inside the ballot is also correctly verified. The second restriction also represents the public verifications of the election data provided on BB by the election auditors EA.

Tally phase. VS selects all the ballots to be tallied by retrieving the last ballot cast for each credential and publishes them on BBtally with the following rule and restriction:

$$\begin{aligned}
\mathsf{R}^{\mathsf{VS}}_{\mathsf{tally}}: \quad & [\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]-[\ \mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ ]\rightarrow[\ \mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ ] \\
\Psi^{\mathsf{VS/EA}}_{\mathsf{tally}}: \quad & \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ @i\ \wedge\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b}')\ @j\ \wedge\ \mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ @k \Rightarrow j \prec i\ \vee\ \mathsf{b} = \mathsf{b}'
\end{aligned}$$

The restriction $\Psi^{\mathsf{VS/EA}}_{\mathsf{tally}}$ ensures the ballot to be tallied is the last one among all cast ballots, which models the action of VS on selecting the correct ballot for each credential as well as the audits by EA for the correct tallying. The full protocol specification $\mathcal{P}_{\mathcal{H}}$ is given in Figure 5.3.

R^0_{ver} : **voter verifies the ballot on** BBcast
[Voted(id, cr, v, b, t), BBcast(cr, b)]
—[Verified(id, cr, v, t), VerB(id, cr, b)]↦[ ]

R^1_{ver} : **voter verifies the ballot on** BBtally
[Voted(id, cr, v, b, t), BBtally(cr, b)]—[Verified(id, cr, v, t)]↦[ ]

R^2_{ver} : **voter verifies there is no ballot on** BBtally
[Voter(id, cr), BBtally(cr, ⊥)]—[Verified(id, cr, ⊥, ⊥)]↦[ ]

R^0_{ver} **can be combined with restrictions below:**

Ψ_{last} : **the verified ballot is currently the last on BB**
$\mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ @i \wedge \mathsf{BBcast}(\mathsf{cr}, \mathsf{b}')\ @j \wedge$
$\mathsf{VerB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ @k \wedge i \prec k \wedge j \prec k \Rightarrow j \prec i \vee \mathsf{b} = \mathsf{b}'$

Ψ_{mine} : **all ballots currently on BB are cast by** id
$\mathsf{VerB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ @i \wedge \mathsf{BBcast}(\mathsf{cr}, \mathsf{b}')\ @j \wedge j \prec i$
$\Rightarrow \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b}')\ @k$

Specification	**Voter instructions**
$\mathcal{V}_1 : (R^0_{ver}; \Psi_{last})$	verify the last ballot on BBcast
$\mathcal{V}_2 : (R^0_{ver}; \Psi_{last} \wedge \Psi_{mine})$	as $\mathcal{V}_1$, and ensure all ballots are theirs
$\mathcal{V}_3 : (R^1_{ver})$	verify the ballot directly on BBtally
$\mathcal{V}_4 : (R^2_{ver})$	verify abstention directly on BBtally

FIGURE 5.1: Individual verification procedures.

Individual verification procedures $\mathcal{V}$**.** In the specification $S_{\mathcal{H}}$, individual verification procedures are intended for verifying the ballot on BB; thus, they are not concerned about verifying the correct vote encryption on the voting platform. Helios allows individual verification of the ballot cast anytime on BB until another ballot is cast for the same credential. In that case, the ballot present on BB is replaced with the new one. It also allows individual verification of an empty ballot, which is ⊥ next to an eligible credential, i.e. a voter can verify on BB that nobody has cast a ballot on their behalf. In addition, we model another version of BB in which no old ballot is replaced with the new one. Instead, the BB displays all cast ballots until the end of the election. Then, whenever a voter performs an individual verification for their last ballot cast, they can also verify that they cast all other ballots present on BB next to their credential, i.e. all other ballots are theirs. Note that this version of BB is not used in Helios. We model this for experimental purposes.

We present the specifications of the abovementioned individual verification procedures in Figure 5.1. The regular procedure, i.e. the individual verification of the last ballot cast anytime during the voting phase, is denoted by $\mathcal{V}_1$. The procedure for which we consider another version of BB, i.e. the voters verify their last ballot cast and all former ballots displayed next to their credentials are theirs, is denoted by $\mathcal{V}_2$. On the other hand, $\mathcal{V}_3$ denotes the procedure

$\mathcal{C}^{\mathsf{V}}_{\mathsf{corr}}$: **corrupt voter to reveal credentials**
[Voter(id, cr), Pwd(id, pwd)]—[Corr(id)]→[Out(⟨id, cr, pwd⟩)]

$\mathcal{C}^{\mathsf{T}}_{\mathsf{key}}$: **corrupt talliers to control the private election key**
[In(sk_{E})]—[BBkey(pk(sk_{E}))]→[SkE(sk_{E}), BBkey(pk(sk_{E}))]

$\mathcal{C}^{\mathsf{VS}}_{\mathsf{cast}}$: **corrupt server to stuff ballots**
[In(⟨cr, b⟩)]—[BBcast(cr, b)]→[BBcast(cr, b)]

$\mathcal{C}^{\mathsf{VR/VS}}_{\mathsf{reg}}$: **corrupt registrar/server to control credentials**
[In(cr), Id(id)]—[Reg(id, cr), BBreg(cr)]→[Voter(id, cr), BBreg(cr)]

$\mathcal{C}^{\mathsf{VP}}_{\mathsf{vote}}$: **corrupt platform to control encryption randomness**
rule $\mathsf{R}^{\mathsf{VP}}_{\mathsf{vote}}$ where Fr(r) is replaced with In(r)

FIGURE 5.2: Rules modelling corrupt parties in Helios.

Adversary Models	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$
Talliers	C	C	C	C
Server	H	C	C	C
Registrar	H	H	C	C
Voting Platform	H	H	H	C

TABLE 5.1: Adversary models for Helios.

that describes the individual verification of the last ballot cast at the end of the election, i.e. in the tally phase, and $\mathcal{V}_4$ represents the one for verifying the abstention.

Adversary models $\mathcal{A}$**.** For Helios, $\mathcal{A}$ models an adversary with the ability to corrupt parties in the ways described in Figure 5.2. The rules that model corrupt parties in the protocol are as follows. The rule $\mathcal{C}^{\mathsf{V}}_{\mathsf{corr}}$ models the corruption of voters any time during the election, where the voters leak their credentials to the adversary. Note that this rule brings flexibility in the way of corruption of the voters so that they can be corrupted after they cast a ballot. Thus, the corrupt voters are not labelled at the beginning of the election as they are modelled in [17]. The rule $\mathcal{C}^{\mathsf{T}}_{\mathsf{key}}$ allows the adversary to control the election's private key since its leakage is not a concern for verifiability. The rule $\mathcal{C}^{\mathsf{VS}}_{\mathsf{cast}}$ models a corrupt server that forgoes the prescribed way of validating ballots before casting them on BB. Therefore, it allows the adversary to stuff any ballot for any credential. Note, however, that the restriction $\Psi^{\mathsf{VS/EA}}_{\mathsf{cast}}$ in the specification $\mathcal{P}_{\mathcal{H}}$ restricts the adversary to cast only valid ballots and for eligible credentials, which is justified by the fact that public checks can ensure it. The rule $\mathcal{C}^{\mathsf{VR/VS}}_{\mathsf{reg}}$ represents a corrupt registrar/server that allows the adversary to determine election credentials for the voters. Finally, the rule $\mathcal{C}^{\mathsf{VP}}_{\mathsf{vote}}$ specifies a corrupt voting platform that allows the adversary to control the randomness used to encrypt the chosen vote by the voter.

Table 5.1 represents the adversary models for Helios, in which corrupt parties are denoted by C, whereas honest parties are by H. In addition to corrupt parties represented in Table 5.1, we consider the network corrupt, i.e. the communication is held through a public channel. Moreover, in all cases, some voters are corrupt. The actions of corrupt parties are described with the rules from Figure 5.2. Whenever a party is corrupt, we replace the rule modelling

its honest behaviour in the specification $\mathcal{P}_{\mathcal{H}}$ with the one modelling its corrupt behaviour. Moreover, when the server is corrupt, we remove the restriction $\Psi^{\mathsf{VS}}_{\mathsf{order}}$ from the specification since a corrupt server may not process the ballots in the correct order. Additionally, the event $\mathsf{Reg}(\mathsf{id}, \mathsf{cr})$ is required to be recorded by an honest party. In $\mathcal{A}_3$ and $\mathcal{A}_4$, we assume they are recorded by voters when they receive their election credentials. This is similar to the case of registration by a corrupt registrar/server. Therefore, we keep the event $\mathsf{Reg}(\mathsf{id}, \mathsf{cr})$ in the rule $\mathcal{C}^{\mathsf{VR/VS}}_{\mathsf{reg}}$.

In the specification of $\mathcal{A}$, the corrupt registrar is not considered a separate case from the corrupt server. One reason is that, in general, the Helios server registers voters. The second reason is that if a separate registrar is used, as in [3], this party should also be responsible for authenticating the voters to accept their ballots. Thus, it forwards all received ballots to the server for recording them to BB. In the case of a corrupt registrar, the adversary can cast any ballot without authentication, and the honest server will publish them. Therefore, this case is not different from the corruption case in $\mathcal{A}_3$.

5.3 Verification Results and Analysis

In this section, we provide the verifiability analysis of Helios with respect to the several scenarios obtained from the individual verification procedures and adversary models defined in Section 5.2. For each scenario, we present the verification results of the automated verification with Tamarin. Then, we analyse the verification results for the concerned adversary models. We show that for some scenarios, Helios guarantees end-to-end verifiability. For others, we capture verifiability attacks, including the well-known ballot stuffing and classical clash attacks and the new versions of clash attacks possible with revoting.

For the analysis, we consider 16 scenarios. Each scenario is formally defined as an e-voting specification $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_i, \mathcal{A}_j)$ that assembles the Helios protocol specification $\mathcal{P}_{\mathcal{H}}$, an individual verification procedure $\mathcal{V}_i$ from Figure 5.1, and an adversary model $\mathcal{A}_j$ from Table 5.1. The Tamarin codes corresponding to the specifications of those scenarios are available online [52]. To check verifiability with respect to those scenarios, we use the formulas described in Figure 4.1, where $\mathsf{tr} = \mathsf{cr}$ and $\mathsf{BBres}(\mathsf{cr}, \mathsf{v}) \equiv \mathsf{BBtally}(\mathsf{cr}, \mathsf{b}) \wedge \mathsf{open}(\mathsf{b}) = \mathsf{v}$. We model the functionality of open with the following equation added to $\mathcal{E}_{\mathcal{H}}$:

$$\mathsf{open}(\langle \mathsf{enc}(\mathsf{v}, \mathsf{pk}_{\mathsf{E}}, \mathsf{r}), \mathsf{p} \rangle) = \mathsf{v}.$$

Moreover, we fix the number of candidates at two in order to specify the zero-knowledge proof equations in $\mathcal{E}_{\mathcal{H}}$. As revote policies, we use $\Omega^{\mathsf{last}}(\mathsf{id}, \mathsf{v}, \mathsf{t})$ for the individual verification procedures $\mathcal{V}_1$ and $\mathcal{V}_2$ that represent the verification any time during the election, and $\Omega^{\mathsf{no}}(\mathsf{id}, \mathsf{v}, \mathsf{t})$ for the procedures $\mathcal{V}_3$ and $\mathcal{V}_4$ that represent the verification performed at the end of the election. Thus, we check each formula from Figure 4.1 with Tamarin and show whether Helios guarantees $\mathsf{E2E}[\mathsf{iv}_{\triangledown}, \mathsf{res}_{\diamond}]$ for the specified scenario $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_i, \mathcal{A}_j)$ and $\triangledown, \diamond \in \{\circ, \bullet\}$.

We present the verification results in Table 5.2, where $[\mathcal{V}_i, \mathcal{A}_j]$ represents $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_i, \mathcal{A}_j)$, the symbol ✓ represents the successful verification, i.e. the security proof, whereas ✗ does the failure, i.e. an attack. The Tamarin execution for any scenario, where all formulas are checked together, typically takes less than a minute. Note that the formulas $\Phi^{\bullet}_{\mathsf{cons}}$ and $\Phi^{\circ}_{\mathsf{cons}}$ in Table 5.2 represent the conjunctions of the formulas $\Phi_{\mathsf{reg1}} \wedge \Phi_{\mathsf{reg2}} \wedge \Phi_{\mathsf{one}}$ and $\Phi^{\circ}_{\mathsf{reg2}} \wedge \Phi_{\mathsf{one}}$, respectively, and the formulas $\Phi^{\triangledown}_{\mathsf{iv2}}$ and $\Phi^{\triangledown}_{\mathsf{iv3}}$ represent both cases, where either $\triangledown = \circ$ or $\triangledown = \bullet$. Recall that

$$\mathsf{SWE2E}[\mathsf{iv}_{\circ}, \mathsf{res}_{\circ}] \quad = \quad \Phi^{\circ}_{\mathsf{iv1}} \wedge \Phi^{\circ}_{\mathsf{iv2}} \wedge \Phi^{\circ}_{\mathsf{iv3}} \wedge \Phi_{\mathsf{eli}} \wedge \Phi^{\circ}_{\mathsf{res}} \wedge \Phi^{\circ}_{\mathsf{cons}}.$$

$[\mathcal{V}_i, \mathcal{A}_j]/\Phi_{\text{type}}$	$\Phi^{\bullet}_{\text{iv1}}$	$\Phi^{\circ}_{\text{iv1}}$	$\Phi^{\triangledown}_{\text{iv2}}$	$\Phi^{\triangledown}_{\text{iv3}}$	Φ_{eli}	$\Phi^{\bullet}_{\text{res}}$	$\Phi^{\circ}_{\text{res}}$	$\Phi^{\bullet}_{\text{cons}}$	$\Phi^{\circ}_{\text{cons}}$
$[\mathcal{V}_i, \mathcal{A}_1],\ i \in \{1,2\}$	✗	✓	✓	✓	✓	✓	✓	✓	✓
$[\mathcal{V}_i, \mathcal{A}_1],\ i \in \{3,4\}$	✓	✓	✓	✓	✓	✓	✓	✓	✓
$[\mathcal{V}_i, \mathcal{A}_2],\ i \in \{1,2\}$	✗	✗	✓	✓	✓	✗	✗	✓	✓
$[\mathcal{V}_i, \mathcal{A}_2],\ i \in \{3,4\}$	✓	✓	✓	✓	✓	✗	✓	✓	✓
$[\mathcal{V}_1, \mathcal{A}_3]$	✗	✗	✗	✓	✓	✗	✗	✗	✓
$[\mathcal{V}_2, \mathcal{A}_3]$	✗	✗	✓	✓	✓	✗	✗	✗	✓
$[\mathcal{V}_3, \mathcal{A}_3]$	✓	✓	✓	✓	✓	✗	✓	✗	✓
$[\mathcal{V}_4, \mathcal{A}_3]$	✓	✓	✗	✓	✓	✗	✓	✗	✓
$[\mathcal{V}_i, \mathcal{A}_4],\ i \in \{1,2\}$	✗	✗	✗	✓	✓	✗	✗	✗	✓
$[\mathcal{V}_i, \mathcal{A}_4],\ i \in \{3,4\}$	✓	✓	✗	✓	✓	✗	✓	✗	✓

TABLE 5.2: Verification results for Helios.

Adversary $\mathcal{A}_1$**.** For the procedure $\mathcal{V}_1$, i.e. the verification of the last ballot cast at any time during the election, we have $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_1, \mathcal{A}_i) \nvDash \Phi^{\bullet}_{\text{iv}}$ for any i, representing the violation of individual verifiability of corrupt voters. In these cases, $\mathcal{A}_i$ casts a ballot using the corrupt voter's credentials after the voter has verified their vote. On the other hand, we have $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_1, \mathcal{A}_1) \vDash \Phi^{\circ}_{\text{iv}}$ for honest voters. Indeed, all other formulas hold for $\mathcal{A}_1$, i.e. $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_1, \mathcal{A}_1) \vDash \mathsf{SE2E}[\mathsf{iv}_{\circ}, \mathsf{res}_{\diamond}]$, for $\diamond \in \{\circ, \bullet\}$, amounting to strong end-to-end verifiability guarantees, i.e. $\mathsf{E2E}[\mathsf{iv}_{\circ}, \mathsf{res}_{\diamond}]$, for honest voters. We also have $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_i, \mathcal{A}_1) \vDash \mathsf{SE2E}[\mathsf{iv}_{\bullet}, \mathsf{res}_{\diamond}]$ for $i \in \{3,4\}$ and $\diamond \in \{\circ, \bullet\}$, guaranteeing strong end-to-end verifiability even for corrupt voters when they perform individual verification at the end of the election. Note that we assume a different BB displaying all cast ballots (not only the last ballot cast for each credential) and require additional voter checks with $\mathcal{V}_2$, i.e. the voters also verify the older ballots cast by them. Intuitively, this deployment of BB with $\mathcal{V}_2$ helps to prevent clash attacks exploited with revoting, as we discuss in $\mathcal{A}_3$. However, it does not help to prevent ballot stuffing since the adversary casts a ballot for the voter's credential after they performed the individual verification. Thus, we obtain the same results for the scenarios $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_1, \mathcal{A}_1)$ and $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_2, \mathcal{A}_1)$.

Adversary $\mathcal{A}_2$**.** We have $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_i, \mathcal{A}_2) \nvDash \Phi^{\bullet}_{\text{res}}$ for any i, representing the well-known ballot stuffing attacks in Helios when the server is corrupt. On the positive side, $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_i, \mathcal{A}_2) \vDash \mathsf{SE2E}[\mathsf{iv}_{\triangledown}, \mathsf{res}_{\circ}]$, for $i \in \{3,4\}$ and $\triangledown \in \{\circ, \bullet\}$, meaning that the server stuffs ballots only for voters who did not perform individual verification at the end of the election. Thus, Helios guarantees weak end-to-end verifiability, i.e. $\mathsf{E2E}[\mathsf{iv}_{\triangledown}, \mathsf{res}_{\circ}]$ for $\triangledown \in \{\circ, \bullet\}$, with the procedures $\mathcal{V}_3$ and $\mathcal{V}_4$.

Adversary $\mathcal{A}_3$**.** We have $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_3, \mathcal{A}_3) \vDash \mathsf{SWE2E}[\mathsf{iv}_{\circ}, \mathsf{res}_{\circ}]$, representing the first automated proof of end-to-end verifiability for Helios when the server is fully corrupt, i.e. the corruption of the server also includes the registration of the voters. Note that $\mathcal{V}_3$ is the procedure performed after the voting phase, whereas $\mathcal{V}_1$ represents the procedure currently deployed and used in [34]. For $\mathcal{V}_1$, in addition to the ballot stuffing attacks (as for $\mathcal{A}_2$), $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_1, \mathcal{A}_3) \nvDash \Phi^{\triangledown}_{\text{iv2}}$, for $\triangledown \in \{\circ, \bullet\}$, corresponding to a new version of the clash attacks [40], which is originally exploited with $\mathcal{A}_4$. The attack scenario is presented in $\mathsf{AS}^1_{\mathcal{H}}$. It can be seen that, if the registrar/server assigns the same credential, i.e. cr_1, to two voters, they can both be happy with their verification results, by verifying BBcast at different points in time. Election auditors cannot detect any irregularity on BB, since the public sequence of ballots is consistent with

revoting performed by cr_1. The server can then stuff a ballot for cr_2, in order to create the impression that ballots from all voters have been taken into account.

Attack Scenario $\mathsf{AS}^1_{\mathcal{H}}$ ***(clash attack by*** $\mathcal{A}_3$***)***

1. VR/VS creates $\mathsf{cr}_1, \mathsf{cr}_2$, and registers both $\mathsf{V}(\mathsf{id}_1)$ and $\mathsf{V}(\mathsf{id}_2)$ with cr_1, resulting in $\mathsf{Reg}(\mathsf{id}_1, \mathsf{cr}_1)$, $\mathsf{Reg}(\mathsf{id}_2, \mathsf{cr}_1)$.
2. $\mathsf{V}(\mathsf{id}_1)$ casts a ballot b_1 and verifies $(\mathsf{cr}_1, \mathsf{b}_1) \in \mathsf{BBcast}$.
3. $\mathsf{V}(\mathsf{id}_2)$ casts a ballot b_2 and verifies $(\mathsf{cr}_1, \mathsf{b}_2) \in \mathsf{BBcast}$.
4. EA sees $(\mathsf{cr}_1, \mathsf{b}_1)$ followed by $(\mathsf{cr}_1, \mathsf{b}_2)$ on BBcast.
Outcome: only one ballot is tallied for id_1 and id_2.

The attack $\mathsf{AS}^1_{\mathcal{H}}$ is particularly effective when BBcast shows only the last ballot cast for each credential, as in the current deployment. However, the other deployment options for BBcast may prevent similar attacks. For example, assume all ballots cast by each voter are displayed on the bulletin board, and voters are instructed to ensure all previous ballots cast for their credential are theirs when they verify their last ballot cast, as in procedure $\mathcal{V}_2$. Then, adapting the above scenario, after id_2 casts b_2, $(\mathsf{cr}_1, \mathsf{b}_1)$, $(\mathsf{cr}_1, \mathsf{b}_2) \in \mathsf{BBcast}$. The voter id_2 has a chance to spot a problem related to cr_1. Therefore, the attack $\mathsf{AS}^1_{\mathcal{H}}$ can be prevented by $\mathcal{V}_2$.

An interesting version of this attack is when id_1 and id_2 choose to abstain. In that case, they can still verify the bulletin board to ensure no ballot is cast on their behalf. The adversary that corrupts the registrar/server to register $\mathsf{id}_1, \mathsf{id}_2$ with the same cr_1 can, again, cast a ballot for cr_2. This entails $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_4, \mathcal{A}_3) \not\models \Phi^{\nabla}_{\mathsf{iv2}}$, for $\nabla \in \{\circ, \bullet\}$. Remarkably, this attack is more difficult to prevent than before. The verification results show that procedures that protect participating voters do not protect the abstainers. The attack $\mathsf{AS}^1_{\mathcal{H}}$ can be prevented by $\mathcal{V}_3$, i.e. verifying ballots directly on BBtally, or by $\mathcal{V}_2$, i.e. verifying the last ballot and ensuring all ballots on BBcast are recognised as theirs by voters. However, it is not possible to prevent clash attacks on abstaining voters; (i) there is no ballot on BBcast, (ii) verification on BBtally also does not help, since two abstaining voters would both be happy seeing no ballot.

Attack Scenario $\mathsf{AS}^2_{\mathcal{H}}$ ***(clash attack by*** $\mathcal{A}_4$***)***

1. VR/VS creates $\mathsf{cr}_1, \mathsf{cr}_2$, and registers both $\mathsf{V}(\mathsf{id}_1)$ and $\mathsf{V}(\mathsf{id}_2)$ with cr_1, resulting in $\mathsf{Reg}(\mathsf{id}_1, \mathsf{cr}_1)$, $\mathsf{Reg}(\mathsf{id}_2, \mathsf{cr}_1)$.
2. VP prepares the same ballot b for $\mathsf{V}(\mathsf{id}_1)$ and $\mathsf{V}(\mathsf{id}_2)$.
3. $\mathsf{V}(\mathsf{id}_1)$ casts the ballot b and verifies $(\mathsf{cr}_1, \mathsf{b}) \in \mathsf{BBtally}$.
4. $\mathsf{V}(\mathsf{id}_2)$ casts the ballot b and verifies $(\mathsf{cr}_1, \mathsf{b}) \in \mathsf{BBtally}$.
Outcome: only one ballot is tallied for id_1 and id_2.

Adversary $\mathcal{A}_4$**.** We capture the classical clash attacks [40], as presented in $\mathsf{AS}^2_{\mathcal{H}}$. There are some notable differences from the scenario of $\mathsf{AS}^1_{\mathcal{H}}$: voters id_1 and id_2 are assumed to vote for the same candidate; voting platforms of both voters is assumed corrupt; the attack is possible even when revoting is not allowed and voters perform the stronger verification procedure $\mathcal{V}_3$. Here, the adversary also needs to create a clash on ballots, and not only on public credentials, since there is at most one ballot on BBtally for each credential. That is where it relies on the voting platform.

Helios with identities. The verification results for the specification of Helios with identities are similar to the one given in Table 5.2, except for the clash attacks caught with the property $\Phi^{\nabla}_{\mathsf{iv2}}$ in $\mathcal{A}_3$ and $\mathcal{A}_4$. The property $\Phi^{\nabla}_{\mathsf{iv2}}$ is trivially satisfied in all scenarios since $\mathsf{cr} = \mathsf{id}$.

Conclusion. Helios is vulnerable to *ballot stuffing attacks* by a corrupt server and *clash attacks* by a corrupt server, registrar, and voting platform. In our analysis, we have captured those attacks with adversary models $\mathcal{A}_2$ and $\mathcal{A}_4$, respectively. In addition, we have discovered new versions of clash attacks exploited with $\mathcal{A}_3$ when revoting is allowed. In the new version, the adversary does not have to corrupt voting platforms; thus, the clash is not required on ballots.

On the other hand, we have experimented the effects of the different individual verification procedures deployed by the protocol. When individual verification is allowed any time during the election with $\mathcal{V}_1$, as in the deployment of Helios, the voters are subjected to ballot stuffing attacks even if they have verified their last ballot cast on BB. However, if the verification is allowed at the end of the election with $\mathcal{V}_3$, Helios provides end-to-end verifiability guarantees even for corrupt voters and even if the server is fully corrupt. Thus, we have obtained the first automated proof of end-to-end verifiability for Helios when the server is fully corrupt. Moreover, we have assumed a different BB displaying all cast ballots by voters, not just the last cast one, and thus a different individual verification procedure $\mathcal{V}_2$, where the voters verify their last cast ballot and also the older ballots cast by them at any time during the election. We have observed that $\mathcal{V}_2$ protects voters against new versions of clash attacks; however, it does not protect them against ballot stuffing.

The findings of our analysis motivate us to perform a similar analysis on Belenios, an e-voting protocol built upon Helios. Belenios introduces a separate registrar to generate election credentials for voters and splits the trust between the registrar and the server. Thus, it prevents ballot stuffing and provides stronger end-to-end verifiability. Our goal is to explore the verifiability of the protocol, especially in the case of revoting.

SETUP PHASE

R^{T}_{key} : **generate election key pair**
$[\ \mathsf{Fr}(\mathsf{sk}_E)\ ] \multimap [\ \mathsf{BBkey}(\mathsf{pk}(\mathsf{sk}_E))\ ] \mapsto [\ \mathsf{SkE}(\mathsf{sk}_E), \mathsf{BBkey}(\mathsf{pk}(\mathsf{sk}_E)), \mathsf{Out}(\mathsf{pk}(\mathsf{sk}_E))\ ]$

R^{A}_{cand} : **determine candidates to be elected**
$\mathsf{let}\ \ \mathsf{vlist} = \langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle\ \ \mathsf{in}$
$[\ \mathsf{In}(\mathsf{vlist})\ ] \multimap [\ \mathsf{BBcand}(\mathsf{v}_1), \ldots, \mathsf{BBcand}(\mathsf{v}_k), \mathsf{Vlist}(\mathsf{vlist})\ ] \mapsto$
$[\ \mathsf{BBcand}(\mathsf{v}_1), \ldots, \mathsf{BBcand}(\mathsf{v}_k), \mathsf{Vlist}(\mathsf{vlist})\ ]$

R^{A}_{id} : **determine identities eligible to vote**
$[\ \mathsf{In}(\mathsf{id})\ ] \multimap [\ ] \mapsto [\ \mathsf{Id}(\mathsf{id})\ ]$

$R^{VR/VS}_{reg}$: **register voter with a public credential**
$[\ \mathsf{Id}(\mathsf{id}), \mathsf{Fr}(\mathsf{cr})\ ] \multimap [\ \mathsf{Reg}(\mathsf{id}, \mathsf{cr}), \mathsf{BBreg}(\mathsf{cr})\ ] \mapsto [\ \mathsf{Voter}(\mathsf{id}, \mathsf{cr}), \mathsf{BBreg}(\mathsf{cr}), \mathsf{Out}(\mathsf{cr})\ ]$

R^{VS}_{pwd} : **generate password for voter authentication**
$[\ \mathsf{Id}(\mathsf{id}), \mathsf{Fr}(\mathsf{pwd})\ ] \multimap [\ ] \mapsto [\ \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd})\ ]$

R^{VS}_{bb} : **setup initial** BBcast **for registered voters**
$[\ \mathsf{BBreg}(\mathsf{cr})\ ] \multimap [\ \mathsf{BBcast}(\mathsf{cr}, \bot)\ ] \mapsto [\ \mathsf{BBcast}(\mathsf{cr}, \bot)\ ]$

VOTING PHASE

R^{VP}_{vote} : **construct a ballot, authenticate and send it to** VS
$\mathsf{let}\ \ \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk}_E, \mathsf{r});\ \ \mathsf{p} = \mathsf{pr}_R(\mathsf{c}, \mathsf{r}, \mathsf{vlist});\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{p} \rangle;\ \ \mathsf{a} = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{b} \rangle)\ \ \mathsf{in}$
$[\ \mathsf{Voter}(\mathsf{id}, \mathsf{cr}), \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd}), \mathsf{BBcand}(\mathsf{v}), \mathsf{Vlist}(\mathsf{vlist}), \mathsf{BBkey}(\mathsf{pk}_E), \mathsf{Fr}(\mathsf{r}), \mathsf{Fr}(\mathsf{t})\ ]$
$\multimap [\ \mathsf{Vote}(\mathsf{id}, \mathsf{v}), \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b}), \mathsf{VoteTime}(\mathsf{id}, \mathsf{v}, \mathsf{t})\ ] \mapsto [\ \mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{b}, \mathsf{t}), \mathsf{Out}(\langle \mathsf{id}, \mathsf{b}, \mathsf{a} \rangle)\ ]$

R^{VS}_{cast} : **authenticate voter, verify and publish ballot**
$\mathsf{let}\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{p} \rangle;\ \ \mathsf{a}' = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{b} \rangle)\ \ \mathsf{in}$
$[\ \mathsf{In}(\langle \mathsf{id}, \mathsf{b}, \mathsf{a} \rangle), \mathsf{Voter}(\mathsf{id}, \mathsf{cr}), \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd}), \mathsf{BBkey}(\mathsf{pk}_E), \mathsf{Vlist}(\mathsf{vlist})\ ]$
$\multimap [\ \mathsf{a}' \equiv \mathsf{a}, \mathsf{ver}_R(\mathsf{p}, \mathsf{c}, \mathsf{pk}_E, \mathsf{vlist}) \equiv \mathsf{true}, \mathsf{VScast}(\mathsf{id}, \mathsf{b}), \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ] \mapsto [\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]$

Ψ^{VS}_{order} : **ensure correct order of ballots**
$\mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ @i\ \wedge\ \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b}')\ @j\ \wedge$
$\mathsf{VScast}(\mathsf{id}, \mathsf{b})\ @k\ \wedge\ \mathsf{VScast}(\mathsf{id}, \mathsf{b}')\ @l\ \wedge\ i \prec j \Rightarrow k \prec l$

$\Psi^{VS/EA}_{cast}$: **ensure ballot validity; can be audited by** EA
$\mathsf{BBcast}(\mathsf{cr}, \mathsf{b}) \Rightarrow \mathsf{BBreg}(\mathsf{cr})\ \wedge\ (\ \mathsf{b} \neq \bot \Rightarrow$
$\mathsf{BBkey}(\mathsf{pk}_E)\ \wedge\ \mathsf{Vlist}(\mathsf{vlist})\ \wedge\ \mathsf{b} = \langle \mathsf{c}, \mathsf{p} \rangle\ \wedge\ \mathsf{ver}_R(\mathsf{p}, \mathsf{c}, \mathsf{pk}_E, \mathsf{vlist}) = \mathsf{true}\ )$

TALLY PHASE

R^{VS}_{tally} : **select ballot to be tallied for a public credential**
$[\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ] \multimap [\ \mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ ] \mapsto [\ \mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ ]$

$\Psi^{VS/EA}_{tally}$: **ensure the last ballot cast is tallied; can be audited by** EA
$\mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ @i\ \wedge\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b}')\ @j\ \wedge\ \mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ @k \Rightarrow j \prec i\ \vee\ \mathsf{b} = \mathsf{b}'$

FIGURE 5.3: Tamarin specification of Helios.

Chapter 6

Belenios and Its Variants

Belenios [18, 8] has built upon Helios to provide stronger end-to-end verifiability with weaker trust assumptions, as described first in [20]. It introduces a separate registrar responsible for generating signing key pairs for the voters but keeps the server responsible for voter authentication. Therefore, voters receive a key pair from the registrar and a password from the server, sign their ballots before casting and use the password for authentication to the server. In this way, Belenios splits the trust between the registrar and the server so that none of them knows both credentials to cast a ballot. Thus, it prevents ballot stuffing by a corrupt server.

The first version of Belenios [20] was described in a computational model and proved to be secure with respect to end-to-end verifiability against a corrupt registrar as well as a corrupt server, provided that they are not simultaneously corrupt. This version has introduced the signature to the ballot structure, keeping the zero-knowledge proof for the vote to show that it is within a valid range. However, in this way, it has inherited the vulnerability in Helios that allows ballot copying attacks [19] against the privacy of the protocol. To prevent those attacks, the zero-knowledge proof in Belenios has been improved [18] to include the public key of the voter so that the ciphertext of the vote is linked to the voter who computes and sings it. Thus, the adversary cannot copy the ciphertext and cast it on behalf of another voter since it cannot generate the related zero-knowledge proof.

The computational model [20] of Belenios has been proved manually, focusing only on verifiability. However, it did not account for revoting, which is a feature affecting verifiability. Another machine-checked security analysis has been performed for the computational model of Belenios in [21] with respect to both privacy and verifiability. The model considers the latest version of Belenios [18] with an improved version of zero-knowledge proof, accounts for revoting and updates the trust assumptions in [20] concerning privacy, i.e. Belenios provides privacy only when the registrar is honest, as shown in [21]. The analysis has been assisted by the tool EasyCrypt, which helped to improve the existing definitions of privacy and verifiability since it allows to catch the details that cannot be seen easily with pen and paper. On the other hand, the symbolic model of Belenios has been analysed in [4] with respect to privacy. However, there is no symbolic proof of verifiability for the general version of Belenios. The symbolic analysis has been performed in [17] for a variant of Belenios, i.e. BeleniosVS, which has several limitations as discussed in Chapter 4.

In this chapter, first, we perform a symbolic analysis for the verifiability of Belenios using the automated verification tool Tamarin and the framework introduced in Chapter 4. Our analysis accounts for revoting and shows that Belenios is not secure even when both the registrar and the server are honest. The adversary can mount an attack against honest voters, only corrupting the communication network when revoting is allowed. In the case of a corrupt registrar, the adversary exploits new versions of clash attacks. Thus, Belenios is not secure against a corrupt registrar or a corrupt server as it is supposed to be.

Second, we propose fixes for the found attacks and improve the verifiability of Belenios without affecting the usability of the protocol. Our solutions require changes in the ballot structure, which will be implemented by the voting platforms and the voting server. We call

Belenios+ the new variant of Belenios, and we prove that Belenios+ is secure with respect to a corrupt registrar or a corrupt server, performing an automated verification with Tamarin.

Last, we perform a verifiability analysis for BeleniosRF [14], which is a variant of Belenios aiming to provide receipt-freeness in addition to end-to-end verifiability. Receipt-freeness and verifiability are often mentioned in the literature as two conflicting properties of e-voting protocols. With this analysis, we aim to evaluate the trade-off between those properties, comparing our verification results with the one for Belenios. In our analysis, we find new attacks against the verifiability of BeleniosRF due to the individual verification procedure being weakened by the receipt-freeness. In BeleniosRF, voters perform individual verification by checking whether there is a ballot next to their credentials on BB since the voting server randomises their ballots, which leads to ballot stuffing and clash attacks. Thus, we conclude that BeleniosRF provides weaker verifiability guarantees than Belenios while providing stronger privacy guarantees with receipt-freeness.

Structure of the chapter. This chapter includes five sections, where the first three are dedicated to Belenios, the fourth is to new variants of Belenios, and the last is to BeleniosRF. In Section 6.1, we give the protocol structural details of Belenios. In Section 6.2, we present its Tamarin specification, and then, in Section 6.3, we provide our verification results with respect to several individual verification procedures and corruption scenarios. In Section 6.4, we propose fixes for the attacks described in Section 6.3 as new variants of Belenios and prove the variant Belenios+ is secure against all attacks. Finally, we present our verifiability analysis for BeleniosRF in Section 6.5.

6.1 Belenios Protocol Structure

Belenios has the following parties described as follows:

- Administrator A is responsible for the election configuration, i.e. determines the candidates and voters eligible for the election and talliers to generate the election key pair.
- Talliers T generate an election key pair: a private key and its corresponding public key, and decrypt the homomorphic tally of the ballots at the end of the election.
- Registrar VR generates a signing key pair for each voter, sends each key pair privately to the corresponding voter, and registers the public keys of the voters as eligible public credentials for the election, i.e. publishes them on the bulletin board.
- Voting server VS generates a login credential, i.e. a password, for each voter identity, accepts ballots from voters if authenticated with the login credential, performs required checks for the validity of the ballots, and then publishes them on the bulletin board. It also manages a log file, denoted by Log, to ensure the consistency of the voter identity-credential pairs, i.e. it does not allow a voter identity to be matched with several eligible credentials and a credential with several identities.
- Voting platform VP allows voters to generate a ballot as a quadruple of a ciphertext obtained encrypting their vote with the election public key, the signature for the ciphertext, and two zero-knowledge proofs; one showing that the vote is within a valid range and another showing that the ciphertext is computed by the voter who signs it. VP, then, allows voters to cast their ballot on the voting server via a login operation.
- Voters V registered to the election as eligible may cast several ballots using their voting platform during the election or abstain from voting.
- Election auditors EA audit the public information recorded on the bulletin board.

Belenios relies on an append-only public bulletin board, denoted by BB, to display the public election data, e.g. the election's public key is displayed on BBkey. In the following, we present the election procedures and the individual verification procedure of Belenios:

Setup phase. A determines the list of candidates $\mathsf{v}_1, \ldots, \mathsf{v}_k$ and voters $\mathsf{id}_1, \ldots, \mathsf{id}_n$ that are eligible for the election, deploys T to generate the election key pair $(\mathsf{sk}_E, \mathsf{pk}_E)$, VR to generate a signing key pair $(\mathsf{sk}_{id}, \mathsf{pk}_{id})$ for each voter id and publish the public keys of the voters as public credentials $\mathsf{cr}_1, \ldots, \mathsf{cr}_n$, where each $\mathsf{cr}_i = \mathsf{pk}_{id_i}$, and VS to generate a login credential pwd for each voter id. The public information is published on the portions of BB as follows:

$$\mathsf{BBkey} : \mathsf{pk}_E; \quad \mathsf{BBcand} : \mathsf{v}_1, \ldots, \mathsf{v}_k; \quad \mathsf{BBreg} : \mathsf{cr}_1, \ldots, \mathsf{cr}_n$$

In this phase, each voter id obtains a signing key pair $(\mathsf{sk}_{id}, \mathsf{cr})$ from VR and a password pwd from VS.

Voting phase. V interacts with VP to construct a ballot:

VP : downloads $\mathsf{pk}_E \in \mathsf{BBkey}$ and $\mathsf{v}_1, \ldots, \mathsf{v}_k \in \mathsf{BBcand}$,
V : selects $\mathsf{v} \in \mathsf{BBcand}$,
VP : encrypts v with pk_E and a randomness $\mathsf{r} : \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk}_E, \mathsf{r})$,
signs c with $\mathsf{sk}_{id} : \mathsf{s} = \mathsf{sign}(\mathsf{c}, \mathsf{sk}_{id})$,
produces a proof of range : $\mathsf{p}_R = \mathsf{pr}_R(\mathsf{c}, \mathsf{r}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle)$,
produces a proof of label : $\mathsf{p}_L = \mathsf{pr}_L(\mathsf{c}, \mathsf{r}, \mathsf{cr})$.

