UNIVERSITÀ
DI TRENTO
Department of Mathematics
Laboratory of Industrial Mathematics and Cryptography

Doctoral programme — XXXVII cycle

COMBINING CRYPTOGRAPHY, RISK ASSESSMENT AND USABILITY FOR SECURE E-VOTING SYSTEMS

Chiara Spadafora

PhD thesis in Mathematics

Supervised by Silvio Ranise

Board:

b1, Prof. Massimo Giulietti	Università degli Studi di Perugia
b2, Prof. Federico Pintore	Università di Trento
b3, Prof. Andrea Visconti	Università degli Studi di Milano

Abstract

Remote electronic voting is a multi-faceted subject that cannot be fully addressed from a single perspective. The design of a secure and effective e-voting system requires a careful balance of mathematical rigor, robust information security measures, and usability considerations. In this thesis, I explore these interconnected dimensions to propose a comprehensive solution, which was called Vote App, that ensures both security and usability. In addition, this thesis introduces the Amun voting protocol, an improved voting protocol derived from the one developed during my master's thesis.

Contents

Chapter 1

Introduction

This chapter introduces electronic voting, starting with an overview of the topic in Section 1.1. In Section 1.2, we discuss the properties that a secure e-voting system must satisfy, followed by a review of related works in Section 1.3. Finally, Section 1.4 presents the main contributions of this thesis.

1.1 Electronic Voting

Electronic voting (e-voting) refers to the various methods used to enable the expression of votes and the counting of preferences through electronic devices. It has become an essential mechanism in modern democracies, using technology to streamline the voting process and increase accessibility. While it promises efficiency and scalability, its use presents several security challenges. E-voting can be divided into two types: physical e-voting, which is conducted in supervised polling places, and remote e-voting, which lacks such supervision and is typically conducted over the Internet or by postal mail [108].

Physical Electronic Voting. *Physical e-voting* [108] assumes trusted human supervision of the voters, procedures, hardware and software in polling places. Electronic voting machines can be categorized based on their function:

- *Direct Recording Voting Machine (DRE).* The Direct Recording Electronic (DRE)[108] voting machine is one of the most widely recognized

types of electronic voting machines, designed to record votes electronically by displaying electronically the ballot. The voter interacts with the machine through physical buttons or a touchscreen interface to select their preferred candidates or choices. Once the vote is cast, it is recorded in the machine's internal memory and the votes are tabulated electronically. The vote tally is typically stored on removable storage devices, such as USB drives, which can be used for a later tabulation of all machines. However, DRE systems do not produce a paper receipt, so there is no physical proof of the vote that can be directly verified by the voter. In the past, several countries including Brazil, Germany, the Netherlands, and Ireland, used DRE voting machines. However, the lack of a paper audit raised significant concerns about the security and transparency of the election process. As a result, many of these countries chose to incorporate a verification mechanism or abandoned DREs altogether in favor of voting systems that provide greater auditability.

- *Voter Verified Paper Audit Trail (VVPAT).* A Voter Verified Paper Audit Trail (VVPAT) [108] or Voter Verified Paper Record (VVPR) is a method of providing feedback to voters using an electronic system to ensure that their vote is accurately recorded. Once the vote is cast, the system generates a paper printout of the vote for the voter to review. The printout often includes the candidate names, the ballot selection, and a unique serial number or QR code. VVPAT systems are primarily used in conjunction with Direct Recording Electronic (DRE) voting machines to allow voters to verify their vote before it is cast and recorded. Today, VVPAT is used in several countries including the United States of America, India, Venezuela, Philippines and Bulgaria.

- *Electronic Ballot Marker (EBM).* An Electronic Ballot Marker (EBM) [108] is a type of voting machine designed to assist voters in marking their selections on a physical ballot. The EBM does not itself count votes or electronically stores them but instead produces a machine-readable and human-verifiable paper ballot. These ballots are typically scanned and counted by an optical scanner. The EBM is equipped with an electronic interface, often a touchscreen or keypad, that voters use to make their selections. After the voter confirms their selections, the EBM prints the completed ballot, which includes both human-readable text and a machine-readable barcode or QR code [108]. EBMs are used in some jurisdictions in the United States of America.

- *Precinct/Central Count Optical Scan (PCOS/CCOS).* A Precinct/Central Count Optical Scan [108] is a machine that combines the use of physi-

cal paper ballots with electronic vote tabulation. In this system, voters mark their choices on paper ballots, which are then scanned by an optical scanner, either at the polling station or in a central location. The scanner reads the marked ballots and tabulates the results electronically. PCOS and CCOS systems are used in the United States of America, Philippines, India, Brazil, and Costa Rica.

Remote Electronic Voting. *Remote E-Voting* [108] assumes no trusted supervision of the voters while voting. Examples of remote e-voting are postal voting and Internet voting, commonly known as *i-voting*. A secure remote e-voting system is harder to achieve because it must rely on stronger security assumptions: voters are often required to register in person, or via an untappable channel, and trust the computing device on which they cast their vote. On the other hand, remote e-voting systems benefit from enhanced transparency through the use of a Web Bulletin Board (WBB), which also enhances the verifiability of the whole process.

The most notable examples of countries using remote electronic voting are Estonia (since 2005), Switzerland (since 2014), and Australia (since 2010). Other countries that use remote electronic voting include Canada, France, Armenia.

Web Bulletin Board. Almost every e-voting protocol relies on a Web Bulletin Board (WBB) [87, 108]. A WBB acts as a log service that implements a publicly readable, insert-only storage. It is typically managed by the election administrator and serves as a transparent and auditable record of election activities. To ensure its security, a WBB relies on several key assumptions:

- it is not possible to forge messages;
- any attempts to present inconsistent or differing views of the log contents to different readers are detectable.

For the integrity of the voting process, a secure system must also protect against potential malicious activity, such as a compromised administrator or WBB attempting to forge entries, redact data, or insert arbitrary ballots or reject valid ones.

Channels. To protect voter privacy and ensure vote integrity in remote electronic voting systems, many protocols often require the existence of untappable or anonymous channels to securely mediate communication between election participants. An untappable channel is a communication channel that guarantees perfect secrecy, either by being physically resistant to eavesdropping or by using cryptographic techniques such as a one-time pad. On the other hand, an anonymous channel ensures that, even if an adversary is able to intercept the communication, they cannot identify the sender. This feature is critical for protecting voter anonymity, as it prevents attackers from associating individual votes with specific voters, thereby protecting against coercion and preserving the secrecy of the voting process.

Stages of an electronic election. Although every country has its own legislation, the stages of an electronic election can be summarized as follows:

1. *Setup phase.* During the setup phase, all relevant information about the voting protocol and procedures is published, and the voting material is generated (e.g., cryptographic keys and look-up tables).

2. *Registration Phase.* In the registration phase, voters are authenticated either *in person* by election officials or *remotely* using their digital identity. Authentication can occur at different stages of the process: either before accessing the voting client or before voting. In the first case (e.g., Civitas [35, 83]), voters must authenticate to gain access to the ballot. In the second case (e.g., Helios [3]), the voting phase is divided into two steps: ballot auditing and ballot casting. This approach allows anyone to verify the correctness of the protocol, while ensuring that only authenticated voters can cast their votes.

3. *Voting Phase.* In the voting phase, voters cast their votes according to the rules of the specific protocol being used. This phase typically involves interactions with a secure voting client, either through a web-based platform or a dedicated application. Some protocols authorize re-voting (e.g., Civitas [35]), which allows voters to change their choices within the voting period. However, re-voting requires careful protocol design to ensure that previous votes are securely replaced and that it cannot be exploited by adversaries attempting to overwrite legitimate votes. The integrity of the voting phase depends on ensuring that votes are cast as intended by the voter, recorded without modifications, and securely stored for tabulation. It must also ensure that the final tally includes only votes from eligible voters, with each voter casting no more

than one ballot. In addition, usability is a key consideration, as the voting interface must be intuitive to minimize errors while maintaining the security of the system.

4. *Tallying Phase.* In the tallying phase, the votes are counted and the results are published. To ensure the anonymity of votes during this process, several cryptographic techniques are commonly used:

   - *Mixing Networks.* Mix-Nets shuffle and anonymize votes through a series of relays, ensuring that the link between voters and their votes is obscured. After anonymization, votes are decrypted and counted. In some systems, the votes are first homomorphically aggregated before being decrypted to produce the final tally.
   - *Blind Signatures.* In protocols using blind signatures, voters authenticate themselves to the authority and present a blinded version of their vote. The authority signs the blinded vote without seeing its contents. The voter then unblinds the signed vote and submits it anonymously to the authority for counting (e.g., [61]). This technique provides both voter authentication and ballot secrecy.

   Some protocols, such as Selene [121], publish the votes unencrypted, allowing the tally to be performed in full public view. Others, known as tally-hiding protocols, prioritize privacy by hiding the entire tally and revealing only the election result, as in Ordinos [90].

5. *Verification Phase.* Remote e-voting increases transparency by allowing verification throughout the election. Voters and third parties can verify the integrity of the voting process at any time, based on the information published during the election. Depending on the protocol, verification can take place in real time or after the election, in order to ensure that the final tally reflects the actual votes cast.

1.2 Properties of an e-voting protocol

Deploying a trustworthy e-voting protocol is a challenging task because it must satisfy conflicting properties: it should preserve both the *integrity* of election results and the *confidentiality* of votes. Ensuring integrity means preventing the tampering or manipulation of votes, while confidentiality ensures that individual votes remain private, even in the face of adversarial

actions. Balancing these requirements is complex, as measures to guarantee one property may inadvertently compromise the other.

Vote Privacy (VP). Vote Privacy [108] is defined as the inability of the adversary to distinguish, given two candidates C_1, C_2, whether voter V_i voted for C_1 or C_2. *Everlasting privacy* is a variation of privacy formally introduced in [104], which focuses on preventing the leaking of votes from attacks that could be carried out in the future. The first practical solution providing everlasting privacy has been proposed by [48], introducing a new primitive called *Commitment Consistent Encryption (CCE)*.

Verifiability. Verifiability [4, 89, 83, 123, 66, 31] requires that the results of tabulation cannot be different than if all votes were announced and tabulated publicly (even if an adversary tries to change the election result). Verifiability can be divided [131, 130, 4] into:

- *Universal Verifiability (UV)* [108]: anyone can check that the published result of an election has been correctly computed;
- *Individual Verifiability:* every voter can check that their vote has been cast correctly and has been accurately counted. It is subdivided into:
  - *cast-as-intended verifiability* (CAIV) [52]: the voter can verify that the complete ballot (i.e. containing the intended vote) is correctly computed and cast;
  - *recorded-as-cast verifiability* (RACV) [105]: the voter can verify that the correct ballot is recorded for the tallying;
  - *tallied-as-recorded verifiability* (TARV) [117]: anyone can verify that all and only the recorded votes are tallied, and with the correct procedure.