In the above computations, sk_{id} is provided by the voter. The proof of range p_R denotes the zero-knowledge proof showing that the vote v is valid within the range $\langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle$. The proof of label p_L represents the zero-knowledge proof showing that the ciphertext is computed by the voter who signs it, i.e. it labels the ciphertext with the public key of the voter who knows the encryption randomness. Thus, VP constructs the ballot as $\mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p}_R, \mathsf{p}_L \rangle$. If V decides to cast b, VP requests login credentials of V, i.e. id and pwd, which prompts a connection to VS. If VS authenticates V, it receives the tuple $\langle \mathsf{cr}, \mathsf{b} \rangle$ on behalf of id and then performs required validity checks on $\mathsf{id}, \mathsf{cr}, \mathsf{b}$, i.e. VS ensures that there is no id' or cr' such that $(\mathsf{id}, \mathsf{cr}') \in \mathsf{Log}$ or $(\mathsf{id}', \mathsf{cr}) \in \mathsf{Log}$ for $\mathsf{id} \neq \mathsf{id}'$ and $\mathsf{cr} \neq \mathsf{cr}'$, then validates $\mathsf{cr} \in \mathsf{BBreg}$, and verifies the signature s and proofs p_R and p_L. Then, it records $(\mathsf{id}, \mathsf{cr})$ in Log and publishes $(\mathsf{cr}, \mathsf{b})$ on BBcast.

Tally phase. At the end of the voting phase, VS selects the *last* ballot recorded on BBcast for each credential and publishes it on $\mathsf{BBtally}$. The final version of $\mathsf{BBtally}$ is:

$$\mathsf{BBtally} : (\mathsf{cr}_1, \mathsf{b}_1), \ldots, (\mathsf{cr}_n, \mathsf{b}_n),$$

where $\mathsf{b} = \perp$ if no ballot was cast for cr. Then, the ciphertexts corresponding to non-empty ballots on $\mathsf{BBtally}$ are combined homomorphically into the ciphertext c encoding the total number of votes for each candidate, which is decrypted by T to obtain the result of the election. T also produces a zero-knowledge proof of correct decryption.

Individual verification. In Belenios, V can verify that VS correctly captures and records their ballot on BB, which is allowed anytime during the election. For the verification, V has to match their ballot with the one present on BB next to their credential. Belenios allows revoting; thus, each time a ballot is received for a credential, the old ballot is replaced with the new one on BB. Thus, V cannot verify an older ballot on BB. If V verifies their last ballot cast on $\mathsf{BBtally}$, then V can ensure that their ballot will be tallied and their vote will be counted on their behalf due to the verifiable tally procedure. The public data on BB allows anyone to verify the produced zero-knowledge proof to ensure the result corresponds to the list of tallied ballots.

Additions with respect to Helios. Belenios extends the ballot structure to include a signature s for the ciphertext and an additional zero-knowledge proof $\mathsf{p_L}$ to ensure that the ciphertext is computed by the voter who signs it. These two components in the ballot structure prevent ballot stuffing attacks and ballot copying attacks [19] in Helios. In addition, Belenios requires VS to ensure consistency of the identity-credential pairs recorded in Log when it receives a ballot from a voter identity for a public credential. This helps to prevent possible attacks related to the corrupt registration of voters with public credentials. These additions are detailed in the following:

- Belenios uses digital signatures for the authenticity of the ballots, i.e. the signature inside the ballot attests that the voter who owns the signing key has generated the ballot. Thus, it ensures that if a ballot is present on the BB next to a public credential, it should have been cast by a voter who owns that credential. Since all the signing key pairs are generated by the registrar and the public keys, i.e. public credentials, are considered eligible, the corrupt server cannot stuff ballots for non-eligible credentials. Furthermore, the corrupt server cannot obtain the signing key of an honest voter since the registrar privately shares it with the voter. In this way, ballot stuffing is prevented for honest voters or non-eligible credentials.

- Helios is vulnerable to ballot copying attacks [19]. In these attacks, the adversary copies a voter's ballot and casts it on behalf of a corrupt voter. This may lead the adversary to learn the vote of an honest voter. Assume there are three voters, one of which is corrupt. Then, the adversary can easily learn the copied vote since it will correspond to the majority of the votes in the outcome. This attack results from the vulnerability of the ballot structure. In Helios, the ballot structure consists of the encryption of the vote and the proof of range, i.e. it does not include any particular information from the voter. Therefore, the copy of the ballot will be valid to be cast by another voter. To prevent such attacks, Belenios brings another zero-knowledge proof, i.e. proof of label, to the ballot structure, which links the encryption, i.e. the ballot, to a public credential. In this way, even if the adversary copies the ballot, it will not be accepted since the public credential of the voter who cast it differs from the one inside the proof. Even if the adversary tries to copy only the ciphertext, it cannot generate a valid proof of label with another credential since it does not know the encryption randomness.

- Belenios requires the following consistency checks in Log by the server:

$$(\mathsf{id}, \mathsf{cr}) \in \mathsf{Log} \ \wedge \ (\mathsf{id}, \mathsf{cr}') \in \mathsf{Log} \ \Rightarrow \ \mathsf{cr} = \mathsf{cr}' \quad \text{and}$$
$$(\mathsf{id}, \mathsf{cr}) \in \mathsf{Log} \ \wedge \ (\mathsf{id}', \mathsf{cr}) \in \mathsf{Log} \ \Rightarrow \ \mathsf{id} = \mathsf{id}'.$$

  With these checks, the server does not allow a corrupt registrar to cheat on registering voter identities with public credentials. For example, a corrupt registrar may attempt to give the same credential to several voters and thus many credentials to a corrupt voter. Due to the consistency checks, the corrupt voter would be able to cast ballots only for a single credential, making other credentials wasted. In addition, the voters who share the same credentials could complain since their ballots are rejected by the server, which may result in the registrar being caught. Therefore, the registrar would refrain from attempting to cheat due to the consistency checks.

6.2 Tamarin Specification of Belenios

In this section, we present the Tamarin specification $\mathcal{S}_B$ of the Belenios protocol. To distinguish different individual verification procedures and adversary models, we separate the

specification $\mathcal{S}_{\mathcal{B}}$ into three components as we have done for Helios in Chapter 5, i.e. $\mathcal{S}_{\mathcal{B}} = (\mathcal{P}_{\mathcal{B}}, \mathcal{V}, \mathcal{A})$, where

- $\mathcal{P}_{\mathcal{B}}$ models the actions of the honest parties,
- $\mathcal{V}$ models individual verification procedures,
- $\mathcal{A}$ models adversarial capabilities.

Each component above has its own rules and restrictions that are intended for the execution of its rules.

The protocol specification $\mathcal{P}_{\mathcal{B}}$. We start to specify $\mathcal{P}_{\mathcal{B}}$ by defining the equation theory $\mathcal{E}_{\mathcal{B}}$ that models the cryptographic primitives used in Belenios. Belenios uses the ElGamal encryption algorithm to encrypt the votes, the Schnorr signature scheme to sign the encrypted votes, a non-interactive zero-knowledge protocol to prove that the encrypted votes are in a valid range of eligible candidates, and the ciphertexts are computed by the eligible credentials who sign them. Thus, the specification requires the following equations defined for $\mathcal{E}_{\mathcal{B}}$:

(1) $\mathsf{dec}(\mathsf{enc}(x, \mathsf{pk}(y), z), y) = x$,
(2) $\mathsf{verify}(\mathsf{sign}(x, y), x, \mathsf{pk}(y)) = \mathsf{true}$,
(3) $(\forall i)\ \ \mathsf{ver}_R(\mathsf{pr}_R(\mathsf{enc}(x_i, y, z), z, \langle x_1, \ldots, x_k\rangle), \mathsf{enc}(x_i, y, z), y, \langle x_1, \ldots, x_k\rangle) = \mathsf{true}$,
(4) $\mathsf{ver}_L(\mathsf{pr}_L(\mathsf{enc}(x, y, z), z, \ell), \mathsf{enc}(x, y, z), \ell) = \mathsf{true}$,

where ℓ in (4) represents a label that will be equal to the public credential of the voter. Note that equations (1) and (3) are the ones defined for $\mathcal{E}_{\mathcal{H}}$ in the Helios specification $\mathcal{S}_{\mathcal{H}}$.

We have described the protocol parties of Belenios in Section 6.1, as abbreviated by A, T, VR, VS, V, EA. Now, we model the actions of those parties with the rules in $\mathcal{P}_{\mathcal{B}}$. We denote any rule in $\mathcal{P}_{\mathcal{B}}$ by $\mathsf{R}^{\mathsf{a}}_{\mathsf{n}}$, where a is the party's abbreviation, and n is the name describing the action in the rule. For simplicity, we remove the symbol ! from the facts, representing persistent facts, since we consider all protocol facts except the special ones (In, Out, and Fr) persistent. We model Belenios, considering the three phases of an election. In the phases, we present the rules differ from the ones described for Helios in Chapter 5.

Setup phase. The rules $\mathsf{R}^{\mathsf{T}}_{\mathsf{key}}$, $\mathsf{R}^{\mathsf{A}}_{\mathsf{cand}}$, $\mathsf{R}^{\mathsf{A}}_{\mathsf{id}}$, $\mathsf{R}^{\mathsf{VS}}_{\mathsf{pwd}}$, and $\mathsf{R}^{\mathsf{VS}}_{\mathsf{bb}}$ are similarly modelled to obtain the election key pair, the candidates, the voter identities, the passwords and the entries for the public credentials of the registered voter identities on BBcast. The specifications of those rules are presented in the full specification of Belenios in Figure 6.5. Unlike Helios, the rule $\mathsf{R}^{\mathsf{VR}}_{\mathsf{reg}}$ models the generation of the signing key pairs for the voters as follows:

$$\mathsf{R}^{\mathsf{VR}}_{\mathsf{reg}}: \ \mathsf{let}\ \ \mathsf{cr} = \mathsf{pk}(\mathsf{sk}_{\mathsf{id}})\ \ \mathsf{in}$$
$$[\ \mathsf{Id}(\mathsf{id}), \mathsf{Fr}(\mathsf{sk}_{\mathsf{id}})\ ] — [\ \mathsf{Reg}(\mathsf{id}, \mathsf{cr}), \mathsf{BBreg}(\mathsf{cr})\ ] \mapsto [\ \mathsf{Cred}(\mathsf{id}, \mathsf{cr}, \mathsf{sk}_{\mathsf{id}}), \mathsf{BBreg}(\mathsf{cr}), \mathsf{Out}(\mathsf{cr})\ ]$$

In the rule $\mathsf{R}^{\mathsf{VR}}_{\mathsf{reg}}$, VR generates a fresh signing key $\mathsf{sk}_{\mathsf{id}}$, computes a public key cr from the signing key, and records it as an eligible public credential on BBreg. Thus, VR associates id with the public credential cr, as recorded in the action fact Reg. In the case of a corrupt registrar, this association would be unreliable. Therefore, in that case, VS makes the association whenever it receives a ballot from the voter id for a public credential.

Voting phase. The rule $\mathsf{R}^{\mathsf{VP}}_{\mathsf{vote}}$ models the ballot generation and cast by VP, where the ballot structure differs from the one in Helios, i.e. $\mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p}_{\mathsf{R}}, \mathsf{p}_{\mathsf{L}}\rangle$. Moreover, it adds cr next to b in the authentication message a since VS does not know the association of public credentials with the voter identities. Before VS accepts a ballot for a credential cr from a voter identity id, as modelled in the rule $\mathsf{R}^{\mathsf{VS}}_{\mathsf{cast}}$, it ensures log consistency for the pair (id, cr) by recording the fact Log and applying the restriction $\Psi^{\mathsf{VS}}_{\mathsf{log}}$. The action fact Log together with the restriction

$\Psi^{\mathsf{VS}}_{\mathsf{log}}$ restricts the rule $\mathsf{R}^{\mathsf{VS}}_{\mathsf{cast}}$ to be executed only for a valid identity-credential pair, i.e. for the respective $(\mathsf{id}, \mathsf{cr})$, there is no another pair $(\mathsf{id}, \mathsf{cr}')$ recorded before in Log for $\mathsf{cr} \neq \mathsf{cr}'$, nor $(\mathsf{id}', \mathsf{cr})$ for $\mathsf{id} \neq \mathsf{id}'$. We model the actions of VP and VS in the following two rules:

$$\begin{aligned}
\mathsf{R}^{\mathsf{VP}}_{\mathsf{vote}} :\ & \mathsf{let}\ \ \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk}_{\mathsf{E}}, \mathsf{r});\ \ \mathsf{s} = \mathsf{sign}(\mathsf{c}, \mathsf{sk}_{\mathsf{id}});\ \ \mathsf{p}_{\mathsf{R}} = \mathsf{pr}_{\mathsf{R}}(\mathsf{c}, \mathsf{r}, \mathsf{vlist});\ \ \mathsf{p}_{\mathsf{L}} = \mathsf{pr}_{\mathsf{L}}(\mathsf{c}, \mathsf{r}, \mathsf{cr});\\
& \quad \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p}_{\mathsf{R}}, \mathsf{p}_{\mathsf{L}} \rangle;\ \ \mathsf{a} = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{b} \rangle)\ \ \mathsf{in}\\
& [\ \mathsf{Cred}(\mathsf{id}, \mathsf{cr}, \mathsf{sk}_{\mathsf{id}}), \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd}), \mathsf{BBcand}(\mathsf{v}), \mathsf{Vlist}(\mathsf{vlist}), \mathsf{BBkey}(\mathsf{pk}_{\mathsf{E}}), \mathsf{Fr}(\mathsf{r}), \mathsf{Fr}(\mathsf{t})\ ]\\
& -[\ \mathsf{Vote}(\mathsf{id}, \mathsf{v}), \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b}), \mathsf{VoteTime}(\mathsf{id}, \mathsf{v}, \mathsf{t})\ ]\!\!\rightarrow\\
& [\ \mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{b}, \mathsf{t}), \mathsf{Out}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{b}, \mathsf{a} \rangle)\ ]
\end{aligned}$$

$$\begin{aligned}
\mathsf{R}^{\mathsf{VS}}_{\mathsf{cast}} :\ & \mathsf{let}\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p}_{\mathsf{R}}, \mathsf{p}_{\mathsf{L}} \rangle;\ \ \mathsf{a}' = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{b} \rangle)\ \ \mathsf{in}\\
& [\ \mathsf{In}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{b}, \mathsf{a} \rangle), \mathsf{BBreg}(\mathsf{cr}), \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd}), \mathsf{BBkey}(\mathsf{pk}_{\mathsf{E}}), \mathsf{Vlist}(\mathsf{vlist})\ ]\\
& -[\ \mathsf{a}' \equiv \mathsf{a}, \mathsf{verify}(\mathsf{s}, \mathsf{c}, \mathsf{cr}) \equiv \mathsf{true}, \mathsf{ver}_{\mathsf{R}}(\mathsf{p}_{\mathsf{R}}, \mathsf{c}, \mathsf{pk}_{\mathsf{E}}, \mathsf{vlist}) \equiv \mathsf{true}, \mathsf{ver}_{\mathsf{L}}(\mathsf{p}_{\mathsf{L}}, \mathsf{c}, \mathsf{cr}) \equiv \mathsf{true},\\
& \quad \mathsf{Log}(\mathsf{id}, \mathsf{cr}), \mathsf{Reg}(\mathsf{id}, \mathsf{cr}), \mathsf{VScast}(\mathsf{id}, \mathsf{b}), \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]\!\!\rightarrow [\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]
\end{aligned}$$

$$\Psi^{\mathsf{VS}}_{\mathsf{log}} :\ \ \mathsf{Log}(\mathsf{id}, \mathsf{cr}) \Rightarrow \neg(\mathsf{Log}(\mathsf{id}, \mathsf{cr}') \wedge \mathsf{cr} \neq \mathsf{cr}') \wedge \neg(\mathsf{Log}(\mathsf{id}', \mathsf{cr}) \wedge \mathsf{id} \neq \mathsf{id}')$$

Unlike Helios, $\mathsf{R}^{\mathsf{VS}}_{\mathsf{cast}}$ records the action fact Reg. In the case of a corrupt registrar, this will allow us to ensure the reliable association of id with cr. However, it can record it only if the voter id casts a ballot. As in Helios, the fact $\mathsf{VoteTime}$ is required for the revote policy. There are also similar restrictions $\Psi^{\mathsf{VS}}_{\mathsf{order}}$ and $\Psi^{\mathsf{VS/EA}}_{\mathsf{cast}}$ to be used with the rule $\mathsf{R}^{\mathsf{VS}}_{\mathsf{cast}}$. They ensure that VS processes the ballots cast by VP in the order they have been sent, any ballot published on BBcast corresponds to a registered credential, and if the ballot is not $\perp$, the signature and the proofs p_{R} and p_{L} inside the ballot are correctly verified.

Tally phase. The rule $\mathsf{R}^{\mathsf{VS}}_{\mathsf{tally}}$ and the restriction $\Psi^{\mathsf{VS/EA}}_{\mathsf{tally}}$ similarly model the tally phase of Belenios. The full protocol specification $\mathcal{P}_{\mathcal{B}}$ is given in Figure 6.5.

Individual verification procedures $\mathcal{V}$. Belenios has the same individual verification procedure as Helios, i.e. it allows individual verification of the ballot cast anytime on BB until another ballot is cast for the same credential. It corresponds to procedure $\mathcal{V}_1$ in Figure 5.1. In addition to the regular procedure $\mathcal{V}_1$, we also use other procedures given in Figure 5.1 to analyse their effects on verifiability. For the second procedure $\mathcal{V}_2$, we assume a different BB in which all cast ballots are displayed until the end of the election, and voters are instructed to verify all the ballots presented on BB next to their credential when they verify their last ballot cast. $\mathcal{V}_3$ allows to verify the ballot tallied for a credential at the end of the election, and $\mathcal{V}_4$ allows to verify the abstention. The rules that specify them are similar to the ones in Figure 5.1, except $\mathcal{V}_4$, for which we replace the fact $\mathsf{Voter}(\mathsf{id}, \mathsf{cr})$ in the rule $\mathsf{R}^{2}_{\mathsf{ver}}$ with the new fact $\mathsf{Cred}(\mathsf{id}, \mathsf{cr}, \mathsf{sk}_{\mathsf{id}})$.

Adversary models $\mathcal{A}$. For Belenios, Table 6.1 represents the adversary models, in which corrupt parties are denoted by C, whereas honest parties are by H. In addition to corrupt parties represented in Table 6.1, we consider the network corrupt, i.e. the communication is held through a public channel. Moreover, in all cases, some voters are corrupt. The corruption rules for talliers and server, i.e. $\mathcal{C}^{\mathsf{T}}_{\mathsf{key}}$ and $\mathcal{C}^{\mathsf{VS}}_{\mathsf{cast}}$, are similar to Helios, whereas for the corruption of voters and registrar, we use the following rules:

$$\begin{aligned}
\mathcal{C}^{\mathsf{V}}_{\mathsf{corr}} :\ & \textbf{corrupt voter to reveal credentials}\\
& [\ \mathsf{Cred}(\mathsf{id}, \mathsf{cr}, \mathsf{sk}_{\mathsf{id}}), \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd})\ ]\!-\![\ \mathsf{Corr}(\mathsf{id})\ ]\!\rightarrow\![\ \mathsf{Out}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{sk}_{\mathsf{id}}, \mathsf{pwd} \rangle)\ ]
\end{aligned}$$

$$\begin{aligned}
\mathcal{C}^{\mathsf{VR}}_{\mathsf{reg}} :\ & \textbf{corrupt registrar to control credentials}\\
& \mathsf{let}\ \ \mathsf{cr} = \mathsf{pk}(\mathsf{sk}_{\mathsf{id}})\ \ \mathsf{in}\\
& [\ \mathsf{In}(\mathsf{sk}_{\mathsf{id}}), \mathsf{Id}(\mathsf{id})\ ]\!-\![\ \mathsf{BBreg}(\mathsf{cr})\ ]\!\rightarrow\![\ \mathsf{Cred}(\mathsf{id}, \mathsf{cr}, \mathsf{sk}_{\mathsf{id}}), \mathsf{BBreg}(\mathsf{cr})\ ]
\end{aligned}$$

where $C^{\mathsf{V}}_{\mathsf{corr}}$ leaks all the voter credentials, and $C^{\mathsf{VR}}_{\mathsf{reg}}$ allows the adversary to determine the signing key of the voter id. To model corrupt voting platforms, similarly, we replace $\mathsf{Fr}(\mathsf{r})$ with $\mathsf{In}(\mathsf{r})$ in the rule $\mathsf{R}^{\mathsf{VP}}_{\mathsf{vote}}$.

Whenever a party is corrupt, we replace the rule modelling its honest behaviour in the specification $\mathcal{P}_{\mathcal{B}}$ with the one modelling its corrupt behaviour. Moreover, when the server is corrupt, we remove the restriction $\Psi^{\mathsf{VS}}_{\mathsf{order}}$ from the specification since a corrupt server may not process the ballots in the correct order. Additionally, the event $\mathsf{Reg}(\mathsf{id}, \mathsf{cr})$ is required to be recorded by an honest party. In $\mathcal{A}_2$ and $\mathcal{A}_3$, it is recorded by the registrar and the server, respectively. However, in $\mathcal{A}_4$ and $\mathcal{A}_5$, we assume it is recorded by voters when they receive their election credentials. This is similar to the case of registration by a corrupt registrar. Therefore, we add the event $\mathsf{Reg}(\mathsf{id}, \mathsf{cr})$ to the rule $C^{\mathsf{VR}}_{\mathsf{reg}}$ in those cases. Note that the adversary model $\mathcal{A}_4$ goes further than standard corruption scenarios assumed for Belenios, where either server or registrar is honest. We show that we can obtain some end-to-end verifiability guarantees (similar to Helios) also when this is not the case.

Adversary Models	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$	$\mathcal{A}_5$
Talliers	C	C	C	C	C
Server	H	C	H	C	C
Registrar	H	H	C	C	C
Voting Platform	H	H	H	H	C

TABLE 6.1: Adversary models for Belenios.

6.3 Verification Results and Analysis

In this section, we provide the verifiability analysis of Belenios with respect to 20 scenarios that are formally defined as an e-voting specification $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_i, \mathcal{A}_j)$, where $\mathcal{P}_{\mathcal{B}}$ is the Belenios specification, $\mathcal{V}_i$ is an individual verification procedure from Section 6.2, and $\mathcal{A}_j$ is an adversary model from Table 6.1. For each scenario, we present the verification results of the automated verification with Tamarin, where its specification is available online [52]. Then, we analyse the verification results for the concerned adversary models. Our analysis shows that Belenios is secure only when the voters verify their votes at the end of the election. It is not secure if the voters verify their votes anytime during the election. Belenios aims to be secure unless the registrar and the server are simultaneously corrupt. However, we capture attacks if either the registrar or the server is corrupt, even when both are honest. The vulnerabilities that enable the attacks are as follows:

- The voting server can accept ballots from some voter id in a different order, i.e. the adversary controlling the communication network may reorder them.
- Since the voting server does not know the association between voter identities and public credentials, it can accept ballots copied from other voters without the adversary needing to change the credentials.

Thus, the adversary is able to reorder ballots, stuff ballots for the honest voters who did not verify, as well as the ones who verified their last ballot cast. Moreover, new versions of clash attacks can be exploited just with a corrupt registrar as we show in $\mathcal{A}_3$.

We present the verification results in Table 6.2, where $[\mathcal{V}_i, \mathcal{A}_j]$ represents $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_i, \mathcal{A}_j)$, the symbol ✓ represents the successful verification, i.e. the security proof, whereas ✗ does the failure, i.e. an attack. The Tamarin execution for any scenario, where all formulas are

checked together, typically takes less than a minute. Note that the formulas $\Phi^{\bullet}_{\mathsf{cons}}$ and $\Phi^{\circ}_{\mathsf{cons}}$ in Table 6.2 represent the conjunctions of the formulas $\Phi_{\mathsf{reg1}} \wedge \Phi_{\mathsf{reg2}} \wedge \Phi_{\mathsf{one}}$ and $\Phi^{\circ}_{\mathsf{reg2}} \wedge \Phi_{\mathsf{one}}$, respectively, and the formulas $\Phi^{\nabla}_{\mathsf{iv2}}$ and $\Phi^{\nabla}_{\mathsf{iv3}}$ represent both cases, where either $\nabla = \circ$ or $\nabla = \bullet$.

$[\mathcal{V}_{\mathbf{i}}, \mathcal{A}_{\mathbf{j}}]/\mathbf{\Phi}_{\mathsf{type}}$	$\Phi^{\bullet}_{\mathsf{iv1}}$	$\Phi^{\circ}_{\mathsf{iv1}}$	$\Phi^{\nabla}_{\mathsf{iv2}}$	$\Phi^{\nabla}_{\mathsf{iv3}}$	Φ_{eli}	$\Phi^{\bullet}_{\mathsf{res}}$	$\Phi^{\circ}_{\mathsf{res}}$	$\Phi^{\bullet}_{\mathsf{cons}}$	$\Phi^{\circ}_{\mathsf{cons}}$
$[\mathcal{V}_i, \mathcal{A}_1],\ i \in \{1,2\}$	✗	✗	✓	✓	✓	✓	✓	✓	✓
$[\mathcal{V}_i, \mathcal{A}_1],\ i \in \{3,4\}$	✓	✓	✓	✓	✓	✓	✓	✓	✓
$[\mathcal{V}_i, \mathcal{A}_2],\ i \in \{1,2\}$	✗	✗	✓	✓	✓	✓	✓	✓	✓
$[\mathcal{V}_i, \mathcal{A}_2],\ i \in \{3,4\}$	✓	✓	✓	✓	✓	✓	✓	✓	✓
$[\mathcal{V}_1, \mathcal{A}_3]$	✗	✗	✗	✓	✓	✓	✓	✓	✓
$[\mathcal{V}_2, \mathcal{A}_3]$	✗	✗	✓	✓	✓	✓	✓	✓	✓
$[\mathcal{V}_3, \mathcal{A}_3]$	✓	✓	✓	✓	✓	✓	✓	✓	✓
$[\mathcal{V}_4, \mathcal{A}_3]$	✓	✓	✗	✓	✓	✓	✓	✓	✓
$[\mathcal{V}_1, \mathcal{A}_4]$	✗	✗	✗	✓	✓	✗	✗	✗	✓
$[\mathcal{V}_2, \mathcal{A}_4]$	✗	✗	✓	✓	✓	✗	✗	✗	✓
$[\mathcal{V}_3, \mathcal{A}_4]$	✓	✓	✓	✓	✓	✗	✓	✗	✓
$[\mathcal{V}_4, \mathcal{A}_4]$	✓	✓	✗	✓	✓	✗	✓	✗	✓
$[\mathcal{V}_1, \mathcal{A}_5]$	✗	✗	✗	✓	✓	✗	✗	✗	✓
$[\mathcal{V}_2, \mathcal{A}_5]$	✗	✗	✓	✓	✓	✗	✗	✗	✓
$[\mathcal{V}_i, \mathcal{A}_5],\ i \in \{3,4\}$	✓	✓	✗	✓	✓	✗	✓	✗	✓

TABLE 6.2: Verification results for Belenios.

Adversary $\mathcal{A}_1$. We have $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_1, \mathcal{A}_1) \not\models \Phi^{\circ}_{\mathsf{iv1}}$, implying that the individual verifiability does not hold for honest voters with $\mathcal{V}_1$, i.e. when voters verify their last ballot cast during the voting phase. In this case, the adversary controlling the communication network exploits an attack against an honest voter, as described in $\mathsf{AS}^1_{\mathcal{B}}$. Assume the voter id_1 casts two ballots b_1 and b_2 respectively, which are reordered and cast by the adversary using the credentials of a corrupt voter id_2. After the adversary casts b_2 as first, the voter verifies it on BBcast, considering it to be counted for them. However, the adversary casts b_1 afterwards, making it to be tallied. In Belenios, the server does not know the association between voter identities and credentials. It associates them when it receives a ballot for a credential from a voter identity. Therefore, it can associate the corrupt voter id_2 with the public credential cr_1 of the honest voter id_1. This attack is not possible in Helios since the registrar and the server agree on the correspondence between identities and credentials. Thus, corrupt voters cannot cast ballots for any credential other than theirs. We note that [18] also mentions a potential alternative design for Belenios where the registrar communicates the association between the voter identities and public credentials to the server before the election starts. That version would not suffer from this attack. The version we analyse is preferred for deployment since it promises everlasting privacy, yet it does pose new problems, as we show.

Similarly, we have $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_2, \mathcal{A}_1) \not\models \Phi^{\circ}_{\mathsf{iv1}}$ for the verification procedure $\mathcal{V}_2$. Considering the attack scenario in $\mathsf{AS}^1_{\mathcal{B}}$, when the voter verifies their second ballot in step 4, there is a single ballot on BBcast, i.e. $\mathsf{b}_2 \in \mathsf{BBcast}$. Therefore, there is no other ballot to be verified on BBcast that it had been cast by them. In reality, the voter may complain that their first ballot is not

Attack Scenario $\mathsf{AS}^1_{\mathcal{B}}$ ***(ballot reordering by*** $\mathcal{A}_1$***)***

1. $\mathsf{V}(\mathsf{id}_1)$ casts a ballot b_1 followed by another ballot b_2.
2. $\mathcal{A}$ blocks b_1 and b_2, corrupts $\mathsf{V}(\mathsf{id}_2)$ and casts b_2 for the credentials $(\mathsf{id}_2, \mathsf{cr}_1)$.
3. VS accepts b_2 coming from $(\mathsf{id}_2, \mathsf{cr}_1)$ and publishes $(\mathsf{cr}_1, \mathsf{b}_2)$ on BBcast.
4. $\mathsf{V}(\mathsf{id}_1)$ successfully verifies $(\mathsf{cr}_1, \mathsf{b}_2) \in$ BBcast.
5. $\mathcal{A}$ casts b_1 for the credentials $(\mathsf{id}_2, \mathsf{cr}_1)$.
6. VS accepts b_1 and publishes $(\mathsf{cr}_1, \mathsf{b}_1)$ on BBcast.
Outcome: b_1 is tallied for cr_1, even if b_2 is the last ballot cast and verified by id_1.

present on BBcast. However, since the voter's last cast is present on BBcast, the voter may feel confident that their ballot will be counted for them and may not complain.

Adversary $\mathcal{A}_2$**.** We obtain $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_i, \mathcal{A}_2) \vDash \mathsf{SE2E}[\mathsf{iv}_\triangledown, \mathsf{res}_\diamond]$ for $i \in \{3, 4\}$ and $\triangledown, \diamond \in \{\circ, \bullet\}$, proving that Belenios indeed satisfies stronger end-to-end verifiability than Helios in this case. We note, however, that the case with $\mathcal{V}_1$ corresponds to the currently recommended procedure in Belenios [8, 18]. Thus, we should be able to prove $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_1, \mathcal{A}_2) \vDash \mathsf{SE2E}[\mathsf{iv}_\circ, \mathsf{res}_\bullet]$, but a variant of the ballot reordering attack presented for $\mathcal{A}_1$ prevents this, i.e. we have $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_1, \mathcal{A}_2) \nvDash \Phi^\circ_{\mathsf{iv}}$. In this variant of ballot reordering attack, as described in $\mathsf{AS}^2_{\mathcal{B}}$, the adversary can reorder and cast the ballots of an honest voter without using the credential of a corrupt voter. Note that the server is corrupt; therefore, it does not process the ballots in the correct order.

Attack Scenario $\mathsf{AS}^2_{\mathcal{B}}$ ***(ballot reordering by*** $\mathcal{A}_2$***)***

1. $\mathsf{V}(\mathsf{id}_1)$ casts a ballot b_1 followed by another ballot b_2.
2. $\mathcal{A}$ blocks b_1 and allows b_2 to be cast.
3. VS accepts b_2 for the credential cr_1 and publishes $(\mathsf{cr}_1, \mathsf{b}_2)$ on BBcast.
4. $\mathsf{V}(\mathsf{id}_1)$ successfully verifies $(\mathsf{cr}_1, \mathsf{b}_2) \in$ BBcast.
5. $\mathcal{A}$ casts b_1 for the credential cr_1.
6. VS accepts b_1 and publishes $(\mathsf{cr}_1, \mathsf{b}_1)$ on BBcast.
Outcome: b_1 is tallied for cr_1, even if b_2 is the last ballot cast and verified by id_1.

Adversary $\mathcal{A}_3$**.** We expect $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_i, \mathcal{A}_3) \vDash \mathsf{SE2E}[\mathsf{iv}_\circ, \mathsf{res}_\bullet]$ for any i, since Belenios is claimed to be secure against a corrupt registrar. However, we find $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_i, \mathcal{A}_3) \nvDash \mathsf{SE2E}[\mathsf{iv}_\circ, \mathsf{res}_\bullet]$ for $i = 1, 2, 4$. Only the scenario with the individual verification $\mathcal{V}_3$, i.e. voters verify their ballots at the end of the election, satisfies strong end-to-end verifiability even for corrupt voters, i.e. $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_3, \mathcal{A}_3) \vDash \mathsf{SE2E}[\mathsf{iv}_\triangledown, \mathsf{res}_\diamond]$ for $\triangledown, \diamond \in \{\circ, \bullet\}$.

One of the attacks we capture against individual verifiability of Belenios with the procedures $\mathcal{V}_1$ and $\mathcal{V}_2$, i.e. $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_i, \mathcal{A}_3) \nvDash \Phi^\circ_{\mathsf{iv1}}$ for $i \in \{1, 2\}$, is the ballot reordering attack presented in $\mathsf{AS}^1_{\mathcal{B}}$. We also discover another attack against the individual verifiability of honest voters, as presented in $\mathsf{AS}^3_{\mathcal{B}}$. In this attack, the adversary blocks the honest voter id_1's ballot b and casts it using the login credentials, i.e. pwd_2 of a corrupt voter id_2. Subsequently, the voter verifies their ballot on BBcast. However, the adversary casts another ballot $\mathsf{b}_{\mathcal{A}}$ for the honest voter's public credential cr_1 before the voting phase ends. Then, the ballot $\mathsf{b}_{\mathcal{A}}$ from the adversary is tallied for cr_1 even if the ballot b from the honest voter is successfully verified. This attack is more serious than ballot reordering attack, since the adversary can choose the vote and generate its own ballot.

Attack Scenario $\mathsf{AS}^3_{\mathcal{B}}$ ***(individual verifiability attack by*** $\mathcal{A}_3$***)***

1. VR registers $\mathsf{V}(\mathsf{id}_1)$ and $\mathsf{V}(\mathsf{id}_2)$ with $\mathsf{Cred}(\mathsf{id}_1, \mathsf{cr}_1, \mathsf{sk}_{\mathsf{id}_1})$ and $\mathsf{Cred}(\mathsf{id}_2, \mathsf{cr}_2, \mathsf{sk}_{\mathsf{id}_2})$ respectively.
2. $\mathcal{A}$ corrupts $\mathsf{V}(\mathsf{id}_2)$ and VR and obtains $\langle \mathsf{pwd}_2, \mathsf{sk}_{\mathsf{id}_1} \rangle$.
3. $\mathsf{V}(\mathsf{id}_1)$ casts a ballot b for the credentials $(\mathsf{id}_1, \mathsf{cr}_1)$.
4. $\mathcal{A}$ blocks b, casts it for the credentials $(\mathsf{id}_2, \mathsf{cr}_1)$.
5. VS accepts b and publishes $(\mathsf{cr}_1, \mathsf{b})$ on BBcast.
6. $\mathsf{V}(\mathsf{id}_1)$ successfully verifies $(\mathsf{cr}_1, \mathsf{b}) \in$ BBcast.
7. $\mathcal{A}$ casts another ballot $\mathsf{b}_{\mathcal{A}}$ for $(\mathsf{id}_2, \mathsf{cr}_1)$.
8. VS accepts $\mathsf{b}_{\mathcal{A}}$ and publishes $(\mathsf{cr}_1, \mathsf{b}_{\mathcal{A}})$ on BBcast.
Outcome: $\mathsf{b}_{\mathcal{A}}$ is tallied for cr_1, even if b is the only ballot cast and verified by id_1.

We also find the clash attack on empty ballots in this case, i.e. $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_4, \mathcal{A}_3) \nvdash \Phi^{\nabla}_{\mathsf{iv2}}$ for $\nabla \in \{\circ, \bullet\}$. In this attack, the corrupt registrar registers two or more voter identities with the same election credentials. Assume the voters did not vote during the election but verified BBtally to ensure nobody cast a ballot for their public credential. Thus, we capture the clash on the credentials. Moreover, we find $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_1, \mathcal{A}_3) \nvdash \Phi^{\nabla}_{\mathsf{iv2}}$ for $\nabla \in \{\circ, \bullet\}$, which is also not expected. It can be explained by the fact that the ballots from honest voters registered with the same election credential are cast by the adversary using the login credential of a corrupt voter. In this way, one of the verified ballots is excluded from the tally. We present this attack in $\mathsf{AS}^4_{\mathcal{B}}$. Note that another version of this attack considering three voters: two are honest, and the third is corrupt, is also possible, as presented in [6].

Attack Scenario $\mathsf{AS}^4_{\mathcal{B}}$ ***(clash attack by*** $\mathcal{A}_3$***)***

1. VR registers $\mathsf{V}(\mathsf{id}_1)$ and $\mathsf{V}(\mathsf{id}_2)$ with the same credentials $\langle \mathsf{cr}, \mathsf{sk}_{\mathsf{id}} \rangle$.
2. $\mathsf{V}(\mathsf{id}_1)$ casts a ballot b_1 for the credentials $(\mathsf{id}_1, \mathsf{cr})$.
3. $\mathcal{A}$ blocks b_1, corrupts $\mathsf{V}(\mathsf{id}_3)$ and casts b_1 for the credentials $(\mathsf{id}_3, \mathsf{cr})$.
4. $\mathsf{V}(\mathsf{id}_1)$ successfully verifies $(\mathsf{cr}, \mathsf{b}_1) \in$ BBcast.
5. $\mathsf{V}(\mathsf{id}_2)$ casts a ballot b_2 for the credentials $(\mathsf{id}_2, \mathsf{cr})$.
6. $\mathcal{A}$ blocks b_2 and casts b_2 for the credentials $(\mathsf{id}_3, \mathsf{cr})$.
7. $\mathsf{V}(\mathsf{id}_2)$ successfully verifies $(\mathsf{cr}, \mathsf{b}_2) \in$ BBcast.
Outcome: only one ballot is tallied for id_1 and id_2 even if both perform a successful verification.

We note that another ballot stuffing attack is possible for an honest voter, which is described as weak ballot stuffing in [6]. Even though we do not capture it with the symbolic definition in Figure 4.1, and it is not an attack on multiset-based election verifiability, we present the attack in $\mathsf{AS}^5_{\mathcal{B}}$. This attack can only be observed by the voter while trying to cast a ballot. In addition, the adversary does not have to control the communication network for this attack. Thus, the consequence is that the honest voter id_1 is not able to cast a vote, no matter what infrastructure is used. This is related to a complementary property of accountability that assigns responsibilities when verifiability fails [39].

Adversary $\mathcal{A}_4$**.** When both the server and the registrar are corrupt, we find, as expected, the same results for Belenios as for Helios. Most notably, we have the positive result $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_3, \mathcal{A}_4) \vdash \mathsf{SWE2E}[\mathsf{iv}_\circ, \mathsf{res}_\circ]$, implying that Belenios satisfies weak end-to-end verifiability with $\mathcal{V}_3$, i.e.

Attack Scenario $\mathsf{AS}^5_{\mathcal{B}}$ ***(weak ballot stuffing by*** $\mathcal{A}_3$***)***

1. VR registers $\mathsf{V}(\mathsf{id}_1)$ and $\mathsf{V}(\mathsf{id}_2)$ with $\mathsf{Cred}(\mathsf{id}_1, \mathsf{cr}_1, \mathsf{sk}_{\mathsf{id}_1})$ and $\mathsf{Cred}(\mathsf{id}_2, \mathsf{cr}_2, \mathsf{sk}_{\mathsf{id}_2})$ respectively.
2. $\mathcal{A}$ corrupts $\mathsf{V}(\mathsf{id}_2)$ and VR and obtains $\langle \mathsf{pwd}_2, \mathsf{sk}_{\mathsf{id}_1} \rangle$.
3. $\mathcal{A}$ casts a ballot $\mathsf{b}_{\mathcal{A}}$ for the credentials $(\mathsf{id}_2, \mathsf{cr}_1)$.
4. VS accepts $\mathsf{b}_{\mathcal{A}}$ and publishes $(\mathsf{cr}_1, \mathsf{b}_{\mathcal{A}})$ on BBcast.
5. $\mathsf{V}(\mathsf{id}_1)$ casts a ballot b for their credentials $(\mathsf{id}_1, \mathsf{cr}_1)$.
6. VS rejects b since it sees inconsistency in logs: $(\mathsf{id}_2, \mathsf{cr}_1)$ vs. $(\mathsf{id}_1, \mathsf{cr}_1)$.
Outcome: id_1 cannot vote, instead, $\mathsf{b}_{\mathcal{A}}$ is tallied for cr_1.

voters verify their ballots at the end of the election, even if both the registrar and the server are corrupt.

Adversary $\mathcal{A}_5$**.** In addition to described attacks above, we have $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_i, \mathcal{A}_4) \nvdash \mathsf{SE2E}[\mathsf{iv}_{\triangledown}, \mathsf{res}_{\circ}]$ for any i and $\triangledown \in \{\circ, \bullet\}$, since we recover the classic clash attacks as in Helios.

Conclusion. Our analysis shows that Belenios is secure against a corrupt registrar or a corrupt server only when the voters verify their ballots at the end of the election, which corresponds to the verification procedure $\mathcal{V}_3$. However, in the real deployment of Belenios, voters are allowed to verify their last ballot cast anytime during the election, as in the procedure $\mathcal{V}_1$. For this procedure, we capture attacks even when both the registrar and server are honest. The attacks we have discovered are exploited against individual verifiability of Belenios with revoting. In many cases, the adversary blocks the ballots of an honest voter and casts them using the login credentials of a corrupt voter, i.e. the honest voter's credential is associated with the corrupt voter's identity by the server. Then, the adversary is able to manipulate the ballot to be tallied for that credential. Similar to Helios, we capture the new versions of clash attacks with a corrupt registrar. The procedure $\mathcal{V}_2$, for which the voters verify their last ballot cast as well as all ballots cast next to their credentials on BB, protects voters against clash attacks, but not against other found attacks. Finally, we prove that Belenios satisfies weak end-to-end verifiability with $\mathcal{V}_3$ when both the registrar and server are corrupt.

6.4 Towards Improved Election Verifiability

Belenios is aimed to be secure against a corrupt server, i.e. the adversary model $\mathcal{A}_2$, or a corrupt registrar, i.e. $\mathcal{A}_3$. However, we have captured verifiability attacks even when both are honest, i.e. $\mathcal{A}_1$. The attacks arise from the vulnerability of the voting server that it does not know the association of the voter identities with public credentials; therefore, it accepts ballots from some voter identity even if another voter identity generated the ballots. On the other hand, the ballot structure $\mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p}_{\mathsf{R}}, \mathsf{p}_{\mathsf{L}} \rangle$ protects against ballot stuffing and copying attacks since the components s, i.e. the signature, and p_{L}, i.e. the proof of label, authenticates b that it was generated by the voter who owns the corresponding credential. However, none of those links b to the voter identity who casts it; thus, the ballot is copied and cast, or the ballots are reordered and cast by another identity.

In this section, we propose solutions to improve the election verifiability of Belenios with respect to adversary models $\mathcal{A}_1$, $\mathcal{A}_2$, or $\mathcal{A}_3$. Our solutions enrich the structure of the label ℓ inside the proof $\mathsf{p}_{\mathsf{L}} = \mathsf{pr}_{\mathsf{L}}(\mathsf{c}, \mathsf{r}, \ell)$ in order to prevent the attacks. The proof p_{L} is a non-interactive Chaum-Pedersen proof of knowledge, where the label $\ell = \mathsf{cr}$ is part of the input to a hash function (SHA-256) that computes the challenge. We propose extending the label with other components so that their concatenation will be the input to the hash function. Thus, our

solutions require changes only on the implementation of the voting platform and the server, making them feasible and transparent for voters. We present the new structure of the label stepwise: first, a label structure that protects against ballot reordering attacks by $\mathcal{A}_1, \mathcal{A}_2$, or $\mathcal{A}_3$; then, a label structure that protects against other attacks by $\mathcal{A}_3$; finally, combining the two labels protects against all attacks by $\mathcal{A}_1, \mathcal{A}_2$, or $\mathcal{A}_3$.

6.4.1 Protection Against Ballot Reordering

The ballot reordering attack is based on reordering of the two ballots cast by a voter so that the last ballot is cast as first by the adversary corrupting the network. The voter verifies their last ballot cast, then the adversary casts the first ballot as last, which results in the first ballot to be tallied for that voter. This attack is prevented if the ballot includes information from the last ballot published on BBcast, i.e. starting from the first ballot, each ballot recursively contains the information from the previous ballot. The solution is as follows.

We assume initially there are empty ballots next to eligible public credentials on BB. Moreover, a specific portion of BB is reserved for displaying the last ballot cast for each credential:

$$\begin{array}{l} \text{(Before voting)} \quad \mathsf{BBlast}: \quad (\mathsf{cr}_1, \bot), \ldots, (\mathsf{cr}_n, \bot) \\ \hline \text{(During voting)} \quad \mathsf{BBlast}: \quad (\mathsf{cr}_1, \mathsf{b}_1), \ldots, (\mathsf{cr}_n, \mathsf{b}_n) \end{array}$$

When the voting platform VP constructs a new ballot for a public credential cr, it fetches from BBlast the last ballot b' associated to cr. Then, in the construction of the proof p_L, instead of cr, VP uses the label $\mathsf{h}(\mathsf{cr}, \mathsf{b}')$, where h is a collision-resistant hash function mapping the pair $(\mathsf{cr}, \mathsf{b}')$ into the appropriate domain for labels:

$$\ell = \mathsf{h}(\mathsf{cr}, \mathsf{b}'); \quad \mathsf{p}_\mathsf{L} = \mathsf{pr}_\mathsf{L}(\mathsf{c}, \mathsf{r}, \ell); \quad \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p}_\mathsf{R}, \mathsf{p}_\mathsf{L}, \ell \rangle.$$

BBcast records all ballots cast for cr, and their order cannot be changed on BB. The voting server VS and the election auditors can look at any two consecutive ballots b' and b cast for a credential cr, recompute $\ell = \mathsf{h}(\mathsf{cr}, \mathsf{b}')$ and verify that

$$\mathsf{ver}_\mathsf{L}(\mathsf{p}_\mathsf{L}, \mathsf{c}, \ell) = \mathsf{true},$$

thereby ensuring that the party constructing b indeed expects it to follow b'. In particular, if an honest voter casts b_2 after b_1, the adversary cannot cast b_2 first, since it would have to generate a proof linking b_2 to an earlier ballot b_0, i.e. $\bot$, which is impossible since the adversary does not know the randomness r_2 in the ciphertext corresponding to b_2. Note that this also prevents $\mathcal{A}$ from delaying both ballots: they would need to be both preceded by b_0 and only one will be accepted. This label structure ensures election verifiability in corruption scenarios when the registrar is honest, i.e. $\mathcal{A}_1$ and $\mathcal{A}_2$, but it also prevents the ballot reordering attack exploited with $\mathcal{A}_3$.

6.4.2 Protection Against a Corrupted Registrar

The main cause of the attacks, in the scenario with a corrupt registrar, i.e. $\mathcal{A}_3$, is that the adversary can block a ballot b of an honest voter and cast it under the identity of a corrupt voter, while maintaining the same public credential associated to b. Subsequently, after the honest voter verified b, the adversary can override it with an own ballot $\mathsf{b}_\mathcal{A}$. In order to prevent this, we enrich the label structure with a commitment to the voter identity. More precisely, during ballot casting for a voter id, the voting platform VP generates a fresh randomness u, constructs the label $\langle \mathsf{cr}, \mathsf{com}(\mathsf{id}, \mathsf{u}) \rangle$ and sends u together with the ballot to the voting server VS. Since the label cannot be reconstructed publicly by election auditors, we explicitly include

it in the ballot. We have:

$$\begin{array}{ll} \mathsf{VP} : & \ell = \langle \mathsf{cr}, \mathsf{com}(\mathsf{id}, \mathsf{u}) \rangle;\ \ \mathsf{p}_\mathsf{L} = \mathsf{pr}_\mathsf{L}(\mathsf{c}, \mathsf{r}, \ell);\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p}_\mathsf{R}, \mathsf{p}_\mathsf{L}, \ell \rangle, \\ \hline \mathsf{VS} : & \text{receives } (\mathsf{cr}, \mathsf{b}, \mathsf{u}) \text{ from VP for a given id; constructs } \ell' = \langle \mathsf{cr}, \mathsf{com}(\mathsf{id}, \mathsf{u}) \rangle, \\ & \text{accepts b if and only if } \ell' = \ell \text{ and } \mathsf{ver}_\mathsf{L}(\mathsf{p}_\mathsf{L}, \mathsf{c}, \ell) = \mathsf{true}. \end{array}$$

In the attack scenario described above, the adversary cannot construct a proof p'_L so that b is accepted by VS under the identity of a corrupt voter. Indeed, the ciphertext in b cannot be detached from the identity of the honest voter. More generally, we prove that this structure of the label is sufficient to ensure election verifiability in the corruption scenarios when the server is honest, i.e. $\mathcal{A}_1$ and $\mathcal{A}_3$. Election auditors can still check the proof p_L on BB, but they will only be able to ensure the ballot is cast for the expected public credential cr and will not have knowledge of the underlying id. Note that we cannot use the voter id directly in the label, as this would reveal the link between id and cr.

6.4.3 Putting the Labels Together

We combine the labels proposed above to protect Belenios against all attacks exploited by $\mathcal{A}_1$, $\mathcal{A}_2$ or $\mathcal{A}_3$ as follows:

$$\ell_1 = \mathsf{h}(\mathsf{cr}, \mathsf{b}'); \quad \ell_2 = \mathsf{com}(\mathsf{id}, \mathsf{u}); \quad \ell = \langle \ell_1, \ell_2 \rangle,$$

where b′ is the last ballot present on BB, u is a fresh randomness. Then, we specify three variants of Belenios with respect to their label structure:

$$\begin{array}{rl} \text{Belenios}_{\text{tr}} : & \ell_{\text{tr}} = \mathsf{h}(\mathsf{cr}, \mathsf{b}'); \\ \text{Belenios}_{\text{id}} : & \ell_{\text{id}} = \langle \mathsf{cr}, \mathsf{com}(\mathsf{id}, \mathsf{u}) \rangle; \\ \text{Belenios+} : & \ell+ = \langle \mathsf{h}(\mathsf{cr}, \mathsf{b}'), \mathsf{com}(\mathsf{id}, \mathsf{u}) \rangle \end{array}$$

Note that ℓ_{tr} depends on cr. That is why it is not necessary to include cr in the tuple as in ℓ_{id}. As we show in the following sections, we perform an automated verification of those variants with Tamarin and obtain the following results for $\diamond \in \{\circ, \bullet\}$:

$$\begin{array}{lrll} & (\ \text{Belenios}_{\text{tr}},\ \mathcal{A}\ ) & \models \mathsf{SE2E}[\mathsf{iv}_\circ, \mathsf{res}_\diamond] & \text{for } \mathcal{A} \in \{\mathcal{A}_1, \mathcal{A}_2\}, \\ & (\ \text{Belenios}_{\text{id}},\ \mathcal{A}\ ) & \models \mathsf{SE2E}[\mathsf{iv}_\circ, \mathsf{res}_\diamond] & \text{for } \mathcal{A} \in \{\mathcal{A}_1, \mathcal{A}_3\}, \\ & (\ \text{Belenios+},\ \mathcal{A}\ ) & \models \mathsf{SE2E}[\mathsf{iv}_\circ, \mathsf{res}_\diamond] & \text{for } \mathcal{A} \in \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\}, \\ \text{while we have} & (\ \text{Belenios},\ \mathcal{A}\ ) & \not\models \mathsf{SE2E}[\mathsf{iv}_\circ, \mathsf{res}_\diamond] & \text{for } \mathcal{A} \in \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\}. \end{array}$$

The property $\mathsf{SE2E}[\mathsf{iv}_\circ, \mathsf{res}_\bullet]$ corresponds to the standard verifiability notion used in [21, 17]. In particular, this notion ensures that, if an honest voter successfully verified a ballot b for a public credential cr, then b is counted in the final tally as the contribution of cr. However, when revoting is allowed, this might not be the case. As we showed in the previous section, even if the last ballot cast is verified by an honest voter, another ballot may be counted in the final tally for the voter's credential cr. Therefore, Belenios does not satisfy $\mathsf{SE2E}[\mathsf{iv}_\circ, \mathsf{res}_\bullet]$. However, the label $\ell = \langle \mathsf{h}(\mathsf{cr}, \mathsf{b}'), \mathsf{com}(\mathsf{id}, \mathsf{u}) \rangle$ used in Belenios+ protects the honest voter against ballot reordering attack, since the adversary cannot manipulate $\mathsf{h}(\mathsf{cr}, \mathsf{b}')$ inside the ballot to reorder the ballots. Moreover, the adversary cannot cast the honest voter's ballot b under the identity of a corrupt voter, since the adversary cannot change the label of the ballot, and thus the proof p_L, which requires the knowledge of the encryption randomness. Similarly, there is no way to cast two ballots from different voters for the same credential due to the commitment inside the ballot and log consistency checks performed by the voting

server. Therefore, the adversary cannot mount clash attacks corrupting the registrar. That is why $\mathsf{SE2E}[\mathsf{iv}_\circ, \mathsf{res}_\bullet]$ holds for Belenios+.

6.4.4 Belenios+ Specification

In this section, we present the Tamarin specification of the Belenios+ protocol, for which we consider the individual verification procedure $\mathcal{V}_1$ from Section 6.2 and adversaries $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ from Table 6.1. The specification of Belenios+ differs from Belenios in the rules $\mathsf{R}^{\mathsf{VS}}_{\mathsf{bb}}$, $\mathsf{R}^{\mathsf{VP}}_{\mathsf{vote}}$, and $\mathsf{R}^{\mathsf{VS}}_{\mathsf{cast}}$, where the first rule introduces linear (consumable) facts $\mathsf{BBlast}(\mathsf{cr}, \bot)$ and $\mathsf{VPlast}(\mathsf{cr}, \bot)$ for tracking the last ballot cast by each credential on BB, and other two rules update the state of those facts each time a ballot is generated or cast. Moreover, the rule $\mathsf{R}^{\mathsf{VP}}_{\mathsf{vote}}$ improves the ballot structure with respect to the label $\ell = \langle \mathsf{h}(\mathsf{cr}, \mathsf{b}_0), \mathsf{com}(\mathsf{id}, \mathsf{u})\rangle$, and therefore extends it with ℓ and adds committed value u to the data sent to VS for ballot casting. On the other hand, $\mathsf{R}^{\mathsf{VS}}_{\mathsf{cast}}$ updates its procedure with respect to the new ballot structure and additional check for the commitment. We provide the Belenios+ specification in Figure 6.6 and the Tamarin codes for each variant of Belenios in [52] online.

In the specification of Belenios+, we use ! to denote persistent facts since we also use linear facts. We highlight the difference between Belenios+ and Belenios in red as follows:

$$\mathsf{R}^{\mathsf{VS}}_{\mathsf{bb}}: \quad [\ !\mathsf{BBreg}(\mathsf{cr})\ ] - [!\mathsf{BBcast}(\mathsf{cr}, \bot)\ ] \rightarrowtail [\ !\mathsf{BBcast}(\mathsf{cr}, \bot), \mathsf{BBlast}(\mathsf{cr}, \bot), \mathsf{VPlast}(\mathsf{cr}, \bot)\ ]$$

For tracking the last ballot on BB, the fact $\mathsf{BBlast}(\mathsf{cr}, \mathsf{b}_0)$ should be consumed each time a ballot is received by VS. For example, for the first ballot b cast for cr, $\mathsf{BBlast}(\mathsf{cr}, \bot)$ is consumed, and a new fact $\mathsf{BBlast}(\mathsf{cr}, \mathsf{b})$ is generated. This is recursively followed by new ballots cast. Since BBlast is used on the voting server's side, we need an additional fact VPlast for the voting platform's side that will be consumed each time a new ballot is generated and cast. For the first ballot cast, VP consumes $\mathsf{VPlast}(\mathsf{cr}, \bot)$ generated by VS in the setup phase. Then, as soon as the server records a ballot b for cr to $\mathsf{BBlast}(\mathsf{cr}, \mathsf{b})$, it also generates a fact $\mathsf{VPlast}(\mathsf{cr}, \mathsf{b})$ which will be consumed by VP if the voter intends to cast another ballot. The following rule models the ballot casting procedure by VP:

$$\begin{aligned}
\mathsf{R}^{\mathsf{VP}}_{\mathsf{vote}}: \quad & \mathsf{let}\ \ \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk}_\mathsf{E}, \mathsf{r});\ \ \mathsf{s} = \mathsf{sign}(\mathsf{c}, \mathsf{sk}_\mathsf{id});\ \ \mathsf{p}_\mathsf{R} = \mathsf{pr}_\mathsf{R}(\mathsf{c}, \mathsf{r}, \mathsf{vlist});\\
& \quad \ell = \langle \mathsf{h}(\mathsf{cr}, \mathsf{b}_0), \mathsf{com}(\mathsf{id}, \mathsf{u})\rangle;\ \ \mathsf{p}_\mathsf{L} = \mathsf{pr}_\mathsf{L}(\mathsf{c}, \mathsf{r}, \ell);\\
& \quad \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p}_\mathsf{R}, \mathsf{p}_\mathsf{L}, \ell\rangle;\ \ \mathsf{a} = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{b}, \mathsf{u}\rangle)\ \ \mathsf{in}\\
& [\ !\mathsf{Cred}(\mathsf{id}, \mathsf{cr}, \mathsf{sk}_\mathsf{id}), !\mathsf{Pwd}(\mathsf{id}, \mathsf{pwd}), !\mathsf{BBcand}(\mathsf{v}), !\mathsf{Vlist}(\mathsf{vlist}), !\mathsf{BBkey}(\mathsf{pk}_\mathsf{E}),\\
& \ \ \mathsf{Fr}(\mathsf{r}), \mathsf{Fr}(\mathsf{t}), \mathsf{Fr}(\mathsf{u}), \mathsf{VPlast}(\mathsf{cr}, \mathsf{b}_0)\ ] - [\ \mathsf{Vote}(\mathsf{id}, \mathsf{v}), \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b}), \mathsf{Last}(\mathsf{id}, \mathsf{t})\ ] \rightarrowtail\\
& [\ !\mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{b}), \mathsf{Out}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{b}, \mathsf{a}, \mathsf{u}\rangle)\ ]
\end{aligned}$$

In the above rule, the ballot is generated according to the new label structure $\ell = \langle \mathsf{h}(\mathsf{cr}, \mathsf{b}_0), \mathsf{com}(\mathsf{id}, \mathsf{u})\rangle$, which requires a commitment to the voter id who generates the ballot, with a randomness u. The commitment randomness u is included in the authentication message a and sent directly to the public channel. Thus, the adversary learns about it but cannot change it due to the zero-knowledge proof p_L and the authentication message a generated with the knowledge of the encryption randomness and the password, respectively. Moreover, the adversary cannot copy the ballot and cast it on behalf of another voter id' since the voter identity id inside the proof p_L will not match with id'. In reality, the commitment randomness u should be sent via a private channel not to violate the privacy of the voter who casts the respective ballot. However, the secrecy of u is not important for verifiability properties; thus, we can send it on the public channel. The rule $\mathsf{R}^{\mathsf{VP}}_{\mathsf{vote}}$ consumes the linear fact $\mathsf{VPlast}(\mathsf{cr}, \mathsf{b}_0)$, thus it can be executed only once for any ballot posted on BB. This mechanism is complemented by the ballot casting rule on the server side:

$$\mathsf{R}^{\mathsf{VS}}_{\mathsf{cast}}: \quad \text{let } \ell' = \langle \mathsf{h}(\mathsf{cr}, \mathsf{b}_0), \mathsf{com}(\mathsf{id}, \mathsf{u})\rangle;\ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p}_\mathsf{R}, \mathsf{p}_\mathsf{L}, \ell\rangle;\ \mathsf{a}' = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{b}, \mathsf{u}\rangle) \text{ in}$$
$$[\ \mathsf{In}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{b}, \mathsf{a}, \mathsf{u}\rangle), !\mathsf{BBreg}(\mathsf{cr}), !\mathsf{Pwd}(\mathsf{id}, \mathsf{pwd}), !\mathsf{BBkey}(\mathsf{pk}_\mathsf{E}), !\mathsf{Vlist}(\mathsf{vlist}), \mathsf{BBlast}(\mathsf{cr}, \mathsf{b}_0)\ ] \multimap [\ \mathsf{a}' \equiv \mathsf{a}, \ell' \equiv \ell, \mathsf{verify}(\mathsf{s}, \mathsf{c}, \mathsf{cr}) \equiv \mathsf{true}, \mathsf{ver}_\mathsf{R}(\mathsf{p}_\mathsf{R}, \mathsf{c}, \mathsf{pk}_\mathsf{E}, \mathsf{vlist}) \equiv \mathsf{true}, \mathsf{ver}_\mathsf{L}(\mathsf{p}_\mathsf{L}, \mathsf{c}, \ell) \equiv \mathsf{true}, \mathsf{Log}(\mathsf{id}, \mathsf{cr}), \mathsf{VScast}(\mathsf{id}, \mathsf{b}), !\mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ] \mapsto [\ !\mathsf{BBcast}(\mathsf{cr}, \mathsf{b}), \mathsf{BBlast}(\mathsf{cr}, \mathsf{b}), \mathsf{VPlast}(\mathsf{cr}, \mathsf{b})\ ]$$

where we receive a ballot from the voter and perform the corresponding validation steps: verifying the password, the signature and the zero-knowledge proofs. The fact containing the last ballot cast is consumed, and new facts are produced for the new ballot: one to be consumed by the voting platform, and one to be consumed by the server when the next ballot is cast. In order to obtain termination, we have a restriction limiting the number of applications of this rule to at most four for each voter.

Limiting the Number of Ballots in Tamarin

To obtain verification results in Tamarin regarding Belenios+ specification, we need to restrict the number of ballots which can be cast by each voter, i.e. we need to limit the number of revotes. Our current specification in Tamarin allows up to four cast ballots, i.e. three revotes. The code does not terminate for a higher bound. We specify the number of ballots allowed to be cast by restrictions in Tamarin. If the allowance is for two ballots, then we use a restriction Ψ_{two} as follows:

$$\Psi_{\mathsf{two}}: \quad \mathsf{TwoTimes}(\mathsf{x})\ @i \wedge \mathsf{TwoTimes}(\mathsf{x})\ @j \wedge \mathsf{TwoTimes}(\mathsf{x})\ @k \Rightarrow i = j \vee i = k \vee j = k$$

This restriction refers to the action fact $\mathsf{TwoTimes}(\langle \mathsf{cr}, \mathsf{'cast'}\rangle)$ used in the rule $\mathsf{R}^{\mathsf{VS}}_{\mathsf{cast}}$, which leads to a limitation on the number of ballots on BBcast. Similarly, to specify an allowance for three and four ballots, we use the following restrictions:

$$\Psi_{\mathsf{three}}: \quad \mathsf{ThreeTimes}(\mathsf{x})\ @i \wedge \mathsf{ThreeTimes}(\mathsf{x})\ @j \wedge \mathsf{ThreeTimes}(\mathsf{x})\ @k \wedge \mathsf{ThreeTimes}(\mathsf{x})\ @l \Rightarrow i = j \vee i = k \vee i = l \vee j = k \vee j = l \vee k = l$$

$$\Psi_{\mathsf{four}}: \quad \mathsf{FourTimes}(\mathsf{x})\ @i \wedge \mathsf{FourTimes}(\mathsf{x})\ @j \wedge \mathsf{FourTimes}(\mathsf{x})\ @k \wedge \mathsf{FourTimes}(\mathsf{x})\ @l \wedge \mathsf{FourTimes}(\mathsf{x})\ @m \Rightarrow i = j \vee i = k \vee i = l \vee i = m \vee j = k \vee j = l \vee j = m \vee k = l \vee k = m \vee l = m$$

For these restrictions, we call the respective action facts $\mathsf{ThreeTimes}(\langle \mathsf{cr}, \mathsf{'cast'}\rangle)$ and $\mathsf{FourTimes}(\langle \mathsf{cr}, \mathsf{'cast'}\rangle)$ in the same server casting rule $\mathsf{R}^{\mathsf{VS}}_{\mathsf{cast}}$.

6.4.5 Verification Results for Belenios$_{\mathsf{tr}}$, Belenios$_{\mathsf{id}}$ and Belenios+

Table 6.3 contains the verification results for the corresponding specifications with Tamarin, available online [52]. We can see the positive results for Belenios+ as the union of the positive results for Belenios$_{\mathsf{tr}}$ and Belenios$_{\mathsf{id}}$ in each of the corruption scenarios $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$. Note that Belenios$_{\mathsf{tr}}$ does not protect against a corrupt registrar, i.e. $\mathcal{A}_3$, and Belenios$_{\mathsf{id}}$ does not protect against a corrupt server, i.e. $\mathcal{A}_2$. Therefore, we do not include those scenarios in the table. In the specifications, we bound the number of ballots per voter since Tamarin does not terminate without such a bound. The table Table 6.4 provides the execution times for the verification of Belenios+ with respect to 2 ballots, 3 ballots, and 4 ballots per voter, which corresponds to at most three revotes per voter.

$\Phi/\mathcal{A}_j$	Belenios			$\text{Belenios}_{\text{tr}}$		$\text{Belenios}_{\text{id}}$		Belenios+		
	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_1$	$\mathcal{A}_3$	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$
$\Phi^{\circ}_{\text{iv1}}$	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓
$\Phi^{\triangledown}_{\text{iv2}}$	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓
$\Phi^{\triangledown}_{\text{iv3}}$	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Φ_{eli}	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
$\Phi^{\diamond}_{\text{res}}$	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Φ_{cons}	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

TABLE 6.3: Verification results for the variants of Belenios.

$\#b/\mathcal{A}_j$	Belenios+		
	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$
2 ballots per voter	18 sec	9 sec	43 sec
3 ballots per voter	1 min	37 sec	3 min 46 sec
4 ballots per voter	14 min 57 sec	15 min 28 sec	26 min 36 sec

TABLE 6.4: Execution times of the automated verification for Belenios+.

6.4.6 Details for the ZKP in $\text{Belenios}_{\text{tr}}$, $\text{Belenios}_{\text{id}}$ and Belenios+

The cryptography underlying Belenios [18, 29] makes use of a cyclic group $\mathbb{G} = \langle g \rangle$ with order q, which is a multiplicative subgroup of $\mathbb{F}_p^*$. Proofs are generated using the Chaum-Pedersen algorithm and made non-interactive by the Fiat-Shamir technique. The algorithm generates a challenge ch and a response re to prove the knowledge of a secret sec corresponding to a public value g^{sec}, and sends (ch, re) as a proof to the verifier. The verifier recomputes ch using the messages g^{sec} and re, and accepts if the computed value matches the one previously received. To generate the proof (ch, re), $w \in \mathbb{Z}_q$ is randomly chosen. Then,

$$ch = h(g^{sec},\ g^{w}) \mod q \quad \text{and} \quad re = w - sec \times ch \mod q$$

are computed. To verify the proof, given (ch, re) and g^{sec}, $A = g^{re}(g^{sec})^{ch}$ is computed and checked that ch is equal to $h(g^{sec}, A)$.

In the implementation of zero-knowledge proofs in Belenios [29], h is specifically the SHA-256 hash function [23], which can take an input up to 2^{64} bits and generates a fixed size output of 256 bits, and the proof $pr_L = proof_L(c, r, cr)$ is computed as follows:

$$ch = SHA256(cr \mid \langle pk, c \rangle \mid g^{w}) \mod q \quad \text{and} \quad re = w - r \times ch \mod q,$$

where $cr \in \mathbb{G}$ is the verification key of the corresponding voter, $w \in \mathbb{Z}_q$, pk is the election public key, $c = (g^r, pk^r g^v)$ is the ciphertext of the vote v. Here, the randomness r is the secret to be proved as a knowledge for a valid encryption. Thus, $A = g^{re}(g^r)^{ch}$ is computed for the verification of pr_L.

Our solutions require to replace cr in the proof pr_L with the following labels:

- $h(cr, b')$ for $\text{Belenios}_{\text{tr}}$,
- $\langle cr, com(id, t) \rangle$ for $\text{Belenios}_{\text{id}}$, and

- $\langle \mathsf{h}(\mathsf{cr}, \mathsf{b}'), \mathsf{com}(\mathsf{id}, \mathsf{t}) \rangle$ for Belenios+.

This means that cr in the generation of challenge ch will be replaced accordingly. For $\text{Belenios}_{\text{tr}}$, we propose to use a collision-resistant hash function h that takes as input cr and the former cast ballot b' on the BB. The hash function h can be SHA-256 for the compatibility within the system, i.e. $\mathsf{h}(\mathsf{cr}, \mathsf{b}') \equiv \mathsf{SHA256}(\mathsf{cr} \mid \mathsf{b}')$. Assume that the former ballot on BB is not empty, i.e. $\mathsf{b}' \neq \perp$. Then, b' will be in the following form:

$$\begin{aligned} \mathsf{b}' &= (\mathsf{c}, \mathsf{s}, \mathsf{pr}_{\mathsf{R}}, \mathsf{pr}_{\mathsf{L}}, \ell') \\ &= ((\mathsf{g}^{\mathsf{r}}, \mathsf{pk}^{\mathsf{r}}\mathsf{g}^{\mathsf{v}}), (\mathsf{ch}_1, \mathsf{re}_1), (\mathsf{ch}_2, \mathsf{re}_2), (\mathsf{ch}_3, \mathsf{re}_3), \mathsf{h}(\mathsf{cr}, \mathsf{b}'')). \end{aligned}$$

Here, s corresponds to a Schnorr-like digital signature which is also a pair of challenge and response in $\mathbb{Z}_{\mathsf{q}}$. Therefore, we have $\mathsf{c} \in \mathbb{G} \times \mathbb{G}$ and $(\mathsf{ch}_{\mathsf{i}}, \mathsf{re}_{\mathsf{i}}) \in \mathbb{Z}_{\mathsf{q}} \times \mathbb{Z}_{\mathsf{q}}$. Note that every element in $\mathbb{G}$ has the same size as p since $\mathbb{G}$ is a subgroup of $\mathbb{F}_{\mathsf{p}}^*$. In the specification [29], the lengths of p and q are taken as 2048 bits and 256 bits, respectively. Together with $\mathsf{cr} \in \mathbb{G}$ and $\ell' \in \mathbb{Z}_{\mathsf{q}}$, the input size for h makes $31 \times 256 \approx 2^{13}$ bits, which is definitely suitable for SHA-256.

For $\text{Belenios}_{\text{id}}$, cr in the challenge ch is replaced with $\langle \mathsf{cr}, \mathsf{com}(\mathsf{id}, \mathsf{t}) \rangle$. This new structure requires a commitment to the voter's identity id with a randomness t. Regarding the cryptography used for Belenios, the commitment can be a Pedersen commitment. In this case, another generator $\bar{\mathsf{g}}$ of $\mathbb{G}$ will be used for the commitment $\mathsf{com}(\mathsf{id}, \mathsf{t}) = \mathsf{g}^{\mathsf{id}}\bar{\mathsf{g}}^{\mathsf{t}} \in \mathbb{G}$ for $\mathsf{t} \in \mathbb{Z}_{\mathsf{q}}$. Thus, the input size of SHA-256 in ch will be increased by 2048 bits since we add a commitment in addition to cr. In a similar fashion, when we enrich pr_{L} by applying $\langle \mathsf{h}(\mathsf{cr}, \mathsf{b}'), \mathsf{com}(\mathsf{id}, \mathsf{t}) \rangle$ as a first argument in the challenge for Belenios+, the input size of SHA-256 will be increased by the output size of $\mathsf{h}(\mathsf{cr}, \mathsf{b}')$, which is 256 bits. Recall that $\mathsf{com}(\mathsf{id}, \mathsf{t})$ has the same length as cr and $\mathsf{h}(\mathsf{cr}, \mathsf{b}')$ is $\mathsf{SHA256}(\mathsf{cr} \mid \mathsf{b}')$ as shown above. Hence, our propositions to enrich the structure of the proof pr_{L} fit well with the cryptographic primitives used in Belenios.

6.5 BeleniosRF

BeleniosRF [14] is a variant of Belenios, which aims for receipt-freeness in addition to end-to-end verifiability. Receipt-freeness provides a voting scheme with stronger privacy guarantees than ballot privacy, ensuring that the voter cannot provide the coercer with a receipt proving how they voted. Belenios is not receipt-free since any voter can record the encryption randomness while generating a ballot and use this as a receipt to prove their vote to the coercer. The coercer can then recompute the ciphertext inside the voter's ballot, match it with the one on BB, and ensure that the vote they desire is cast. On the other hand, the voter cannot provide such a receipt in BeleniosRF since it requires a randomisation operation by the server when it receives a ballot, i.e. the server randomises the ballot before recording it to BB. Thus, the ballot recorded on BB differs from the one cast for a credential. Therefore, the encryption randomness would not help the coercer to ensure the vote is cast, which protects honest voters against a coercer. Besides honest voters, even corrupt voters cannot provide any receipt when cooperating with the coercer. In this sense, BeleniosRF satisfies a strong notion of receipt-freeness.

Privacy and verifiability are often mentioned in the literature as two conflicting properties of e-voting protocols. Privacy requires the sensitive data in an election to be hidden, for example, the link between a vote and the voter who casts it. On the other hand, verifiability requires revealing sufficient data to verify all the election procedures. Thus, there is a trade-off between the data to be hidden and the data to be revealed, i.e. privacy and verifiability. Furthermore, this tension is accentuated when we consider receipt-freeness, since the data allowing individual verification could potentially be used as a receipt for the coercer. In this section, we

perform a security analysis of BeleniosRF with respect to verifiability, aiming to evaluate the trade-off between receipt-freeness and verifiability. We use the tool ProVerif for the automated verification of BeleniosRF. According to our observation, ProVerif is better at handling complex equational theories than Tamarin, i.e. the ProVerif codes terminate faster when the cryptographic primitives require complex equations. Our verification results obtained with ProVerif show that BeleniosRF provides weaker verifiability guarantees than Belenios (when revoting is not allowed or voters verify their ballots in the tally phase). Thus, in particular to Belenios, we confirm that receipt-freeness weakens verifiability while strengthening privacy.

6.5.1 BeleniosRF Protocol Structure

BeleniosRF deploys an additional randomisation server to randomise the ballots before recording them on BB. It may also deploy a single server, as in Belenios, improving it to enable randomisation operation. The randomisation relies on a cryptographic primitive called signatures on randomisable ciphertexts [12], instantiated from Groth-Shai proofs [30]. Note that in the previous sections we have defined the ballot structure of Belenios as $\mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p}_\mathsf{R}, \mathsf{p}_\mathsf{L} \rangle$. For modelling BeleniosRF, we combine the two proofs p_L and p_R into one $\mathsf{p} = \mathsf{pr}(\mathsf{c}, \mathsf{v}, \mathsf{r}, \mathsf{cr})$, as in the version of zero-knowledge proof presented in [18]. The setup and tally phases of BeleniosRF are similar to Belenios. Thus, we focus on the voting phase and the individual verification procedure of BeleniosRF.

In the voting phase, the voting platform of a voter id having the credentials $(\mathsf{sk}_\mathsf{id}, \mathsf{cr})$ generates a ballot of the form $\mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p} \rangle$, i.e.

$$\begin{array}{lcll} \mathsf{c} & = & \mathsf{enc}(\mathsf{c}, \mathsf{pk}_\mathsf{E}, \mathsf{r}) & \textit{(encryption)} \\ \mathsf{s} & = & \mathsf{sign}(\mathsf{s}, \mathsf{pk}_\mathsf{E}) & \textit{(signature)} \\ \mathsf{p} & = & \mathsf{pr}(\mathsf{c}, \mathsf{v}, \mathsf{r}, \mathsf{cr}) & \textit{(zero-knowledge proof)} \end{array}$$

for a chosen vote v, using a fresh randomness r. When the voter decides to cast it, the ballot is sent to the server after it authenticates the voter with a login operation. Then, the server randomises the ballot b with a fresh randomness r' or forwards it to the randomisation server, obtains the ballot $\mathsf{b}' = \langle \mathsf{c}', \mathsf{s}', \mathsf{p}' \rangle$, where

$$\begin{array}{lcll} \mathsf{c}' & = & \mathsf{renc}(\mathsf{c}, \mathsf{pk}_\mathsf{E}, \mathsf{r}') & \textit{(re-encryption)} \\ \mathsf{s}' & = & \mathsf{resign}(\mathsf{s}, \mathsf{pk}_\mathsf{E}, \mathsf{r}') & \textit{(adaptation of signature)} \\ \mathsf{p}' & = & \mathsf{repr}(\mathsf{p}, \mathsf{pk}_\mathsf{E}, \mathsf{r}', \mathsf{cr}) & \textit{(adaptation of proof)} \end{array}$$

and publishes it on BBcast. BeleniosRF does not allow revoting in order to protect against a corrupt server. Assume the voter casts two ballots b_1 and b_2 for different votes v_1 and v_2. The server should randomise the first ballot as b'_1 and the second ballot as b'_2. However, for the second ballot, instead of randomising b_2, a corrupt server can re-randomise the ballot b'_1 and obtains b''_1. The voter will not realise that the ballot b''_1 published on BBcast encodes v_1 instead of v_2. Due to this weakness, revoting is not allowed in BeleniosRF. Therefore, all the ballots recorded on BBcast are directly tallied in the tally phase.

The individual verification procedure for a voter with credential cr consists in verifying that there is a ballot b' associated to cr on $\mathsf{BBtally}$, and verifying that b' contains a valid signature with respect to cr. Intuitively, this should be sufficient to ensure election verifiability if either the registrar or the server is not corrupt. If the server is honest, it will log the identity of voters together with their public credentials. Thus, even if a voter's signing key is corrupt due to a corrupt registrar, the adversary should not be able to cast a ballot for the credential of an honest voter. Symmetrically, if the registrar is honest, the only party who can construct valid ballots for a public credential is the corresponding voter - verifiability follows even for a corrupt server since revoting is not allowed.

6.5.2 BeleniosRF Specification

In this section, we present the ProVerif specification of the BeleniosRF protocol, for which we consider the individual verification procedure $\mathcal{V}_3$ from Section 6.2, and the adversary models $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ from Table 6.1. The ProVerif specification of a protocol can be easily obtained from its Tamarin specification. Therefore, we only present the equational theory, the processes for ballot casting by a voting platform and then by the server, and the process for the individual verification in BeleniosRF, where it differs from Belenios. The full specification is given in Figure 6.7.

BeleniosRF uses the ElGamal encryption algorithm to encrypt the votes, the asymmetric Waters signature scheme to sign the encrypted votes and then adapt signatures for the randomised ciphertexts, and Groth-Shai proofs to prove the knowledge of encryption, associating the ciphertext with the public credential of the voter who generates the ballot. Thus, the specification requires the following equations:

(1) $\mathsf{dec}(\mathsf{enc}(\mathsf{x}, \mathsf{pk}(\mathsf{y}), \mathsf{z}), \mathsf{y}) = \mathsf{x}$,
(2) $\mathsf{verify}(\mathsf{sign}(\mathsf{x}, \mathsf{y}), \mathsf{x}, \mathsf{pk}(\mathsf{y})) = \mathsf{true}$,
(3) $\mathsf{ver}(\mathsf{pr}(\mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}), \mathsf{x}, \mathsf{z}, \mathsf{w}), \mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}), \mathsf{y}, \mathsf{w}) = \mathsf{true}$,
(4) $\mathsf{renc}(\mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}_1), \mathsf{y}, \mathsf{z}_2) = \mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}_2)$,
(5) $\mathsf{resign}(\mathsf{sign}(\mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}_1), \mathsf{w}), \mathsf{y}, \mathsf{z}_2) = \mathsf{sign}(\mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}_2), \mathsf{w})$,
(6) $\mathsf{repr}(\mathsf{pr}(\mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}_1), \mathsf{x}, \mathsf{z}_1, \mathsf{w}), \mathsf{y}, \mathsf{z}_2, \mathsf{w})) = \mathsf{pr}(\mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}_2), \mathsf{x}, \mathsf{z}_2, \mathsf{w})$,

where the equation (4) equates re-encryption of the ciphertext with an encryption so that it can be decrypted with the equation (1). Similarly, the equations (5) and (6) adapt the signature and the proof for the re-encryption so that they can be verified with the equations (2) and (3), respectively.