  In [14], the combination of cast-as-intended, recorded-as-cast, and tallied-as-recorded, is called *End-to-end*.
- *Eligibility Verifiability (EV):* [38, 131] not only can ballots be cast exclusively by eligible voters, but *anyone* can verify that every vote in the final tally was submitted by an eligible voter. The paper [131] also adds *and there is at most one vote per voter*. In Chapter 6 we divide this property into two properties that anyone should be able to verify, to do a more precise analysis:

  - *EV1*: all valid votes have been cast by eligible voters;
  - *EV2*: all valid votes have been cast by distinct voters.

A stronger property is *accountability* [89, 90]: if the election results does not match how voters actually voted, then this is not only detected but misbehaving parties can be identified.

Auditability (A). Auditability [1] requires that it must be possible to examine the processes used to collect and count the votes in order to confirm the accuracy of the results.

Correctness (CO). Correctness [83] requires that an adversary cannot preempt, alter, or cancel the votes of honest voters.

Fairness (FA). Fairness [43] requires that no information about how many votes each candidate has received can be learned until the voting results are published.

Coercion Resistance (CR). Coercion resistance (see [83]) requires that an adversary cannot learn any additional information about the votes other than what is revealed by the results of tabulation. In other words, voters should not be able to prove whether or how they voted, even if they can interact with the adversary while voting. For a more detailed analysis of coercion-resistance, see Chapter 4 and Section 1.3.2.

The requirements presented so far are critical to ensuring a secure voting system, but they do not properly cover the whole CIA triad of information security (i.e., Confidentiality, Integrity, Availability). In particular, availability is often neglected. Therefore, we propose to consider two new properties:

- **Right to Vote (RTV).** Right to vote requires that all eligible voters are able to cast a valid vote.
- **Successful Completion (SC).** Successful completion requires that the election process manages to reach the end, publishing the result of the tallying.

1.2.1 Security Property Framework for Threat Analysis

As part of the threat analysis, we grouped the presented properties (see Section 1.2) into three main aggregated properties (see [98] and Table 1.2). This approach helped structure the subsequent threat analysis, providing a clearer framework for identifying and addressing vulnerabilities (the detailed threat analysis is presented in Chapter 6). The characterization of the aggregated properties was executed in accordance with the identification of their commonalities, which entailed the classification into discrete categories (e.g., system integrity, freedom to vote, and verifiability).

The first aggregated property, namely *Functional and Security Requirements*, refers to the fact that a voting system should allow an election to be carried out properly, allowing all and only the eligible voters to select their choices and record those correctly without disclosing them before the final tally. The second aggregated property, *Vote Freedom*, concerns the ability of a voter to express their preference without undue influence. Finally, *Verifiability* refers to the feasibility of checking every step of the election process and assessing its correctness. In Table 1.2, the properties constituting the three aggregated properties are summarized.

Council of Europe Recommendations. The Council of Europe (CoE) provides 49 recommendations for e-voting systems, organized in 8 high-level, technology-independent principles [43]. An overview of the principles is presented in Table 1.1.

Following the aforementioned aggregation of the properties of an e-voting system, we identify the applicable CoE recommendations, which can be grouped in the same way. In Table 1.2, for each aggregated property, we list the constituent properties and associate the corresponding CoE recommendations and principles. Note that some principles and properties are outside the scope of this security assessment. For example, the recommendations in Principle I: Universal Suffrage focus on human-computer interface issues, addressing usability and accessibility.

1.3 Related Works

1.3.1 Real-World E-Voting Protocols

This section provides an overview of e-voting protocols used in practice, describing their design and key features at a high level.

ElectionGuard. ElectionGuard [12] is not a complete election system, but rather a toolkit designed to enable end-to-end (E2E) verifiability. It has been used in U.S. elections, such as the Wisconsin 2020 Primary Election and to secure the voting process for the Democratic Caucus in the U.S. House of Representatives. Voters are provided tracking codes to verify that their vote was correctly counted, while any verifier can check the integrity and consistency of the whole election process. Votes are encrypted using an additively homomorphic encryption scheme, such that the ciphertexts' sum corresponds to the encryption of the sum of the plaintexts. Election Guard does not address the problem of coercion.

Helios Voting. Helios [3] is an open-source, web-based election protocol that has been used by organizations such as the IACR and Princeton University. It offers end-to-end verifiability and transparency, allowing everyone to audit the election results. It is closely related to Benaloh's protocol [13] and was firstly presented by Ben Adida in 2008 [3], then other versions have been proposed (e.g. [5, 16, 50, 15, 41]).

One of its key features is the postponed authentication of voters, where anyone can propose a vote and test its encryption, but only authenticated voters can cast their vote. The protocol is not receipt free and can be vulnerable to coercion attacks.

Helios is also vulnerable to ballot replay attacks, but this can be mitigated by ensuring that malformed ballots cannot be cast, i.e., by avoiding the use of zero as encryption randomness [124].

Belenios. Belenios [38, 63] is an electronic voting protocol built upon Helios, that provides vote privacy and verifiability even against a corrupt voting server. However, like Helios, Belenios is not coercion-resistant. Moreover the voter is assumed to use an honest voting device. These limitations have been overcome with *BeleniosRF* [31] and *BeleniosVS* [37].

- *BeleniosRF* [31] is an extension of *Belenios* designed to address the issue of coercion resistance. The novelty of BeleniosRF is that it is receipt free and verifiable without assuming an untappable channel but under the only assumption that the adversary cannot indefinitely eavesdrop the communication channel between voter and voting server. It satisfies *strong verifiability* and *strong receipt freeness*, which implies that vote privacy is also satisfied [42]. However, if re-voting was allowed, BeleniosRF could not be strongly verifiable. Moreover the protocol assumes that the registrar and the voting server cannot collude otherwise they could undetectably change any vote. This could be avoided by having every voter generate their own key pair, at the expense of usability.

- *BeleniosVS* [37] was built upon BeleniosRF and provides privacy and verifiability against a corrupt voting server and a corrupt voting device. To provide verifiability against a corrupt voting device, BeleniosVS puts alongside the voting device an auditing device with the assumption that at most one of the two devices can be corrupt.

IVXV (Estonian Internet Voting System). The IVXV protocol [72] follows the Estonian e-voting framework. IVXV mimics the double envelope postal voting: the inner privacy preserving envelope is replaced by encrypting the vote with the election's public key and the outer authentication and integrity layer is provided by signing the encrypted vote with the voter's digital signature (via ID-Card).

The protocol achieves individual verifiability since everyone can check that their vote was cast-as-intended and recorded-as-cast thanks to a verification application. The protocol also achieves a weak form of universal verifiability.

The voter can decide to vote multiple times online and one in person: in case of paper voting the paper vote has the supremacy, otherwise only the last electronic vote is preserved. This flexibility helps protect against certain forms of coercion, as it allows the voter to cast their vote in a secure environment outside of the online system: if coerced a voter can always choose to go voting in person.

iVote. iVote [22] is an online voting system developed by the New South Wales Electoral Commission (NSWEC) to facilitate remote voting, particularly for individuals with disabilities. *iVote* is not coercion resistant and does not satisfy the property of universal verifiability, but allows individual voters

to verify their votes.

In 2021, during the New South Wales Local Government elections, iVote experienced significant technical issues, leading to some voters being unable to access the system [109].

Swiss Post Voting System. The Swiss Post Voting System [136] is a return-code-based remote online voting system designed to ensure individual verifiability, universal verifiability, and vote secrecy. Before the election, each voter receives a printed code sheet. To begin voting, the voter enters an alphanumeric code for authentication and selects their preferred options. The system generates a return code for each selected option. The voter then verifies that the return codes match those printed on their voting card. If there is any discrepancy, the voter can abort the process and report the issue to the election authorities. If the codes match, the vote is confirmed as reflecting the voter's intention, and the voter proceeds by entering a confirmation code. The system then acknowledges the successful vote submission by sending a final return code, which the voter checks against the printed code sheet.

In the event of an interruption after submitting the vote, the voter can resume the process by logging in again, even on a different device.

In 2024, an asymmetric distributed protocol was introduced to improve the system, in line with Measure A.5 of the Catalogue of Measures by the Confederation and Cantons. This enhanced protocol reduces the trust assumptions needed to ensure vote privacy and verifiability.

Features comparison. Table 1.3.1 highlights the voter experience and the technologies used in the implementation of the previously presented e-voting systems. *Receipt* indicates whether it is possible to reveal the content of the vote to a third party using the receipt given by the protocol.

1.3.2 Related Works on Coercion Resistance

To mitigate coercion many strategies have been proposed:

- *Tracking numbers.* Selene [121] is an e-voting protocol that provides verifiability while mitigating the risk of coercion. It assigns a unique tracking number to each voter, which can be used to verify that their vote has been correctly recorded on a public bulletin board. To prevent coercion, voters do not learn their tracking numbers until after their

votes have been posted on the board. However, challenges arise when a coercer is also a voter, who may be able to identify the coerced voter's tracker. The scheme is most effective in environments with low coercion threats, providing a balance between transparency and security. This tracking problem has been overcome with Hyperion [49].

- *Fake credentials.* The JCJ protocol [83] introduced the concept of fake credentials. This approach allows voters to generate fake credentials, indistinguishable from valid ones, that can be handed to coercers without compromising their legitimate voting rights. Votes cast with fake credentials are discarded during the election's cleansing phase. More discussion on this can be found in Chapters 3, 4 and 6.

- *Decoy Tokens.* The Amun protocol [99] addresses over-the-shoulder coercion by introducing decoy tokens. These tokens provide voters with a means to fulfill coercer demands without compromising the integrity of their true vote. This strategy enhances resistance to immediate, physical coercion while requiring careful implementation to manage tokens securely and intuitively.

- *Masking.* The masking technique [9] counters coercion by providing each voter with a unique, secret mask, which is used to blind their true vote during normal voting. If coerced to vote for a different choice, the voter computes a fake mask such that the resulting blinded vote still remains a vote for their actual preference. This approach ensures plausible deniability, as the coerced vote appears valid to the coercer, while the true preference is recorded.

- *Hedgehogs.* Chaum et al. [33] propose a voting system which allows any voter to effectively nullify their vote in an irrevocable and anonymous manner. This approach enables each voter to enlist one or more trusted agents, referred to as *hedgehogs*. Using a zero-knowledge proof, the voter or their hedgehogs can nullify the vote by demonstrating knowledge of the voter's secret key, ensuring that the action is both untraceable and permanent.

1.4 Summary of Contributions

The contributions of this thesis span multiple disciplinary areas due to the interdisciplinary nature of the research. Below is a list of the major results,

categorized into different areas of focus, along with references to more detailed explanations.

Chapters 3 and 4. First, from a cryptographic and protocol design perspective, the following contributions are presented:

1. a remote e-voting protocol with linear vote tallying (see Chapter 3), which has been implemented in a real-world application, *Vote App*, and is supported by a proof of security (see Chapter 4) with, in particular:
   (a) a novel method for managing credentials, aimed at improving the user experience while complying with all the security requirements (see Sections 3.6.3 and 3.7);
   (b) a ballot encoding tailored to the Italian electoral law, with associated zero-knowledge proofs for ballot correctness (see Sections 3.8.1 and 3.8.3).

Chapter 5. In addition, this thesis contributes to the design of a secure e-voting architecture, specifically:

1. an architecture for an OAuth 2.0-based [71] e-voting solution by identifying relevant OAuth 2.0 specifications and best practices to provide a secure e-voting protocol (see [133]);
2. a new token, called `CAT`, to authorize ballot casting, designed to prevent brute-forcing of the credential while maintaining unlinkability between vote and voter (see Section 5.2).

Chapter 6. Regarding threat modeling and risk assessment, the following contributions were made:

1. the adaptation of STRIDE methodology [126] and LINDDUN framework [94] to the e-voting context (see Section 6.2.1);
2. the identification of the most significant attackers for an e-voting system, with particular emphasis on a new attacker called *Coercer* (see Section 6.2.2);

3. the definition of new impact factors specifically tailored to the e-voting scenario, the adaptation of the likelihood factors of the OWASP Risk Rating methodology [110], along with an improved methodology for risk computation (see Section 6.2.3);

4. the application of this new methodology to identify threats and assess risks in *Vote App* (see Section 6.3).

Appendix A. Finally, with regard to *Vote App*, this thesis also addresses its usability, which is an essential aspect to take into account for the practical adoption of an internet voting system. The key contributions in this area include:

1. a usability study to evaluate the effectiveness of a selected glyph alphabet in assisting users in verifying the integrity of long byte strings (see Appendix A.3);

2. a usability study of the key features of *Vote App* (see Appendix A.4).

For reference, *Vote App* interfaces are available in Appendix A.6.

Chapter 7. Another contribution, still in the context of e-voting, is a further development of the e-voting protocol designed during my master's thesis [134], extended to support elections with more than two candidates and to allow voters to express multiple preferences.

1.4.1 Publications

The aforementioned contributions have been presented to the scientific community in the following publications:

- *Riccardo Longo, Umberto Morelli, Chiara Spadafora, and Alessandro Tomasi. "Adaptation of an i-voting scheme to Italian Elections for Citizens Abroad". In: University of Tartu Press (2022).* ([97], Chapter 3);

- *Riccardo Longo, Chiara Spadafora, and Francesco Antonio Marino. "Vote App: Verifiable and Coercion-Resistant I-Voting". In: De Cifris Koine 1 (Apr. 2024), pp. 107–109* ([96], Chapter 3);
- *C. Spadafora, M. Bitussi, R. Longo, F. Antonio Marino, U. Morelli, A. Sharif, C. Spadafora, and A. Tomasi. Coercion-resistant i-voting with short PIN and OAuth 2.0. E-Vote-ID 2023. 2023.* ([133], Chapter 5);
- *Riccardo Longo, Majid Mollaeefar, Umberto Morelli, Chiara Spadafora, Alessandro Tomasi, and Silvio Ranise. "Modeling and Assessing Coercion Threats in Electronic Voting". In: Crisis 2024: 19th International Conference on Risks and Security of Internet and Systems. Springer, 2024* ([98][1], Chapter 6);
- *Riccardo Longo. and Chiara Spadafora. "Amun: Securing E-Voting Against Over-the-Shoulder Coercion". In: Proceedings of the 21st International Conference on Security and Cryptography - SECRYPT. INSTICC. SciTePress, 2024* ([99], Chapter 7);
- *Laura Cristiano and Chiara Spadafora. "Enhancing Usability in E-Voting Systems: Balancing Security and Human Factors with the HC3 Framework". In: HCI International 2024 Posters. 26th International Conference on Human-Computer Interaction, Proceedings, Part VI. CCIS Springer Nature, 2024.* ([47], Appendix A);
- *Laura Cristiano, Riccardo Longo, and Chiara Spadafora. "Click and Cast - Assessing the Usability of Vote App". In: E-Vote-ID 2024. Bonn: Gesellschaft für Informatik, 2024, pp. 67–84* ([46], Appendix A);
- *Simone Brunello, Laura Cristiano, Riccardo Longo, and Chiara Spadafora. "E-Voting with Confidence: Usability Challenges in Man- ual Integrity Checks". In: To appear in International Conference on Human-Computer Interaction. Springer. 2025* ([23], Appendix A).

From now on, unless stated otherwise, the terms electronic voting and e-voting will refer exclusively to *internet voting*.

The research conducted during my PhD, and presented in this thesis, was supported by a joint laboratory between Fondazione Bruno Kessler (FBK) and the Istituto Poligrafico e Zecca dello Stato (IPZS).

[1] This article was awarded Best Paper at *Crisis 2024: 19th International Conference on Risks and Security of Internet and Systems* [45].

Table 1.1: Overview of the Council of Europe principles for e-voting [43].

Principle	Name	Description
I	Universal Suffrage	The e-voting system must be easy to use for all voters, including those with disabilities. Remote voting should be optional unless it is universally accessible.
II	Equal Suffrage	All voting channels must present information equally, and secure methods are needed to aggregate the results. Strong voter authentication must be implemented.
III	Free Suffrage	The system must protect voter autonomy by ensuring that votes are freely cast and verified. It must also allow voters to confirm their choices and be notified of casting an invalid vote.
IV	Secret Suffrage	The e-voting process must maintain the secrecy of the vote and prevent any unauthorized disclosure of votes or voter identities, including during storage and transmission.
V	Regulatory and Organizational Requirements	E-voting must be introduced gradually and accompanied by updated legislation that defines roles and responsibilities for all the parties involved.
VI	Transparency and Observation	The authorities managing the e-voting system must ensure transparency, they must provide clear information to voters and allow external observation of the operation of the system.
VII	Accountability	E-voting systems must be subject to evaluation, certification, and ongoing auditing to ensure compliance with legal and democratic principles. Any problems must be reported and addressed.
VIII	Reliability and Security of the System	The electoral management body is responsible for the security and reliability of the e-voting system, including the protection of data and the response to potential threats.

Table 1.2: Mapping of aggregated properties to the constituent properties from Sec. 1.2 and the corresponding CoE recommendations and principles on e-voting systems.

Aggregated Property	**Constituent Properties**	**CoE Principles**	**CoE Recommendations**
Functional and Security Require-ments	SC, EV1, RTV, CO, FA	II: Equal suffrage IV: Secret suffrage VIII: Reliability and security	7, 8 24 45
Vote Freedom	CR, VP	III: Free suffrage IV: Secret suffrage	10 19, 23, 25, 26
Verifiability	CAIV, RACV, TARV, UV, EV2	II: Equal suffrage III: Free suffrage V: Regulatory and organizational VIII: Reliability and security	9 15, 16, 17, 18 30 49

Table 1.3: Summary of voter experience and implementation technologies. *CR* indicates whether the system is Coercion-Resistant or not. The final row provides a summary of *Vote App*, the voting protocol discussed in this thesis.

	Auth. Mechanism	**Voting Method**	**Verif. Method**	**Receipt**	**CR**
ElectionGuard	Smartcard-based at polling stations, unknown remoely	Voting machine or mobile app	Browser-based	No	No
Helios	Password-based or third-party OpenID or OAuth provider	Browser-based	Browser-based	No	No
Belenios	Password-based, CAS or third-party OpenID provider	Browser-based	Browser-based	No	No
IVXV	eID or Estonian mobileID-based	Desktop app	Mobile app (Android/iOS)	Yes	No
iVote	Password-based	Browser-based or phone voting	Mobile app (Android/iOS)	Yes	No
Swiss Post	Code-based	Browser-based	Browser-based	Yes	No
Vote App	eID	Mobile app (Android/iOS)	Browser-based or Mobile app (Android/iOS)	No	Yes

Chapter 2

Mathematical Background

This chapter provides the essential mathematical background needed to understand the rest of the thesis.

2.1 Introductory Definitions

Definition 1 (Negligible Function)**.** *Let $f : \mathbb{N} \to \mathbb{R}$ be a function. We say that f is* negligible in λ *if for every $c \in \mathbb{N}$ there exists $m \in \mathbb{N}$ such that*

$$|f(\lambda)| < \frac{1}{\lambda^c} \quad \forall \lambda > m.$$

We will say that, given two distributions $\mathcal{D}_1, \mathcal{D}_2$, they are *indistinguishable* if, informally speaking, it is impossible to decide whether, given some data, its original distribution is $\mathcal{D}_1$ or $\mathcal{D}_2$.

Definition 2 (Indistinguishability)**.** *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two probability distributions. We say that $\mathcal{D}_1$ and $\mathcal{D}_2$ are indistinguishable if, for any algorithm $\mathbb{B}$, there exists a negligible function ν such that for all $n \in \mathbb{N}$*

$$\Pr[\mathbb{B}(\mathcal{D}_1(n)) = 1] - \Pr[\mathbb{B}(\mathcal{D}_2(n)) = 1] \leq \nu(n).$$

The algorithm $\mathbb{B}$ in Definition 2 is often called *distinguisher*.

From now on, let p be a prime and $\mathbb{G}$ a group of order p.