The figure Figure 6.1 presents three processes representing the action of the voting platform that generates a ballot for the voter's choice and casts it, the action of the server that validates the ballot, randomises it and records on BBcast, the action of the voter who performs the individual verification for their ballot on $\mathsf{BBtally}$. Specifically, the voting platform receives the voter's choice v, the election's public key pk_E, and the credentials of the voter id recorded in the tables Cred and Pwd. Then, it generates a fresh randomness r, uses it to encrypt the vote v, signs the encryption c, and generates a proof p for the knowledge of the encryption. The ballot in the form of $\mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p} \rangle$ is authenticated with the help of a hash function as we did for the Tamarin specification of Belenios and sent to the public channel. The table Voted records the information of the vote to be used for the individual verification.

The voting server receives a ballot for a voter id and a credential cr with an authentication message a from the public channel. It validates the voter id and their credential cr, i.e. checks its log file to ensure the consistency of the pair $(\mathsf{id}, \mathsf{cr})$, which is modelled using a restriction with respect to the event $\mathsf{Log}(\mathsf{id}, \mathsf{cr})$. It validates the ballot, i.e. verifies the authentication with respect to the pwd of the voter id, the credential on BBcast, the signature s and the proof p. Then, it randomises the ballot with a fresh randomness r' and obtains $\mathsf{b}' = \langle \mathsf{c}', \mathsf{s}', \mathsf{p}' \rangle$, which is recorded in both the event and the table BBcast for the credential cr.

For the individual verification, the voter recalls the table Voted and gets the ballot information recorded on $\mathsf{BBtally}$. Since the ballot is randomised, it is different than the one cast by the voter. Thus, the voter can only perform validity checks on the signature and the proof inside the ballot. The verification of both implies a successful verification of the voter. BeleniosRF does not allow revoting, and therefore one ballot for each credential is accepted by the server. The individual verification is performed on the ballot cast by the server in the tally phase. Note that since there is no revoting, we omit t from the event $\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{t})$ and simply specify $\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v})$.

```
let VotingPlatformVote(v, pk_E) =
  get Cred(id, cr, sk_id) in
  get Pwd(= id, pwd) in
  new r;
  let c = enc(v, pk_E, r) in
  let s = sign(c, sk_id) in
  let p = pr(c, v, r, cr) in
  let b = (c, s, p) in
  let a = h(id, pwd, cr, b) in
  event Vote(id, v);
  insert Voted(id, cr, v);
  out(pub, (id, cr, b, a)).

let VoterVer(id) =
  get Voted(= id, cr, v) in
  get BBkey(pk_E) in
  get BBtally(= cr, (c, s, p)) in
  if verify(s, c, cr) = true then
  if ver(p, c, pk_E, cr) = true then
  event Verified(id, cr, v).
```

```
let ServerCast(pk_E) =
  get BBcast(cr, = ⊥) in
  get Pwd(id, pwd) in
  in(pub, (= id, = cr, (c, s, p), a);
  event Log(id, cr);
  let b = (c, s, p) in
  if h(id, pwd, cr, b) = a then
  if verify(s, c, cr) = true then
  if ver(p, c, pk_E, cr) = true then
  new r';
  let c' = renc(c, pk_E, r') in
  let s' = resign(s, pk_E, r') in
  let p' = repr(p, pk_E, r', cr) in
  let b' = (c', s', p') in
  event BBcast(cr, b');
  insert BBcast(cr, b').
```

FIGURE 6.1: Processes specified for ballot casting and individual verification procedures in BeleniosRF.

Adversary model $\mathcal{A}_1$**.** In this model, we consider corrupt talliers that allow adversary to control the election's private key, corrupt voters who leak all their credentials to the adversary anytime during the election. We assume both the registrar and the server are honest. Therefore, the process modelling the talliers generating the election key pair receives the election's private key from the public channel, i.e. we replace the first line of the process of TallierKey with the line in(pub, sk_E). We add a process VoterCorr, presented in Figure 6.2, to specify the corruption of the voters which leaks the credentials recorded in the tables Cred and Pwd to the public channel.

Adversary model $\mathcal{A}_2$**.** For A_2, we assume the voting server is corrupt in addition to talliers and voters. Therefore, the corrupt server accepts any valid ballot coming from the network, without authenticating the sender. It still validates the ballots before recording them to BB since all the ballots on BB are validated by the election auditors. In $\mathcal{A}_2$, the process ServerCast is replaced with the process CorruptedServerCast described in Figure 6.2.

Adversary model $\mathcal{A}_3$**.** For $\mathcal{A}_3$, we consider a corrupt registrar in addition to talliers and voters. The corrupt registrar allows adversary to control signing keys of the voters so that it can register two or more voters with the same election credentials. At the same time, it allows adversary to generate valid ballots since it has all the signing key pairs. We model the corrupt registrar changing the first line of the process RegisterReg with the line in(pub, sk_{id}); as we did for corrupt talliers.

6.5.3 BeleniosRF Verification Results and Analysis

Belenios satisfies all the notions of end-to-end election verifiability, i.e. SE2E[iv_∇, $\mathsf{res}_\diamond$] for $\nabla, \diamond \in \{\circ, \bullet\}$, when revoting is not allowed or the individual verification is allowed in the tally phase. Similar to Belenios, BeleniosRF is also expected to give the same verifiability

```
let CorruptedServerCast(pk_E) =
  get BBcast(cr, = ⊥) in
  in(pub, (= cr, (c, s, p));
  let b = (c, s, p) in
  if verify(s, c, cr) = true then
  if ver(p, c, pk_E, cr) = true then
  event BBcast(cr, b);
  insert BBcast(cr, b).

let VoterCorr =
  get Cred(id, cr, sk_id) in
  get Pwd(= id, pwd) in
  event Corr(id);
  out(pub, (id, pwd, cr, sk_id)).
```

FIGURE 6.2: Corruption of voters and voting server in BeleniosRF.

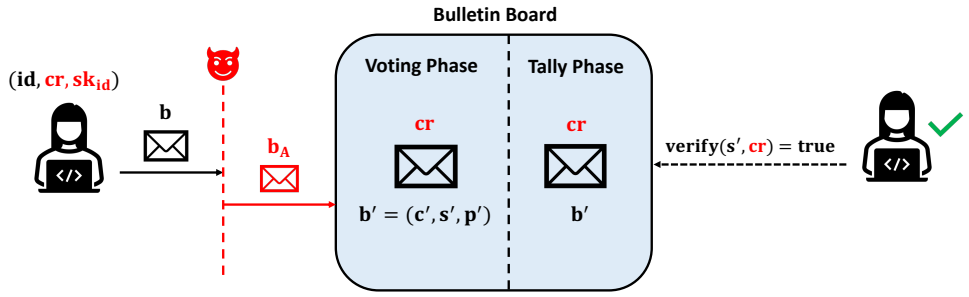


FIGURE 6.3: Attack on individual verifiability by a corrupt registrar in BeleniosRF.

guarantees with $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$, i.e. when both the registrar and server are not corrupt at the same time. However, our verification results show that BeleniosRF is vulnerable to attacks for honest voters with $\mathcal{A}_3$ and for corrupt voters with $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$. The verification results for both Belenios and BeleniosRF obtained with ProVerif are presented in Table 6.5, for which each corruption scenarios the execution of the codes ends within several seconds.

Attacks. We find an attack against individual verifiability in the case of a corrupt registrar, i.e. the scenario $\mathcal{A}_3$. The weakness comes from the fact that individual verification does not directly check the ballot cast by voter, but its randomisation. The properties $\Phi^{\nabla}_{\text{iv1}}$ and $\Phi^{\nabla}_{\text{iv2}}$ in $\mathsf{SE2E}[\mathsf{iv}_{\nabla}, \mathsf{res}_{\diamond}]$ are violated for $\nabla, \diamond \in \{\circ, \bullet\}$, because of ballot stuffing and clash attacks as illustrated in Figure 6.3 and Figure 6.4, respectively. Ballot stuffing occurs for an honest voter when the adversary drops the voter's ballot, generates another ballot for its desired vote using the signing credentials of the voter and casts it using the login credential of a corrupt voter. The honest voter verifies their vote upon seeing a ballot with their signature. However, the ballot on BB encodes the adversary's vote instead of the voter's vote. Therefore, $\Phi^{\circ}_{\text{iv1}}$ is violated. Clash attack occurs when a corrupt registrar gives the same signing key pair $(\mathsf{cr}, \mathsf{sk}_{\mathsf{id}})$ to two honest voters id_1 and id_2. Assume the two voters cast their ballots separately relying on their credential cr. A ballot b from one of the voters is accepted, randomised, and published on BB. For the second voter, the adversary drops the ballot. In the tally phase, both voters can verify that, for the tuple $(\mathsf{cr}, \mathsf{b}')$ on BB, b' is the randomisation of a ballot submitted with their credential, verifying the attached signature and the proof since they share the same credentials. However, only one vote will be counted for these two voters.

A similar individual verifiability attack is possible in the case of a corrupt voter, even when both the registrar and server are honest as in $\mathcal{A}_1$. Assume the voter has leaked its credentials unwittingly and become corrupt. The adversary corrupting the communication network drops the voter's ballot and cast another ballot for a different vote. In this case, the voter can perform a successful verification by verifying the signature inside the ballot.

Adversary Models		$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$
Talliers		C	C	C
Registrar		H	H	C
Server		H	C	H
Voting Platform		H	H	H
Belenios*	SE2E[$\mathsf{iv}_\bullet, \mathsf{res}_\bullet$]	✓	✓	✓
	SE2E[$\mathsf{iv}_\circ, \mathsf{res}_\bullet$]	✓	✓	✓
	SE2E[$\mathsf{iv}_\bullet, \mathsf{res}_\circ$]	✓	✓	✓
	SE2E[$\mathsf{iv}_\circ, \mathsf{res}_\circ$]	✓	✓	✓
BeleniosRF†	SE2E[$\mathsf{iv}_\bullet, \mathsf{res}_\bullet$]	✗	✗	✗
	SE2E[$\mathsf{iv}_\circ, \mathsf{res}_\bullet$]	✓	✓	✗
	SE2E[$\mathsf{iv}_\bullet, \mathsf{res}_\circ$]	✗	✗	✗
	SE2E[$\mathsf{iv}_\circ, \mathsf{res}_\circ$]	✓	✓	✗

$*$: no revoting or voter verification in the tally phase
$\dagger$: no revoting

TABLE 6.5: Verification results for Belenios and BeleniosRF.

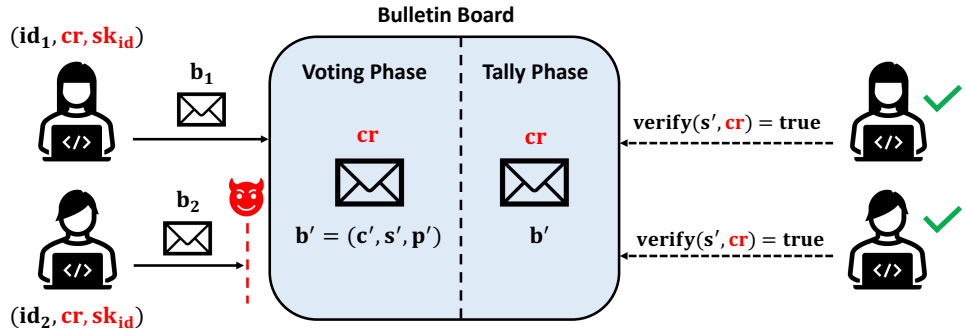


FIGURE 6.4: Clash attack by a corrupt registrar in BeleniosRF.

However, the ballot belongs to the adversary, and the corresponding vote of the adversary will be tallied for that voter. As a result, $\Phi^{\bullet}_{\mathsf{iv1}}$ is violated in the scenario $\mathcal{A}_1$ as well as $\mathcal{A}_2$ and $\mathcal{A}_3$. In Belenios, the same scenario does not lead to an attack, but in BeleniosRF it does.

Related attacks. BeleniosVS [17] is a variant of Belenios that uses the same ballot randomisation feature as BeleniosRF, in order to achieve better privacy in the case of a dishonest voting platform. In consequence, a similar clash attack as ours applies to BeleniosVS in the case of a corrupt registrar, and the authors of [17] have already observed this. As a fix, they propose that the voting server sends an acknowledgement to the voter when the ballot is successfully cast. In an attack scenario similar to the one above (from Figure 6.4), this should counter the dropping of the ballot b_2 by the network, since the voter id_2 would notice that no acknowledgement was sent. A similar fix could also help in BeleniosRF if the channel between the voting platform and the server that forwards the ballot and the acknowledgement, is assumed to be secure. Otherwise, the adversary corrupting the channel, could also impersonate the voting server and spoof the acknowledgement.

Proofs. BeleniosRF satisfies election verifiability for honest voters who verify their ballots, i.e. SE2E[$\mathsf{iv}_\diamond, \mathsf{res}_\diamond$] for $\diamond \in \{\circ, \bullet\}$, when the registrar is honest. In this case, the talliers and the voting server can be fully corrupt as in the scenarios $\mathcal{A}_1$ and $\mathcal{A}_2$.

Conclusion. Our findings show that BeleniosRF, the receipt-free variant of Belenios, provides weaker verifiability guarantees than Belenios. There is indeed a tension between receipt-freeness and verifiability of Belenios. The ways how to resolve it is an open question. Selene [47] is another voting protocol that aims to provide both receipt-freeness and end-to-end verifiability, applying different methods than the randomisation of the ballots for receipt-freeness. This motivates us to evaluate the tension between receipt-freeness and verifiability in a different protocol and see whether both receipt-freeness and verifiability can be achieved or in which conditions.

SETUP PHASE

R^{T}_{key} : **generate election key pair**

$[\ \mathsf{Fr}(\mathsf{sk}_E)\ ] \dashv [\ \mathsf{BBkey}(\mathsf{pk}(\mathsf{sk}_E))\ ] \mapsto [\ \mathsf{SkE}(\mathsf{sk}_E), \mathsf{BBkey}(\mathsf{pk}(\mathsf{sk}_E)), \mathsf{Out}(\mathsf{pk}(\mathsf{sk}_E))\ ]$

R^{A}_{cand} : **determine candidates to be elected**

$\mathsf{let}\ \ \mathsf{vlist} = \langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle\ \ \mathsf{in}$

$[\ \mathsf{In}(\mathsf{vlist})\ ] \dashv [\ \mathsf{BBcand}(\mathsf{v}_1), \ldots, \mathsf{BBcand}(\mathsf{v}_k), \mathsf{Vlist}(\mathsf{vlist})\ ] \mapsto$
$[\ \mathsf{BBcand}(\mathsf{v}_1), \ldots, \mathsf{BBcand}(\mathsf{v}_k), \mathsf{Vlist}(\mathsf{vlist})\ ]$

R^{A}_{id} : **determine identities eligible to vote**

$[\ \mathsf{In}(\mathsf{id})\ ] \dashv [\ ] \mapsto [\ \mathsf{Id}(\mathsf{id})\ ]$

R^{VR}_{reg} : **register voter with a signing key pair**

$\mathsf{let}\ \ \mathsf{cr} = \mathsf{pk}(\mathsf{sk}_{id})\ \ \mathsf{in}$

$[\ \mathsf{Id}(\mathsf{id}), \mathsf{Fr}(\mathsf{sk}_{id})\ ] \dashv [\ \mathsf{Reg}(\mathsf{id}, \mathsf{cr}), \mathsf{BBreg}(\mathsf{cr})\ ] \mapsto [\ \mathsf{Cred}(\mathsf{id}, \mathsf{cr}, \mathsf{sk}_{id}), \mathsf{BBreg}(\mathsf{cr}), \mathsf{Out}(\mathsf{cr})\ ]$

R^{VS}_{pwd} : **generate password for voter authentication**

$[\ \mathsf{Id}(\mathsf{id}), \mathsf{Fr}(\mathsf{pwd})\ ] \dashv [\ ] \mapsto [\ \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd})\ ]$

R^{VS}_{bb} : **setup initial** BBcast **for registered voters**

$[\ \mathsf{BBreg}(\mathsf{cr})\ ] \dashv [\ \mathsf{BBcast}(\mathsf{cr}, \bot)\ ] \mapsto [\ \mathsf{BBcast}(\mathsf{cr}, \bot)\ ]$

VOTING PHASE

R^{VP}_{vote} : **construct a ballot, authenticate and send it to** VS

$\mathsf{let}\ \ \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk}_E, \mathsf{r});\ \ \mathsf{s} = \mathsf{sign}(\mathsf{c}, \mathsf{sk}_{id});\ \ \mathsf{p}_R = \mathsf{pr}_R(\mathsf{c}, \mathsf{r}, \mathsf{vlist});\ \ \mathsf{p}_L = \mathsf{pr}_L(\mathsf{c}, \mathsf{r}, \mathsf{cr});$
$\mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p}_R, \mathsf{p}_L \rangle;\ \ \mathsf{a} = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{b} \rangle)\ \ \mathsf{in}$

$[\ \mathsf{Cred}(\mathsf{id}, \mathsf{cr}, \mathsf{sk}_{id}), \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd}), \mathsf{BBcand}(\mathsf{v}), \mathsf{Vlist}(\mathsf{vlist}), \mathsf{BBkey}(\mathsf{pk}_E), \mathsf{Fr}(\mathsf{r}), \mathsf{Fr}(\mathsf{t})\ ]$
$\dashv [\ \mathsf{Vote}(\mathsf{id}, \mathsf{v}), \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b}), \mathsf{VoteTime}(\mathsf{id}, \mathsf{v}, \mathsf{t})\ ] \mapsto [\ \mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{b}, \mathsf{t}), \mathsf{Out}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{b}, \mathsf{a} \rangle)\ ]$

R^{VS}_{cast} : **authenticate voter, verify and publish ballot**

$\mathsf{let}\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p}_R, \mathsf{p}_L \rangle;\ \ \mathsf{a}' = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{b} \rangle)\ \ \mathsf{in}$

$[\ \mathsf{In}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{b}, \mathsf{a} \rangle), \mathsf{BBreg}(\mathsf{cr}), \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd}), \mathsf{BBkey}(\mathsf{pk}_E), \mathsf{Vlist}(\mathsf{vlist})\ ]$
$\dashv [\ \mathsf{a}' \equiv \mathsf{a}, \mathsf{verify}(\mathsf{s}, \mathsf{c}, \mathsf{cr}) \equiv \mathsf{true}, \mathsf{ver}_R(\mathsf{p}_R, \mathsf{c}, \mathsf{pk}_E, \mathsf{vlist}) \equiv \mathsf{true}, \mathsf{ver}_L(\mathsf{p}_L, \mathsf{c}, \mathsf{cr}) \equiv \mathsf{true},$
$\mathsf{Log}(\mathsf{id}, \mathsf{cr}), \mathsf{Reg}(\mathsf{id}, \mathsf{cr}), \mathsf{VScast}(\mathsf{id}, \mathsf{b}), \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ] \mapsto [\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]$

Ψ^{VS}_{order} : **ensure correct order of ballots**

$\mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ @i\ \wedge\ \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b}')\ @j\ \wedge$
$\mathsf{VScast}(\mathsf{id}, \mathsf{b})\ @k\ \wedge\ \mathsf{VScast}(\mathsf{id}, \mathsf{b}')\ @l\ \wedge\ i \prec j \Rightarrow k \prec l$

Ψ^{VS}_{log} : **ensure log consistency; can be audited by** EA

$\mathsf{Log}(\mathsf{id}, \mathsf{cr}) \Rightarrow \neg(\mathsf{Log}(\mathsf{id}, \mathsf{cr}') \wedge \mathsf{cr} \neq \mathsf{cr}') \wedge \neg(\mathsf{Log}(\mathsf{id}', \mathsf{cr}) \wedge \mathsf{id} \neq \mathsf{id}')$

$\Psi^{VS/EA}_{cast}$: **ensure ballot validity; can be audited by** EA

$\mathsf{BBcast}(\mathsf{cr}, \mathsf{b}) \Rightarrow \mathsf{BBreg}(\mathsf{cr})\ \wedge\ (\ \mathsf{b} \neq \bot \Rightarrow \mathsf{BBkey}(\mathsf{pk}_E) \wedge \mathsf{Vlist}(\mathsf{vlist})\ \wedge\ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p}_R, \mathsf{p}_L \rangle$
$\wedge\ \mathsf{verify}(\mathsf{s}, \mathsf{c}, \mathsf{cr}) = \mathsf{true}\ \wedge\ \mathsf{ver}_R(\mathsf{p}_R, \mathsf{c}, \mathsf{pk}_E, \mathsf{vlist}) = \mathsf{true}\ \wedge\ \mathsf{ver}_L(\mathsf{p}_L, \mathsf{c}, \mathsf{cr}) = \mathsf{true}\ )$

TALLY PHASE

R^{VS}_{tally} : **select ballot to be tallied for a public credential**

$[\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ] \dashv [\ \mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ ] \mapsto [\ \mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ ]$

$\Psi^{VS/EA}_{tally}$: **ensure the last ballot cast is tallied; can be audited by** EA

$\mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ @i\ \wedge\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b}')\ @j\ \wedge\ \mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ @k \Rightarrow j \prec i\ \vee\ \mathsf{b} = \mathsf{b}'$

FIGURE 6.5: Tamarin specification of Belenios.

SETUP PHASE

R^{T}_{key} : **generate election key pair**
$[\ \mathsf{Fr}(\mathsf{sk_E})\ ] \text{—}[!\mathsf{BBkey}(\mathsf{pk}(\mathsf{sk_E}))\ ] \mapsto [\ !\mathsf{SkE}(\mathsf{sk_E}), !\mathsf{BBkey}(\mathsf{pk}(\mathsf{sk_E})), \mathsf{Out}(\mathsf{pk}(\mathsf{sk_E}))\ ]$

R^{A}_{cand} : **determine candidates to be elected**
$\mathsf{let}\ \ \mathsf{vlist} = \langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle\ \ \mathsf{in}$
$[\ \mathsf{In}(\mathsf{vlist})\ ] \text{—}[!\mathsf{BBcand}(\mathsf{v}_1), \ldots, !\mathsf{BBcand}(\mathsf{v}_k), !\mathsf{Vlist}(\mathsf{vlist})\ ] \mapsto$
$[\ !\mathsf{BBcand}(\mathsf{v}_1), \ldots, !\mathsf{BBcand}(\mathsf{v}_k), !\mathsf{Vlist}(\mathsf{vlist})\ ]$

R^{A}_{id} : **determine identities eligible to vote**
$[\ \mathsf{In}(\mathsf{id})\ ] \text{—}[\ ] \mapsto [\ !\mathsf{Id}(\mathsf{id})\ ]$

R^{VR}_{reg} : **register voter with a signing key pair**
$\mathsf{let}\ \ \mathsf{cr} = \mathsf{pk}(\mathsf{sk_{id}})\ \ \mathsf{in}$
$[\ !\mathsf{Id}(\mathsf{id}), \mathsf{Fr}(\mathsf{sk_{id}})\ ] \text{—}[\ \mathsf{Reg}(\mathsf{id}, \mathsf{cr}), !\mathsf{BBreg}(\mathsf{cr})\ ] \mapsto [\ !\mathsf{Cred}(\mathsf{id}, \mathsf{cr}, \mathsf{sk_{id}}), !\mathsf{BBreg}(\mathsf{cr}), \mathsf{Out}(\mathsf{cr})\ ]$

R^{VS}_{pwd} : **generate password for voter authentication**
$[\ !\mathsf{Id}(\mathsf{id}), \mathsf{Fr}(\mathsf{pwd})\ ] \text{—}[\ ] \mapsto [\ !\mathsf{Pwd}(\mathsf{id}, \mathsf{pwd})\ ]$

R^{VS}_{bb} : **setup initial** BBcast **for registered voters**
$[\ !\mathsf{BBreg}(\mathsf{cr})\ ] \text{—}[!\mathsf{BBcast}(\mathsf{cr}, \bot)\ ] \mapsto [\ !\mathsf{BBcast}(\mathsf{cr}, \bot), \mathsf{BBlast}(\mathsf{cr}, \bot), \mathsf{VPlast}(\mathsf{cr}, \bot)\ ]$

VOTING PHASE

R^{VP}_{vote} : **construct a ballot, authenticate and send it to** VS
$\mathsf{let}\ \ \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk_E}, \mathsf{r});\ \ \mathsf{s} = \mathsf{sign}(\mathsf{c}, \mathsf{sk_{id}});\ \ \ell = \langle \mathsf{h}(\mathsf{cr}, \mathsf{b}_0), \mathsf{com}(\mathsf{id}, \mathsf{u}) \rangle;\ \ \mathsf{p_R} = \mathsf{pr_R}(\mathsf{c}, \mathsf{r}, \mathsf{vlist});$
$\mathsf{p_L} = \mathsf{pr_L}(\mathsf{c}, \mathsf{r}, \ell);\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p_R}, \mathsf{p_L}, \ell \rangle;\ \ \mathsf{a} = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{b}, \mathsf{u} \rangle)\ \ \mathsf{in}$
$[\ !\mathsf{Cred}(\mathsf{id}, \mathsf{cr}, \mathsf{sk_{id}}), !\mathsf{Pwd}(\mathsf{id}, \mathsf{pwd}), !\mathsf{BBcand}(\mathsf{v}), !\mathsf{Vlist}(\mathsf{vlist}), !\mathsf{BBkey}(\mathsf{pk_E}),$
$\mathsf{Fr}(\mathsf{r}), \mathsf{Fr}(\mathsf{t}), \mathsf{Fr}(\mathsf{u}), \mathsf{VPlast}(\mathsf{cr}, \mathsf{b}_0)\ ] \text{—}[\ \mathsf{Vote}(\mathsf{id}, \mathsf{v}), \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b}), \mathsf{VoteTime}(\mathsf{id}, \mathsf{v}, \mathsf{t})\ ] \mapsto$
$[\ !\mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{b}, \mathsf{t}), \mathsf{Out}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{b}, \mathsf{a}, \mathsf{u} \rangle)\ ]$

R^{VS}_{cast} : **authenticate voter, verify and publish ballot**
$\mathsf{let}\ \ \ell' = \langle \mathsf{h}(\mathsf{cr}, \mathsf{b}_0), \mathsf{com}(\mathsf{id}, \mathsf{u}) \rangle;\ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p_R}, \mathsf{p_L}, \ell \rangle;\ \ \mathsf{a}' = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{b}, \mathsf{u} \rangle)\ \ \mathsf{in}$
$[\ \mathsf{In}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{b}, \mathsf{a}, \mathsf{u} \rangle), !\mathsf{BBreg}(\mathsf{cr}), !\mathsf{Pwd}(\mathsf{id}, \mathsf{pwd}), !\mathsf{BBkey}(\mathsf{pk_E}), !\mathsf{Vlist}(\mathsf{vlist}), \mathsf{BBlast}(\mathsf{cr}, \mathsf{b}_0)\ ]$
$\text{—}[\ \mathsf{a}' \equiv \mathsf{a}, \ell' \equiv \ell, \mathsf{verify}(\mathsf{s}, \mathsf{c}, \mathsf{cr}) \equiv \mathsf{true}, \mathsf{ver_R}(\mathsf{p_R}, \mathsf{c}, \mathsf{pk_E}, \mathsf{vlist}) \equiv \mathsf{true}, \mathsf{ver_L}(\mathsf{p_L}, \mathsf{c}, \ell) \equiv \mathsf{true},$
$\mathsf{Log}(\mathsf{id}, \mathsf{cr}), \mathsf{Reg}(\mathsf{id}, \mathsf{cr}), \mathsf{VScast}(\mathsf{id}, \mathsf{b}), !\mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ] \mapsto$
$[\ !\mathsf{BBcast}(\mathsf{cr}, \mathsf{b}), \mathsf{BBlast}(\mathsf{cr}, \mathsf{b}), \mathsf{VPlast}(\mathsf{cr}, \mathsf{b})\ ]$

Ψ^{VS}_{order} : **ensure correct order of ballots**
$\mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ @i\ \wedge\ \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b}')\ @j\ \wedge$
$\mathsf{VScast}(\mathsf{id}, \mathsf{b})\ @k\ \wedge\ \mathsf{VScast}(\mathsf{id}, \mathsf{b}')\ @l\ \wedge\ i \prec j \Rightarrow k \prec l$

Ψ^{VS}_{log} : **ensure log consistency; can be audited by** EA
$\mathsf{Log}(\mathsf{id}, \mathsf{cr}) \Rightarrow \neg(\mathsf{Log}(\mathsf{id}, \mathsf{cr}')\ \wedge\ \mathsf{cr} \neq \mathsf{cr}')\ \wedge\ \neg(\mathsf{Log}(\mathsf{id}', \mathsf{cr})\ \wedge\ \mathsf{id} \neq \mathsf{id}')$

$\Psi^{VS/EA}_{cast}$: **ensure ballot validity; can be audited by** EA
$!\mathsf{BBcast}(\mathsf{cr}, \mathsf{b}) \Rightarrow !\mathsf{BBreg}(\mathsf{cr})\ \wedge\ (\mathsf{b} \neq \bot \Rightarrow !\mathsf{BBkey}(\mathsf{pk_E}) \wedge !\mathsf{Vlist}(\mathsf{vlist})\ \wedge\ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p_R}, \mathsf{p_L}, \ell \rangle$
$\wedge\ \mathsf{verify}(\mathsf{s}, \mathsf{c}, \mathsf{cr}) = \mathsf{true}\ \wedge\ \mathsf{ver_R}(\mathsf{p_R}, \mathsf{c}, \mathsf{pk_E}, \mathsf{vlist}) = \mathsf{true}\ \wedge\ \mathsf{ver_L}(\mathsf{p_L}, \mathsf{c}, \ell) = \mathsf{true})$

TALLY PHASE

R^{VS}_{tally} : **select ballot to be tallied for a public credential**
$[\ !\mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ] \text{—}[!\mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ ] \mapsto [\ !\mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ ]$

$\Psi^{VS/EA}_{tally}$: **ensure the last ballot cast is tallied; can be audited by** EA
$!\mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ @i \wedge !\mathsf{BBcast}(\mathsf{cr}, \mathsf{b}')\ @j \wedge !\mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ @k \Rightarrow j \prec i\ \vee\ \mathsf{b} = \mathsf{b}'$

FIGURE 6.6: Tamarin specification of Belenios+.

```
let TallierKey =
  new sk_E;
  let pk_E = pk(sk_E) in
  insert SkE(sk_E);
  event BBkey(pk_E);
  insert BBkey(pk_E);
  out(pub, pk_E).

let RegistrarReg(id) =
  new sk_id;
  let cr = pk(sk_id) in
  insert Cred(id, cr, sk_id);
  event Reg(id, cr);
  event BBreg(cr);
  insert BBreg(cr);
  out(pub, cr).

let VotingPlatformVote(v, pk_E) =
  get Cred(id, cr, sk_id) in
  get Pwd(= id, pwd) in
  new r;
  let c = enc(v, pk_E, r) in
  let s = sign(c, sk_id) in
  let p = pr(c, v, r, cr) in
  let b = (c, s, p) in
  let a = h(id, pwd, cr, b) in
  event Vote(id, v);
  insert Voted(id, cr, v);
  out(pub, (id, cr, b, a)).

let VoterVer(id) =
  get Voted(= id, cr, v) in
  get BBkey(pk_E) in
  get BBtally(= cr, (c, s, p)) in
  if verify(s, c, cr) = true then
  if ver(p, c, pk_E, cr) = true then
  event Verified(id, cr, v).

let AdminCand =
  in(pub, (v_1, v_2));
  event BBcand(v_1);
  event BBcand(v_2);
  insert BBcand(v_1);
  insert BBcand(v_2).

let AdminId =
  in(pub, id);
  insert Id(id).

let ServerPwd(id) =
  new pwd;
  insert Pwd(id, pwd).

let ServerBB =
  get BBreg(cr) in
  insert BBcast(cr, ⊥).

let ServerCast(pk_E) =
  get BBcast(cr, = ⊥) in
  get Pwd(id, pwd) in
  in(pub, (= id, = cr, (c, s, p), a));
  event Log(id, cr);
  let b = (c, s, p) in
  if h(id, pwd, cr, b) = a then
  if verify(s, c, cr) = true then
  if ver(p, c, pk_E, cr) = true then
  new r';
  let c' = renc(c, pk_E, r') in
  let s' = resign(s, pk_E, r') in
  let p' = repr(p, pk_E, r', cr) in
  let b' = (c', s', p') in
  event BBcast(cr, b');
  insert BBcast(cr, b').

let ServerTally =
  get BBcast(cr, b) in
  event BBtally(cr, b);
  insert BBtally(cr, b).
```

FIGURE 6.7: ProVerif specification of BeleniosRF.

Chapter 7

Selene and Its Variants

Selene [47] is a recently proposed e-voting protocol that aims to provide end-to-end verifiability, simplifying the individual verification procedure for voters so that they verify their votes directly in the election result. This is achieved by providing each voter with a tracker and publishing the result as tracker-vote pairs on the bulletin board. Thus, voters need to find their tracker and match their vote for individual verification instead of matching their complex cryptographic ballot with the one on the bulletin board. Selene gives assurance that no two voters receive the same tracker due to the cryptographic primitives used by the protocol. The idea is that the trackers are held in encrypted and committed form next to the public credentials of the voters. Using private trapdoor keys associated with each voter and randomness obtained from talliers, voters can open the commitment and obtain their trackers. The randomness is sent to the respective voter after the election result is published so that the voter can generate a fake randomness in order to resist coercion. The fake randomness allows the voter to obtain a tracker associated with the vote the coercer desires. Thus, Selene also provides receipt-freeness.

Three analyses [36, 13, 26] have been performed for the privacy of Selene. In [36], it was aimed to discover the limits of the protection of Selene against coercion, i.e. the exact limits of the coercer's ability with an analysis based on multi-agent logic. Thus, a simple multi-agent model of Selene was checked with respect to the formulas obtained from Alternating-time Temporal Logic (ATL) representing coercion. In [13], a symbolic analysis was performed for the ballot privacy and receipt-freeness of Selene. The symbolic models with respect to those properties were checked with Tamarin, and the first automated proofs of Selene were provided. Moreover, the findings of the analysis [13] confirmed that Selene is receipt-free unless the coercer is also a voter, and the fake randomness provided by the voter opens the commitment to the coercer's tracker, as also mentioned in [47]. The ballot privacy definition was refined in [26] against a corrupt voting server manipulating the bulletin board. The computational model of Selene was checked with EasyCrypt, and a machine-checked proof of ballot privacy was provided. On the other hand, there is no proof of verifiability for Selene.

To improve the tracker management in Selene and prevent such tracker collision with the coercer, the Hyperion e-voting protocol was proposed in [46] as a variant of Selene. In Hyperion, trackers are directly computed from the public trapdoor keys of the voters. They are randomised through exponentiation operations, where the exponentiation randomness is kept secret by talliers and revealed at the end of the election. Tracker collision attacks are defeated by the individual views of the bulletin board, i.e. each voter has their own view of the bulletin board, where all the trackers in the outcome are re-randomised with additional randomness generated for the respective voter. Thus, the voter is able to retrieve their tracker with the information provided by talliers, verify their vote, and fake the tracker if coerced. However, the coercer will not find their tracker from the voter's view of the bulletin board.

Selene and Hyperion aim to provide both end-to-end-verifiability and receipt-freeness. In this chapter, we perform a similar verifiability analysis of those protocols, as we did for BeleniosRF, in order to evaluate the tension between receipt-freeness and verifiability in different

protocols. Selene's receipt-freeness is weaker than BeleniosRF since the voters can provide a receipt, i.e. their encryption randomness, to the coercer, which allows them to verify the ballot published on the bulletin board before the tally. In order to remove such weakness, Selene can be modified to apply the same randomisation mechanism deployed by BeleniosRF, as mentioned in [47]. We also perform a verifiability analysis for this variant, i.e. the so-called SeleneRF. For the analysis, we consider the same adversary models for the three protocols and assume any party is corrupt except the administrator. Moreover, we extend the adversary's abilities to corrupt voting platforms so that the adversary determines the whole ballot cast by the voter, not only the encryption randomness used for the encryption of the vote. Our findings show that Selene and its variants satisfy end-to-end verifiability for honest voters who verified their votes, even if the talliers and voting platforms are fully corrupt. From a foundational perspective, the novelty of this chapter consists in using a non-trivial linking event $\mathsf{Link}(\mathsf{cr}, \mathsf{tr})$, where $\mathsf{cr} \neq \mathsf{tr}$.

Structure of the chapter: This chapter includes three sections, where the first is dedicated to Selene and SeleneRF, the second is to Hyperion, and the third is to the verifiability analysis of those protocols. Specifically, in Section 7.1 and Section 7.2, we present the protocol structures and the ProVerif specifications of Selene, SeleneRF, and Hyperion. Then, in Section 7.3, we provide our verification results and verifiability analysis for those protocols.

7.1 Selene

In this section, we present the protocol details of Selene. First, we describe the protocol structure of Selene, introducing its protocol parties with their roles, the cryptographic primitives used, and the election procedures followed by the parties according to the setup, voting, and tally phases, the individual verification procedure followed by voters, and the procedure that allows them to resist coercion. Then, we provide its ProVerif specification details with respect to seven adversary models in which any party can be corrupt other than the administrator. We also consider a stronger adversary for modelling a corrupt voting platform that allows the adversary to determine the voter's ballot, not only the encryption randomness, as considered in the cases of Helios and Belenios. Finally, we describe the features of SeleneRF and give its specification details that are different from Selene.

7.1.1 Selene Protocol Structure

Selene has the following parties along with their roles:

- Administrator A is responsible for the election configuration, i.e. determines the candidates and voters eligible for the election, talliers to generate the election key pair, and trackers to be assigned to the voters.
- Talliers T generate the election public key in a distributed way, where no tallier knows the private part of the key generated by another tallier (unless they are not both corrupt). Then, at least the threshold number of talliers, say k out of t, decrypt the set of ciphertexts corresponding to the ballots cast by the voters. The decryption procedure relies on mixnets, i.e. re-encryption, shuffling, and decryption of the ciphertexts. Talliers also contribute to the generation of encrypted trackers and trapdoor commitments to the encrypted trackers. Trapdoor commitments ensure voters about their assigned trackers.
- Registrar VR registers the public keys of the voters as public credentials for the election. VR may also generate a signing key pair for each registered voter according to the

deployment of the protocol. The authors in [47] do not explicitly mention a registrar for registering the public keys. They assume that voters are registered with their public keys before the start of the election. We consider that this registration is performed relying on a registrar. The case when voters register themselves is equivalent to the case of an honest registrar (if they use an honest device) or a corrupt registrar (if they use a corrupt one).

– Voting server VS generates a login credential, i.e. a password, for each registered voter, accepts ballots from the voters if they are authenticated with the login credential, and then publishes them on the bulletin board.

– Voting platform VP allows voters to generate a ballot as a triple, i.e. an encryption of their vote with the election public key, a signature for the ciphertext, and a zero-knowledge proof of knowledge of the vote. VP, then, allows voters to cast their ballot on the voting server via a login operation.

– Voters V who are registered for the election generate a trapdoor key pair: a private key and its corresponding public key. The public keys are used to generate the trapdoor commitments by talliers, and then the private keys allow voters to open the commitments after the election. The voters may cast several ballots during the election using their voting platform or abstain from voting.

Selene deploys an append-only secure public bulletin board, denoted by BB, that displays the public election data: the election's public key, the eligible candidates, the public credentials of registered voters, the encrypted trackers, the trapdoor commitments, the cast ballots, the votes as the election result (recorded next to the trackers), and the produced zero-knowledge proofs for the correctness of the operations done by the election authorities. BB allows voters to verify their votes at the end of the election and anybody to verify the data published on BB, e.g. the correct decryption of the ciphertexts. The data on BB is displayed in portions; for example, the election's public key is recorded and displayed in the portion of BBkey.