2.2 Discrete Logarithm Assumptions

2.2.1 Decisional Diffie-Hellman Assumption

Let a, b, $\xi \in \mathbb{Z}_p^*$ be chosen at random and g be a generator of a cyclic group $\mathbb{G}$ of order p. The *Decisional Diffie-Hellman Problem* (DDH) consists in constructing an algorithm

$$\mathbb{B}\left(g, A = g^a, B = g^b, T\right) \to \{0, 1\}$$

to distinguish between the tuples $\left(g, A, B, g^{ab}\right)$ and $\left(g, A, B, g^{\xi}\right)$, outputting respectively 1 and 0. The advantage of $\mathbb{B}$ in this case is written as:

$$Adv_{\mathbb{B}} = \left|\mathbb{P}\left[\mathbb{B}\left(g, A, B, g^{ab}\right) = 1\right] - \mathbb{P}\left[\mathbb{B}\left(g, A, B, g^{\xi}\right) = 1\right]\right|,$$

where the probability is taken over the random choice of the generator g, of the exponents a, b, $\xi \in \mathbb{Z}_p^*$, and of the random bits possibly consumed by $\mathbb{B}$ to compute the response.

Definition 3 (DDH Assumption)**.** *The Decisional Diffie-Hellman assumption holds if no probabilistic polynomial-time algorithm $\mathbb{B}$ has a non-negligible advantage in solving the DDH problem.*

Residue Classes of Integers. Let p be a prime, and $\mathbb{Z}_p^*$ the multiplicative subgroup of $\mathbb{Z}_p$. We define $\mathbb{Z}_p^r$ as the set of r-th residues in $\mathbb{Z}_p^*$, i.e. $\mathbb{Z}_p^r = \{y \in \mathbb{Z}_p^* \text{ for which } \exists x \in \mathbb{Z}_p^* : y \equiv x^r \mod p\}$. When p is a prime for which $p - 1 = qr$ with q a prime that is not a divisor of $r \in \mathbb{Z}$, then $\mathbb{Z}_p^r$ is a cyclic subgroup of $\mathbb{Z}_p$ with order q, and for each $y \in \mathbb{Z}_p$, $y \in \mathbb{Z}_p^r$ if and only if $y^q \equiv 1 \mod p$.

If p and q are big enough (e.g. with length 4096 bits and 256 bits) and $r/2$ is prime, the DDH assumption of Definition 3 is supposed to hold in $\mathbb{Z}_p^r$.

2.2.2 q-Strong Decisional Diffe-Hellman Assumption

q-SDH-I [19, 8]. Let g be a generator of $\mathbb{G}$. Let $y \in \mathbb{Z}_p^*$, $c \in \mathbb{Z}_p$ be chosen uniformly at random.

The *q-Strong Diffie-Hellman Problem I (q-SDH-I)* [19, 8] consist in constructing an algorithm

$$\mathbb{B}(g, g^y, \ldots, g^{y^q}) \to \left(c, g^{\frac{1}{y+c}}\right).$$

$\mathbb{B}$ has advantage ϵ in solving the q-SDH-I problem if

$$Adv_{\mathbb{B}} = \left| \mathbb{P}\left[\mathbb{B}(g, g^y, \ldots, g^{y^q}) = \left(c, g^{\frac{1}{y+c}} \right) \right] \right| \geq \epsilon,$$

where the probability is taken over the random choice of the generator g and c and of the random bits possibly consumed by $\mathbb{B}$ to compute the response.

Definition 4 (q-Strong Diffie-Hellman Assumption I (q-SDH-I))**.** *The q-Strong Diffie-Hellman assumption I holds if no probabilistic polynomial-time algorithm $\mathbb{B}$ has a non-negligible advantage in solving the q-SDH-I problem.*

q-SDH-II [19, 8]. Let g_1, g_2 be two generators of $\mathbb{G}$. Let $y \in \mathbb{Z}_p^*$, $(x_i, r_i) \in \mathbb{Z}_p^2$ for all $1 \leq i \leq q-1$ be chosen uniformly at random, and let $B_i = (g_1 g_2^{x_i})^{\frac{1}{y+r_i}}$ for all $1 \leq i \leq q-1$.

The *q-Strong Diffie-Hellman Problem II (q-SDH-II)* [19, 8] consist in constructing an algorithm

$$\mathbb{B}(g_1, g_2, g_2^y, B_1, \ldots, B_{q-1}) \rightarrow \left(x, r, A = (g_1 g_2^x)^{\frac{1}{y+r}} \right).$$

$\mathbb{B}$ has advantage ϵ in solving the q-SDH-II problem if

$$Adv_{\mathbb{B}} = \left| \mathbb{P}\left[\mathbb{B}(g_1, g_2, g_2^y, B_1, \ldots, B_{q-1}) = \left(x, r, A = (g_1 g_2^x)^{\frac{1}{y+r}} \right) \right] \right| \geq \epsilon,$$

where the probability is taken over the random choice of the generators g_1, g_2 and (x_i, r_i) and of the random bits possibly consumed by $\mathbb{B}$ to compute the response.

Definition 5 (q-Strong Diffie-Hellman Assumption II (q-SDH-II))**.** *The q-Strong Diffie-Hellman assumption II holds if no probabilistic polynomial-time algorithm $\mathbb{B}$ has a non-negligible advantage in solving the q-SDH-II problem.*

2.2.3 Strong Decision Diffie-Hellman Inversion Assumption

SDDHI-I [8, 26]. Let $a, r, \xi \in \mathbb{Z}_p^*$ be chosen at random and g be a generator of $\mathbb{G}$. Let $\mathcal{O}_a(\cdot)$ be an oracle that on input $z \in \mathbb{Z}_p^*$ outputs $g^{\frac{1}{a+z}}$.

The *Strong Decision Diffie-Hellman Inversion Problem I (SDDHI-I)* consists in constructing an algorithm, that do not query the oracle on r,

$$\mathbb{B}^{\mathcal{O}_a}(g, g^a, T) \to \{0, 1\}$$

to distinguish between the tuples $\left(g, g^a, g^{\frac{1}{a+r}}\right)$ and $\left(g, g^a, g^{\xi}\right)$, outputting respectively 1 and 0. The advantage of $\mathbb{B}$ in this case is written as:

$$Adv_{\mathbb{B}} = \left|\mathbb{P}\left[\mathbb{B}^{\mathcal{O}_a}\left(g, g^a, g^{\frac{1}{a+r}}\right) = 1\right] - \mathbb{P}\left[\mathbb{B}^{\mathcal{O}_a}\left(g, g^a, g^{\xi}\right) = 1\right]\right|,$$

where the probability is taken over the random choice of the generator g, of the exponents $a, r, \xi \in \mathbb{Z}_p^*$, and of the random bits possibly consumed by $\mathbb{B}$ to compute the response.

Definition 6 (Strong Decision Diffie-Hellman Inversion Assumption I (SDDHI-I))**.** *The Strong Decision Diffie-Hellman Inversion assumption I holds if no probabilistic polynomial-time algorithm $\mathbb{B}$ has a non-negligible advantage in solving the SDDHI-I problem.*

SDDHI-II [8, 26]. Let $a, r, x \in \mathbb{Z}_p^*$ be chosen at random and g_1, g_2 be two generators of $\mathbb{G}$ and $\Xi \in \mathbb{G}$ random. Let $\mathcal{O}_a(\cdot)$ be an oracle that on input $z, t \in \mathbb{Z}_p^*$ outputs $(g_1 g_2^t)^{\frac{1}{a+z}}$.

The *Strong Decision Diffie-Hellman Inversion Problem II (SDDHI-II)* consists in constructing an algorithm, that do not query the oracle on r and x,

$$\mathbb{B}^{\mathcal{O}_a}(g_1, g_2, g_2^a, T) \to \{0, 1\}$$

to distinguish between the tuples $\left(g_1, g_2, g_2^a, (g_1 g_2^x)^{\frac{1}{a+r}}\right)$ and (g_1, g_2, g_2^a, Ξ), outputting respectively 1 and 0. The advantage of $\mathbb{B}$ in this case is written as:

$$Adv_{\mathbb{B}} = \left|\mathbb{P}\left[\mathbb{B}^{\mathcal{O}_a}\left(g_1, g_2, g_2^a, (g_1 g_2^x)^{\frac{1}{a+r}}\right) = 1\right] - \mathbb{P}\left[\mathbb{B}^{\mathcal{O}_a}(g_1, g_2, g_2^a, \Xi) = 1\right]\right|,$$

where the probability is taken over the random choice of the generators g_1, g_2, of Ξ, of the exponents $a, r, x \in \mathbb{Z}_p^*$, and of the random bits possibly consumed by $\mathbb{B}$ to compute the response.

Definition 7 (Strong Decision Diffie-Hellman Inversion Assumption II (SDDHI-II))**.** *The Strong Decision Diffie-Hellman Inversion assumption II holds if no probabilistic polynomial-time algorithm $\mathbb{B}$ has a non-negligible advantage in solving the SDDHI-II problem.*

2.3 Hash Functions

Hash functions are somewhat the digital equivalent of a fingerprint: they compress a file in a digest that represents it, so that (ideally) no two different files correspond to the same digest. Moreover it should be almost impossible to reconstruct the whole file from the digest, but at the same time it should be easy to show that a given file corresponds to a given digest.

A hash function H can be idealized as a function whose set of inputs is the set of all possible binary strings, denoted by $(\mathbb{F}_2)^*$, while its set of possible outputs is the set of all binary strings of given length (called *digest*).

A cryptographic hash function should have the following properties:

- *Efficiency*: it is quick to compute the hash value for any given message;
- *Pre-image resistance*: given y it is infeasible to find x such that $y = f(x)$;
- *Second pre-image resistance*: given x_1 it is infeasible to find $x_2 \neq x_1$ such that $f(x_1) = f(x_2)$;
- *Collision resistance*: it is infeasible to find two values $x_1 \neq x_2$ such that $f(x_1) = f(x_2)$;
- *Avalanche effect*: a small change to a message should change the hash value so extensively that a new hash value appears uncorrelated with the old hash value.

Hash functions[1] are widely used in various cryptographic schemes and protocols, such as digital signatures, commitments, and zero-knowledge proofs.

2.3.1 Random Oracle Model

The Random Oracle Model [11] is an idealised security model useful to analyze and prove the security of many cryptographic protocols. Given two sets $\mathbb{A}$ and $\mathbb{B}$, a random oracle is a publicly-accessible oracle $\mathcal{O} : \mathbb{A} \to \mathbb{B}$ that behaves as a perfectly random function. From a theoretical point of view no efficient algorithm can possibly be used as random oracle, since a truly random function would have a description that is exponentially large. In

[1] In this thesis, *hash function* will always denote a *cryptographic* hash function.

practice, random oracles are often substituted by hash functions. A protocol is secure in the random oracle model if it is secure when all the pseudo-random functions (usually hash functions) are replaced by random oracles, which does not change answer when a query is repeated. It is evident that the model under consideration is consistent with the aforementioned properties of a cryptographic hash function. This model is in opposition with the Standard Model in which the adversary is only limited by the amount of time and computation available. A scheme is secure in the standard model if it can be proven secure only using complexity assumptions.