Cryptographic primitives: Selene relies on ElGamal encryption algorithm, which is also used for re-encryption, a digital signature scheme, Pedersen trapdoor commitments, and non-interactive zero-knowledge proofs.

- *Encryption:* A message m is encrypted with the public key pk and fresh randomness r, and a ciphertext $\mathsf{c} = \mathsf{enc}(\mathsf{m}, \mathsf{pk}, \mathsf{r})$ is obtained.

- *Re-encryption:* The ciphertext c is re-encrypted with the public key pk and fresh randomness r', and a ciphertext $\mathsf{c}' = \mathsf{renc}(\mathsf{enc}(\mathsf{m}, \mathsf{pk}, \mathsf{r}), \mathsf{pk}, \mathsf{r}')$ is obtained.

- *Decryption:* Any ciphertext c that is encryption with the public key pk is decrypted with the corresponding private key sk and the message m is revealed, i.e.

$$\mathsf{m} = \mathsf{dec}(\mathsf{enc}(\mathsf{m}, \mathsf{pk}, \mathsf{r}), \mathsf{sk}).$$

- *Signing:* A message m is signed with a signing key sk and the signature $\mathsf{s} = \mathsf{sign}(\mathsf{m}, \mathsf{sk})$ is obtained, which is verified with the corresponding public key pk as follows:

$$\mathsf{verify}(\mathsf{sign}(\mathsf{m}, \mathsf{sk}), \mathsf{pk}) = \mathsf{true}.$$

- *Trapdoor commitment:* A trapdoor commitment cm is made to a message m with the public trapdoor key tpk of the party that will open the commitment, and the commitment

randomness r, i.e.

$$\mathsf{cm} = \mathsf{com}(\mathsf{m}, \mathsf{tpk}, \mathsf{r}).$$

Then, the party that will open the commitment cm receives the commitment randomness r from the respective party, and reveals the message using their private trapdoor key tsk as follows:

$$\mathsf{m} = \mathsf{open}(\mathsf{com}(\mathsf{m}, \mathsf{tpk}, \mathsf{r}), \mathsf{tsk}, \mathsf{r}).$$

- *Interaction between encryption and trapdoor commitment:* The fact that Selene uses the ElGamal encryption and Pedersen commitment allows the following computation of a commitment to tracker:

  1. Assume $\mathsf{enc}(\mathsf{tr}, \mathsf{pk}, \mathsf{r}_3)$ is an encryption of tr, and $\mathsf{enc}(\mathsf{tpk}^{\mathsf{r}}, \mathsf{pk}, \mathsf{r}_1)$ is an encryption of tpk raised to some randomness r. Then, we can obtain an encryption of the commitment as follows:

  $$\mathsf{enc}(\mathsf{tpk}^{\mathsf{r}}, \mathsf{pk}, \mathsf{r}_1) \cdot \mathsf{enc}(\mathsf{tr}, \mathsf{pk}, \mathsf{r}_3) = \mathsf{enc}(\mathsf{com}(\mathsf{tr}, \mathsf{tpk}, \mathsf{r}), \mathsf{pk}, \mathsf{r}_1 + \mathsf{r}_3).$$

  2. The encryption of the commitment is decrypted with the private key sk, and the commitment cm is revealed:

  $$\mathsf{cm} = \mathsf{dec}(\mathsf{enc}(\mathsf{com}(\mathsf{tr}, \mathsf{tpk}, \mathsf{r}), \mathsf{pk}, \mathsf{r}_1 + \mathsf{r}_3), \mathsf{sk}) = \mathsf{com}(\mathsf{tr}, \mathsf{tpk}, \mathsf{r}).$$

- *Non-interactive zero-knowledge proofs:* A proof can be produced to prove:

  1. the knowledge of the message m for the encryption $\mathsf{enc}(\mathsf{m}, \mathsf{pk}, \mathsf{r})$, i.e.

  $$\pi_{\mathsf{enc}} = \mathsf{pr}_{\mathsf{enc}}(\mathsf{enc}(\mathsf{m}, \mathsf{pk}, \mathsf{r}), \mathsf{m}, \mathsf{r}),$$

  2. the knowledge of the encryption $\mathsf{enc}(\mathsf{m}, \mathsf{pk}, \mathsf{r}_1)$ for the re-encryption $\mathsf{enc}(\mathsf{m}, \mathsf{pk}, \mathsf{r}_2)$, i.e.

  $$\pi_{\mathsf{renc}} = \mathsf{pr}_{\mathsf{renc}}(\mathsf{enc}(\mathsf{m}, \mathsf{pk}, \mathsf{r}_2), \mathsf{enc}(\mathsf{m}, \mathsf{pk}, \mathsf{r}_1), \mathsf{r}_2),$$

  3. the correct decryption of $\mathsf{enc}(\mathsf{m}, \mathsf{pk}, \mathsf{r})$ with the private key sk, i.e.

  $$\pi_{\mathsf{dec}} = \mathsf{pr}_{\mathsf{dec}}(\mathsf{enc}(\mathsf{m}, \mathsf{pk}, \mathsf{r}), \mathsf{m}, \mathsf{sk}),$$

  4. that the terms $\mathsf{enc}(\mathsf{tpk}^{\mathsf{r}}, \mathsf{pk}, \mathsf{r}_1)$ and $\mathsf{enc}(\mathsf{r}, \mathsf{pk}, \mathsf{r}_2)$ that are computed for a trapdoor commitment have the same randomness r, i.e.

  $$\pi_{\mathsf{rand}} = \mathsf{pr}_{\mathsf{rand}}(\mathsf{enc}(\mathsf{tpk}^{\mathsf{r}}, \mathsf{pk}, \mathsf{r}_1), \mathsf{enc}(\mathsf{r}, \mathsf{pk}, \mathsf{r}_2), \mathsf{r}, \mathsf{r}_1, \mathsf{r}_2),$$

  5. the correct computation of the commitment $\mathsf{com}(\mathsf{tr}, \mathsf{tpk}, \mathsf{r})$ from $\mathsf{enc}(\mathsf{tpk}^{\mathsf{r}}, \mathsf{pk}, \mathsf{r}_1)$, $\mathsf{enc}(\mathsf{r}, \mathsf{pk}, \mathsf{r}_2)$, and $\mathsf{enc}(\mathsf{tr}, \mathsf{pk}, \mathsf{r}_3)$, i.e.

  $$\pi_{\mathsf{com}} = \mathsf{pr}_{\mathsf{com}}(\mathsf{tpk}, \mathsf{enc}(\mathsf{tpk}^{\mathsf{r}}, \mathsf{pk}, \mathsf{r}_1), \mathsf{enc}(\mathsf{r}, \mathsf{pk}, \mathsf{r}_2), \mathsf{enc}(\mathsf{tr}, \mathsf{pk}, \mathsf{r}_3), \mathsf{com}(\mathsf{tr}, \mathsf{tpk}, \mathsf{r}), \mathsf{sk}, \mathsf{r}).$$

Main idea. In Selene, the following information is published on BB for a voter credential cr:

$$\mathsf{BB} : \mathsf{cr}, \mathsf{tpk}, \mathsf{com}(\mathsf{tr}, \mathsf{tpk}, \mathsf{r}), \mathsf{enc}(\mathsf{tr}, \mathsf{pk}_{\mathsf{E}}, -), \mathsf{enc}(\mathsf{r}, \mathsf{pk}_{\mathsf{E}}, -), \mathsf{enc}(\mathsf{v}, \mathsf{pk}, -), \pi$$

where tpk is the voter's public trapdoor key, $\mathsf{com}(\mathsf{tr}, \mathsf{tpk}, \mathsf{r})$ is the commitment to their tracker, $\mathsf{enc}(\mathsf{tr}, \mathsf{pk}_{\mathsf{E}}, -)$, $\mathsf{enc}(\mathsf{r}, \mathsf{pk}_{\mathsf{E}}, -)$, and $\mathsf{enc}(\mathsf{v}, \mathsf{pk}, -)$ are the encryptions of their tracker, commitment randomness, and vote, respectively, and π is the combination of all the produced proofs.

At the end of the election, talliers will obtain tr, r, and v by decryption. Then, they can identify the corresponding voter and send them the randomness so that they can compute the tracker. The voter opens their commitment $\mathsf{com}(\mathsf{tr}, \mathsf{tpk}, \mathsf{r})$ with the randomness r received from talliers and their private trapdoor key, i.e. $\mathsf{tr} = \mathsf{open}(\mathsf{cm}, \mathsf{tsk}, \mathsf{r})$, which allows them to find the pair $(\mathsf{tr}, \mathsf{v})$ on BB and match their vote.

We describe the election procedures and the individual verification procedure of Selene as follows:

Setup phase. A determines the list of candidates $\mathsf{v}_1, \dots, \mathsf{v}_k$ and voters $\mathsf{id}_1, \dots, \mathsf{id}_n$ that are eligible for the election. It delegates T to generate the election key pair $(\mathsf{sk}_E, \mathsf{pk}_E)$, VR to generate a signing key pair $(\mathsf{sk}_{id}, \mathsf{pk}_{id})$, and VS to generate a login credential pwd for each voter id. Thus, each voter id obtains a signing key pair $(\mathsf{sk}_{id}, \mathsf{pk}_{id})$ from VR and a password pwd from VS. VR publishes the public keys of the voters as public credentials $\mathsf{cr}_1, \dots, \mathsf{cr}_n$, where each $\mathsf{cr}_i = \mathsf{pk}_{id_i}$. At this stage, the information available on BB is as follows:

$$\mathsf{BBkey} : \mathsf{pk}_E; \quad \mathsf{BBcand} : \mathsf{v}_1, \dots, \mathsf{v}_k; \quad \mathsf{BBreg} : \mathsf{cr}_1, \dots, \mathsf{cr}_n$$

Voters generate their trapdoor key pairs, i.e. a pair of $(\mathsf{tsk}, \mathsf{tpk})$, and publish the public trapdoor key tpk.

$$\mathsf{BBtpk} : (\mathsf{cr}_1, \mathsf{tpk}_1), \dots, (\mathsf{cr}_n, \mathsf{tpk}_n)$$

Moreover, A generates n-trackers, i.e. $\mathsf{tr}'_1, \dots, \mathsf{tr}'_n$, to be used for the verification of the votes by voters and publishes trackers with their trivial encryptions, i.e. $\mathsf{etr}'_1, \dots, \mathsf{etr}'_n$. The trivial encryptions are obtained using a fix randomness r_{fix} to encrypt each tracker tr' such that $\mathsf{etr}' = \mathsf{enc}(\mathsf{tr}', \mathsf{pk}_E, \mathsf{r}_{fix})$. T re-encrypts the set of (trivially) encrypted trackers through a mixnet and obtains $\mathsf{etr}_1, \dots, \mathsf{etr}_n$:

$$\mathsf{tr}'_1, \dots, \mathsf{tr}'_n \xrightarrow[\textit{encryption}]{\textit{trivial}} \mathsf{etr}'_1, \dots, \mathsf{etr}'_n \xrightarrow[\textit{mixing}]{\textit{re-encryption}} \mathsf{etr}_1, \dots, \mathsf{etr}_n, \pi_{\mathsf{renc}}$$

where $\mathsf{etr} = \mathsf{renc}(\mathsf{etr}', \mathsf{pk}_E, \mathsf{r}')$. T also produces a zero-knowledge proof π_{renc} for the correct re-encryption. The following information is posted on BB:

$$\mathsf{BBtr} : \mathsf{tr}'_1, \dots, \mathsf{tr}'_n; \quad \mathsf{BBetr}' : \mathsf{etr}'_1, \dots, \mathsf{etr}'_n; \quad \mathsf{BBetr} : \mathsf{etr}_1, \dots, \mathsf{etr}_n, \pi_{\mathsf{renc}}$$

Encrypted trackers are assigned to the voters, i.e. each etr_i is assigned to the trapdoor public key tpk_i. Thus, we have:

$$\mathsf{BBatr} : (\mathsf{cr}_1, \mathsf{tpk}_1, \mathsf{etr}_1), \dots, (\mathsf{cr}_n, \mathsf{tpk}_n, \mathsf{etr}_n)$$

Then, T commits to the trackers with a fresh randomness r, i.e. it generates a trapdoor commitment cm for each trapdoor public key tpk as follows:

$$(1) \quad \mathsf{tpk}^{\mathsf{r}}, \mathsf{r} \xrightarrow{\textit{encryption}} \mathsf{enc}(\mathsf{tpk}^{\mathsf{r}}, \mathsf{pk}_E, \mathsf{r}_1), \mathsf{enc}(\mathsf{r}, \mathsf{pk}_E, \mathsf{r}_2), \pi_{\mathsf{rand}}$$

$$(2) \quad \mathsf{enc}(\mathsf{tpk}^{\mathsf{r}}, \mathsf{pk}_E, \mathsf{r}_1), \mathsf{enc}(\mathsf{tr}, \mathsf{pk}_E, \mathsf{r}_3) \xrightarrow{\textit{multiplication \& decryption}} \mathsf{com}(\mathsf{tr}, \mathsf{tpk}, \mathsf{r}), \pi_{\mathsf{com}}$$

In (1), T encrypts both $\mathsf{tpk}^{\mathsf{r}}$ and r and produces the zero-knowledge proof π_{rand} that the randomness r is the same in both terms, i.e.

$$\pi_{\mathsf{rand}} = \mathsf{pr}_{\mathsf{rand}}(\mathsf{enc}(\mathsf{tpk}^{\mathsf{r}}, \mathsf{pk}_E, \mathsf{r}_1), \mathsf{enc}(\mathsf{r}, \mathsf{pk}_E, \mathsf{r}_2), \mathsf{r}, \mathsf{r}_1, \mathsf{r}_2).$$

In (2), T multiplies the encryption of $\mathsf{tpk}^{\mathsf{r}}$ with the encryption of tr, i.e. etr, then decrypts the multiplication and obtains the commitment cm, i.e. it computes

$$\mathsf{decom}(\mathsf{enc}(\mathsf{tpk}^{\mathsf{r}}, \mathsf{pk}_{\mathsf{E}}, \mathsf{r}_1), \mathsf{enc}(\mathsf{tr}, \mathsf{pk}_{\mathsf{E}}, \mathsf{r}_3), \mathsf{sk}_{\mathsf{E}}) = \mathsf{com}(\mathsf{tr}, \mathsf{tpk}, \mathsf{r}).$$

The correctness of this operation relies on homomorphic properties of the ElGamal encryption, and on its similarity with the Pedersen commitment, as explained above. To prove the correctness of the computation, T produces the proof π_{com}, where

$$\pi_{\mathsf{com}} = \mathsf{pr}_{\mathsf{com}}(\mathsf{tpk}, \mathsf{enc}(\mathsf{tpk}^{\mathsf{r}}, \mathsf{pk}_{\mathsf{E}}, \mathsf{r}_1), \mathsf{enc}(\mathsf{r}, \mathsf{pk}_{\mathsf{E}}, \mathsf{r}_2), \mathsf{enc}(\mathsf{tr}, \mathsf{pk}_{\mathsf{E}}, \mathsf{r}_3), \mathsf{com}(\mathsf{tr}, \mathsf{tpk}, \mathsf{r}), \mathsf{sk}_{\mathsf{E}}, \mathsf{r}).$$

Thus, for each public credential cr, the following tuple is published on BBcast:

$$\mathsf{BBcast} : (\mathsf{cr}, \mathsf{tpk}, \mathsf{etr}, \mathsf{cm}, \perp),$$

where $\mathsf{etr} = \mathsf{enc}(\mathsf{tr}, \mathsf{pk}_{\mathsf{E}}, \mathsf{r}_{\mathsf{tr}})$ and $\mathsf{cm} = \mathsf{com}(\mathsf{tr}, \mathsf{tpk}, \mathsf{r})$. The last part of the tuple will be filled with the ballot of the voter.

Voting phase. V interacts with VP to construct a ballot:

VP : downloads $\mathsf{pk}_{\mathsf{E}} \in \mathsf{BBkey}$ and $\mathsf{v}_1, \ldots, \mathsf{v}_{\mathsf{k}} \in \mathsf{BBcand}$,
V : selects $\mathsf{v} \in \mathsf{BBcand}$,
VP : encrypts v with pk_{E} and a randomness r_{v} : $\mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk}_{\mathsf{E}}, \mathsf{r}_{\mathsf{v}})$,
signs c with $\mathsf{sk}_{\mathsf{id}}$: $\mathsf{s} = \mathsf{sign}(\mathsf{c}, \mathsf{sk}_{\mathsf{id}})$,
produces a proof of knowledge of v : $\mathsf{p} = \mathsf{pr}_{\mathsf{enc}}(\mathsf{c}, \mathsf{v}, \mathsf{r}_{\mathsf{v}})$.

In the above computations, the voter provides $\mathsf{sk}_{\mathsf{id}}$ for the signature. The proof p denotes the non-interactive zero-knowledge proof of the knowledge of the vote v. Thus, VP constructs the ballot as a triple, i.e. $\mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p} \rangle$. If V decides to cast b, VP requests login credentials of V, i.e. id and pwd, which will prompt a connection to VS. If VS authenticates V, it receives the tuple $\langle \mathsf{cr}, \mathsf{b} \rangle$ on behalf of id and then verifies the signature s and the proof p. Then, it records b in the position $*$ of the tuple $(\mathsf{cr}, \mathsf{tpk}, \mathsf{etr}, \mathsf{cm}, *)$ on BBcast. In case of revoting, the existing ballot is replaced with the new one.

Tally phase. At the end of the voting phase, VS retrieves the *last* ballot published on BBcast for each credential and publishes it on $\mathsf{BBtally}$, i.e. for the credential cr, the following tuple is published:

$$\mathsf{BBtally} : (\mathsf{cr}, \mathsf{tpk}, \mathsf{etr}, \mathsf{cm}, \mathsf{b}),$$

where $\mathsf{b} = \perp$ if no ballot was cast. Then, the encrypted tracker etr and the ciphertext c from the ballot $\mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p} \rangle$ are extracted for each such tuple on $\mathsf{BBtally}$, and the pair $(\mathsf{etr}, \mathsf{c})$ is sent to T for the decryption. T, first, re-encrypts and shuffles the list of such pairs and then decrypts all, producing the zero-knowledge proof π_{dec} for the correct decryption. Thus, the list of tracker-vote pairs, i.e. $(\mathsf{tr}, \mathsf{v})$, and the proof π_{dec} are published on BB as the election result:

$$\mathsf{BBres} : (\mathsf{tr}_1, \mathsf{v}_1), \ldots, (\mathsf{tr}_{\mathsf{n}}, \mathsf{v}_{\mathsf{n}}), \pi_{\mathsf{dec}}.$$

At this stage, voters do not know which tracker is theirs.

Individual verification. Selene allows individual verification of the votes directly in the election result. Therefore, votes are published next to trackers as additional credentials for vote verification. The trackers are generated and assigned to the voters after a couple of operations performed by T at the beginning of the election. However, neither T nor voters know which trackers were assigned to them due to the verifiable mixnets. T cannot change the assigned trackers due to the commitments and produced zero-knowledge proofs.

To verify their vote, voters obtain the commitment randomness r from T via a secure channel. Then, they open the commitment cm using their private trapdoor key tsk as follows:

$$\mathsf{open}(\mathsf{cm}, \mathsf{tsk}, \mathsf{r}) = \mathsf{open}(\mathsf{com}(\mathsf{tr}, \mathsf{tpk}, \mathsf{r}), \mathsf{tsk}, \mathsf{r}) = \mathsf{tr},$$

which allows them to verify the vote next to tr on BBres.

Receipt-freeness. The coercer may force a voter to cast a particular vote, say v'. Then, at the end of the election, the coercer could ask for the commitment randomness sent by T, compute the corresponding tracker, and check whether the vote next to it is v'. To prevent such coercion, the cryptographic primitives used in Selene allow voters to generate a fake commitment randomness r' that opens the commitment to a different tracker tr' on the bulletin board, for which the vote is v'. Thus, the commitment randomness is sent after the result is published on the bulletin board, i.e. to allow voters to generate their fake commitment randomness if coerced. Selene provides some level of receipt-freeness since the voter can give the fake randomness to the coercer. However, there is a risk for the voter that the coercer could itself be a voter, and the randomness generated by the voter could open their commitment to the coercer's tracker, i.e. trackers may collude.

7.1.2 ProVerif Specification of Selene

In this section, we present the ProVerif specification of the Selene protocol, focusing on the details specific to Selene. These are its equational theory, the extraction of trackers and the linking events, and its individual verification procedure for voters. At the end of the section, we provide the full specification of Selene in Figure 7.6 and Figure 7.7.

Equational theory. Selene relies on ElGamal encryption, a digital signature scheme, Pedersen trapdoor commitments, and non-interactive zero-knowledge proofs, as described in Section 7.1.1. Mixnets are used for anonymising trackers and ciphertexts corresponding to the votes before decryption. However, we do not explicitly model mixnets in the specification, but only the re-encryption operation they perform. This is sufficient for verifiability purposes since we do not attempt to hide the permutation of the ballots when computing the outcome. Thus, cryptography requires the following equations:

(1) $\mathsf{dec}(\mathsf{enc}(\mathsf{x}, \mathsf{pk}(\mathsf{y}), \mathsf{z}), \mathsf{y}) = \mathsf{x}$,
(2) $\mathsf{verify}(\mathsf{sign}(\mathsf{x}, \mathsf{y}), \mathsf{x}, \mathsf{pk}(\mathsf{y})) = \mathsf{true}$,
(3) $\mathsf{ver}_{\mathsf{enc}}(\mathsf{pr}_{\mathsf{enc}}(\mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}), \mathsf{x}, \mathsf{z}), \mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}), \mathsf{y}) = \mathsf{true}$,
(4) $\mathsf{ver}_{\mathsf{dec}}(\mathsf{pr}_{\mathsf{dec}}(\mathsf{enc}(\mathsf{x}, \mathsf{pk}(\mathsf{y}), \mathsf{z}), \mathsf{x}, \mathsf{y}), \mathsf{enc}(\mathsf{x}, \mathsf{pk}(\mathsf{y}), \mathsf{z}), \mathsf{x}, \mathsf{pk}(\mathsf{y})) = \mathsf{true}$,

(5) $\mathsf{renc}(\mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}_1), \mathsf{y}, \mathsf{z}_2) = \mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}_2)$,
(6) $\mathsf{ver}_{\mathsf{renc}}(\mathsf{pr}_{\mathsf{renc}}(\mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}_2), \mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}_1), \mathsf{z}_2), \mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}_2), \mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}_1), \mathsf{y}) = \mathsf{true}$,

(7) $\mathsf{ver}_{\mathsf{rand}}(\mathsf{pr}_{\mathsf{rand}}(\mathsf{enc}(\mathsf{x}^{\mathsf{r}}, \mathsf{y}, \mathsf{z}_1), \mathsf{enc}(\mathsf{r}, \mathsf{y}, \mathsf{z}_2), \mathsf{r}, \mathsf{z}_1, \mathsf{z}_2), \mathsf{enc}(\mathsf{x}^{\mathsf{r}}, \mathsf{y}, \mathsf{z}_1), \mathsf{enc}(\mathsf{r}, \mathsf{y}, \mathsf{z}_2), \mathsf{y}) = \mathsf{true}$,
(8) $\mathsf{decom}(\mathsf{enc}(\mathsf{x}^{\mathsf{r}}, \mathsf{pk}(\mathsf{y}), \mathsf{z}_1), \mathsf{enc}(\mathsf{w}, \mathsf{pk}(\mathsf{y}), \mathsf{z}_3), \mathsf{y}) = \mathsf{com}(\mathsf{w}, \mathsf{x}, \mathsf{r})$,
(9) $\mathsf{ver}_{\mathsf{com}}(\mathsf{pr}_{\mathsf{com}}(\mathsf{x}, \mathsf{enc}(\mathsf{x}^{\mathsf{r}}, \mathsf{pk}(\mathsf{y}), \mathsf{z}_1), \mathsf{enc}(\mathsf{r}, \mathsf{pk}(\mathsf{y}), \mathsf{z}_2), \mathsf{enc}(\mathsf{w}, \mathsf{pk}(\mathsf{y}), \mathsf{z}_3), \mathsf{com}(\mathsf{w}, \mathsf{x}, \mathsf{r}), \mathsf{y}, \mathsf{r}),$
$\mathsf{x}, \mathsf{enc}(\mathsf{x}^{\mathsf{r}}, \mathsf{pk}(\mathsf{y}), \mathsf{z}_1), \mathsf{enc}(\mathsf{r}, \mathsf{pk}(\mathsf{y}), \mathsf{z}_2), \mathsf{enc}(\mathsf{w}, \mathsf{pk}(\mathsf{y}), \mathsf{z}_3), \mathsf{com}(\mathsf{w}, \mathsf{x}, \mathsf{r}), \mathsf{pk}(\mathsf{y})) = \mathsf{true}$,

(10) $\mathsf{open}(\mathsf{com}(\mathsf{x}, \mathsf{pk}(\mathsf{y}), \mathsf{z}), \mathsf{y}, \mathsf{z}) = \mathsf{x}$,
(11) $\mathsf{com}(\mathsf{x}_2, \mathsf{pk}(\mathsf{y}), \mathsf{fake}(\mathsf{x}_1, \mathsf{x}_2, \mathsf{y}, \mathsf{z})) = \mathsf{com}(\mathsf{x}_1, \mathsf{pk}(\mathsf{y}), \mathsf{z})$,
(12) $\mathsf{open}(\mathsf{com}(\mathsf{x}_1, \mathsf{pk}(\mathsf{y}), \mathsf{z}), \mathsf{y}, \mathsf{fake}(\mathsf{x}_1, \mathsf{x}_2, \mathsf{y}, \mathsf{z})) = \mathsf{x}_2$,
(13) $\mathsf{fake}(\mathsf{x}_2, \mathsf{x}_3, \mathsf{y}, \mathsf{fake}(\mathsf{x}_1, \mathsf{x}_2, \mathsf{y}, \mathsf{z})) = \mathsf{fake}(\mathsf{x}_1, \mathsf{x}_3, \mathsf{y}, \mathsf{z})$.

The equations (1-4) represent the cryptography required for the ballot generation, validation, and decryption of the votes, which are common in many e-voting protocols. Specifically, they model (1) asymmetric encryption/decryption, (2) signing/verification, (3) zero-knowledge for

the knowledge of the vote/verification, and (4) zero-knowledge proof for correct decryption, respectively. The equations (5) and (6) are added to model re-encryption and the related zero-knowledge proof for correct re-encryption. The equation (8) is used for the generation of the trapdoor commitments, i.e. it allows talliers to compute a commitment to a tracker from encryption of that tracker without decrypting it. The zero-knowledge proofs π_{rand} and π_{com} generated for the commitment randomness and the correct computation of the commitment are verified using the equations (7) and (9), respectively.

We use a previously considered theory in [24, 13] to model trapdoor commitments and verify privacy-type properties. The main property is that a commitment can be opened with the corresponding private trapdoor key, as in equation (10). Furthermore, equation (11) models that a private trapdoor key and the commitment randomness can be used to generate another randomness that opens the commitment to any desired value. As shown in [13], equations (12) and (13) are added to make this theory convergent in ProVerif.

Extraction of trackers and the Link ***events:*** The trackers are generated by A at the beginning of the election and recorded on BB with their trivial encryptions. Then, they are re-encrypted by T through a re-encryption mixnet and assigned to the voter public credentials. If T correctly re-encrypts and shuffles the trackers, then each re-encrypted tracker should have been uniquely assigned to a voter credential. Therefore, if the zero-knowledge proof produced by T for the correct re-encryption is verified, we can ensure that each credential is linked to a tracker on BB. In our specification, we formally link each credential to a tracker that can be extracted from public information and proofs associated to it on BB to ensure the existence of the link and the consistency between public credentials and trackers. Therefore, we assume the pre-tracking information is recorded on $\mathsf{BBptr}(\mathsf{cr}, \mathsf{etr}, \pi_{\mathsf{renc}})$, which allows us to extract the tracker tr from its encryption etr, if the proof π_{renc} produced for the re-encryption is verified. Specifically, the extraction requires the following equation:

$$\mathsf{extract}(\mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}), \mathsf{y}) = \mathsf{x},$$

i.e. the function extract retrieves the term x from its encryption $\mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z})$. Note that etr is associated with a public credential cr on BBptr. Thus, we can link tr to cr after its extraction from etr. The following information flow on the BB describes how the trackers are extracted from BBptr and then linked to the credentials:

BBreg :	cr_1	…	cr_n	
BBtr :	tr'_1	…	tr'_n	
BBetr′ :	etr'_1	…	etr'_n	
BBetr :	etr_1	…	etr_n	$\pi_{\mathsf{renc}} = \pi_1 \circ \ldots \circ \pi_n$
BBptr :	$(\mathsf{cr}_1, \mathsf{etr}_1, \pi_1)$	…	$(\mathsf{cr}_n, \mathsf{etr}_n, \pi_n)$	
	π_1 : verified	…	π_n : verified	
	$\mathsf{tr}_1 = \mathsf{extract}(\mathsf{etr}_1, \pi_1)$	…	$\mathsf{tr}_n = \mathsf{extract}(\mathsf{etr}_n, \pi_n)$	
	$\mathsf{Link}(\mathsf{cr}_1, \mathsf{tr}_1)$	…	$\mathsf{Link}(\mathsf{cr}_n, \mathsf{tr}_n)$	
BBres :	$(\mathsf{tr}_1, \mathsf{v}_1)$	…	$(\mathsf{tr}_n, \mathsf{v}_n)$	π_{res}

Individual verification: Selene allows individual verification of the votes at the end of the election on BBres. The voters verify their votes after they obtain their associated trackers and

then find them on BBres. To obtain their tracker tr, voters receive their commitment randomness r from T, and using it and their private trapdoor key tsk, they open their commitment cm, i.e. $tr = open(cm, tsk, r)$. If the tracker tr matches one of the trackers on BBres, then the voter can verify the vote v next to it. We assume that the commitment associated with the voter's public trapdoor key is recorded on $BBvtr(cr, tpk, cm)$. Thus, we similarly describe the flow on BB that allows voters to obtain their commitments from BBvtr, open them, and obtain their trackers:

BBreg :	cr_1	…	cr_n
BBtpk :	(cr_1, tpk_1)	…	(cr_n, tpk_n)
BBvtr :	(cr_1, tpk_1, cm_1)	…	(cr_n, tpk_n, cm_n)
	$tr_1 = open(cm_1, tsk_1, r_1)$	…	$tr_n = open(cm_n, tsk_n, r_n)$
BBres :	(tr_1, v_1)	…	(tr_n, v_n) π_{res}

We model the voter's trapdoor key generation, ballot cast using their voting platform, and individual verification as specified in Figure 7.1. The voter's vote v is recorded in the table Voted when the voting platform generates the ballot for the vote and casts it on behalf of the voter. Then, the voter verifies the vote v, first receiving the commitment randomness r from the table Com, then opening their commitment cm recorded in the table BBvtr, and last matching the tracker tr and the vote v on BBres. Note that the talliers record the commitment randomness in the table Com for each credential. Then, the voters with those credentials get the commitment randomness from Com, which models the private communication of the commitment randomness to the voter.

```
let VoterTpk(id) =
  get Cred(= id, cr, sk_id) in
  new tsk;
  let tpk = pk(tsk) in
  insert Td(id, tpk, tsk);
  insert BBtpk(cr, tpk).

let VoterVer(id) =
  get Td(= id, tpk, tsk) in
  get Voted(= id, cr, v) in
  get BBvtr(= cr, = tpk, cm) in
  get BBres(tr, = v) in
  get Com(= cr, r) in
  if open(cm, tsk, r) = tr then
  event Verified(id, cr, v).
```

```
let VotingPlatformVote(v, pk_E) =
  get Cred(id, cr, sk_id) in
  get Pwd(= id, pwd) in
  new r;
  let c = enc(v, pk_E, r) in
  let s = sign(c, sk_id) in
  let p = pr_enc(c, v, r) in
  let b = (c, s, p) in
  let a = h(id, pwd, cr, b) in
  event Vote(id, v);
  insert Voted(id, cr, v);
  out(pub, (id, cr, b, a)).
```

FIGURE 7.1: Processes specified for the voters and voting platforms in Selene.

Some details about our ProVerif specification of Selene are as follows:

1. The re-encrypted trackers are assigned to the voters if the proof of re-encryption attached is verified in the process AdminAssign. Note that this process gets one random entry from table Etr and another from table BBtpk; therefore, it models the shuffling

of the re-encrypted trackers before being assigned to the voters. We ensure any re-encrypted tracker is assigned to at most one voter with a restriction that we do not present in Figure 7.6 but in the ProVerif code [52] provided online.

2. The table Compair in the process TallierCom records the encryptions of $\mathsf{tpk}^{\mathsf{r}}$ and r as a and b, together with the proof π_{rand}. Similarly, the table Cm records the commitment cm and the proof π_{com}. The entries of these two tables are used by the voting server to verify the related proofs. Then, it records the commitment cm next to etr on BBcast.

3. Like Belenios, we model the server as it checks the consistency between voter identities and public credentials, recording each identity-credential pair on the table Log.

```
let VoterCorr =
  get Cred(id, cr, sk_id) in
  get Td(= id, tpk, tsk) in
  get Pwd(= id, pwd) in
  event Corr(id);
  out(pub, (id, pwd, cr, sk_id, tpk, tsk)).
```

```
let CorruptedVotingPlatformVote(v, pk_E) =
  get Cred(id, cr, sk_id) in
  get Pwd(= id, pwd) in
  out(pub, (id, pwd, cr, sk_id, v));
  in(pub, (= id, = cr, b, a);
  event Vote(id, v);
  insert Voted(id, cr, v);
  out(pub, (id, cr, b, a)).
```

FIGURE 7.2: Processes specified for the corrupt voters and corrupt voting platforms in Selene.

Adversary models $\mathcal{A}$. For Selene, we assume other than the administrator, any party can be corrupt, allowing $\mathcal{A}$ to manipulate the election data:

- ***Corrupt talliers*** allow $\mathcal{A}$ to determine the private key of the election and any other data they need to provide on BB, e.g. etr and $\mathsf{p}_{\mathsf{renc}}$. However, since the data on BB can be audited by anybody, e.g. the zero-knowledge proofs can be verified, talliers first verify the correctness of the data received from $\mathcal{A}$ and then provide them on BB. In the specifications that $\mathcal{A}$ corrupts the talliers, we model the processes TallierKey, TallierEtr, TallierCom, and TallierDec as receiving the data they need to provide from the public channel, i.e. from $\mathcal{A}$. Note that in addition to verifiable public data, talliers receive the commitment randomness from $\mathcal{A}$ in the process TallierCom, which allows $\mathcal{A}$ to manipulate the tracker assigned to a voter if $\mathcal{A}$ has already obtained the private trapdoor key tsk of the voter.

- A ***corrupt registrar*** allows $\mathcal{A}$ to determine the signing key pair for each voter. In the specifications that $\mathcal{A}$ corrupts the registrar, we model the process for registration of the credentials, i.e. RegistrarReg, as receiving $(\mathsf{cr}, \mathsf{sk}_{\mathsf{id}})$ from $\mathcal{A}$.

- A ***corrupt server*** accepts any ballot for any credential without authentication, but it still verifies the signature and the proof inside the ballot as they can be verified publicly on BB. We model a corrupt server with the process CorruptServerCast as specified in Figure 7.4.

- ***Corrupt voting platforms*** allow $\mathcal{A}$ to construct a ballot on behalf of the voter and compute its corresponding authentication, revealing the voter credentials to $\mathcal{A}$ except for the trapdoor key pair, i.e. $\mathcal{A}$ obtains $(\mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{sk}_{\mathsf{id}}, \mathsf{v})$ from the voting platform and constructs a ballot b and its authentication a instead of the platform. Here, we extend the adversary's abilities to construct any ballot compared to the one considered for

Helios and Belenios in the previous chapters, where it is allowed only to choose the encryption randomness. Thus, we model the corrupt voting platforms with the process CorruptedVotingPlatformVote in Figure 7.2 as they communicate with $\mathcal{A}$ during the ballot generation through the public channel.

– ***Corrupt voters*** leak all their credentials to $\mathcal{A}$, i.e. $(\mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{sk}_{\mathsf{id}}, \mathsf{tpk}, \mathsf{tsk})$, which is modelled in all specifications with the process VoterCorr described in Figure 7.2.

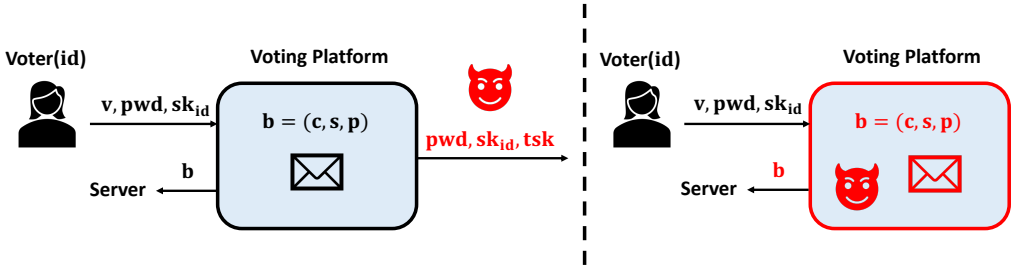


FIGURE 7.3: Corrupt voter (left) vs corrupt voting platform (right).

Figure 7.3 illustrates the difference between a corrupt voter and a corrupt voting platform. In the latter, we let the adversary construct the ballot as it wishes, providing $\mathcal{A}$ with the required credentials. Yet, contrary to a corrupt voter, we assume the voter's private trapdoor key is secret. The reason is that voters generate trapdoor key pairs at the beginning of the election. Thus, the procedure of trapdoor key generation is independent of the procedure of ballot casting. In practice, the private trapdoor key can be stored on special trusted hardware after generated. Therefore, it protects the key during ballot casting. Even if there is no such hardware, the voters should use a separate device to verify their votes other than the one used for ballot casting in case the corrupt device can trick the voter about the verification. In this case, the tracker key should be generated in the verification device, making it secret during ballot casting.

The adversary models we consider for our analysis are defined by the subsets of the corrupt talliers, registrar, server, and voting platform described above. Other than them, we assume the communication network and some voters are corrupt in all scenarios. Specifically, we define seven scenarios from $\mathcal{A}_1$ to $\mathcal{A}_7$, as presented in Table 7.1.

Adversary Models	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$	$\mathcal{A}_5$	$\mathcal{A}_6$	$\mathcal{A}_7$
Talliers	H	C	C	C	C	C	C
Registrar	H	H	C	H	H	C	H
Server	H	H	H	C	H	H	C
Voting Platform	H	H	H	H	C	C	C

TABLE 7.1: Adversary models for Selene, SeleneRF, and Hyperion.

7.1.3 SeleneRF and Its ProVerif Specification

Selene relies on a public bulletin board that displays all the public election data, including the cast ballots. Even though it provides an easy verification procedure for the voters to verify their votes in the clear at the end of the election, any voter can still verify that their ballot is captured by the voting server and thus published on BB. Therefore, coercion is possible if the voter records the encryption randomness while generating a ballot and then gives it to

the coercer as a receipt. If the coercer knows the voter's public credential or public trapdoor key, they can verify that the ballot contains their desired vote. To prevent such attacks, the method used for BeleniosRF can be deployed by the protocol, as discussed in [47]. This method is based on the randomisation of the ballots by the server after it receives from voters and validates them. In this way, even if the coercer obtains the encryption randomness, they cannot verify the ballot on BB that it contains their desired vote.