2.4 Commitments

Commitment schemes are a cryptographic primitive that allows a sender to commit to a chosen value while keeping it hidden to others. Later, by sending some extra values, the sender is able to reveal the committed value to the receiver.

Definition 8 (Commitment Scheme)**.** *A commitment scheme [21] is a couple* (`Commit`, `Verify`) *such that:*

1. *(Commitment)* `Commit`*: a probabilistic algorithm that takes the message m to commit with some random value r as input and outputs the commitment c and a decommitment value d.*

2. *(Decomittment)* `Verify`*: a deterministic algorithm that takes the commitment c, the message m and the decommitment value d and outputs true if the verification succeeds, false otherwise.*

A commitment scheme must be binding and hiding, namely that it should be difficult to change a committed value once the commitment is sent and that the commitment does not reveal anything about the original message:

- *Binding:* it is infeasible to find values $m' \neq m$ and d, d' such that $\texttt{Verify}(c, m, d) = \texttt{Verify}(c, m', d') = \texttt{true}$. We say that the commitment is *perfectly binding* if the binding property holds even if the adversary has unbounded computational power, otherwise it is *computationally binding.*

- *Hiding:* Let $[c_1, d_1] = \texttt{Commit}(m_1, r_1)$ and $[c_2, d_2] = \texttt{Commit}(m_2, r_2)$ with $m_1 \neq m_2$, then it is infeasible for an attacker having only c_1, c_2,

m_1 and m_2 to distinguish which c_i corresponds to which m_i. We say that the commitment is *perfectly hiding* if the hiding property holds even if the adversary has unbounded computational power, otherwise it is *computationally hiding*.

Notice that perfect hiding and perfect binding are mutually exclusive properties, in fact in a perfectly binding commitment c can be decommitted in at most one way, so a computationally unbounded adversary can violate the hiding property via a brute-force search.

2.4.1 Hash-Based Commitments

A simple commitment scheme to a message m based on a cryptographic hash function H may be instantiated as follows ($||$ denotes concatenation):

- $\texttt{Commit}(m, r) = c = H(m||r)$ with $r \xleftarrow{\$} \{0,1\}^\lambda$;
- $\texttt{Verify}(c, m, r) : c \stackrel{?}{=} H(m||r)$.

This scheme is perfectly binding if H is collision resistant and perfectly hiding if the distribution of digests is computationally indistinguishable from uniformly random output [20]. The former depends on the hashing algorithm; for the latter to be the case, the input must have sufficiently high entropy [138].

2.5 Digital Signatures

Digital signatures are used to certify integrity and authenticity of an electronic document, and to provide non-repudiation.

Definition 9 (Digital Signature Scheme)**.** *A Digital Signature Scheme is a tuple* $(\texttt{KeyGen}, \texttt{Sign}, \texttt{Ver})$ *such that:*

- (Key Generation) `KeyGen`*: given a security parameter* κ *generates a public key* `pk`*, that is published, and a secret key* `sk`*.*
- (Signing) `Sign`*: given a message* m *and the secret key* `sk`*, computes a digital signature* s *of* m*.*

- (Verifying) `Ver`*: given a message m, a signature s and the public key* `pk`*, it outputs whether or not s is a valid signature of m computed by the secret key corresponding to* `pk`*.*

The previous algorithms usually require a source of random bits to operate securely.

A common security requirement for a Digital Signature Scheme is the difficulty of forging a signature of a new message, given a signing oracle that provides signatures of a chosen message (commonly known as an *existential forgery*).

In e-voting protocols digital signatures are usually employed to provide authentication of the messages exchanged, in particular, they can be used to certify that a ballot is cast by an authorized voter.

2.5.1 Blind Signatures

A particular kind of digital signatures, called Blind Signatures, has some properties which make them particularly interesting for e-voting protocols.

With blind signatures the content of a message can be disguised (blinded) before it is signed, while the resulting blind signature can be publicly verified against the original message. This method enables election authorities to approve a voter's ballot without requiring the disclosure of its contents. This process attests to the voter's eligibility as an elector with the right to vote, while safeguarding their privacy.

Blind signatures therefore provide unlinkability, which prevents the signer from linking the blinded message it signs to a later un-blinded version that it may be called upon to verify.

The first blind signature scheme was proposed by Chaum [32], and was implemented with RSA signatures [120]. Successively new schemes have been developed (e.g., [27, 25]).

2.6 Elliptic Curves

Elliptic curves play a fundamental role in modern cryptography and number theory. Formally, an elliptic curve over a field K is defined as the set of

solutions to the Weierstrass equation: $E : y^2 = x^3 + ax + b$ together with the point at the infinity $\mathcal{O}$, where $a, b \in K$ with $4a^3 + 27b^2 \neq 0$.

In order to apply the theory of elliptic curves to cryptography, we need to look at elliptic curves whose points have coordinates in a finite field $K = \mathbb{F}_p$.

Suppose we denote the set of points on the elliptic curve by

$$E(\mathbb{F}_q) = \{(x, y) : x, y \in \mathbb{F}_p \mid y^2 = x^3 + ax + b\} \bigcup \{\mathcal{O}\}$$

It is possible to define an addition operation on $E(\mathbb{F}_q)$ so that $(E(\mathbb{F}_q), +)$ is an abelian group, with neutral element $\mathcal{O}$. If the field is finite, the resulting group is finite and typically either cyclic or the direct product of two cyclic groups.

Definition 10 (Elliptic Curve Discrete Logarithm Problem (ECDLP))**.** *Let E be an elliptic curve over the finite field $\mathbb{F}_p$ and let P and Q be points in $E(\mathbb{F}_p)$. The Elliptic Curve Discrete Logarithm Problem is the problem of finding an integer n such that $Q = nP$.*

The best-known algorithms for solving ECDLP have sub-exponential complexity, while for the classical discrete logarithm problem, there exist index-calculus methods with quasi-exponential complexity. This fundamental difference underlies the advantage of elliptic curve-based constructions in cryptographic applications.

2.6.1 Edwards Curves

Edward Curves [51] are a particular family of elliptic curves defined over a field K by the equation $x^2 + y^2 = c^2(1 + x^2y^2)$ where $c \in K$, $c^5 - c \neq 0$, and the characteristic of K is different from 2. The formulas for addition differ from those of the Weierstrass form.

Let $P_1, P_2 \in \mathcal{E}$, $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, then their sum $P_3 = (x_3, y_3) = P_1 + P_2$ is given by

$$(x_3, y_3) = \left(\frac{x_1y_2 + x_2y_1}{c(1 + x_1x_2y_1y_2)}, \frac{y_1y_2 - x_1x_2}{c(1 - x_1x_2y_1y_2)} \right).$$

A notable property of this addition law is that it remains unchanged in the case of doubling. The neutral element of the group is $(0, c)$, while the inverse of a point (x, y) is given by $(-x, y)$.

Later, Bernstein and Lange showed in [17] that the family of Edwards curves could be extended with the equation $E : -x^2 + y^2 = 1 + dx^2y^2$ over $\mathbb{F}_q$, where q is a prime power congruent to 1 modulo 4 and $d \in \mathbb{F}_q$ is a non square element. The condition that d is not a square implies that $E(\mathbb{F}_q)$ forms a group with neutral element $(0, 1)$ under the twisted Edwards addition law: given $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, their sum $P_3 = (x_3, y_3) = P_1 + P_2$ is given by

$$(x_3, y_3) = \left(\frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 + x_1x_2}{1 - dx_1x_2y_1y_2} \right).$$

2.6.2 EdDSA

In [18] the authors present a new signature scheme called EdDSA which uses twisted Edwards curves instead of Weierstrass curves.

Let $q \equiv 1 \mod 4$ be a prime power, d a non-square element of $\mathbb{F}_q$ and $E(\mathbb{F}_q) = \{(x, y) \in \mathbb{F}_q \times \mathbb{F}_q \mid -x^2 + y^2 = 1 + dx^2y^2\}$.

Let b be an integer, $b \geq 10$, H an hash function producing a $2b$-bit output, $B \in E(\mathbb{F}_q)$, with $B \neq (0, 1)$ and $2^{b-4} \leq l \leq 2^{b-3}$ a prime such that $lB = 0$. We denote with $\underline{\star}$ a $(b-1)$-bit encoding of $\star$.

Protocol 1 (EdDSA)**.** The Edwards-curve Digital Signature Algorithm is defined by the tuple $(\texttt{KeyGen}, \texttt{Sign}, \texttt{Ver})$ where:

1. `KeyGen`. An EdDSA secret key is a b-bit string k randomly chosen. The hash $H(k) = (h_0, h_1, \ldots, h_{2b-1})$ determines an integer

$$a = 2^{b-2} + \sum_{i=3}^{b-3} 2^i h_i,$$

   which in turn determines the public key A, which is the encoding of the point $A = aB$.

2. `Sign`. Let m be the message to be signed with secret key k. Define $r = H(h_0, \ldots, h_{2b-1}, m)$ where we interpret $2b$-bit strings in little-endian form as integers in $\{0, 1, \ldots, 2^{2b} - 1\}$. Let $R = rB$, compute $s \equiv r + aH(\underline{R}, \underline{A}, m)) \mod l$. The signature of m is then the $2b$-bit string $(\underline{R}, \underline{s})$.

3. `Ver`. In order to verify the signature $(\underline{R}, \underline{s})$ the verifier:

   (a) computes R, A from $\underline{R}$, $\underline{A}$ and checks if are elements of $E(\mathbb{F}_q)$;

   (b) computes s from $\underline{s}$ and checks $s \in \{0, 1, \ldots, l-1\}$;

   (c) computes $H(\underline{R}, \underline{A}, m)$ and then checks the equality $8sB = 8R + 8AH(\underline{R}, \underline{A}, m)$.

The verifier rejects the alleged signature any of these checks fail, if the parsing fails or if the group equation does not hold. To see that signatures pass verification, simply multiply $8B$ by the equation $s = r + aH(\underline{R}, \underline{A}, m) + zl$, and use the fact that $lB = 0$, to see that

$$8sB = 8rB + 8aBH(\underline{R}, \underline{A}, m) + 8zlB = 8R + 8AH(\underline{R}, \underline{A}, m).$$

2.7 Polynomial Interpolation

Given $N+1$ distinct points $(x_0, \ldots, x_N)$ and $(y_0, \ldots, y_N)$ generated by a function $y = f(x)$, the problem of *Polynomial Interpolation* consists in finding a polynomial $p(x)$ of degree N such that $p(x_i) = y_i$ for all $i = 0, \ldots, N$.