In our analysis, we aim to experiment with the effect of randomisation of the ballots on the server's side on the verifiability of Selene. Therefore, we model SeleneRF, a variant of Selene that provides receipt-freeness. The specification of SeleneRF is slightly different from Selene, where the cryptography is extended to adapt signature and zero-knowledge proofs according to the re-encryption of the ciphertext inside the ballot. Therefore, we add the following two equations to the equational theory of Selene:

$$\text{(14)}\quad \mathsf{resign}(\mathsf{sign}(\mathsf{enc}(x, y, z_1), w), y, z_2) = \mathsf{sign}(\mathsf{enc}(x, y, z_2), w),$$
$$\text{(15)}\quad \mathsf{repr}(\mathsf{pr}(\mathsf{enc}(x, y, z_1), x, z_1), y, z_2)) = \mathsf{pr}(\mathsf{enc}(x, y, z_2), x, z_2).$$

In addition to the equational theory, the process of the server for casting the voter's ballot to BB is extended with the randomisation abilities of the server, as specified in Figure 7.4. Thus, the server generates a randomness r' after it validates the ballot $b = \langle c, s, p\rangle$, re-encrypts the ciphertext c and adapts the signature s and the proof p of knowledge of the vote with respect to the re-encryption of the vote. Thanks to the cryptography deployed, the server does not require the signing key of the voter to adapt the signature. The server records the randomised ballot $b = \langle c', s', p'\rangle$ on BBcast. Therefore, the ciphertext c computed with the encryption randomness r generated by the voter differs from the randomised ciphertext c' on BBcast, which means that the voters cannot provide a receipt to the adversary.

```
let ServerCast(pk_E) =
  get BBcast(cr, tpk, etr, cm, = ⊥) in
  get Pwd(id, pwd) in
  in(pub, (= id, = cr, (c, s, p), a);
  event Log(id, cr);
  let b = (c, s, p) in
  if h(id, pwd, cr, b) = a then
  if verify(s, c, cr) = true then
  if ver_enc(p, c, pk_E) = true then
  new r';
  let c' = renc(c, pk_E, r') in
  let s' = resign(s, pk_E, r') in
  let p' = repr(p, pk_E, r') in
  let b' = (c', s', p') in
  insert BBcast(cr, tpk, etr, cm, b').
```

```
let CorruptedServerCast(pk_E) =
  get BBcast(cr, tpk, etr, cm, = ⊥) in
  in(pub, (= cr, (c, s, p));
  let b = (c, s, p) in
  if verify(s, c, cr) = true then
  if ver_enc(p, c, pk_E) = true then
  insert BBcast(cr, tpk, etr, cm, b).
```

FIGURE 7.4: The processes for an honest and corrupt server in SeleneRF.

The adversary models we consider for SeleneRF are the same as those for Selene, including the processes that describe them. The process modelling the corrupt server in both Selene and SeleneRF is provided in Figure 7.4.

7.2 Hyperion

In this section, we present the protocol details of Hyperion as done for Selene. First, we describe its protocol structure, including its protocol parties with their different roles than Selene, the cryptographic primitives used, and the election procedures followed by the parties. Then, we provide its ProVerif specification details with respect to the seven adversary models described for Selene. Hyperion prevents tracker collision, providing each voter with an individual view of the bulletin board. This feature is required for receipt-freeness of the protocol. Since we focus on verifiability, we do not consider this option and exclude the related details.

7.2.1 Hyperion Protocol Structure

Hyperion has similar parties to the ones in Selene but the role of talliers and voters are modified as follows:

– Talliers T generate the election public key in a distributed way, and at least the threshold number of talliers, say k out of t, decrypt the set of ciphertexts corresponding to the ballots cast by the voters. The decryption procedure relies on a mixnet, i.e. re-encryption and shuffling, and then decryption of the ciphertexts. Talliers also contribute to the distributed generation of the trackers from the public trapdoor keys: they raise each trapdoor public key to a fresh secret randomness to obtain a term that we call a pre-tracker; then, they raise all the pre-trackers to a common randomness using an exponentiation mix and obtain trackers that allow voters to verify their votes in the election outcome.

– Voters V registered for the election generate a trapdoor key pair: a private key and its corresponding public key. They sign the public key and produce proof of knowledge of their private key. They register the public trapdoor key next to their public credential. The voters cast their ballots using their voting platform during the election.

Cryptographic primitives: Hyperion relies on ElGamal encryption algorithm, which is also used for re-encryption, a digital signature scheme, exponentiation mixes, and non-interactive zero-knowledge proofs. The encryption, re-encryption, decryption, and signing are as in described for Selene. Assume g is the generator of the cyclic multiplicative group of the order q, which is used for the cryptographic operations of Hyperion.

- *Trapdoor key generation:* A trapdoor key pair $(\mathsf{tpk}, \mathsf{tsk})$ is generated by selecting a random element tsk between 1 and q and computing $\mathsf{tpk} = \mathsf{g}^{\mathsf{tsk}}$.

- *Exponentiation mixes:* The exponentiation mix M mixes a set of input messages, i.e. M_{in}, raising each message in M_{in} to the power of s and mixing the set, and obtains the set of output messages, i.e. $\mathsf{M}_{\mathsf{out}}$. In Hyperion, they mix the set of exponentiated trapdoor keys in the form $\mathsf{tpk}^{\mathsf{r}}$, where r is randomness used for the exponentiation and obtain the terms in the form $\mathsf{tpk}^{\mathsf{rs}}$.

  Assume there are n public trapdoor keys, where each $\mathsf{tpk}_{\mathsf{i}}$ is raised to the power of r_{i} such that the pre-trackers are $\mathsf{tpk}_1^{\mathsf{r}_1}, \ldots, \mathsf{tpk}_{\mathsf{n}}^{\mathsf{r}_{\mathsf{n}}}$. Then, they are put through M as follows:

$$(\mathsf{tpk}_1^{\mathsf{r}_1}, \ldots, \mathsf{tpk}_{\mathsf{n}}^{\mathsf{r}_{\mathsf{n}}}) \xrightarrow{\mathsf{M}} (\mathsf{tpk}_1^{\mathsf{r}_1 \mathsf{s}}, \ldots, \mathsf{tpk}_{\mathsf{n}}^{\mathsf{r}_{\mathsf{n}} \mathsf{s}})$$

- *Non-interactive zero-knowledge proofs:* The proofs π_{enc}, π_{renc}, and π_{dec} are similar to the ones in Selene. In addition, a proof is produced to prove the correct exponentiation g^{x} from g and secret x as follows:

$$\pi_{\mathsf{exp}} = \mathsf{pr}_{\mathsf{exp}}(\mathsf{g}, \mathsf{x}, \mathsf{g}^{\mathsf{x}}).$$

Hyperion also relies on an append-only public bulletin board, denoted by BB, to display the public election data. In the following, we present the election procedures of Hyperion, where it differs from Selene, and its individual verification procedure:

Setup phase. Voters generate their trapdoor key pairs, i.e. a pair of $(\mathsf{tsk}, \mathsf{tpk})$, where $\mathsf{tpk} = \mathsf{g}^{\mathsf{tsk}}$ (g is the generator of the cyclic group), sign their public trapdoor key tpk with $\mathsf{sk}_{\mathsf{id}}$, i.e. $\mathsf{s}_{\mathsf{td}} = \mathsf{sign}(\mathsf{tpk}, \mathsf{sk}_{\mathsf{id}})$, and produce proof of knowledge of tsk, i.e. $\mathsf{p}_{\mathsf{td}} = \mathsf{pr}_{\mathsf{exp}}(\mathsf{g}, \mathsf{tsk}, \mathsf{tpk})$. Then, they post the tuple $\mathsf{td} = (\mathsf{tpk}, \mathsf{s}_{\mathsf{td}}, \mathsf{p}_{\mathsf{td}})$ next to their public credential cr on BB:

$$\mathsf{BBtd} : (\mathsf{cr}_1, \mathsf{td}_1), \ldots, (\mathsf{cr}_n, \mathsf{td}_n)$$

The trapdoor public key tpk allows T to compute a tracker, i.e. a verification credential, for the voter in the tally phase, and the others s_{td} and p_{td} justify that tpk is generated by the voter registered with cr and valid. In [46], td is sent together with the ballot during the voting phase. We assume they are published at the beginning of the election to allow revoting. VS verifies all the signatures and proofs inside the trapdoor tuples and prepares BBcast for the voting phase. For each credential cr, the following tuple is published on BBcast:

$$\mathsf{BBcast} : (\mathsf{cr}, \mathsf{tpk}, \perp)$$

where the last entry will be filled with the ballot of the voter if they cast one.

Tally phase. At the end of the voting phase, VS retrieves the *last* ballot published on BBcast for each credential and publishes it on $\mathsf{BBtally}$, i.e. for the credential cr, the following tuple is published:

$$\mathsf{BBtally} : (\mathsf{cr}, \mathsf{tpk}, \mathsf{b}),$$

where $\mathsf{b} = \perp$ if no ballot was cast. Then, T raises each public trapdoor key tpk and g to a fresh randomness r, and publishes $\mathsf{tpk}^{\mathsf{r}}$ next to the ballot on BBexp:

$$\mathsf{BBexp} : (\mathsf{cr}, \mathsf{tpk}, \mathsf{b}, \mathsf{tpk}^{\mathsf{r}}),$$

while keeping g^{r} secret. We call each $\mathsf{tpk}^{\mathsf{r}}$ on BBexp a pre-tracker. Recall that each ballot b is of the form $\mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p} \rangle$. Thus, from each tuple recorded on BBexp, T retrieves the pair $(\mathsf{tpk}^{\mathsf{r}}, \mathsf{c})$, i.e. the pair of pre-tracker and ciphertext. All such pairs are put through a mixnet, where T raises each pre-tracker $\mathsf{tpk}^{\mathsf{r}}$ to common randomness s, i.e. obtains the tracker $\mathsf{tr} = \mathsf{tpk}^{\mathsf{rs}}$, and re-encrypts each ciphertext c, i.e. obtains c', producing proofs of exponentiation and re-encryption, i.e.

$$\pi_{\mathsf{renc}} = \mathsf{pr}_{\mathsf{renc}}(\mathsf{c}', \mathsf{c}, \mathsf{r}'); \quad \pi_{\mathsf{exp}} = \mathsf{pr}_{\mathsf{exp}}(\mathsf{tpk}^{\mathsf{r}}, \mathsf{s}, \mathsf{tr}).$$

In this way, each pre-tracker is anonymised, and its corresponding ciphertext is randomised. In the meantime, T also raises g^{r} to common randomness s and obtains the secret, i.e. g^{rs}, to be shared with the corresponding voter to reveal their tracker. For verifiability, we assume that g^{s} is also published on BB. Then, T decrypts each such randomised ciphertext, producing proof of correct decryption, i.e. $\mathsf{p}_{\mathsf{dec}}$. These two steps taken by T can be summarised in the following:

$$\begin{array}{lll} (1) & (\mathsf{tpk}^{\mathsf{r}}, \mathsf{c}) & \xrightarrow[\textit{mixing}]{\textit{exponentiation \& re-encryption}} (\mathsf{tr}, \mathsf{c}'), \pi_{\mathsf{exp}}, \pi_{\mathsf{renc}} \\ (2) & (\mathsf{tr}, \mathsf{c}') & \xrightarrow{\textit{decryption}} (\mathsf{tr}, \mathsf{v}), \pi_{\mathsf{dec}} \end{array}$$

Thus, the list of tracker-vote pairs, i.e. $(\mathsf{tr}, \mathsf{v})$, is published as the election result:

$$\mathsf{BBres} : (\mathsf{tr}_1, \mathsf{v}_1), \ldots, (\mathsf{tr}_n, \mathsf{v}_n),$$

At this stage, voters do not know which tracker is theirs.

Individual verification. Like Selene, Hyperion allows voters to verify their votes directly in the election result. They receive the secret g^{rs} from T via a secure channel. Then, they compute their tracker using their private trapdoor key tsk as follows:

$$\mathsf{tr} = g^{rs\,\mathsf{tsk}} (= g^{\mathsf{tsk}\,rs}),$$

which allows them to verify their vote v next to the tracker tr on BBres.

Receipt-freeness. Similar to Selene, if the voter is coerced to vote for v', they can generate a fake secret that allows them to compute a tracker tr' on the bulletin board, for which the vote is v'. However, as in Selene, the tracker tr' could be the coercer's tracker and lead the coercer to detect the voter's disobedience. To prevent tracker collision, Hyperion provides an individual view of the bulletin board for each voter, raising all the trackers in the result with another fresh randomness r_i specific to that voter. The secret is also modified with respect to the randomness, i.e. it is raised to r_i. Thus, the voter verifies their vote, computing their tracker with the modified secret. In case of coercion, the voter can generate a fake secret to compute any other tracker on the bulletin board. However, this time, the coercer cannot detect the voter's cheating since, on this particular view of the bulletin board, the coercer does not know which tracker is theirs.

7.2.2 ProVerif Specification of Hyperion

In this section, we present the ProVerif specification of the Hyperion protocol, focusing on the details specific to Hyperion. These are its equational theory, the extraction of trackers and the linking events, and its individual verification procedure for voters. At the end of the section, we provide the full specification of Hyperion in Figure 7.8.

Equational theory. As a variant of Selene, Hyperion has similar cryptographic primitives, i.e. an encryption algorithm, a digital signature scheme, and a non-interactive zero-knowledge proof protocol. In addition to them, Hyperion uses a re-encryption and exponentiation mixes. Thus, we model the cryptography used by Hyperion with the following equations:

(1) $\mathsf{dec}(\mathsf{enc}(\mathsf{x}, \mathsf{pk}(\mathsf{y}), \mathsf{z}), \mathsf{y}) = \mathsf{x}$,
(2) $\mathsf{verify}(\mathsf{sign}(\mathsf{x}, \mathsf{y}), \mathsf{x}, \mathsf{pk}(\mathsf{y})) = \mathsf{true}$,
(3) $\mathsf{ver}_{\mathsf{enc}}(\mathsf{pr}_{\mathsf{enc}}(\mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}), \mathsf{x}, \mathsf{z}), \mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}), \mathsf{y}) = \mathsf{true}$,
(4) $\mathsf{ver}_{\mathsf{dec}}(\mathsf{pr}_{\mathsf{dec}}(\mathsf{enc}(\mathsf{x}, \mathsf{pk}(\mathsf{y}), \mathsf{z}), \mathsf{x}, \mathsf{y}), \mathsf{enc}(\mathsf{x}, \mathsf{pk}(\mathsf{y}), \mathsf{z}), \mathsf{x}, \mathsf{pk}(\mathsf{y})) = \mathsf{true}$,

(5) $\mathsf{renc}(\mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}_1), \mathsf{y}, \mathsf{z}_2) = \mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}_2)$,
(6) $\mathsf{ver}_{\mathsf{renc}}(\mathsf{pr}_{\mathsf{renc}}(\mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}_2), \mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}_1), \mathsf{z}_2), \mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}_2), \mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}_1), \mathsf{y}) = \mathsf{true}$,

(7) $(g^{\mathsf{x}})^{\mathsf{y}} = (g^{\mathsf{y}})^{\mathsf{x}}$,
(8) $((g^{\mathsf{x}})^{\mathsf{y}})^{\mathsf{z}} = ((g^{\mathsf{y}})^{\mathsf{z}})^{\mathsf{x}}$,
(9) $\mathsf{ver}_{\mathsf{exp}}(\mathsf{pr}_{\mathsf{exp}}(\mathsf{x}, \mathsf{y}, \mathsf{x}^{\mathsf{y}}), \mathsf{x}, \mathsf{x}^{\mathsf{y}}) = \mathsf{true}$.

The equations (1-4) are similar to the ones in Selene that are required for the ballot generation, validation, and decryption of the votes. We use the equations (5) and (6) to model the operations of the re-encryption mix, i.e. re-encryption and the verification of the proof produced for correct encryption. For the exponentiation mix, we need to consider a model of exponentiation that is more general than the usual model used in ProVerif, which only allows capturing Diffie-Hellman-like interactions with two exponentiations. We need three exponentiations since trackers are generated through three exponentiations, i.e. $\mathsf{tr} = ((g^{\mathsf{tsk}})^{\mathsf{r}})^{\mathsf{s}}$. For this, we consider the equations (7) and (8). We note that this theory is still incomplete, as in general, one needs to cover any number of exponentiations in order to reason about security

and not only to be able to execute the protocol. Finally, we use equation (9) to model the verification of the proof produced for correct exponentiation.

Extraction of trackers and the Link ***events:*** As in Selene, due to the public information on $\mathsf{BBptr}(\mathsf{cr}, \mathsf{tpk}^{\mathsf{r}}, \pi_{\mathsf{exp}})$ and the zero-knowledge proof π_{exp}, we can extract the tracker tr from its pre-tracker $\mathsf{tpk}^{\mathsf{r}}$ and link it to the corresponding credential. Specifically, the extraction requires the following equation:

$$\mathsf{extract}(\mathsf{x}, \mathsf{pr}_{\mathsf{exp}}(\mathsf{x}, \mathsf{y}, \mathsf{z})) = \mathsf{z},$$

i.e. the function $\mathsf{extract}$ retrieves the term z from the proof π_{exp}. Note that pre-tracker $\mathsf{tpk}^{\mathsf{r}}$ is associated with a public credential cr on BBptr. Thus, we can link tr to cr after its extraction from $\mathsf{tpk}^{\mathsf{r}}$ and π_{exp}.

BBreg :	cr_1	…	cr_n	
BBtally :	$(\mathsf{cr}_1, \mathsf{tpk}_1, \mathsf{b}_1)$	…	$(\mathsf{cr}_n, \mathsf{tpk}_n, \mathsf{b}_n)$	
BBexp :	$(\mathsf{cr}_1, \mathsf{tpk}_1, \mathsf{b}_1, \mathsf{tpk}_1^{\mathsf{r}_1})$	…	$(\mathsf{cr}_n, \mathsf{tpk}_n, \mathsf{b}_n, \mathsf{tpk}_n^{\mathsf{r}_n})$	
	$\mathsf{b}_1 = (\mathsf{c}_1, \mathsf{s}_1, \mathsf{p}_1)$	…	$\mathsf{b}_n = (\mathsf{c}_n, \mathsf{s}_n, \mathsf{p}_n)$	
BBmix :	$(\mathsf{tr}_1, \mathsf{c}'_1)$	…	$(\mathsf{tr}_n, \mathsf{c}'_n)$	$\pi_{\mathsf{exp}} = (\pi_1 \circ \ldots \circ \pi_n), \pi_{\mathsf{renc}}$
BBptr :	$(\mathsf{cr}_1, \mathsf{tpk}_1^{\mathsf{r}_1}, \pi_1)$	…	$(\mathsf{cr}_n, \mathsf{tpk}_n^{\mathsf{r}^n}, \pi_n)$	
	π_1 : verified	…	π_n : verified	
	$\mathsf{tr}_1 = \mathsf{extract}(\mathsf{tpk}_1^{\mathsf{r}_1}, \pi_1)$	…	$\mathsf{tr}_n = \mathsf{extract}(\mathsf{tpk}_n^{\mathsf{r}^n}, \pi_n)$	
	$\mathsf{Link}(\mathsf{cr}_1, \mathsf{tr}_1)$	…	$\mathsf{Link}(\mathsf{cr}_n, \mathsf{tr}_n)$	
BBres :	$(\mathsf{tr}_1, \mathsf{v}_1)$	…	$(\mathsf{tr}_n, \mathsf{v}_n)$	π_{res}

Individual verification procedure. In Hyperion, voters compute the tracker tr with the secret t received from the talliers, raising it to their private trapdoor key tsk, i.e. $\mathsf{tr} = \mathsf{t}^{\mathsf{tsk}}$. If the tracker tr matches one of the trackers on BBres, then the voter can verify the vote v next to it. The secret should be in the form $\mathsf{t} = \mathsf{g}^{\mathsf{rs}}$ to allow voter to compute the corresponding tracker $\mathsf{tr} = \mathsf{tpk}^{\mathsf{rs}}$ computed from their pre-tracker. We assume that the voter's public trapdoor key is recorded on $\mathsf{BBvtr}(\mathsf{cr}, \mathsf{tpk})$. Thus, we similarly describe the flow on BB that allows voters to compute their trackers:

BBreg :	cr_1	…	cr_n	
BBvtr :	$(\mathsf{cr}_1, \mathsf{tpk}_1)$	…	$(\mathsf{cr}_n, \mathsf{tpk}_n)$	
	$\mathsf{tr}_1 = \mathsf{t}_1{}^{\mathsf{tsk}_1}$	…	$\mathsf{tr}_n = \mathsf{t}_n{}^{\mathsf{tsk}_n}$	
BBres :	$(\mathsf{tr}_1, \mathsf{v}_1)$	…	$(\mathsf{tr}_n, \mathsf{v}_n)$	π_{res}

We model the voter's trapdoor key generation and individual verification as specified in Figure 7.5.The process VoterVer models the individual verification of the voter id. The voter id, first, gets their private trapdoor key from the table Td and receives the secret t from T, i.e. the voter gets the entry t recorded for their credential cr from table Exp_2. Then, the voter computes their tracker with tsk, i.e. they compute $\mathsf{tr} = \mathsf{t}^{\mathsf{tsk}}$. The tracker tr allows the voter to

check the vote v next to it on BBres. If v matches the one recorded in table Voted, the voter successfully ends the individual verification.

```
let VoterTpk(id) =
  get Cred(= id, cr, sk_id) in
  new tsk;
  let tpk = g^tsk in
  let s_td = sign(tpk, sk_id) in
  let p_td = pr_exp(g, tsk, tpk) in
  let td = (tpk, s_td, p_td) in
  insert Td(id, tpk, tsk);
  insert BBtd(cr, td).

let VoterVer(id) =
  get Td(= id, tpk, tsk) in
  get Voted(= id, cr, = tpk, v) in
  get BBres(tr, = v) in
  get Exp_2(= cr, t) in
  if tr = t^tsk then
  event Verified(id, cr, v).
```

FIGURE 7.5: The processes specified for the voters in Hyperion.

Adversary models $\mathcal{A}$**.** We consider the same adversary models as Selene, referring to Table 7.1. The specifications of a corrupt registrar, server, voting platforms and voters are similar to the ones in Selene. In the specifications that $\mathcal{A}$ corrupts the talliers, we model the processes TallierKey, TallierExp, TallierMix, and TallierDec, as receiving the data they need to provide from the public channel, i.e. from $\mathcal{A}$.

7.3 Verification Results and Analysis

In this section, we present the verifiability analysis of Selene, SeleneRF, and Hyperion with respect to 7 corruption scenarios. For each scenario, we give the verification results of the automated verification with ProVerif, where its specification is available online [52]. As presented in Table 7.2, we have obtained the same verification results for each of those protocols. In Table 7.2, the symbol ✓ represents the successful verification, i.e. the security proof, whereas ✗ does the failure, i.e. an attack. We explain them in the following.

Adversary Models	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$	$\mathcal{A}_5$	$\mathcal{A}_6$	$\mathcal{A}_7$
Talliers	H	C	C	C	C	C	C
Registrar	H	H	C	H	H	C	H
Server	H	H	H	C	H	H	C
Voting Platform	H	H	H	H	C	C	C
SE2E[$iv_\bullet$, $res_\bullet$]	✓	✗	✗	✗	✗	✗	✗
SE2E[$iv_\circ$, $res_\bullet$]	✓	✓	✓	✓	✗	✗	✗
SE2E[$iv_\bullet$, $res_\circ$]	✓	✗	✗	✗	✗	✗	✗
SE2E[$iv_\circ$, $res_\circ$]	✓	✓	✓	✓	✓	✓	✓

TABLE 7.2: Verification results for Selene, SeleneRF and Hyperion.

Attacks. A trust assumption in Selene is that the voters performing verification are honest. On the other hand, the talliers can be fully corrupt. If both the verifying voter and the talliers are corrupt, there is a trivial attack against election verifiability. The main idea is that talliers can use the private trapdoor key of the voter to compute fake randomness that opens their commitment to a different tracker. If that tracker points to the same vote as chosen by the voter, this results in a clash on trackers, i.e. two voters open the same tracker. This scenario

is formally represented by the properties $\mathsf{iv}_\bullet$ (corrupt voters) and adversary models $\mathcal{A}_2 - \mathcal{A}_7$ (corrupt talliers) in Table 7.2.

Another notable attack is ballot stuffing (i.e. attack against $\mathsf{res}_\bullet$) when the voting platform is corrupt. Indeed, in that case, the adversary can replace the vote of the voter with any other vote, as soon as the voter does not verify their vote. This shows that the tracking mechanism in Selene protects voters against corrupt platforms only if they verify their votes, as expected.

Proofs. Notable cases are the positive results in corruption scenarios $\mathcal{A}_3, \mathcal{A}_4$ and $\mathcal{A}_6, \mathcal{A}_7$. Assuming that the voting platform is honest, in the scenarios $\mathcal{A}_3, \mathcal{A}_4$, the (standard) verifiability property, i.e. $\mathsf{SE2E}[\mathsf{iv}_\circ, \mathsf{res}_\bullet]$, is proved if either the registrar or the voting server is honest, like in Belenios. Note that the registrar is an abstraction we introduce for Selene to represent the credential generation mechanism. It does not exist as an independent party in Selene. Furthermore, if the voting platform is corrupt, the results for $\mathcal{A}_6, \mathcal{A}_7$ show that the weaker notion of end-to-end verifiability, i.e. $\mathsf{SE2E}[\mathsf{iv}_\circ, \mathsf{res}_\circ]$, holds, meaning that ballot stuffing is possible for honest voters only if they did not verify their votes.

Conclusion. Our analysis confirms that Selene provides end-to-end verifiability for honest voters who verify their votes in the election outcome, even if all the talliers and voting platforms are fully corrupt. We obtain the same verification results in all scenarios for Selene, SeleneRF, and Hyperion, implying that they provide the same end-to-end verifiability guarantees. Thus, SeleneRF is much more secure than Selene since it provides stronger receipt-freeness and the same level of end-to-end verifiability. On the other hand, Hyperion can be preferred over Selene due to the its efficient computation of trackers and similar verifiability guarantees. It also improves the receipt-freeness of Selene, preventing tracker collisions with the individual views of the bulletin board. However, the individual view of the bulletin board may not be practical considering the number of voters, e.g. in a large-scale election.

```
let AdminCand =
  in(pub, (v1, v2));
  event BBcand(v1);
  event BBcand(v2);
  insert BBcand(v1);
  insert BBcand(v2).

let AdminId =
  in(pub, id);
  insert Id(id).

let TallierKey =
  new skE;
  let pkE = pk(skE) in
  insert SkE(skE);
  event BBkey(pkE);
  insert BBkey(pkE);
  out(pub, pkE).

let RegistrarReg(id) =
  new skid;
  let cr = pk(skid) in
  insert Cred(id, cr, skid);
  event Reg(id, cr);
  event BBreg(cr);
  insert BBreg(cr);
  out(pub, cr).

let ServerPwd(id) =
  new pwd;
  insert Pwd(id, pwd).

let VoterTpk(id) =
  get Cred(= id, cr, skid) in
  new tsk;
  let tpk = pk(tsk) in
  insert Td(id, tpk, tsk);
  insert BBtpk(cr, tpk).

let AdminTr(pkE) =
  new tr′;
  let etr′ = enc(tr′, pkE, rfix) in
  insert BBtr(tr′, etr′).

let TallierEtr(pkE) =
  get BBtr(tr′, etr′) in
  new r;
  let etr = renc(etr′, pkE, r) in
  let prenc = prrenc(etr, tr′, r) in
  insert Etr(etr, prenc).

let AdminAssign(pkE) =
  get BBtpk(cr, tpk) in
  get BBtr(tr′, etr′) in
  get Etr(etr, prenc) in
  if verrenc(prenc, etr, tr′, pkE) = true then
  insert BBetr(cr, tpk, etr),
  insert BBptr(cr, tpk, etr, prenc, pkE).

let TallierCom(pkE, skE) =
  get BBetr(cr, tpk, etr) in
  get Etr(= etr, prenc) in
  new r, r1, r2;
  insert Com(cr, r);
  let a = enc(tpkr, pkE, r1) in
  let b = enc(r, pkE, r2) in
  let prand = prrand(a, b, r, r1, r2) in
  insert Compair(cr, tpk, a, b, prand);
  let cm = decom(a, etr, skE) in
  let pcom = prcom(tpk, a, b, etr, cm, skE, r) in
  insert Cm(cr, tpk, etr, cm, pcom).

let ServerBB =
  get BBetr(cr, tpk, etr) in
  get Compair(= cr, = tpk, a, b, prand) in
  get Cm(= cr, = tpk, = etr, cm, pcom) in
  if verrand(prand, a, b, pkE) = true then
  if vercom(pcom, tpk, a, b, etr, cm, pkE) = true then
  insert BBvtr(cr, tpk, cm),
  insert BBcast(cr, tpk, etr, cm, ⊥).
```

FIGURE 7.6: ProVerif specification for the setup procedure in Selene.

```
let VotingPlatformVote(v, pk_E) =
  get Cred(id, cr, sk_id) in
  get Pwd(= id, pwd) in
  new r;
  let c = enc(v, pk_E, r) in
  let s = sign(c, sk_id) in
  let p = pr_enc(c, v, r) in
  let b = (c, s, p) in
  let a = h(id, pwd, cr, b) in
  event Vote(id, v);
  insert Voted(id, cr, v);
  out(pub, (id, cr, b, a)).

let Extraction =
  get BBptr(cr, tpk, etr, p_renc, pk_E) in
  get BBtr(tr', etr') in
  if ver_renc(p_renc, etr, etr', pk_E) = true then
  let tr = extract(etr, pk_E) in
  event Link(cr, tr);
  insert Linked(cr).

let VoterVer(id) =
  get Td(= id, tpk, tsk) in
  get Voted(= id, cr, v) in
  get BBvtr(= cr, = tpk, cm) in
  get BBres(tr, = v) in
  get Com(= cr, r) in
  if open(cm, tsk, r) = tr then
  event Verified(id, cr, v).

let ServerCast(pk_E) =
  get BBcast(cr, tpk, etr, cm, = ⊥) in
  get Pwd(id, pwd) in
  in(pub, (= id, = cr, (c, s, p), a);
  event Log(id, cr);
  let b = (c, s, p) in
  if h(id, pwd, cr, b) = a then
  if verify(s, c, cr) = true then
  if ver_enc(p, c, pk_E) = true then
  insert BBcast(cr, tpk, etr, cm, b).

let ServerTally =
  get BBcast(cr, tpk, etr, cm, b) in
  insert BBtally(cr, tpk, etr, cm, b).

let TallierDec(sk_E) =
  get BBtally(cr, tpk, etr, cm, (c, s, p)) in
  let tr = dec(etr, sk_E) in
  let v = dec(c, sk_E) in
  let p_dec1 = pr_dec(etr, tr, sk_E) in
  let p_dec2 = pr_dec(c, v, sk_E) in
  insert Dec(etr, tr, p_dec1, c, v, p_dec2).

let ServerRes(pk_E) =
  get Linked(cr) in
  get BBtally(= cr, tpk, etr, cm, (c, s, p)) in
  get Dec(= etr, tr, p_dec1, = c, v, p_dec2) in
  if ver_dec(p_dec1, etr, tr, pk_E) = true then
  if ver_dec(p_dec2, c, v, pk_E) = true then
  event BBres(tr, v);
  insert BBres(tr, v);
  out(pub, (tr, v)).
```

FIGURE 7.7: ProVerif specification for ballot casting and tally procedures in Selene.

```
let AdminCand =
  in(pub, (v1, v2));
  insert BBcand(v1); BBcand(v2).

let AdminId =
  in(pub, id); insert Id(id).

let TallierKey =
  new skE; insert SkE(skE);
  event BBkey(pk(skE));
  insert BBkey(pk(skE)); out(pub, pk(skE)).

let RegistrarReg(id) =
  new skid; let cr = pk(skid) in
  insert Cred(id, cr, skid);
  event Reg(id, cr); BBreg(cr);
  insert BBreg(cr); out(pub, cr).

let ServerPwd(id) =
  new pwd; insert Pwd(id, pwd).

let VoterTpk(id, cr, skid) =
  new tsk; let tpk = g^tsk in
  let std = sign(tpk, skid) in
  let ptd = prexp(g, tsk, tpk) in
  insert Td(id, tpk, tsk); BBtd(cr, (tpk, std, ptd)).

let ServerBB(cr) =
  get BBtd(= cr, (tpk, std, ptd)) in
  if verify(std, tpk, cr) = true then
  if verexp(ptd, g, tpk) = true then
  insert BBcast(cr, tpk, ⊥).

let VotingPlatformVote(v, pkE) =
  get Cred(id, cr, skid) in
  get Td(= id, tpk, tsk) in
  get Pwd(= id, pwd) in
  new r; let c = enc(v, pkE, r) in
  let s = sign(c, skid) in
  let p = prenc(c, v, r) in
  let a = h(id, pwd, cr, tpk, (c, s, p)) in
  event Vote(id, v); insert Voted(id, cr, tpk, v);
  out(pub, (id, cr, tpk, (c, s, p), a)).

let ServerCast(pkE) =
  get BBcast(cr, tpk, = ⊥) in
  get Pwd(id, pwd) in
  in(pub, (= id, = cr, = tpk, (c, s, p), a);
  event Log(id, cr);
  if h(id, pwd, cr, tpk, (c, s, p)) = a then
  if verify(s, c, cr) = true then
  if verenc(p, c, pkE) = true then
  insert BBcast(cr, tpk, (c, s, p)).

let ServerTally =
  get BBcast(cr, tpk, b) in
  insert BBtally(cr, tpk, b).

let TallierExp =
  get BBtally(cr, tpk, b) in new r;
  insert Exp1(cr, g^r); BBexp(cr, tpk, b, tpk^r).

let TallierMix(s, pkE) =
  get BBexp(cr, tpk, (c, s, p), tpk^r) in
  get Exp1(= cr, g^r) in
  insert Exp2(cr, g^rs);
  let tr = tpk^rs in
  let pexp = prexp(tpk^r, s, tr) in
  new r'; let c' = renc(c, pkE, r') in
  let prenc = prenc(c', c, r') in
  insert BBmix(tr, pexp, c', prenc);
  insert BBptr(cr, tpk^r, tr, pexp).

let TallierDec(skE) =
  get BBmix(tr, pexp, c', prenc) in
  let pdec = prdec(c', dec(c', skE), skE) in
  insert BBdec(tr, c', dec(c', skE), pdec).

let Extraction =
  get BBptr(cr, tpk^r, tr, pexp) in
  if verexp(pexp, tpk^r, tr) = true then
  let tr = extract(tpk^r, pexp) in
  event Link(cr, tr);
  insert Linked(cr).

let ServerRes(pkE) =
  get Linked(cr) in
  get BBtally(= cr, tpk, b) in
  get BBexp(= cr, = tpk, = b, tpk^r) in
  get BBmix(tr, pexp, c', prenc) in
  get BBdec(= tr, = c', v, pdec) in
  if verexp(pexp, tpk^r, tr) = true then
  if verrenc(prenc, c', c, pkE) = true then
  if verdec(pdec, c, v, pkE) = true then
  event BBres(tr, v);
  insert BBres(tr, v);
  out(pub, (tr, v)).

let VoterVer(id) =
  get Td(= id, tpk, tsk) in
  get Voted(= id, cr, = tpk, v) in
  get BBres(tr, = v) in
  get Exp2(= cr, g^rs) in
  if tr = g^rstsk then
  event Verified(id, cr, v).
```

FIGURE 7.8: ProVerif specification of Hyperion.

Chapter 8

The Estonian E-Voting Protocol

Estonia has started to deploy internet voting for national elections in 2005. In the parliamentary elections of 2011, the rate of internet voters increased to 24,3% [50] among participating voters. With this high rate, the security of the adopted e-voting protocol has been under scrutiny, and an early paper [31] has shown that ballot manipulating attacks are possible by a corrupt voting device. Following this, the protocol was improved [32] to allow individual verifiability, allowing voters to verify their votes within a specified timeframe. The improved protocol was used in the local elections of 2013. In those elections, a group of researchers observed the election procedures, conducted some experiments, and prepared a technical report [49] on the security of the Estonian e-voting protocol. This report pointed out some implementation vulnerabilities and ghost-clicking attacks against the newly introduced individual verifiability mechanisms. The protocol has then been further improved [33] to have better verifiability, i.e. individual and universal verifiability, for the parliamentary elections of 2015. However, a recent attack [44] on individual verifiability shows that the efforts to make a more secure protocol are still not sufficient.

The Estonian e-voting protocol allows revoting to protect voters against coercion from its first deployment. Due to the first attack exploited by a corrupt voting device, the protocol has been improved to provide an individual verification mechanism [32], allowing voters to verify their last ballot cast only for a limited time after they cast. This mechanism is safe to protect the voters against coercion, i.e. it provides receipt-freeness since the receipt used for the verification becomes invalid after the verification time expires. In the recent version [33] of the protocol, the individual verification procedure has been modified to allow voters to verify any ballot within its specified verification timeframe, i.e. if the voter casts two ballots subsequently, they can verify the first within its verification timeframe even if the second ballot is the last in the voting server. This seems like to provide stronger receipt-freeness compared to the previous version since the voter does not have to wait for revoting if coerced. However, it brings some drawbacks since a corrupt voting device now can manipulate the receipts obtained from revoting and mislead the voter, as shown in [44].

In this chapter, we perform a security analysis for the recent version [33] of the Estonian e-voting protocol, applying our verification framework to evaluate its verifiability, and the standard privacy definition [24] to evaluate its privacy. Our analysis confirms the individual verifiability attack [44] when the voting device is corrupt. In addition, we discover ballot reordering and ballot copying attacks [19] when the network is corrupt. To prevent all mentioned attacks, we propose two protocol variants that improve its security. Our solutions achieve both ballot privacy and end-to-end verifiability, as we proved with ProVerif and Tamarin.

Structure of the chapter. This chapter includes five sections. In Section 8.1 and Section 8.2, we present the protocol structure and the Tamarin specification details of the Estonian e-voting protocol. In Section 8.3 and Section 8.4, we present the attacks we capture in our analysis and the solutions that improve the security of the protocol. Finally, in Section 8.5, we provide our verification results and verifiability analysis.

8.1 EEV Protocol Structure

There have been several iterations of the Estonian e-voting (EEV) protocol, for which several security analyses have been performed [31, 32, 49, 33]. We present an overview of the most recent version of the protocol, IVXV, from [33], which aims to fill the security gaps in previous versions and enable homomorphic tallying based on ElGamal encryption.

Smartcards and PKI. A Public Key Infrastructure (PKI) provides voters and protocol parties with signature key pairs certified by a Certification Authority (CA). All the materials sent between parties are signed with the sender's signing key and verified with the verification key inside the certificate. For eligible voters, the signing key is stored within a personal electronic identity card (EID) that allows them to sign messages of their choice securely.

Protocol parties. The EEV protocol is deployed by a central voting system that is maintained by multiple election parties, which are responsible for organising the election and computing the final result. Furthermore, several external parties provide additional functionalities, like ballot time-stamping, registration, and audit of election data. Altogether, we have the following parties:

Election parties: EO, VC, IBBP. ***External parties:*** CA, RS, TMS, DA, V.

EO :	Election Organiser;	CA :	Certification Authority;
VC :	Vote Vollector;	RS :	Registration Service;
IBBP :	I-Ballot Box Processor;	TMS :	Time Marking Service;
		DA :	Data Auditors;
		V :	Voters.

- EO is responsible for the election configuration, i.e. it determines the list of candidates and the list of eligible voters holding an EID card, which stores a signature key pair $(\mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}})$ certified by the CA for the voter id. EO is also responsible for the tally: it generates an election key pair $(\mathsf{pk}_{\mathsf{E}}, \mathsf{sk}_{\mathsf{E}})$, makes pk_{E} available to any party in the protocol, and decrypts the final ballots to be tallied.

- VC collects ballots from voters via their voting applications during the voting phase. It interacts with the TMS and RS to timestamp and acquire registration confirmation for each ballot. It stores all the received information and sends a receipt confirmation to the voter.

- IBBP starts the tally procedure by verifying all the ballots and registration confirmations in the lists received from VC and RS and determines the list of ballots to be tallied. It extracts the ciphertext from each ballot, determining the final list of ciphertexts to be decrypted (after anonymisation via a mixnet or homomorphic combination).

- TMS signs the ballot information received from VC to which it adds the time of request (if the corresponding voter certificate is still valid) and sends the signature back to VC.

- RS validates the ballot registration request from VC and signs it to generate the registration confirmation sent back to VC. Furthermore, RS stores the record in its list of records.

- DA audits all parties by: (1) verifying the list of ballots provided by VC for well-formedness and eligibility, i.e. the eligibility of voters who cast the ballots and the correctness of their signatures; (2) verifying the registration confirmations stored by RS and the consistency between the list from RS and the one from VC; (3) verifying the

correct processing of ballots by IBBP leading to final ballots to be tallied; this involves redoing the same procedure and performing consistency checks for the data obtained from VC and RS; and (4) verifying the proofs generated in the tally phase, e.g. the proofs of correct decryption or mixing.