2.7.1 Lagrange Polynomials

Given $N+1$ pairs (x_i, y_i), the interpolation polynomial in the *Lagrange form* is the linear combination

$$p(x) = \sum_{k=0}^{N} y_k \cdot \lambda_k(x) \quad \text{of} \quad \lambda_k(x) = \prod_{j=0, j\neq k}^{N} \frac{x - x_j}{x_k - x_j}$$

where the $\lambda_k(x)$ are called *Lagrange Coefficients*.

Using the same technique it is possible to interpolate a polynomial in the exponent. Given $N+1$ pairs (x_i, g^{y_i}) and the Lagrange coefficients, we can easily compute:

$$\prod_{k=0}^{N} (g^{y_i})^{\lambda_k(x)} = g^{\sum_{k=0}^{N} y_k \cdot \lambda_k(x)} = g^{p(x)}.$$

2.8 Secret Sharing Schemes

Secret sharing is a basic technique in Secure Multi-Party Computation which allows to divide a secret s into multiple shares that can be distributed among n parties in such a way that with less than $k \leq n$ shares no information can be gained about s, while with k shares the whole secret can be reconstructed.

The classical protocols require a *dealer* who shares a secret among n parties. In some cases it is desirable to remove the dealer, so that no one knows the secret shared until it is reconstructed.

Moreover, some protocols also require the parties to verify the consistency of their shares, so verifiable secret sharing protocols should be employed.

These two conditions can be achieved with the following protocol derived from Feldman's protocol (see [57, 141]):

Protocol 2. Let p be a prime, let $\mathbb{G}$ be a cyclic group of order p with generator g for which the DDH assumption holds. Let $(P_1, \ldots, P_n)$ be n participants in the protocol, each one having a secret $\hat{s}_i \in \mathbb{Z}_p$, for all $1 \leq i \leq n$. A verifiable t-out-of-n sharing of $s = \sum_{i=1}^{n} \hat{s}_i$ can be achieved through the following protocol:

1. P_i, $\forall 1 \leq i \leq n$, generates uniformly at random $(\hat{a}_{i,1}, \ldots, \hat{a}_{i,t-1}) \in \mathbb{Z}_p^{t-1}$;
2. P_i, $\forall 1 \leq i \leq n$, computes the polynomial $f_i(x) = \sum_{k=1}^{t-1} \hat{a}_{i,k} \cdot x^k + \hat{s}_i$;
3. for every $1 \leq j \leq n$, P_i sends to P_j the share $(j, s_{i,j})$ with $s_{i,j} = f_i(j)$;
4. P_i publishes the commitments $A_{i,k} = g^{\hat{a}_{i,k}}$ for $1 \leq k \leq t-1$ and $A_{i,0} = g^{\hat{s}_i}$;
5. for $1 \leq j \leq n$, P_j can verify the share $s_{i,j}$ received by P_i by checking that $g^{s_{i,j}} = \prod_{k=0}^{t-1} (A_{i,k})^{j^k}$;
6. P_i computes (after having received and verified all n shares) $s_i = \sum_{k=1}^{n} s_{k,i}$;
7. the share of P_i of the secret s is (i, s_i).

To reconstruct the secret it is sufficient to interpolate at least t points to reconstruct the polynomial, and then evaluate it in 0 to compute s.

2.9 Encryption Schemes

Asymmetric or *public-key cryptography* is one of the two main families of encryption schemes in which the receiver holds a secret-public key pair, while the sender has access only to the public key. Formally, it is defined as follows:

Definition 11 (Public-Key Encryption Scheme)**.** *A public-key encryption scheme is defined by the tuple* (`Setup`, `KeyGen`, `Enc`, `Dec`) *where:*

- (Setup) `Setup`*: on input a security parameter* λ*, it outputs public parameters* `pp`*.*
- (Key Generation) `KeyGen`*: it is a probabilistic algorithm that takes as input the public parameters* `pp`*, the security parameter* λ *and outputs a public key* `pk` *and the corresponding secret key* `sk`*.*
- (Encryption) `Enc`*: it is a probabilistic algorithm that takes as input a public key* `pk` *and a message, called plaintext, and outputs an encryption of it, called ciphertext;*
- (Decryption) `Dec`*: it is a deterministic algorithm which takes as input a secret key* `sk` *and a ciphertext and outputs a message or failure.*

From now on, the term *encryption* will always refer to the use of a public-key encryption scheme.

Homomorphic Encryption is a type of encryption that allows to operate some transformations to encrypted messages without the need to decrypt and re-encrypt them. That is, an encryption function E is homomorphic with respect to the function f if there is an efficiently computable function g such that $g(E(m)) = E(f(m))$. This property is very useful in many protocols, which exploit it to manipulate private data without disclosing it.

2.9.1 Modified ElGamal Cryptosystem

The *Modified ElGamal cryptosystem (M-ElGamal)* [83] is a public-key encryption protocol which extends the classical ElGamal cryptosystem (Section 11.5 [20]), with security based on the DDH (Decisional Diffie-Hellman) assumption.

Protocol 3 (Modified ElGamal Cryptosystem)**.** The Modified ElGamal Cryptosystem is defined by the tuple $(\texttt{Setup}, \texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$ where:

1. `Setup`: let p be a prime and $\mathbb{G}$ a cyclic group of order p for which the DDH assumption holds. Let g_1 and g_2 be two generators of $\mathbb{G}$, the tuple $(p, \mathbb{G}, g_1, g_2)$ defines the parameters of the cryptosystem;

2. `KeyGen`: let $x_1, x_2 \in \mathbb{Z}_p$ be chosen uniformly at random and let $h = g_1^{x_1} \cdot g_2^{x_2}$. The private key is the pair (x_1, x_2), the public key is h;

3. `Enc`: let $M \in \mathbb{G}$ be the message to be encrypted for the public key h, and let $r \in \mathbb{Z}_p$ chosen uniformly at random: the ciphertext is computed as the tuple $(\alpha, \beta, \gamma) = (g_1^r, g_2^r, h^r \cdot M)$;

4. `Dec`: let (α, β, γ) be the ciphertext to be decrypted with the secret key (x_1, x_2): the message is recovered as $M = \gamma/(\alpha^{x_1} \cdot \beta^{x_2})$.

For succinctness of notation, we will denote with $E_h^r[M]$ the encryption of a message $M \in \mathbb{G}$ computed with randomness $r \in \mathbb{Z}_p$ for the public key h, assuming that g_1 and g_2 are fixed public parameters. Note that the scheme is multiplicatively homomorphic, where the homomorphic multiplication of ciphertext is the point-wise multiplication over $\mathbb{G}^3$: $E_h^r[M] \cdot E_h^{r'}[M'] = E_h^{r+r'}[M \cdot M']$.

2.9.1.1 Modified Exponential ElGamal

The *Modified Exponential ElGamal Cryptosystem* is a variant of the cryptosystem where the ciphertext space is $\mathbb{Z}_p$ and is additively homomorphic, but the decryption is limited to small values of the plaintext.

For the exponential variant, it is sufficient to fix an additional generator g_0 of $\mathbb{G}$ during the setup, and then encrypt a message $m \in \mathbb{Z}_p$, by defining $M = g_0^m$. For the decryption, once M is recovered, we need to compute $m = \log_{g_0} M$, which for small values of m can be done with the help of some precomputation.

Similarly to before, we will denote with $\mathcal{E}_h^r[m]$ the encryption of a message $m \in \mathbb{Z}_p$ computed with randomness $r \in \mathbb{Z}_p$ for the public key h, assuming that g_0, g_1, and g_2 are fixed public parameters. Note that the scheme is additively homomorphic, where the homomorphic addition of ciphertexts is the point-wise multiplication over $\mathbb{G}^3$: $\mathcal{E}_h^r[m] \cdot \mathcal{E}_h^{r'}[m'] = \mathcal{E}_h^{r+r'}[m + m']$.

2.9.2 Threshold Modified (Exponential) ElGamal

Let $(P_1, \ldots, P_n)$ be the participants of the system, each P_i equipped with a secret pair $(\xi_{1,i}, \xi_{2,i}) \in \mathbb{Z}_p^2$. A threshold *t-out-of-n* modified (exponential) ElGamal cryptosystem can be achieved in the following way:

Protocol 4 (Threshold Modified (Exponential) ElGamal)**.** The Threshold Modified (Exponential) ElGamal Cryptosystem is defined by the tuple $(\texttt{Setup}, \texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$ where :

1. *(Setup)* `Setup`: let p be a prime and $\mathbb{G}$ a cyclic group of order p for which the DDH assumption holds. Let g_1 and g_2 be two generators of $\mathbb{G}$, the tuple $(p, \mathbb{G}, g_1, g_2)$ defines the parameters of the cryptosystem;

2. `KeyGen`:

   (a) P_i, $1 \leq i \leq n$, performs Protocol 2 with $\hat{s}_i = \xi_{1,i}$ (using generator g_1) to retrieve the share $(i, x_{1,i})$ and with $\hat{s}_i = \xi_{2,i}$ (using generator g_2) to retrieve the share $(i, x_{2,i})$;

   (b) using the values $g_1^{\xi_{1,i}}$ and $g_2^{\xi_{2,i}}$ published during the previous step, anyone can compute the public key h of the cryptosystem as $h = \prod_{i=1}^{n} g_1^{\xi_{1,i}} g_2^{\xi_{2,i}}$.

   (c) Note that each P_i owns a t-out-of-n share of the private key (x_1, x_2), which is $(x_{1,i}, x_{2,i})$.

3. `Enc`: same as the standard version of the cryptosystem.

4. `Dec`: let (α, β, γ) be the ciphertext to be decrypted:

   (a) P_i computes and broadcasts the couple $(i, \alpha^{x_{1,i}} \cdot \beta^{x_{2,i}})$;

   (b) after *at least* $2t + 1$ participants P_j broadcast $(i, \alpha^{x_{1,j}} \cdot \beta^{x_{2,j}})$, P_i computes the Lagrangian interpolation (see Section 2.7.1) of the points in $x = 0$ which yields h^r.

The parties $(P_1, \ldots, P_n)$ can *verifiably decrypt* a ciphertext by attaching to the decryption a threshold proof of correct decryption (see Protocol 12).