– V uses a voting application to start a voting session with VC. In the session, VC authenticates V via their EID card and provides a list of candidates from which V selects their vote. The voting application generates a ballot for V's choice, sends it to VC, and receives a registration confirmation in return, which allows it to form a QR code. V uses a verification application installed on another device to verify their vote with the QR code provided by the voting application. The QR code allows the verification application to interact with VS, gets the corresponding ballot, and opens it to v for the individual verification of V. Against a corrupt voting application or a corrupt voting device, the voters are recommended to use a separate device to verify their votes. Individual verification with the QR is allowed only for a limited time after the ballot is cast, e.g. 30 minutes, and voters can revote during the election. In the rest of the chapter, we denote the voting and verification applications by VoteApp and VerApp, respectively.

Unlike other protocols, EEV does not rely on a public bulletin board. For simplicity of presentation, we assume a private bulletin board BB containing all the information that should be available to all parties in the election and in particular to data auditors. After the setup, BB includes the election's public key and eligibility information. We describe the election procedures and the individual verification procedure of EEV as follows:

Setup phase. EO generates the election key pair $(\mathsf{pk}_\mathsf{E}, \mathsf{sk}_\mathsf{E})$ and determines the list of candidates $\mathsf{v}_1, \ldots, \mathsf{v}_\mathsf{k}$ and voters $\mathsf{id}_1, \ldots, \mathsf{id}_\mathsf{n}$ that are eligible for the election. The election public key pk_E is shared with any party in the election, and the list of voters is made available to any authority. Each voter id holds an EID card which has the tuple $\langle \mathsf{id}, \mathsf{pk}_\mathsf{id}, \mathsf{sk}_\mathsf{id}, \mathsf{cert}_\mathsf{id} \rangle$ inside, where $\mathsf{cert}_\mathsf{id}$ is certified by CA as follows: $\mathsf{cert}_\mathsf{id} = \mathsf{sign}(\langle \mathsf{id}, \mathsf{pk}_\mathsf{id} \rangle, \mathsf{sk}_\mathsf{CA})$. Similarly, the services TMS, RS and VC hold certificates $\mathsf{cert}_\mathsf{TMS}$, $\mathsf{cert}_\mathsf{RS}$, and $\mathsf{cert}_\mathsf{VC}$ for their key pairs.

Voting phase. In Figure 8.1, we give an illustration of the voting phase. V uses their VoteApp to create a ballot $\mathsf{b} = \langle \mathsf{c}, \mathsf{s} \rangle$, where $\mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk}_\mathsf{E}, \mathsf{r})$ is the encryption of their chosen candidate v with fresh randomness r, and $\mathsf{s} = \mathsf{sign}(\mathsf{c}, \mathsf{sk}_\mathsf{id})$ is their corresponding signature. The EID card is first used to authenticate the voter to VC and then for signing the ciphertext containing their vote. The VoteApp sends the voter identity and the ballot to VC, which verifies the eligibility of the voter and the validity of the signature, acquires a timestamp on the ballot from the TMS, creates a fresh identifier vid for b, and registers vid, b with RS. If all checks and registration are successful, it stores the record in its database and sends the tuple $\langle \mathsf{vid}, \mathsf{reg} \rangle$ back to the VoteApp, where reg is a signature on $\langle \mathsf{vid}, \mathsf{b} \rangle$ by RS. The tuple $\langle \mathsf{vid}, \mathsf{reg} \rangle$ represents confirmation of the receipt of the ballot by VC and registration confirmation of that ballot by RS. The identifier vid, together with the encryption randomness r generated before, is used by VoteApp to display a QR code that can be used for verifying the ballot on a separate verification device.

Tally phase. During the voting phase, VC records the ballots received for the voter id in $\mathsf{Stored}_\mathsf{id}$, whereas RS records their registration confirmations in Reg. In the tally phase, IBBP collects Reg and $\mathsf{Stored}_\mathsf{id}$ for each voter id, then validates each ballot in $\mathsf{Stored}_\mathsf{id}$ and verifies that it is recorded in Reg. Conversely, it also checks that no registered ballot has been removed. If no problem is detected, IBBP retrieves the last ballot b^ℓ recorded in $\mathsf{Stored}_\mathsf{id}$ for each voter and extracts its ciphertext c^ℓ. The list of resulting ciphertexts $\mathsf{c}_1, ..., \mathsf{c}_\mathsf{n}$ is anonymised by re-encryption or homomorphic combination, and the resulting combined ciphertext (or list of ciphertexts) is sent to EO for decryption. EO uses the election private key sk_E to decrypt the

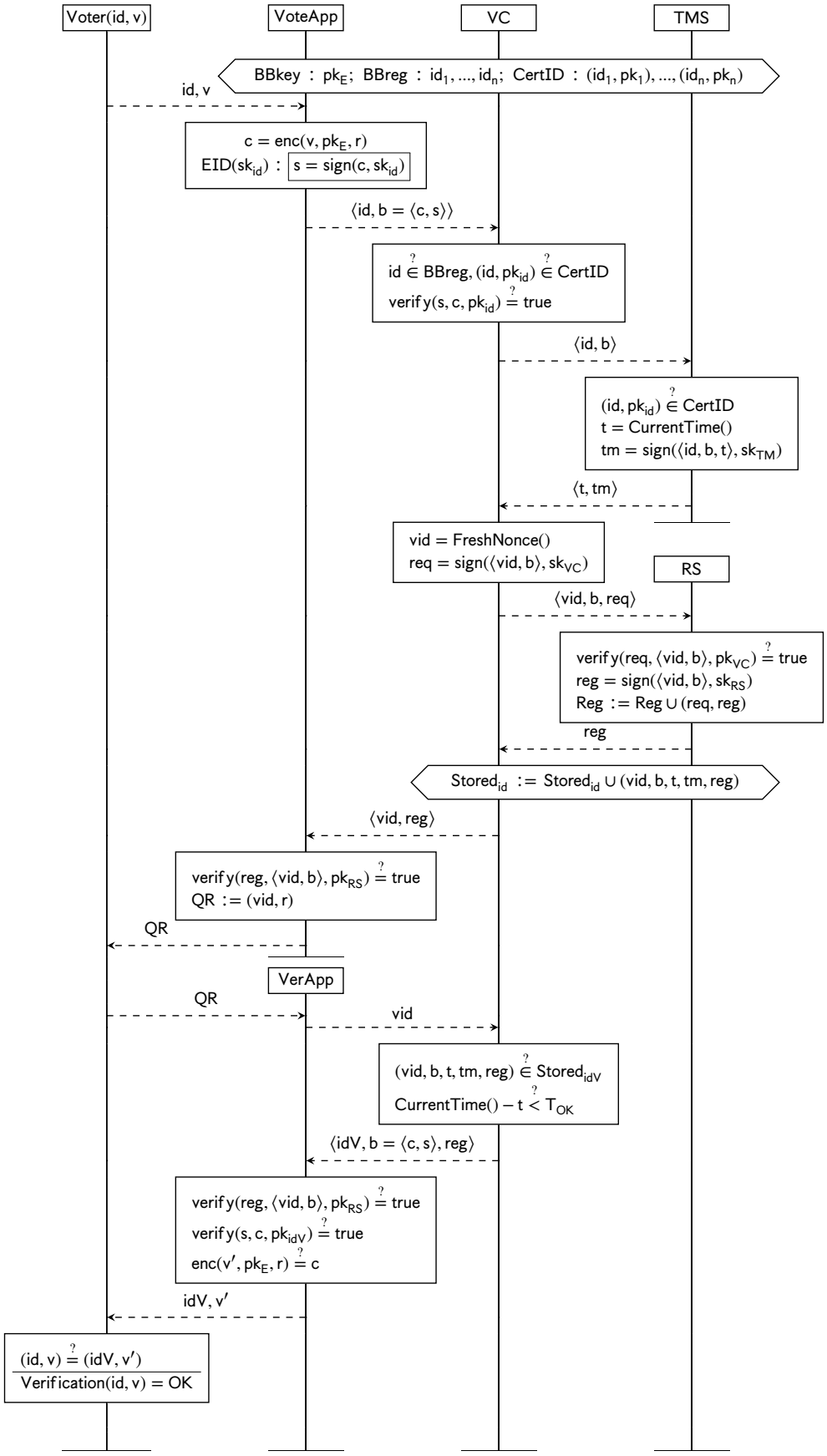


FIGURE 8.1: Illustration of EEV protocol.

ciphertext(s), publishing the outcome and the proof of correct decryption. All these operations can be audited by DA, who can request the necessary data from VC, RS, IBBP, or read it from a jointly maintained bulletin board.

Individual verification. During the voting phase, voters can verify that their ballots reached the VC and that they encode their desired votes. For this, they scan the QR $= (\mathsf{vid}, \mathsf{r})$ code displayed on their VoteApp (after they submit their ballot) with their VerApp, which sends vid to VC to request the ballot recorded for that vid. The VC retrieves the tuple $\langle \mathsf{id}, \mathsf{b} = \langle \mathsf{c}, \mathsf{s} \rangle, \mathsf{reg} \rangle$ corresponding to vid from its database and sends the tuple back. The VerApp first verifies s and reg. Second, using the randomness r from the QR code, it can determine whether the ciphertext c recorded by VC encodes a valid vote v', by simply recomputing the encryption algorithm for eligible candidates. In this case, v' is displayed to the voter, who concludes successful verification if v' matches the expected vote v. Note that VC allows verification of any vote cast within a timeframe, and not only of the last vote. This will be exploited by Pereira's attack discussed later.

Receipt-freeness. The EEV allows individual verification only within a timeframe specified by the protocol, usually 30 minutes. This feature together with revoting helps to achieve receipt-freeness. Indeed, after the timeframe expires, the QR code becomes useless to learn which vote was cast, and the voter can undetectably revote. Thus, the QR code cannot effectively function as a receipt. Another feature that somehow complements this one is that, even if the QR code corresponds to a ballot that is not the last one cast, the individual verification will be successful as long as we are within the prescribed timeframe. Thus, the coercer will not know whether the vote they desire will be counted.

8.2 Tamarin Specification of the EEV protocol

In this section, we present the Tamarin specification of the EEV protocol, focusing on the details specific to EEV . These are its equational theory, the extraction of trackers and the linking events, and its individual verification procedure for voters. At the end of the section, we provide the full specification of the EEV protocol in Figure 7.6 and Figure 7.7.

Equational theory. The EEV protocol relies on ElGamal encryption and digital signature scheme. Thus, cryptography requires the following two equations:

$$\begin{aligned} &(1) \quad \mathsf{dec}(\mathsf{enc}(\mathsf{x}, \mathsf{pk}(\mathsf{y}), \mathsf{z}), \mathsf{y}) = \mathsf{x}, \\ &(2) \quad \mathsf{verify}(\mathsf{sign}(\mathsf{x}, \mathsf{y}), \mathsf{x}, \mathsf{pk}(\mathsf{y})) = \mathsf{true}, \end{aligned}$$

where they are used for the encryption/decryption and signing/verification, respectively.

EID ***cards and certificates:*** In this specification, we do not model the revocation of the certificates during the election. We assume that each voter is registered to the election with their valid certificates, and they remain valid during the election. Since CA is assumed to be honest, we model the generation of EID cards and certification of the public keys stored in EID cards with a single rule as follows:

$$\mathsf{R}_{\mathsf{eid}} : \quad \mathsf{let}\ \ \mathsf{pk}_{\mathsf{id}} = \mathsf{pk}(\mathsf{sk}_{\mathsf{id}})\ \ \mathsf{in}$$
$$[\ \mathsf{In}(\mathsf{id}), \mathsf{Fr}(\mathsf{sk}_{\mathsf{id}})\ ] - [\ ] \rightarrow [\ !\mathsf{CertID}(\mathsf{id}, \mathsf{pk}_{\mathsf{id}}), !\mathsf{EID}(\mathsf{id}, \mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}), \mathsf{Out}(\mathsf{pk}_{\mathsf{id}})\ ]$$

Similarly, we specify a single rule for the signing key pair generation of the election parties and the certification of their public key.

Voting and QR ***codes:*** In e-voting protocols, typically, the ballot casting procedure consists of just submitting the generated ballot to the voting server. In the EEV protocol, the communication is two-way, i.e. after the ballot is submitted, a registration confirmation is received

from VC, where it is used to form a QR code for the verification of the vote encoded in the respective ballot. Thus, we model two rules for the actions of the VoteApp instead of one as we done in the previous chapters.

$$\begin{aligned} \mathsf{R}_{\mathsf{vote}}: \quad & \mathsf{let}\ \ c = \mathsf{enc}(v, \mathsf{pk}_E, r);\ \ s = \mathsf{sign}(c, \mathsf{sk}_{id});\ \ b = \langle c, s\rangle\ \ \mathsf{in} \\ & [\ !\mathsf{Voter}(id), !\mathsf{EID}(id, \mathsf{pk}_{id}, \mathsf{sk}_{id}), !\mathsf{BBcand}(v), !\mathsf{BBkey}(\mathsf{pk}_E), \mathsf{Fr}(r), \mathsf{Fr}(t_v)\ ] \\ & -[\ \mathsf{Vote}(id, v, t_v), \mathsf{VoteB}(id, b), \mathsf{VoteTime}(id, v, t_v)\ ]\rightarrow[\ \mathsf{St}(id, v, b, r, t_v), \mathsf{Out}(\langle id, b\rangle)\ ] \end{aligned}$$

$$\begin{aligned} \mathsf{R}_{\mathsf{vid}}: \quad & [\ \mathsf{In}(\langle vid, reg\rangle), \mathsf{St}(id, v, b, r, t_v), !\mathsf{CertRS}(\mathsf{pk}_{RS})\ ] \\ & -[\ \mathsf{verify}(reg, \mathsf{h}(\langle vid, \mathsf{h}(b)\rangle), \mathsf{pk}_{RS}) \equiv \mathsf{true}\ ]\rightarrow[\ !\mathsf{Voted}(id, v, t_v, vid, r)\ ] \end{aligned}$$

In the rule $\mathsf{R}_{\mathsf{vote}}$, the ballot b is generated for the voter id, signed with sk_{id} stored in EID, and sent to VC. The vote information to be used in the second rule is recorded in St. After receiving the ballot, VC processes it and sends its registration confirmation reg with a vote identifier vid. VoteApp receives the tuple $\langle vid, reg\rangle$, verifies the registration confirmation with the public key of RS and records the vote information in Voted. The terms vid, r represent the QR code formed by the VoteApp. The randomness r inside the code allows VerApp to encrypt all the candidates and find the one matching with the ciphertext.

Timeframes and individual verification. EEV allows individual verification only within a specific timeframe after the ballot is cast. To model a timeframe, we have two rules that determine its start, its end and its public label $t, to which the protocol parties can refer for determining its status:

$$\begin{aligned} \mathsf{R}_{\mathsf{start}}: \quad & [\ ]-[\ ]\rightarrow[\ !\mathsf{StartTime}(\$t)\ ] \\ \mathsf{R}_{\mathsf{end}}: \quad & [\ !\mathsf{StartTime}(\$t)\ ]-[\ \mathsf{EndTime}(\$t)\ ]\rightarrow[\ ] \end{aligned}$$

The notation $t in Tamarin means that the first rule can create an unbounded number of timeframes with different public labels $t. When VC processes a ballot, it determines its timeframe $t, consuming the fact StartTime, and stores the corresponding label in a fact Ver(vid, $t) that will be used to prohibit the verification of the corresponding ballot if its timeframe has expired. The rule modelling the individual verification procedure is the following:

$$\begin{aligned} \mathsf{R}_{\mathsf{ver}}: \quad & \mathsf{let}\ \ b = \langle c, s\rangle;\ \ c' = \mathsf{enc}(v, \mathsf{pk}_E, r)\ \ \mathsf{in} \\ & [\ !\mathsf{Voted}(id, v, t_v, vid, r), !\mathsf{Stored}(id, \mathsf{pk}_{id}, b, vid, t_s, tm, reg), \\ & \ \ !\mathsf{Ver}(vid, t), !\mathsf{BBkey}(\mathsf{pk}_E), !\mathsf{CertID}(id, \mathsf{pk}_{id}), !\mathsf{CertRS}(\mathsf{pk}_{RS})\ ] \\ & -[\ \mathsf{verify}(s, c, \mathsf{pk}_{id}) \equiv \mathsf{true}, \mathsf{verify}(reg, \mathsf{h}(\langle vid, \mathsf{h}(b)\rangle), \mathsf{pk}_{RS}) \equiv \mathsf{true}, \\ & \ \ c \equiv c', \mathsf{CheckTime}(t), \mathsf{Verified}(id, v, t_v)\ ]\rightarrow[\ ] \end{aligned}$$

The verification of the timeframe in this rule is modelled by the action fact CheckTime(t) in conjunction with a restriction. The following restriction on the action fact CheckTime ensures that individual verification cannot be performed after the timeframe has expired:

$$\Psi_{\mathsf{ver}}: \quad \mathsf{CheckTime}(x) \Rightarrow \neg\mathsf{EndTime}(x)$$

In addition to the timeframe, the rule verifies the signature on ballot b and registration confirmation reg. Then, it computes a ciphertext c' using the randomness r in the QR code, as recorded in Voted. If c' matches with the ciphertext on the ballot, verification is completed recording an action fact $\mathsf{Ver}(id, v, t_v)$.

Adversary models $\mathcal{A}$. We consider the adversary A with the abilities to corrupt underlying infrastructure, i.e. the communication network, voters, voting applications, and the election parties: RS and VC. At least one of the election parties, VC or RS, should be honest since

they could collude without being noticed by DA. They could, for example, send a registration confirmation to the voter, even though they do not store the ballot for the final tally. The verification application is also assumed honest. We assume a set of corrupt voters, whose credentials are leaked to the attacker. In most models we assume a corrupt communication network, but sometimes we assume an honest network to test whether attacks are also possible in that case. We also assume CA and TMS are honest, but IBBP can be corrupt since its actions can be audited DA. Specifically, we define seven scenarios from $\mathcal{A}_1$ to $\mathcal{A}_6$, as presented in Table 8.1.

Adversary Models	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$	$\mathcal{A}_5$	$\mathcal{A}_6$
Network	H	C	H	C	C	C
VoteApp	H	H	C	C	C	C
RS	H	H	H	H	C	H
VC	H	H	H	H	H	C

TABLE 8.1: Adversary models for the EEV protocol.

- ***An honest network*** allows the communication between VoteApp and VC to be through a private channel. Thus, VC processes the multiple ballots received in the order of they have been cast.

- ***A corrupt network*** allows $\mathcal{A}$ to remove, insert, and reorder ballots between VoteApp and VC.

– ***A corrupt*** VoteApp allows $\mathcal{A}$ to determine the ciphertext that will be used in the cast ballot. In this case, the EID card is assumed to be secure, i.e. it is not compromised and keeps the signing key secret. Thus, VoteApp uses the EID card to sign the ballot. VoteApp may also manipulate the registration confirmation received by the voter, allowing QR code to be manipulated by $\mathcal{A}$.

- ***A corrupt*** RS leaks its signing key to $\mathcal{A}$, allowing it to sign any ballot information from VC.

- ***A corrupt*** VC leaks its signing key to $\mathcal{A}$, reveals the vote identifier vid that is generated to store a ballot, and does not perform any check on the stored ballots. It may also allow individual verification even if the timeframe for that vid was expired.

– ***Corrupt voters*** leak all their credentials to $\mathcal{A}$, including the signing key pairs stored in their EID cards.

8.3 Attacks against the EEV Protocol

In this section, we present three attacks against the EEV protocol, for which two are against its verifiability, and one is against its privacy. The first verifiability attack proposed in [44] by Pereira is based on vote manipulation by a corrupt voting application or device without being detected by the voter. The second verifiability attack is ballot reordering, in which the adversary reorders the ballots cast by a voter corrupting the communication network. The third attack is a ballot copying attack against the privacy of the protocol, where the adversary copies the ballot of the targeted voter and learns the vote inside, just looking at the outcome. In our analysis, we capture the verifiability attacks with our framework, and the privacy attack using the standard definition of [24] on our ProVerif models.

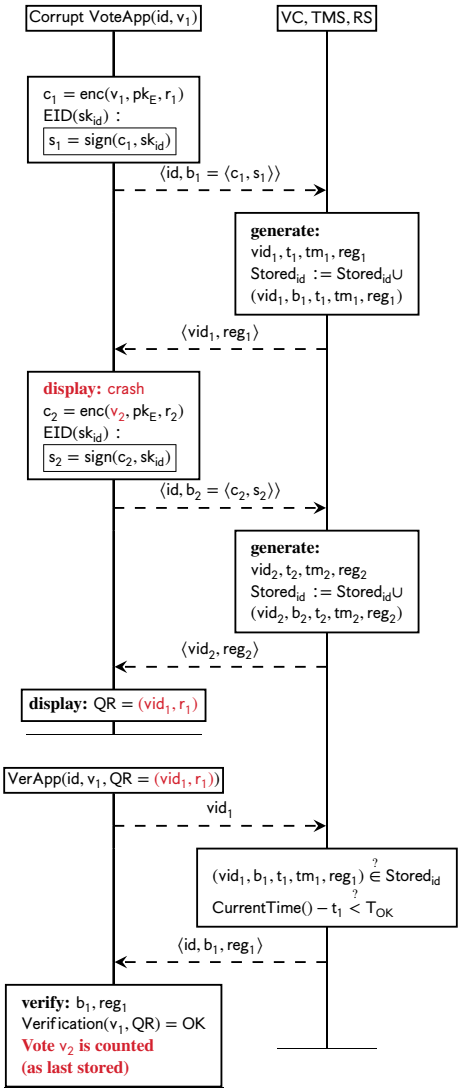
Pereira's attack [44] against verifiability. This attack assumes a corrupt voting application/device that manipulates the session started by the voter and does not require a corrupt network. As in Figure 8.2a, the voter id starts a session with VC and submits a ballot b_1 for the desired vote v_1. The VC processes b_1 and sends its confirmation $\langle \mathsf{vid}_1, \mathsf{reg}_1 \rangle$ back. Instead of displaying the $\mathsf{QR} = (\mathsf{vid}_1, \mathsf{r}_1)$ for the verification of v_1, the corrupt $\mathsf{VoteApp}$ displays crash on the screen. Thus, the voter attempts to generate another ballot for v_1; however, the corrupt $\mathsf{VoteApp}$ encrypts v_2 instead and submits the corresponding b_2 (here v_2 is the vote desired by the adversary). After receiving confirmation for b_2, $\mathsf{VoteApp}$ displays the first ballot's $\mathsf{QR} = (\mathsf{vid}_1, \mathsf{r}_1)$ for verification. Then, the voter verifies v_1 with QR code using an honest VerApp, and since all the verification checks pass, the voter expects v_1 to be tallied. However, b_2 corresponding v_2 is selected for id in the tally phase. This attack targets the individual verifiability of the voter, exploiting the verification procedure allowing any vote's verification if its timeframe has not expired. It also violates the result integrity since the corrupt voting application/device stuffs a ballot that the adversary desires.

Ballot reordering attack against verifiability. We discover a ballot reordering attack with the corrupt network, similar to the one found for Belenios in Chapter 6. Here it is notable that the voting application can be honest. In this attack, the adversary manipulates the order of the ballots when the voter revotes successively. As in Figure 8.2b, the voter id casts two successive ballots b_1 and b_2, corresponding to v_1 and v_2. The adversary $\mathcal{A}$ in the network blocks the first ballot b_1, and submits the second ballot b_2, as the first. The VC processes b_2 and sends its confirmation to the voter. Before the session ends, $\mathcal{A}$ achieves to submit b_1 and blocks its confirmation. The voter verifies v_2 as the last vote cast; however, the ballot b_1 corresponding to v_1 is tallied as last stored, which violates individual verifiability.

Ballot copying attack against privacy. We capture a ballot copying attack against privacy, which belongs to a class first described in [19] against Helios. In its simplest version, the attack consists in copying the ballot cast by one honest voter and recasting it in the name of a dishonest voter. This is sufficient to violate privacy in simple scenarios: assume two honest voters A and B and one corrupt voter C that cast ballots $\mathsf{b}_\mathsf{A} = \langle \mathsf{c}_\mathsf{A}, \mathsf{s}_\mathsf{A} \rangle$, $\mathsf{b}_\mathsf{B} = \langle \mathsf{c}_\mathsf{B}, \mathsf{s}_\mathsf{B} \rangle$, and $\mathsf{b}_\mathsf{C} = \langle \mathsf{c}_\mathsf{A}, \mathsf{s}_\mathsf{C} \rangle$, respectively, where the adversary copies the ciphertext c_A in A's ballot to generate a ballot for C. If the result comes out, e.g. as two a's and one b, the attacker infers that A voted for a. As shown in [42], this type of attack also leads to privacy violations in more general scenarios as well. If revoting is disallowed, this attack can be countered by ballot weeding by auditors. However, if revoting is allowed, the adversary can use one of the two ciphertexts and submit it for a malicious voter, which would not be detected by weeding.

In the case of EEV, the ballot copying attack is possible as soon as the voting network is corrupt and revoting is performed. In practice, one may consider that the connection to VC happens over a secure channel. Since there is no public bulletin board, the attack is then not as simple to perform as in Helios. However, stronger security is desirable, as the adversary may compromise the secure channel as well. Another attack on privacy was recently described in [43]. This attack is outside the scope of our model as it uses the homomorphic properties of the encryption scheme in order to modify the vote inside the ballot and make it equal to a value that, when published in the outcome, will be an outlier that will give a hint about the original value of the vote or will leak the votes of other voters. Both attacks can be prevented by adding a zero-knowledge proof to the ballot. In order to counter our attack, in addition to the non-malleability of the ballot (that is sufficient to counter the attack in [43]), the zero-knowledge proof will need to make sure that the ciphertext cannot be detached from the public key of the voter.

(A) Pereira's attack.



(B) Ballot reordering attack.

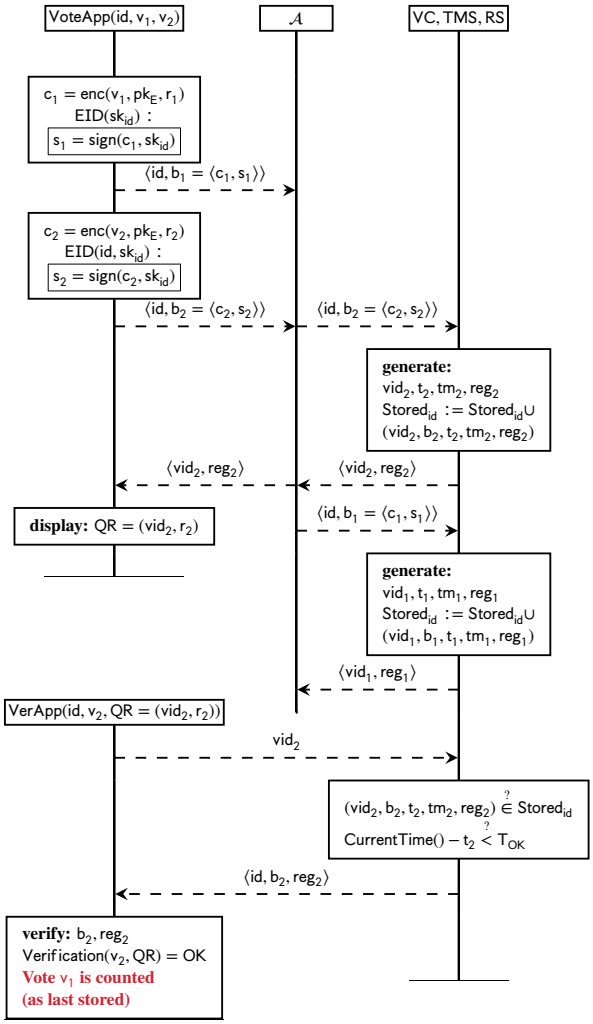


FIGURE 8.2: Illustration of the verifiability attacks against EEV protocol.

8.4 Solutions to Attacks

We consider the problem of strengthening the EEV protocol to counter the privacy and individual verifiability attacks presented in Section 8.3 and prove the resulting protocol secure considering the adversary models from Table 8.1. Our attacks are related to the verifiability attacks in [44] and the privacy attacks in [43]. These papers have discussed possible solutions and informally argued why they should help. However, Pereira [44] cautioned that some of the suggested improvements may not protect in all corruption scenarios and may also affect receipt-freeness. In our attempts at formal verification, we could confirm that the improvements suggested in [44] are not sufficient to obtain security in general. However, we show that one of the proposed solutions can be proved secure (for verifiability) in a stronger corruption model than suggested in [44] by making a small change to the communication infrastructure and asking the voter to perform additional verification steps. All variants proposed in [44] are still subject to the privacy attack, and furthermore, the addition of the zero-knowledge proof proposed in [43] is not sufficient to protect against the ballot copying attack. We propose two distinct improved variants of EEV that build upon the ideas from [44, 43] and we prove them

to be secure with respect to verifiability and receipt-freeness. The difference between the two variants is in the usability of the verification procedure: improved usability comes at the price of decreased security, both for the verifiability and the receipt-freeness properties.

8.4.1 Insufficiency of Current Solutions

Four approaches are proposed in [44] to mitigate Pereira's attack. Two of them are clearly in conflict with receipt-freeness in EEV, i.e. providing a public bulletin board or accepting at most one ballot per voter, and thus we do not consider them. Two further variants are as follows:

$$1.\ \mathsf{EEV}_{\mathsf{Ntfy}} \equiv \mathsf{EEV} + \mathsf{Ntfy}(\mathsf{id}, \mathsf{vid}) \qquad 2.\ \mathsf{EEV}_{\mathsf{Last}} \equiv \mathsf{EEV} + \mathsf{Last}(\mathsf{id}, \mathsf{b}).$$

The variant $\mathsf{EEV}_{\mathsf{Ntfy}}$ uses a feedback channel that informs the voter each time a ballot is cast for their respective id; this feedback should contain the identifier vid of the ballot. We represent this feature using the notation $\mathsf{Ntfy}(\mathsf{id}, \mathsf{vid})$. The variant $\mathsf{EEV}_{\mathsf{Last}}$ allows voters to verify only the last ballot stored in their name. We represent this feature by $\mathsf{Last}(\mathsf{id}, \mathsf{b})$. Intuitively, $\mathsf{EEV}_{\mathsf{Ntfy}}$ should prevent Pereira's attack because the corrupt $\mathsf{VoteApp}$ cannot hide anymore from the voter that their first ballot has been correctly cast. One problem with this approach mentioned in [44] is that the corrupt $\mathsf{VoteApp}$ may attempt to alter the vid in confirmations. However, suppose the feedback channel is implemented on a different device, e.g. a mobile phone that also acts as a verification device. In that case, we can distribute trust by asking the voter to confirm that the identifier vid is the same on both devices. This is the main addition to $\mathsf{EEV}_{\mathsf{Ntfy}}$ that we perform in one of our improved EEV variants. Another question left open by [44] is who should be responsible for the notification. The most obvious party is VC, but if the attacker also corrupts this party in addition to $\mathsf{VoteApp}$, then the security breaks down.

Concerning $\mathsf{EEV}_{\mathsf{Last}}$, if VC is honest, Pereira's attack should not be possible: the voter is not able to verify the ballot corresponding to vid_1 after the ballot corresponding to vid_2 is cast. If VC is malicious, it could, for example, delay the registration and storage of the second ballot until after the voter verifies their first ballot. Furthermore, in our adversary model, where we assume a corrupt network, the adversary could delay the second ballot even without corrupting the VC. The ballot reordering attack is also possible in this case. We will solve these problems by adding a simple notification on a feedback channel every time a new ballot is cast for the voter. The notification is simplified with respect to $\mathsf{EEV}_{\mathsf{Ntfy}}$, since it does not have to include the vid, only to notify voters that a ballot is cast in their name.

The privacy attacks from [43] are based on corrupting the ballot of an honest voter so that the vote encoded inside can reveal information about the initial vote cast. The solution proposed in [43] to mitigate this attack is to add a zero-knowledge proof that ensures the ballot cannot be modified without detection. Formally, these zero-knowledge proofs can be expressed by the equation $\mathsf{ver}(\pi, \mathsf{enc}(\mathsf{v}, \mathsf{pk}_\mathsf{E}, \mathsf{r}), \mathsf{pk}_\mathsf{E}) = \mathsf{true}$ where $\pi = \mathsf{zkp}(\mathsf{enc}(\mathsf{v}, \mathsf{pk}_\mathsf{E}, \mathsf{r}), \mathsf{v}, \mathsf{r})$ is a non-malleable zero-knowledge proof that will be invalid for any ciphertext c' different from $\mathsf{enc}(\mathsf{v}, \mathsf{pk}_\mathsf{E}, \mathsf{r})$. This prevents the vote v inside the ciphertext constructed by an honest voter from being modified without detection. However, this does not prevent our ballot copying attack, since the adversary can take a ballot, extract the ciphertext and the zero-knowledge proof, and reuse them without modification to cast a vote on behalf of a malicious voter. To prevent this from happening, we will use an enhanced version of the zero-knowledge proof that allows attaching a label to the ciphertext. Similarly to the solution used in Belenios [18], we can then use the voter's public key as a label, which will prevent ballots from being copied from one voter to another.

8.4.2 Our Solutions: EEV_* and EEV_+

We propose two variants of EEV, where we improve the solutions from [44] to achieve secure individual verifiability. We note that ballot stuffing is still possible for voters that do not verify their ballots if their voting applications are corrupt. We also use an enhanced version of the zero-knowledge proof to counter the privacy attacks. We obtain two protocol variants as follows:

1. $\mathsf{EEV}_* \equiv \mathsf{EEV} + \mathsf{Ntfy}(\mathsf{id}, \mathsf{vid}) + (\text{voter checks}) + (\text{EO BB}) + (\text{labeled ZKP})$,
2. $\mathsf{EEV}_+ \equiv \mathsf{EEV} + \mathsf{Last}(\mathsf{id}, \mathsf{b}) + \mathsf{Ntfy}(\mathsf{id}) + (\text{EO BB}) + (\text{labeled ZKP})$.

In Figure 8.3, we highlight the additions of EEV_+ and EEV_* to EEV in red.

EEV_* augments $\mathsf{EEV}_{\mathsf{Ntfy}}$ with explicit voter checks to counter a corrupt VoteApp, and with a private bulletin board BB that will be observed by EO and audited by DA to monitor the state of registered and stored ballots. Recall that one problem with $\mathsf{EEV}_{\mathsf{Ntfy}}$ as proposed in [44] is that a corrupt VC could manipulate the notifications on the feedback channel. Instead, we propose this common bulletin board for the election parties to monitor each other and to allow the EO (or the DA) to notify voters that a ballot is recorded in their name. The notification on the feedback channel is (id, vid) for every ballot processed by EO. Then, we require the voter to check that the identifier vid from the latest notification matches the identifier vid' from the $\mathsf{QR} = (\mathsf{vid}', \mathsf{r})$ code displayed by the VoteApp for the ballot to be verified.

EEV_+ augments $\mathsf{EEV}_{\mathsf{Last}}$ with a notification on a feedback channel, but which does not require sophisticated voter checks, e.g. comparing two distinct vid's like in EEV_*. The notification contains only the information that a ballot was cast in the name of that voter. Then, we require the voter to consider verification failed if a notification arrives after they perform the verifiability check. This makes $\mathsf{EEV}_{\mathsf{Last}}$ more usable and more intuitive, since it is natural to expect that only the last ballot should count. However, we find with Tamarin that this is not sufficient to obtain individual verifiability when the VC is corrupt, so only the ballot reordering attack is countered, and not Pereira's. The property of receipt-freeness is also weakened, as we explain below.

In both EEV_* and EEV_+, the EO/DA monitor the private bulletin board to detect ballots registered by RS and stored by VC. Recall that we assume that either the VC or the RS is trusted, thus the state of the bulletin board will reflect the final state agreed between parties. Thus, EO can ensure that a ballot is stored, registered and will not be removed. Whenever EO notices a ballot reaching the agreed upon store, it sends a notification through the feedback channel to the voter who cast the ballot.

Labeled zero-knowledge proofs. To prevent ballot copy attacks, the VC must ensure that no ciphertext in any ballot is accepted twice. For that, we adopt the solution provided for Belenios [18], i.e. we improve ballot structure by adding zero-knowledge proof labeled with the public key $\mathsf{pk}_{\mathsf{id}}$ of the voter that proves the public key owner created the ciphertext knowing the encryption randomness r. Implementing zero-knowledge proofs in this way will not raise difficulties in the EEV protocol since every eligible voter id holds an EID card keeping the signature pair $(\mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}})$. Formallly, the ballot structure will be improved as $\mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{p} \rangle$, where $\mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk}_{\mathsf{E}}, \mathsf{r})$, $\mathsf{s} = \mathsf{sign}(\mathsf{c}, \mathsf{sk}_{\mathsf{id}})$, and $\mathsf{p} = \mathsf{zkp}(\mathsf{enc}(\mathsf{v}, \mathsf{pk}_{\mathsf{E}}, \mathsf{r}), \mathsf{v}, \mathsf{r}, \mathsf{pk}_{\mathsf{id}}))$, and p is verified through the equation:

$$\mathsf{ver}(\mathsf{zkp}(\mathsf{enc}(\mathsf{v}, \mathsf{pk}_{\mathsf{E}}, \mathsf{r}), \mathsf{v}, \mathsf{r}, \mathsf{pk}_{\mathsf{id}}), \mathsf{enc}(\mathsf{v}, \mathsf{pk}_{\mathsf{E}}, \mathsf{r}), \mathsf{pk}_{\mathsf{E}}, \mathsf{pk}_{\mathsf{id}}) = \mathsf{true}.$$

Verifiability vs Receipt-freeness vs Usability. Comparing the two variants, Tamarin shows that EEV_* provides stronger verifiability since we obtain a security proof even with a corrupt VC, which is not the case for EEV_+. Indeed, the notifications in Pereira's attack could both

$\mathcal{A}_j/\Phi$	EEV			EEV+			EEV $*$		
	Φ_{iv}	$\Phi^{\bullet}_{E2E}$	Φ°_{E2E}	Φ_{iv}	$\Phi^{\bullet}_{E2E}$	Φ°_{E2E}	Φ_{iv}	$\Phi^{\bullet}_{E2E}$	Φ°_{E2E}
$\mathcal{A}_1$	✓	✓	✓	✓	✓	✓	✓	✓	✓
$\mathcal{A}_2$	✗	✗	✗	✓	✓	✓	✓	✓	✓
$\mathcal{A}_3$	✗	✗	✗	✓	✗	✓	✓	✗	✓
$\mathcal{A}_4$	✗	✗	✗	✓	✗	✓	✓	✗	✓
$\mathcal{A}_5$	✗	✗	✗	✓	✗	✓	✓	✗	✓
$\mathcal{A}_6$	✗	✗	✗	✗	✗	✗	✓	✗	✓

TABLE 8.2: Verification results for the EEV protocol.

happen before the voter performs verification. EEV_* also provides stronger receipt-freeness since allowing to verify the last ballot, like in EEV_+, opens a window where the coercer can detect a misbehaving voter - if the voter revotes while the coercer's ballot is still within its validity timeframe. However, EEV_+ is more usable. The voter does not need to perform any check related to the notification as it is in EEV_*, except for ensuring that no additional notification is received. In EEV_*, the voter should know for which vid they get a notification and which vid they use to verify their ballot. Thus, they should handle vid correctly to ensure individual verifiability and be careful about the order of vid in case of revoting.

8.5 Verification Results and Analysis

In this section, we present the verifiability analysis of the EEV protocol with respect to six corruption scenarios described in Table 8.1. For each scenario, we give the verification results of the automated verification with Tamarin, where its specification is available online [52]. As presented in Table 8.2, we have improved the individual verifiability and end-to-end verifiability of the EEV protocol. In Table 8.2, the symbol ✓ represents the successful verification, i.e. the security proof, whereas ✗ does the failure, i.e. an attack. We explain them in the following.