2.10 Zero Knowledge Proofs

A zero-knowledge proof (ZKP) is a cryptographic protocol which allows one party (the prover $\mathcal{P}$) to convince another party (the verifier $\mathcal{V}$) about the truth of some statement, without revealing anything else to the verifier. In the context of e-voting, we would like the protocol to be non-interactive and designated verifier, i.e. the prover publishes some sort of evidence of the truthfulness of the statement that can be independently verified by only the designated parties later on.

2.10.1 General Properties and Structure

Given a language $\mathcal{L}$ and a common input x then the three basic properties of a zero-knowledge proof are:

Definition 12 (Completeness)**.** *If $x \in \mathcal{L}$ (i.e. the prover is honest) then the verifier should accept the proof with probability* 1*.*

Definition 13 (Soundness)**.** *If $x \notin \mathcal{L}$ (i.e. the prover wants to convince the verifier to know something that it does not know or the validity of a property that is actually false) then the verifier should only accept with negligible probability.*

Definition 14 (Zero-Knowledge)**.** *For every verifier $\mathcal{V}$ there exists an efficient simulator that can generate transcripts that are indistinguishable from real interaction between a real prover and $\mathcal{V}$.*

The third property guarantees that the verifier learns nothing from the interaction, except that $x \in \mathcal{L}$.

In some cases, we assume that the verifier is honest. In this context, an *honest* verifier means that the verifier follows the protocol correctly, without trying to cheat or gain additional knowledge beyond what is strictly necessary to verify the claim.

Definition 15 (Honest-Verifier Zero-Knowledge)**.** *For every honest verifier $\mathcal{V}$ (i.e., follows the protocol correctly, does not deviate from the prescribed steps or tries to gain additional information) there exists an efficient simulator that can generate transcripts that are indistinguishable from real interaction between a real prover and an honest verifier $\mathcal{V}$.*

In a ZKP, in order to prove that $x \in \mathcal{L}$, $\mathcal{P}$ has access to a witness w, and uses it to convince $\mathcal{V}$ that $x \in \mathcal{L}$ without disclosing w.

Definition 16 (Witness)**.** *A value w is a witness of the statement $x \in \mathcal{L}$, if the existence of w implies $x \in \mathcal{L}$, and given w it is easy to check that $x \in \mathcal{L}$.*

The vast majority of ZKPs are derived from sigma protocols, a general class of interactive protocols with a commitment-challenge-response structure.

Definition 17 (Sigma Protocol)**.** *Let w be a witness of $x \in \mathcal{L}$, a Sigma Protocol is an interactive protocol used to build ZKP with the following structure:*

1. *$\mathcal{P}$ randomly generates a value t which will be used to mask w, then sends to $\mathcal{V}$ a commitment I of t;*
2. *$\mathcal{V}$ randomly generates a challenge c and sends it to $\mathcal{P}$;*
3. *$\mathcal{P}$ derives from t, c, and w a response z and sends it to $\mathcal{V}$;*
4. *$\mathcal{V}$ checks that z satisfies some properties in relation to c, I, x, $\mathcal{L}$: if the verification succeeds then $\mathcal{V}$ accepts the proof, otherwise it rejects it.*

In order to achieve soundness, some ZKP require multiple iterations (called rounds) of the sigma protocol. Others use a single round, but often this comes at the cost of only achieving a somewhat weaker notion of zero-knowledge.

To achieve the zero-knowledge property, given c in advance it should be possible for $\mathcal{P}$ to compute I and a response z without knowing w, in such a way that the checks performed by $\mathcal{V}$ in the last step succeed without problems.

2.10.2 Fiat-Shamir Non-Interactive ZK Proofs

In many contexts, such as public verifiability in e-voting protocols, it is necessary to prove a statement (in zero-knowledge) to many parties, so an interactive protocol becomes quite inconvenient. In these cases it is much preferable if the prover can publish some sort of evidence of the truthfulness of the statement that can be independently verified by all the relevant parties later on. Such sort of ZKP is called Non-Interactive Zero-Knowledge

Proof (NIZKP), and the Fiat-Shamir technique can be used to transform an interactive sigma protocol (see Definition 17) into a NIZKP by exploiting a hash function modeled as a random oracle (see Section 2.3.1).

The technique requires to derive the challenge value c deterministically by hashing all the public values involved in the ZKP, including the commitment I and the common value x. This method assures that $\mathcal{P}$ cannot choose c before I, so a single-round ZKP can be transformed in a NIZKP that works as follows:

1. $\mathcal{P}$ performs the first step as in the ZKP, derives $c = H(x, I)$, then computes z as in the third step of the ZKP, and publishes (x, I, z);
2. $\mathcal{V}$ computes $c = H(x, I)$, then performs the check on z, I, c, and x as in the last step of the ZKP.

2.10.3 Cramer-Damgård-Schoenmakers OR Proofs

Many applications require to prove that one of two statements is true without revealing which one. That is, given a common value x and two languages $\mathcal{L}_0$ and $\mathcal{L}_1$, a prover $\mathcal{P}$ wants to convince a verifier $\mathcal{V}$ that either $x \in \mathcal{L}_0 \vee x \in \mathcal{L}_1$, without revealing which statement is actually true.

Given a ZKP (or NIZKP) for $\mathcal{L}_0$ and $\mathcal{L}_1$, with a challenge $c \in \mathbb{Z}_n$ for some $n \in \mathbb{Z}$ (or equivalently with a challenge that may be deterministically derived from an integer modulo n), the Cramer-Damgård-Schoenmakers technique [44] allows to build a ZKP (or NIZKP) for the disjunction $\mathcal{L}_0 \vee \mathcal{L}_1$. The method exploits the ability of the prover to simulate the proof if c is known in advance (see Definition 17) and the fact that given $c \in \mathbb{Z}_n$ you can freely choose $c_0 \in \mathbb{Z}_n$ and in consequence fix $c_1 \in \mathbb{Z}_n$ such that $c = c_0 + c_1$.

Let w be a witness for $x \in \mathcal{L}_b$ for $b \in \{0, 1\}$, the OR-ZKP proceeds as follows:

1. $\mathcal{P}$ computes and sends to $\mathcal{V}$ the commitments (I_0, I_1):
   (a) chooses t and computes its commitment I_b as in the ZKP;
   (b) chooses $c_{1-b} \in \mathbb{Z}_n$ uniformly at random and computes compatible commitment I_{1-b} and response z_{1-b} that correctly simulate a proof for $x \in \mathcal{L}_{1-b}$ (for which $\mathcal{P}$ does not have a witness);
2. $\mathcal{V}$ responds with a challenge $c \in \mathbb{Z}_n$;

3. $\mathcal{P}$ computes $c_b = c - c_{1-b}$, then computes a response v_b using c_b, t and w as in the ZKP, then sends to $\mathcal{V}$ the challenges and responses (c_0, v_0, c_1, v_1);

4. $\mathcal{V}$ accepts the proof if and only if $c = c_0 + c_1$ and both the ZKP checks for $(\mathcal{L}_0, x, I_0, c_0, v_0)$ and $(\mathcal{L}_1, x, I_1, c_1, v_1)$ verify the validity of the tuples.

Note that this construction is easily adapted into a NIZKP by deriving deterministically $c = H(x, I_0, I_1)$ and publishing $(x, I_0, I_1, c_0, c_1, v_0, v_1)$.

Moreover this same technique can be easily extended to prove the disjunction of multiple languages $\vee_{i=1}^{n} \mathcal{L}_i$. In fact, given $c \in \mathbb{Z}_n$ it is possible to freely choose c_i for $1 \leq i < n$ and fix $c_n = c - \sum_{i=1}^{n-1} c_i$, so $n-1$ proofs can be simulated by freely choosing c_i, has a forced challenge under the constraint $c = \sum_{i=1}^{n} c_i$ so cannot be simulated, but the verifier cannot know which proof is real.

2.10.4 Explicit Constructions of NIZK Proofs

In the following proofs we assume p to be a prime, $\mathbb{G}$ a cyclic group of order p and $\mathcal{H}_p$ a hash function with output in $\mathbb{Z}_p$. To avoid possible attacks, the inputs to $\mathcal{H}_p$ should always include the public parameters such as the group generators and the public keys. To simplify the notation, in the following we omit them but we assume that they are included.

Protocol 5 (Proof of Plaintext Knowledge)**.** Using the protocol of Okamoto [125], a prover $\mathcal{P}$ may prove to a verifier $\mathcal{V}$ that a tuple C has the form $C = (\ell^\rho, w^\rho, h^\rho \cdot g^m)$, for secret $\rho, m \in \mathbb{Z}_p$ and four public group generators (ℓ, w, h, g) of $\mathbb{G}$, in the following way:

1. $\mathcal{P}$ chooses uniformly at random $t_1, t_2 \in \mathbb{Z}_p$;

2. $\mathcal{P}$ computes the commitments $I = h^{t_1} \cdot g^{t_2}$, $I_1 = \ell^{t_1}$ and $I_2 = w^{t_1}$;

3. $\mathcal{P}$ computes the challenge $c = \mathcal{H}_p(I, I_1, I_2, C, \ell, w, h, g)$;

4. $\mathcal{P}$ computes the response $z_1 = t_1 + c \cdot \rho$ and $z_2 = t_2 + c \cdot m$;

5. $\mathcal{P}$ sends the tuple (I, I_1, I_2, z_1, z_2) to $\mathcal{V}$.

To check the validity of this proof:

1. $\mathcal{V}$ computes the challenge $c \stackrel{?}{=} \mathcal{H}_p(I, I_1, I_2, C, \ell, w, h, g)$;

2. $\mathcal{V}$ computes

$$\ell^{z_1} \stackrel{?}{=} I_1 \cdot (C[1])^c \qquad w^{z_1} \stackrel{?}{=} I_2 \cdot (C[2])^c \qquad h^{z_1} \cdot g^{z_2} \stackrel{?}{=} I \cdot (C[3])^c;$$

and accepts if the equations hold.

Note that, given a commitment $\hat{g}^m$ of m (where $\hat{g} \in \mathbb{G}$ is a known generator), the proof above can be extended to prove that g (in C) and $\hat{g}$ have the same exponent by adding to the NIZKP the value $I = \hat{g}^{t_2}$, and adding the check that

$$\hat{g}^{z_2} \stackrel{?}{=} I \cdot (\hat{g}^m)^c.$$

Protocol 6 (Proof of Representation Equality)**.** Following [125], a prover $\mathcal{P}$ may prove to a verifier $\mathcal{V}$ the equality of the two public representations $C_1 = (h^{\rho_1} \cdot g^{\mu})$, $C_2 = (h^{\rho_2} \cdot g^{\mu})$ for a secret $\mu \in \mathbb{Z}_p$, $\rho_1 \neq \rho_2$ and two public group generators (h, g) of $\mathbb{G}$, in this way:

1. $\mathcal{P}$ chooses uniformly at random $t \in \mathbb{Z}_p$;

2. $\mathcal{P}$ computes the commitment $I = h^t$;

3. $\mathcal{P}$ computes the challenge $c = \mathcal{H}_p(I, C_1, C_2, h, g)$;

4. $\mathcal{P}$ computes the response $z = t + c \cdot (\rho_1 - \rho_2)$;

5. $\mathcal{P}$ sends the pair (I, z) to $\mathcal{V}$.