Note that we apply Definition 5 to the EEV protocol, where $\mathsf{tr} = \mathsf{cr} = \mathsf{id}$ and $\nabla = \circ$. In this case, Φ°_{iv2} is trivial. Thus, we check the following property on the Tamarin models of the EEV protocol:

$$\mathsf{SE2E}[\mathsf{iv}_\circ, \mathsf{res}_\diamond] \quad = \quad \Phi^{\circ}_{iv1} \wedge \Phi^{\circ}_{iv3} \wedge \Phi_{eli} \wedge \Phi^{\diamond}_{res} \wedge \Phi_{one},$$

where $\diamond = \{\bullet, \circ\}$. For simplicity, we denote Φ°_{iv1}, $\mathsf{SE2E}[\mathsf{iv}_\circ, \mathsf{res}_\bullet]$, and $\mathsf{SE2E}[\mathsf{iv}_\circ, \mathsf{res}_\circ]$ by Φ_{iv}, $\Phi^{\bullet}_{E2E}$, and Φ°_{E2E}, respectively.

With the adversary model $\mathcal{A}_2$, we capture the ballot reordering attack against verifiability and the ballot copy attack against privacy in EEV. We prevent ballot reordering attack in both EEV_+ and EEV_*. With the model $\mathcal{A}_3$ where the voting device is corrupt, we capture Pereira's attack in EEV. On the other hand, the variants EEV_+ and EEV_* provide a security proof for Φ_{iv}. With $\mathcal{A}_4$ and $\mathcal{A}_5$ where we extend the model with a corrupt network and a corrupt RS, we still have security proof for Φ_{iv} in the variants. However, with $\mathcal{A}_6$ where VC is corrupt in addition to corrupt voting device and network, just EEV_* is proved to be secure for Φ_{iv}. With the models $\mathcal{A}_i$ for $i = 3, 4, 5$, no variant satisfies $\Phi^{\bullet}_{E2E}$ since there is ballot stuffing by a corrupt voting device if the voters did not verify their votes. Thus, we prove the weak notion of end-to-end verifiability, i.e. Φ°_{E2E}, for those cases, guaranteeing that the adversary can only

cast ballots for corrupt voters and for honest voters if they did not verify their votes. As we proved security for Φ_{iv} in EEV_* with the model $\mathcal{A}_6$, we also obtain a security proof for $\Phi^{\circ}_{\mathsf{E2E}}$ in EEV_*.

Conclusion. Our analysis confirms Pereira's attack against individual verifiability of the EEV protocol. It also allowed us to capture new attacks against security of the protocol, i.e. ballot reordering and ballot copying attacks. An interesting open question is how to deploy the solutions that we discussed and proved in practice, and how to achieve the best trade-off between verifiability, receipt-freeness and usability.

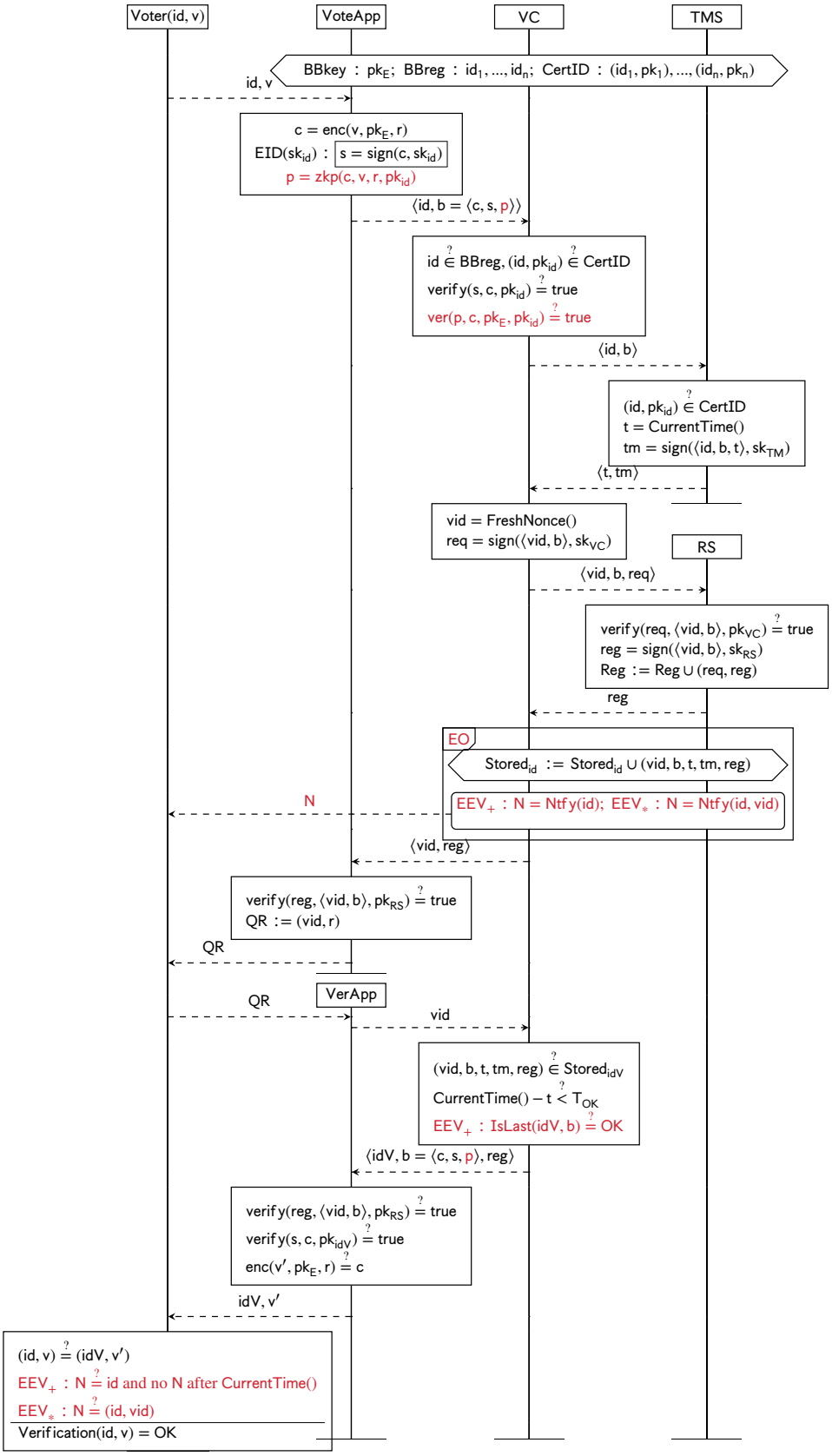


FIGURE 8.3: Illustration of EEV protocols. In red are EEV_+ and EEV_* additions.

SETUP PHASE

R_{eid} : **generate an** EID **card for identities**
$[\ In(id), Fr(sk_{id})\ ] -[\ ]\rightarrow [\ !CertID(id, pk(sk_{id})), !EID(id, pk(sk_{id}), sk_{id}), Out(pk(sk_{id}))\ ]$

R^{TM}_{cert} : **generate a key pair and get certified**
$[\ Fr(sk_{TM})\ ] -[\ ]\rightarrow [\ !CertTM(pk(sk_{TM})), !SkTM(sk_{TM}), Out(pk(sk_{TM}))\ ]$

R^{RS}_{cert} : **generate a key pair and get certified**
$[\ Fr(sk_{RS})\ ] -[\ ]\rightarrow [\ !CertRS(pk(sk_{RS})), !SkRS(sk_{RS}), Out(pk(sk_{RS}))\ ]$

R^{VC}_{cert} : **generate a key pair and get certified**
$[\ Fr(sk_{VC})\ ] -[\ ]\rightarrow [\ !CertVC(pk(sk_{VC})), !SkVC(sk_{VC}), Out(pk(sk_{VC}))\ ]$

R^{EO}_{key} : **generate election key pair**
$[\ Fr(sk_{E})\ ] -[\ BBkey(pk(sk_{E}))\ ]\rightarrow [\ !BBkey(pk(sk_{E})), !SkE(sk_{E}), Out(pk(sk_{E}))\ ]$

R^{EO}_{cand} : **determine candidates to be elected**
$[\ In(v)\ ] -[\ ]\rightarrow [\ !BBcand(v)\ ]$

R^{EO}_{voter} : **determine identities eligible to vote**
$[\ In(id), !CertID(id, pk_{id})\ ] -[\ BBreg(id)\ ]\rightarrow [\ !Voter(id)\ ]$

VOTING PHASE

R_{start} : **start verification time frame**
$[\ \ ] -[\ ]\rightarrow [\ !StartTime(\$t)\ ]$

R_{end} : **end verification time frame**
$[\ !StartTime(\$t)\ ] -[\ !EndTime(\$t)\ ]\rightarrow [\ \ ]$

R^{V}_{vote} : **construct a ballot and send it to** VC
$\text{let}\ \ c = enc(v, pk_E, r);\ \ s = sign(c, sk_{id});\ \ b = \langle c, s\rangle\ \ \text{in}$
$[\ !Voter(id), !EID(id, pk_{id}, sk_{id}), !BBcand(v), !BBkey(pk_E), Fr(r), Fr(t_v)\ ]$
$-[\ Vote(id, v), VoteB(id, b), VoteTime(id, v, t_v)\ ]\rightarrow [\ St(id, v, b, r, t_v), Out(\langle id, b\rangle)\ ]$

R^{VC}_{accept} : **verify and accept ballot, and send** h(c) **to** TM
$[\ In(\langle id, \langle c, s\rangle\rangle), !Voter(id), !CertID(id, pk_{id}), Fr(vid)\ ]$
$-[\ verify(s, c, pk_{id}) \equiv true\ ]\rightarrow [\ Vid(id, pk_{id}, b, vid), Out(\langle pk_{id}, h(c)\rangle)\ ]$

R^{TM}_{time} : **stamps time to** h(c) **and sends it back to** VC
$\text{let}\ \ tm = sign(\langle pk_{id}, hash, t_s\rangle, sk_{TM})\ \ \text{in}$
$[\ In(\langle pk_{id}, hash\rangle), !CertID(id, pk_{id}), !SkTM(sk_{TM}), Fr(t_s)\ ] -[\ ]\rightarrow [\ Out(\langle t_s, tm\rangle)\ ]$

R^{VC}_{req} : **verify time mark, generate** req **and send it to** RS
$\text{let}\ \ req = sign(\langle vid, h(b)\rangle, sk_{VC})\ \ \text{in}$
$[\ In(\langle t_s, tm\rangle), Vid(id, pk_{id}, b, vid), !SkVC(sk_{VC}), !CertTM(pk_{TM})\ ]$
$-[\ verify(tm, \langle pk_{id}, h(c), t_s\rangle, pk_{TM}) \equiv true\ ]\rightarrow$
$[\ Tm(id, pk_{id}, b, vid, t_s, tm), Out(\langle vid, h(b), req\rangle)\ ]$

R^{RS}_{reg} : **register** $\langle$vid, h(b)$\rangle$ **and send confirmation to** VC
$\text{let}\ \ reg = sign(h(\langle vid, hash\rangle), sk_{RS})\ \ \text{in}$
$[\ In(\langle vid, hash, req\rangle), !SkRS(sk_{RS}), !CertVC(pk_{VC})\ ]$
$-[\ verify(req, \langle vid, hash\rangle, pk_{VC}) \equiv true\ ]\rightarrow [\ !Reg(req, reg), Out(reg)\ ]$

R^{VC}_{store} : **verify registration confirmation and store ballot**
$[\ In(reg), Tm(id, pk_{id}, b, vid, t_s, tm), !CertRS(pk_{RS}), !StartTime(t)\ ]$
$-[\ Eq(verify(reg, h(\langle vid, h(b)\rangle)), pk_{RS}), true), StoreB(id, b)\ ]\rightarrow$
$[\ !Stored(id, pk_{id}, b, vid, t_s, tm, reg), !Ver(vid, t), Out(\langle vid, reg\rangle)\ ]$

Ψ^{VC}_{order} : **store ballots in the correct order**
$VoteB(id, b_1)\ @i\ \wedge\ VoteB(id, b_2)\ @j\ \wedge$
$StoreB(id, b_1)\ @k\ \wedge\ StoreB(id, b_2)\ @l\ \wedge\ i \prec j \Rightarrow k \prec l$

R^{V}_{vid} : **verify registration confirmation and capture** QR **code**
$[\ In(\langle vid, reg\rangle), St(id, v, b, r, t_v), !CertRS(pk_{RS})\ ]$
$-[\ verify(reg, h(\langle vid, h(b)\rangle), pk_{RS}) \equiv true\ ]\rightarrow [\ !Voted(id, v, t_v, vid, r)\ ]$

FIGURE 8.4: Setup and voting phase of the EEV protocol specification.

TALLY PHASE

$\mathsf{R}^{\mathsf{IBBP}}_{\mathsf{verify}}$: **verify the ballot stored by** VC **and check whether it was registered**

$[\ !\mathsf{Voter}(\mathsf{id}), !\mathsf{Stored}(\mathsf{id}, \mathsf{pk}_{\mathsf{id}}, \mathsf{b}, \mathsf{vid}, \mathsf{t}_s, \mathsf{tm}, \mathsf{reg}), !\mathsf{Reg}(\mathsf{req}, \mathsf{reg}), !\mathsf{CertID}(\mathsf{id}, \mathsf{pk}_{\mathsf{id}}),$
$!\mathsf{CertRS}(\mathsf{pk}_{\mathsf{RS}}), !\mathsf{CertVC}(\mathsf{pk}_{\mathsf{VC}})\ ] \multimap [\ \mathsf{verify}(\mathsf{s}, \mathsf{c}, \mathsf{pk}_{\mathsf{id}}) \equiv \mathsf{true},$
$\mathsf{verify}(\mathsf{req}, \langle \mathsf{vid}, \mathsf{hash}\rangle, \mathsf{pk}_{\mathsf{VC}}) \equiv \mathsf{true}, \mathsf{verify}(\mathsf{reg}, \mathsf{h}(\langle \mathsf{vid}, \mathsf{h}(\mathsf{b})\rangle), \mathsf{pk}_{\mathsf{RS}}) \equiv \mathsf{true}\ ] \mapsto$
$[\ !\mathsf{IBBPVerified}(\mathsf{id}, \mathsf{pk}_{\mathsf{id}}, \mathsf{b}, \mathsf{vid}, \mathsf{t}_s, \mathsf{tm}, \mathsf{reg})\ ]$

$\mathsf{R}^{\mathsf{IBBP}}_{\mathsf{tally}}$: **select the last ballot stored by** VC **for the tally**

$[\ !\mathsf{IBBPVerified}(\mathsf{id}, \mathsf{pk}_{\mathsf{id}}, \mathsf{b}, \mathsf{vid}, \mathsf{t}_s, \mathsf{tm}, \mathsf{reg})\ ] \multimap [\ \mathsf{BBtally}(\mathsf{id}, \mathsf{b})\ ] \mapsto [\ ]$

$\Psi^{\mathsf{IBBP}}_{\mathsf{tally}}$: **select the last ballot stored by** VC

$\mathsf{StoreB}(\mathsf{id}, \mathsf{b})\ @i \ \wedge\ \mathsf{StoreB}(\mathsf{id}, \mathsf{b}')\ @j \ \wedge\ \mathsf{BBtally}(\mathsf{id}, \mathsf{b})\ @k \Rightarrow j \prec i \ \vee\ \mathsf{b} = \mathsf{b}'$

INDIVIDUAL VERIFICATION

$\mathsf{R}^{\mathsf{V}}_{\mathsf{ver}}$: **voter verifies the ballot within verification timeframe**

$\mathsf{let}\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}\rangle;\ \ \mathsf{c}' = \mathsf{enc}(\mathsf{v}, \mathsf{pk}_{\mathsf{E}}, \mathsf{r})\ \ \mathsf{in}$
$[\ !\mathsf{Voted}(\mathsf{id}, \mathsf{v}, \mathsf{t}_{\mathsf{v}}, \mathsf{vid}, \mathsf{r}), !\mathsf{Stored}(\mathsf{id}, \mathsf{pk}_{\mathsf{id}}, \mathsf{b}, \mathsf{vid}, \mathsf{t}_s, \mathsf{tm}, \mathsf{reg}), !\mathsf{Ver}(\mathsf{vid}, \mathsf{t}), !\mathsf{BBkey}(\mathsf{pk}_{\mathsf{E}}),$
$!\mathsf{CertID}(\mathsf{id}, \mathsf{pk}_{\mathsf{id}}), !\mathsf{CertRS}(\mathsf{pk}_{\mathsf{RS}})\ ]$
$\multimap [\ \mathsf{verify}(\mathsf{s}, \mathsf{c}, \mathsf{pk}_{\mathsf{id}}) \equiv \mathsf{true}, \mathsf{verify}(\mathsf{reg}, \mathsf{h}(\langle \mathsf{vid}, \mathsf{h}(\mathsf{b})\rangle), \mathsf{pk}_{\mathsf{RS}}) \equiv \mathsf{true}, \mathsf{c} \equiv \mathsf{c}'$
$\mathsf{CheckTime}(\mathsf{t}), \mathsf{Verified}(\mathsf{id}, \mathsf{v}, \mathsf{t}_{\mathsf{v}})\ ] \mapsto [\ ]$

$\Psi^{\mathsf{V}}_{\mathsf{ver}}$: **check whether time frame has expired**

$\mathsf{CheckTime}(\mathsf{t}) \Rightarrow \neg \mathsf{EndTime}(\mathsf{t})$

FIGURE 8.5: Tally phase and individual verification procedure of the EEV protocol specification.

Chapter 9

Conclusion

In this thesis, we provide a general formal verification framework to evaluate the verifiability of e-voting protocols bridging the gap between theoretical analysis and practical implementation by considering real-world e-voting protocols, such as Helios, Belenios, Selene, and the Estonian e-voting protocol, as case studies. Our framework allows automated verification, accounts for revoting, and captures both existing and new verifiability attacks. Our analyses highlight the importance of the correct and realistic modelling of the protocols and their automated verification. To identify vulnerabilities of the protocols under different corruption scenarios, they should be modelled with their features affecting the security. For example, revoting is a feature that many e-voting protocols provide to protect against coercion or as a general usability feature. However, this feature has not been modelled in the verifiability analyses of e-voting protocols, except for the computational model of Belenios. Thus, the attacks we discovered are lacking in the analyses of Helios or the literature. Moreover, the automated verification of the protocols allowed us to capture those attacks, emphasising the advantages of the comprehensive analysis provided by automated verification over manual verification.

9.1 Our Findings and Open Questions

- ***In particular to our definition:***
  - Our definition can evaluate the verifiability of the protocols according to the different levels of end-to-end verifiability.
  - It can be applied to different types of protocols that publish the election result as a set of votes or a set of tracker-vote pairs.
  - It can be instantiated for different revoting policies, i.e. a trivial policy modelling no revote or the policy selecting the last vote cast.
  - It can be checked through for both specifications of Tamarin and ProVerif, without making any change in the formulas.
- ***In particular to revoting, attacks, and individual verification procedures:***
  - There are new attacks arising from revoting. So far, two main attacks have been described in the literature against the verifiability of e-voting protocols: ballot stuffing and clash attacks. We captured ballot reordering attacks exploited by the adversary corrupting the communication network and new versions of clash attacks requiring only the registrar to be corrupt. In the original scenario of clash attacks, the registrar, server, and voting platforms are all required to be corrupt to exploit the attacks.
  - Individual verification procedures allowed by the protocols could be insufficient to ensure individual verifiability. If the voters are allowed to verify their ballots

at any time during the election, as in the real deployments of the protocols that allow revoting, the voters could be subjected to the attacks mentioned above. The most secure procedure requires verifying the ballots after the voting period has ended. However, e-voting protocols, in general, provide long periods for ballot casting. If the voter casts a ballot at the very beginning of the election, waiting for the verification until the end of the voting period could make them abandon the procedure.

- The deployment of the bulletin board that displays all the ballots cast for a voter credential without replacing them with the new ones is efficient in capturing the new versions of clash attacks if a different individual verification procedure is provided. We assumed the voters verify their last ballot cast and also the previous ballots cast by them. Thus, they perform a successful verification if all the existing ballots for their credential are theirs. This individual verification procedure protects voters against clash attacks but does not protect against other found attacks.

- ***In particular to case studies:***

  - Helios is vulnerable to known attacks as well as new attacks found with revoting. However, we prove the weaker notion of end-to-end verifiability against a partially or fully corrupt server if the voters verify their ballots after the voting phase ends. In this notion of verifiability, ballot stuffing is allowed for honest voters who did not verify their ballots. We mean by a fully corrupt server that it is also responsible for registering voters, i.e. it plays the roles of both server and registrar.
  - Belenios is vulnerable to newly found attacks, i.e. ballot reordering attacks and new versions of clash attacks in the cases of a corrupt network, a corrupt voting server, or a corrupt registrar. Thus, it is not secure as it is supposed to be against a corrupt registrar or a corrupt server. We find ballot reordering attacks even when both are honest, and the adversary corrupts the network. The attacks are exploited against the vulnerability of the voting server that it does not know the order of the ballots, if the network is corrupt, and the correspondence between voter identities and their credentials. Thus, the adversary can block the ballots from honest voters and cast them under the identities of corrupt voters. However, similar to Helios, we prove the weaker notion of end-to-end verifiability against all these three cases if the voters verify their ballots after the voting phase ends. Note that both Helios and Belenios allow individual verification at any time in their real deployments.
  - BeleniosRF is vulnerable to individual verifiability attacks and also new versions of clash attacks. BeleniosRF aims to provide receipt-freeness in addition to end-to-end verifiability, using an additional randomisation server that randomises each received ballot before publishing it on the bulletin board. Therefore, the voters only verify that there exists a ballot next to their credentials. In this way, even if receipt-freeness is achieved, the individual verification procedure is weakened and does not fully protect voters against ballot alteration or clash attacks.
  - Selene satisfies end-to-end verifiability if honest voters verify their votes, even if all other parties are corrupt. Thus, we provide the first automated proof of verifiability of Selene.
  - SeleneRF, the variant of Selene that provides a stronger receipt-freeness due to the randomisation of the ballots, as in BeleniosRF, achieves the same level of end-to-end verifiability as Selene.

  - Hyperion, another variant of Selene that improves tracker management due to the different cryptographic primitives employed, achieves the same level of end-to-end verifiability as Selene.
  - The Estonian e-voting protocol is susceptible to individual verifiability attacks. We capture the known attack proposed by Pereira and new ballot reordering attacks. Thus, it does not satisfy end-to-end verifiability unless all parties and the underlying infrastructure are honest.

- ***In particular to end-to-end verifiability vs. receipt-freeness:***

  - In the case of Belenios, its receipt-free variant, i.e. BeleniosRF, weakens the verifiability guarantees provided by Belenios. We observe the tension between receipt-freeness and end-to-end verifiability in this case. The tension can be reduced with an improvement of BeleniosRF concerning its individual verification procedure. The ways and the methods that achieve stronger end-to-end verifiability are open questions.
  - In the case of Selene, its variant, SeleneRF, preserves the verifiability guarantees provided by Selene. Thus, without sacrificing verifiability, a stronger receipt-freeness can be achieved by SeleneRF.

- ***In particular to improvements of Belenios and the Estonian e-voting protocols:***

  - We handled the attack scenarios of Belenios when the network, server, or registrar is corrupt. The attacks exploited the voting server's vulnerability in that it knows the order of the ballots, if the network is corrupt, and the correspondence between voter identities and their credentials. Identifying this, we provided solutions to remove these vulnerabilities and prevent those attacks. Our solutions require changes in the implementation of the voting platform and the ballot structure. Thus, they do not affect the usability of the protocol. We proved that the solution modelled in the variant Belenios+ guarantees end-to-end verifiability in all attack scenarios.
  - We provided two solutions to prevent individual verifiability attacks in the Estonian e-voting protocol that are exploited by a corrupt voting application. Our solutions improve the individual verifiability and end-to-end verifiability of the protocol, but they require additional checks performed by the voters when they verify their votes. Even though we proved that deploying the solutions provides better security, their feasibility and practicality are open questions.

- ***In particular to automated verification tools Tamarin and ProVerif:***

  - Tamarin allows to use timepoints in both restrictions and lemmas, which makes it more expressive than ProVerif to specify the correct and realistic model of the protocol. For example, we can specify a restriction in Tamarin to select the last ballot cast ordering the time of the events representing the ballots recorded by the voting server. On the other hand, in ProVerif, timepoints are only allowed in lemmas, on the right side of the implications, which limits to express what we want to formulate.
  - ProVerif is better at handling equational theory. In our first attempt to model Selene in Tamarin, we encountered termination problems in Tamarin. We observed that Tamarin could not handle complicated equations together with a realistic model of Selene, such as the ones modelling the commitments, where they are allowed to open with fake randomness. On the other hand, we could obtain our verification results with the attempt in ProVerif.

- Both tools were insufficient to model the abstraction of the exponentiation, which is needed to model Hyperion. Therefore, we had to skip modelling the individual view of bulletin boards as suggested in the protocol since it requires quadruple exponentiation. For the triple exponentiation, we could model the two equations in ProVerif required for the specification of Hyperion. For completeness, we tried to add complementary equations. However, these resulted in failure, i.e. they left the execution non-terminated. It was worse in Tamarin since even these two equations did not work.
- To model Belenios+ in Tamarin, we needed to model a rule that will be executed recursively since it represents a ballot-casting procedure, where each ballot is generated with the information from the previous ballot. We could not achieve to execute this rule infinitely many times, and we had to restrict its execution to four times in order to make Tamarin terminate.

Thus, the tools should be improved with respect to those aspects to be used more for the automated verification of various cryptographic protocols.

In conclusion, we achieved to provide a general verifiability framework that can also be applied to other protocols not presented in this thesis to evaluate their verifiability. Our framework can also be used for the identification of vulnerabilities, i.e. the formulas falsified in the definition may give a hint about the vulnerabilities. The ultimate goal of e-voting protocols is to satisfy both strong end-to-end verifiability and receipt-freeness and provide them even if all the parties are corrupt. Towards this, our framework can contribute by providing a systematic and automated way to formally verify election verifiability.

Bibliography

[1] Ben Adida. "Helios: Web-based Open-Audit Voting". In: *17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*. 2008, pp. 335–348. URL: http://www.usenix.org/events/sec08/tech/full_papers/adida/adida.pdf.

[2] Ben Adida and C. Andrew Neff. "Ballot Casting Assurance". In: *USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'06*. 2006. URL: https://www.usenix.org/conference/evt-06/ballot-casting-assurance.

[3] Ben Adida et al. "Electing a university president using open-audit voting: Analysis of real-world use of Helios". In: *2009 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections*. Usenix, 2009.

[4] Myrto Arapinis, Véronique Cortier, and Steve Kremer. "When Are Three Voters Enough for Privacy Properties?" In: *Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part II*. Ed. by Ioannis G. Askoxylakis et al. Vol. 9879. Lecture Notes in Computer Science. Springer, 2016, pp. 241–260. DOI: 10.1007/978-3-319-45741-3_13. URL: https://doi.org/10.1007/978-3-319-45741-3_13.

[5] S. Baloglu et al. "Election Verifiability in Receipt-free Voting Protocols". In: *2023 2023 IEEE 36th Computer Security Foundations Symposium (CSF) (CSF)*. Los Alamitos, CA, USA: IEEE Computer Society, 2023, pp. 63–78. DOI: 10.1109/CSF57540.2023.00005. URL: https://doi.ieeecomputersociety.org/10.1109/CSF57540.2023.00005.

[6] Sevdenur Baloglu et al. "Election Verifiability Revisited: Automated Security Proofs and Attacks on Helios and Belenios". In: *34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021*. IEEE, 2021, pp. 1–15. DOI: 10.1109/CSF51468.2021.00019. URL: https://doi.org/10.1109/CSF51468.2021.00019.

[7] Sevdenur Baloglu et al. "Provably Improving Election Verifiability in Belenios". In: *Electronic Voting*. Ed. by Robert Krimmer et al. Cham: Springer International Publishing, 2021, pp. 1–16. ISBN: 978-3-030-86942-7.

[8] *Belenios - Verifiable Online Voting System*. https://belenios.org/. URL: https://belenios.org/.

[9] Josh Benaloh. "Simple Verifiable Elections". In: *2006 USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'06, Vancouver, BC, Canada, August 1, 2006*. Ed. by Dan S. Wallach and Ronald L. Rivest. USENIX Association, 2006. URL: https://www.usenix.org/conference/evt-06/simple-verifiable-elections.

[10] Josh Daniel Cohen Benaloh. "Verifiable Secret-ballot Elections". AAI8809191. PhD thesis. New Haven, CT, USA: Yale University, 1987.

[11] Bruno Blanchet. "Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif". In: *Foundations and Trends in Privacy and Security* 1.1-2 (2016), pp. 1–135. ISSN: 2474-1558. DOI: 10.1561/3300000004. URL: http://dx.doi.org/10.1561/3300000004.

[12] Olivier Blazy et al. “Signatures on Randomizable Ciphertexts”. In: *Public Key Cryptography - PKC 2011 - 14th International Conference on Practice and Theory in Public Key Cryptography, Taormina, Italy, March 6-9, 2011. Proceedings*. Ed. by Dario Catalano et al. Vol. 6571. Lecture Notes in Computer Science. Springer, 2011, pp. 403–422. DOI: 10.1007/978-3-642-19379-8_25. URL: https://doi.org/10.1007/978-3-642-19379-8_25.

[13] Alessandro Bruni, Eva Drewsen, and Carsten Schürmann. “Towards a Mechanized Proof of Selene Receipt-Freeness and Vote-Privacy”. In: *Electronic Voting - Second International Joint Conference, E-Vote-ID 2017, Bregenz, Austria, October 24-27, 2017, Proceedings*. Ed. by Robert Krimmer et al. Vol. 10615. Lecture Notes in Computer Science. Springer, 2017, pp. 110–126. DOI: 10.1007/978-3-319-68687-5_7. URL: https://doi.org/10.1007/978-3-319-68687-5_7.

[14] Pyrros Chaidos et al. “BeleniosRF: A Non-interactive Receipt-Free Electronic Voting Scheme”. In: *23rd ACM Conference on Computer and Communications Security (CCS’16)*. Vienna, Austria: ACM, 2016, pp. 1614–1625. DOI: 10.1145/2976749.2978337.

[15] David Chaum, Peter Y. A. Ryan, and Steve A. Schneider. “A Practical Voter-Verifiable Election Scheme”. In: *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*. 2005, pp. 118–139. DOI: 10.1007/11555827_8. URL: https://doi.org/10.1007/11555827_8.

[16] 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), 18-21 May 2008, Oakland, California, USA*. 2008, pp. 354–368. DOI: 10.1109/SP.2008.32. URL: https://doi.org/10.1109/SP.2008.32.

[17] 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*. 2019, pp. 367–381. DOI: 10.1109/CSF.2019.00032. URL: https://doi.org/10.1109/CSF.2019.00032.

[18] Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. “Belenios: A Simple Private and Verifiable Electronic Voting System”. In: *Foundations of Security, Protocols, and Equational Reasoning - Essays Dedicated to Catherine A. Meadows*. Ed. by Joshua D. Guttman et al. Vol. 11565. Lecture Notes in Computer Science. Springer, 2019, pp. 214–238. DOI: 10.1007/978-3-030-19052-1_14. URL: https://doi.org/10.1007/978-3-030-19052-1_14.

[19] Véronique Cortier and Ben Smyth. “Attacking and Fixing Helios: An Analysis of Ballot Secrecy”. In: *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF 2011, Cernay-la-Ville, France, 27-29 June, 2011*. IEEE Computer Society, 2011, pp. 297–311. DOI: 10.1109/CSF.2011.27. URL: https://doi.org/10.1109/CSF.2011.27.

[20] Véronique Cortier et al. “Election verifiability for Helios under weaker trust assumptions”. In: *ESORICS - European Symposium on Research in Computer Security*. Springer International Publishing. 2014, pp. 327–344.

[21] Véronique Cortier et al. “Machine-Checked Proofs for Electronic Voting: Privacy and Verifiability for Belenios”. In: *31st IEEE Computer Security Foundations Symposium, Oxford, United Kingdom, July 9-12, 2018*. 2018, pp. 298–312. DOI: 10.1109/CSF.2018.00029. URL: https://doi.org/10.1109/CSF.2018.00029.

[22] Véronique Cortier et al. "Type-Based Verification of Electronic Voting Protocols". In: *4th International Conference on Principles of Security and Trust, London, UK, April 11-18, 2015*. Vol. 9036. Lecture Notes in Computer Science. Springer, 2015, pp. 303–323. ISBN: 978-3-662-46665-0. DOI: 10.1007/978-3-662-46666-7_16. URL: https://doi.org/10.1007/978-3-662-46666-7_16.

[23] Quynh Dang. *Secure Hash Standard (SHS)*. en. 2012. DOI: https://doi.org/10.6028/NIST.FIPS.180-4.

[24] Stéphanie Delaune, Steve Kremer, and Mark Ryan. "Verifying privacy-type properties of electronic voting protocols". In: *J. Comput. Secur.* 17.4 (2009), pp. 435–487. DOI: 10.3233/JCS-2009-0340. URL: https://doi.org/10.3233/JCS-2009-0340.

[25] W. Diffie and M. Hellman. "New directions in cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638.

[26] C. Dragan et al. "Machine-Checked Proofs of Privacy Against Malicious Boards for Selene & Co". In: *2022 2022 IEEE 35th Computer Security Foundations Symposium (CSF) (CSF)*. Los Alamitos, CA, USA: IEEE Computer Society, 2022, pp. 319–331. DOI: 10.1109/CSF54842.2022.00021. URL: https://doi.ieeecomputersociety.org/10.1109/CSF54842.2022.00021.

[27] Taher ElGamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara*. 1984, pp. 10–18. DOI: 10.1007/3-540-39568-7_2.

[28] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. "A Practical Secret Voting Scheme for Large Scale Elections". In: *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*. ASIACRYPT '92. Berlin, Heidelberg: Springer-Verlag, 1992, 244–251. ISBN: 3540572201.

[29] Stéphane Glondu. *Belenios specification*. 2023 [Online]. URL: https://www.belenios.org/specification.pdf.

[30] Jens Groth and Amit Sahai. "Efficient Non-interactive Proof Systems for Bilinear Groups". In: *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*. Ed. by Nigel P. Smart. Vol. 4965. Lecture Notes in Computer Science. Springer, 2008, pp. 415–432. DOI: 10.1007/978-3-540-78967-3_24. URL: https://doi.org/10.1007/978-3-540-78967-3_24.

[31] Sven Heiberg, Peeter Laud, and Jan Willemson. "The Application of I-Voting for Estonian Parliamentary Elections of 2011". In: *E-Voting and Identity - Third International Conference, VoteID 2011, Tallinn, Estonia, September 28-30, 2011, Revised Selected Papers*. Ed. by Aggelos Kiayias and Helger Lipmaa. Vol. 7187. Lecture Notes in Computer Science. Springer, 2011, pp. 208–223. DOI: 10.1007/978-3-642-32747-6_13. URL: https://doi.org/10.1007/978-3-642-32747-6_13.

[32] Sven Heiberg and Jan Willemson. "Verifiable internet voting in Estonia". In: *6th International Conference on Electronic Voting: Verifying the Vote, EVOTE 2014, Lochau / Bregenz, Austria, October 29-31, 2014*. Ed. by Robert Krimmer and Melanie Volkamer. IEEE, 2014, pp. 1–8. DOI: 10.1109/EVOTE.2014.7001135. URL: https://doi.org/10.1109/EVOTE.2014.7001135.

[33] Sven Heiberg et al. "Improving the Verifiability of the Estonian Internet Voting Scheme". In: *Electronic Voting - First International Joint Conference, E-Vote-ID 2016, Bregenz, Austria, October 18-21, 2016, Proceedings*. Ed. by Robert Krimmer et al. Vol. 10141. Lecture Notes in Computer Science. Springer, 2016, pp. 92–107. DOI: 10.1007/978-3-319-52240-1_6. URL: https://doi.org/10.1007/978-3-319-52240-1_6.

[34] *Helios - Verifiable Online Elections*. https://heliosvoting.org/. URL: https://heliosvoting.org/.

[35] *IACR elections*. https://www.iacr.org/elections/. URL: https://www.iacr.org/elections/.

[36] Wojciech Jamroga, Michal Knapik, and Damian Kurpiewski. "Model Checking the SELENE E-Voting Protocol in Multi-agent Logics". In: *Electronic Voting*. Ed. by Robert Krimmer et al. Cham: Springer International Publishing, 2018, pp. 100–116. ISBN: 978-3-030-00419-4.

[37] Ari Juels, Dario Catalano, and Markus Jakobsson. "Coercion-resistant electronic elections". In: *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005, Alexandria, VA, USA, November 7, 2005*. 2005, pp. 61–70. DOI: 10.1145/1102199.1102213. URL: http://doi.acm.org/10.1145/1102199.1102213.

[38] Steve Kremer, Mark Ryan, and Ben Smyth. "Election Verifiability in Electronic Voting Protocols". In: *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings*. Ed. by Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou. Vol. 6345. Lecture Notes in Computer Science. Springer, 2010, pp. 389–404. DOI: 10.1007/978-3-642-15497-3_24. URL: https://doi.org/10.1007/978-3-642-15497-3_24.

[39] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. "Accountability: definition and relationship to verifiability". In: *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*. ACM, 2010, pp. 526–535. ISBN: 978-1-4503-0245-6. DOI: 10.1145/1866307.1866366. URL: https://doi.org/10.1145/1866307.1866366.

[40] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. "Clash Attacks on the Verifiability of E-Voting Systems". In: *Proceedings of the 33rd IEEE Symposium on Security and Privacy, 21-23 May 2012, San Francisco, California, USA*. IEEE Computer Society, 2012, pp. 395–409. ISBN: 978-0-7695-4681-0. DOI: 10.1109/SP.2012.32. URL: https://doi.org/10.1109/SP.2012.32.

[41] Simon Meier et al. "The TAMARIN Prover for the Symbolic Analysis of Security Protocols". In: *25th International Conference on Computer Aided Verification*. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013. DOI: 10.1007/978-3-642-39799-8_48.

[42] David Mestel, Johannes Mueller, and Pascal Reisert. "How Efficient are Replay Attacks against Vote Privacy? A Formal Quantitative Analysis". In: *35th IEEE Computer Security Foundations Symposium, CSF*. 2022.

[43] Johannes Müller. "Breaking and Fixing Vote Privacy of the Estonian E-Voting Protocol IVXV". In: *7th Workshop on Advances in Secure Electronic Voting, FC22*. 2022. URL: https://orbilu.uni.lu/handle/10993/49442.

[44] Olivier Pereira. "Individual Verifiability and Revoting in the Estonian Internet Voting System". In: *7th Workshop on Advances in Secure Electronic Voting, FC22*. 2022. URL: https://eprint.iacr.org/2021/1098.

[45] *ProVerif*. URL: https://bblanche.gitlabpages.inria.fr/proverif/.

[46] Peter Y. A. Ryan, Peter B. Roenne, and Simon Rastikian. "Hyperion: An Enhanced Version of the Selene End-to-End Verifiable Voting Scheme". In: *Sixth International Joint Conference on Electronic Voting E-Vote-ID*. University of Tartu Press, 2021.

[47] Peter Y. A. Ryan, Peter B. Rønne, and Vincenzo Iovino. "Selene: Voting with Transparent Verifiability and Coercion-Mitigation". In: *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*. Vol. 9604. Lecture Notes in Computer Science. Springer, 2016, pp. 176–192. DOI: 10.1007/978-3-662-53357-4_12. URL: https://doi.org/10.1007/978-3-662-53357-4_12.

[48] Benedikt Schmidt et al. "Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties". In: *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*. Ed. by Stephen Chong. IEEE Computer Society, 2012, pp. 78–94. DOI: 10.1109/CSF.2012.25. URL: https://doi.org/10.1109/CSF.2012.25.

[49] Drew Springall et al. "Security Analysis of the Estonian Internet Voting System". In: *Proceedings of the 21st ACM Conference on Computer and Communications Security*. ACM. Nov. 2014.

[50] *Statistics about internet voting in estonia*. URL: https://www.valimised.ee/en/archive/statistics-about-internet-voting-estonia.

[51] *Tamarin Prover*. URL: https://tamarin-prover.github.io.

[52] *Tamarin specifications for the variants of Belenios*. URL: https://github.com/sbaloglu/.