To check the validity of this proof:

1. $\mathcal{V}$ computes the challenge $c \stackrel{?}{=} \mathcal{H}_p(I, C_1, C_2, h, g)$;

2. $\mathcal{V}$ computes

$$h^z \stackrel{?}{=} I \cdot \left(\frac{C_1}{C_2}\right)^c;$$

and accepts if the equation holds.

Protocol 7 (Proof of Equality of Two Discrete Logarithms)**.** Let g_1, g_2 be two public generators of $\mathbb{G}$, let $h_1 = g_1^\rho, h_2 = g_2^\rho \in \mathbb{G}$ public, and $\rho \in \mathbb{Z}_p$ secret. Using the protocol of Chaum and Pedersen [125], a prover $\mathcal{P}$ may prove to a verifier $\mathcal{V}$ that $\log_{g_1} h_1 = \log_{g_2} h_2$ without disclosing ρ, in this way:

1. $\mathcal{P}$ chooses uniformly at random $t \in \mathbb{Z}_p$;
2. $\mathcal{P}$ computes the commitments $I_1 = g_1^t$, $I_2 = g_2^t$ in $\mathbb{G}$;
3. $\mathcal{P}$ computes the challenge $c = \mathcal{H}_p(I_1, I_2, h_1, h_2, g_1, g_2)$;
4. $\mathcal{P}$ computes the response $z = t + c \cdot \rho$;
5. $\mathcal{P}$ sends the tuple (I_1, I_2, z) to $\mathcal{V}$.

To check the validity of this proof:

1. $\mathcal{V}$ computes the challenge $c \stackrel{?}{=} \mathcal{H}_p(I_1, I_2, h_1, h_2, g_1, g_2)$
2. $\mathcal{V}$ computes
$$g_1^z \stackrel{?}{=} I_1 \cdot h_1^c, \qquad g_2^z \stackrel{?}{=} I_2 \cdot h_2^c;$$

and accepts if the equations hold.

Protocol 8 (Proof of Inequality of Two Discrete Logarithms)**.** Let ℓ_1, ℓ_2 be two public generators of $\mathbb{G}$, let $x, w \in \mathbb{Z}_p$ with $x \neq w$ be secret, and let $h_1 = \ell_1^x$ and $h_2 = \ell_2^w$ be public. A prover $\mathcal{P}$ may prove to a verifier $\mathcal{V}$ that $\log_{\ell_1} h_1 \neq \log_{\ell_2} h_2$ in the following way [125]:

1. $\mathcal{P}$ chooses $t \in \mathbb{Z}_p$ uniformly at random and computes $C = (\frac{\ell_2^x}{h_2})^t \in \mathbb{G}$.
2. $\mathcal{P}$ proves in ZK to $\mathcal{V}$ the existence of two scalars $\mu_1, \mu_2 \in \mathbb{Z}_p$ such that $C = \ell_2^{\mu_1} \cdot (\frac{1}{h_2})^{\mu_2} = F \cdot B$ and $1_\mathbb{G} = \ell_1^{\mu_1} \cdot (\frac{1}{h_1})^{\mu_2} = E \cdot D$:
   (a) $\mathcal{P}$ proves that $\log_{\ell_2} F = \log_{\ell_1} E$ and $\log_{1/h_2} B = \log_{1/h_1} D$ as in Section 7 (with the adjustment that all common values and commitments are included in the hash computation);
   (b) $\mathcal{V}$ accepts if it accepts these proofs and if $C \stackrel{?}{=} F \cdot B$ and $1 \stackrel{?}{=} E \cdot D$ are satisfied.

3. $\mathcal{V}$ accepts if it accepts in the previous step, and if $C \stackrel{?}{\neq} 1$ is verified.

Protocol 9 (Proof of Plaintext Equality)**.** Let $(p, \mathbb{G}, g_1, g_2)$ be the parameter set for an instance of the M-ElGamal Cryptosystem (see Section 2.9.1), T a public key, and (α, β, γ) a ciphertext. A Prover $\mathcal{P}$, that knows the secret randomness s used to compute the ciphertext, may prove to a Verifier $\mathcal{V}$ that (α, β, γ) is the encryption of $M \in \mathbb{G}$ in the following way [124]:

1. $\mathcal{P}$ chooses uniformly at random $t \in \mathbb{Z}_p$;
2. $\mathcal{P}$ computes the commitments $I_1 = g_1^t$, $I_2 = g_2^t$, $I_3 = T^t$;
3. $\mathcal{P}$ computes the challenge $c = \mathcal{H}_p(\alpha, \beta, \gamma, I_1, I_2, I_3)$;
4. $\mathcal{P}$ computes the response $z = t + s \cdot c$;
5. $\mathcal{P}$ sends the tuple (z, I_1, I_2, I_3) to $\mathcal{V}$

To check the validity of this proof:

1. $\mathcal{V}$ computes the challenge $c \stackrel{?}{=} \mathcal{H}_p(\alpha, \beta, \gamma, I_1, I_2, I_3)$;
2. $\mathcal{V}$ computes:

$$g_1^z \stackrel{?}{=} I_1 \cdot \alpha^c, \qquad g_2^z \stackrel{?}{=} I_2 \cdot \beta^c, \qquad T^z \stackrel{?}{=} I_3 \cdot \left(\frac{\gamma}{M}\right)^c;$$

and accepts if the equations hold.

Protocol 10 (Proof of Plaintext Discrete Logarithm Equality)**.** Let $(p, \mathbb{G}, g_1, g_2)$ be the parameter set for an instance of the M-ElGamal Cryptosystem, T a public key, $g_3, o \in \mathbb{G}$ two public generators, and $C_1 = (\alpha_1, \beta_1, \gamma_1)$, $C_2 = (\alpha_2, \beta_2, \gamma_2)$ two ciphertexts. A Prover $\mathcal{P}$, that knows the secret randomnesses s_1, s_2 used to compute the two ciphertexts, may prove to a Verifier $\mathcal{V}$ that C_1 is the encryption of g_3^x and C_2 is the encryption of o^x, for $x \in \mathbb{Z}_p$, in the following way:

1. $\mathcal{P}$ selects $t \in \mathbb{Z}_p$ uniformly at random;
2. $\mathcal{P}$ computes the commitments $I_1 = g_1^t$, $I_2 = g_2^t$, $I_3 = T^t$, $I_4 = (\frac{g_3}{o})^t$;
3. $\mathcal{P}$ computes the challenge $c = \mathcal{H}_p(C_1, C_2, I_1, I_2, I_3, I_4)$

4. $\mathcal{P}$ computes the response $z_1 = t + (s_1 - s_2) \cdot c$, $z_2 = t + x \cdot c$;
5. $\mathcal{P}$ sends the tuple $(z_1, z_2, I_1, I_2, I_3, I_4)$ to $\mathcal{V}$.

To check the validity of this proof:

1. $\mathcal{V}$ computes the challenge $c \stackrel{?}{=} \mathcal{H}_p(C_1, C_2, I_1, I_2, I_3, I_4)$;
2. $\mathcal{V}$ computes:

$$g_1^{z_1} \stackrel{?}{=} I_1 \cdot \left(\frac{\alpha_1}{\alpha_2}\right)^c, \qquad g_2^{z_1} \stackrel{?}{=} I_2 \cdot \left(\frac{\beta_1}{\beta_2}\right)^c, \qquad T^{z_1} \cdot \left(\frac{g_3}{o}\right)^{z_2} \stackrel{?}{=} I_3 \cdot I_4 \cdot \left(\frac{\gamma_1}{\gamma_2}\right)^c;$$

and accepts the proof if all the equations hold.

Protocol 11 (Disjunctive Proof of Plaintext Equality)**.** Let $(p, \mathbb{G}, g_0, g_1, g_2)$ be the parameter set for an instance of the M-Exp-ElGamal Cryptosystem, T a public key, and (α, β, γ) a ciphertext. A Prover $\mathcal{P}$, that knows the secret randomness s used to compute the ciphertext, may prove to a Verifier $\mathcal{V}$ that (α, β, γ) is the encryption of a value $0 \leq m \leq \nu$, with $\nu \in \mathbb{Z}^+$, in the following way [124]:

1. For each $0 \leq i \leq \nu$, if $i = m$ $\mathcal{P}$ performs the first step of a plaintext equality proof (Protocol 9) to obtain t_m, $I_{1,m}, I_{2,m}, I_{3,m}$, otherwise it chooses $z_i, c_i \in \mathbb{Z}_p$ uniformly at random and computes the commitments:

$$I_{1,i} = g_1^{z_i} \alpha^{-c_i}, \qquad I_{2,i} = g_2^{z_i} \beta^{-c_i}, \qquad I_{3,i} = T^{z_i} \left(\frac{\gamma}{g_0^i}\right)^{-c_i};$$

2. $\mathcal{P}$ computes the challenge $c = \mathcal{H}_p(\alpha, \beta, \gamma, (I_{1,i}, I_{2,i}, I_{3,i})_{i=0}^{\nu},)$;
3. $\mathcal{P}$ computes the $m-th$ response $c_m = c - \sum_{i=0, i\neq m}^{\nu} c_i$, $z_m = t_m + s \cdot c_m$;
4. $\mathcal{P}$ sends the tuple $(c_i, z_i, I_{1,i}, I_{2,i}, I_{3,i})_{i=0}^{\nu}$ to $\mathcal{V}$.

To check the validity of this proof:

1. $\mathcal{V}$ computes the challenge $c \stackrel{?}{=} \mathcal{H}_p(\alpha, \beta, \gamma, (I_{1,i}, I_{2,i}, I_{3,i})_{i=0}^{\nu},)$;

2. $\mathcal{V}$ computes:

$$g_1^{z_i} \stackrel{?}{=} I_{1,i} \cdot \alpha^{c_i}, \qquad g_2^{z_i} \stackrel{?}{=} I_{2,i} \cdot \beta^{c_i}, \qquad T^{z_i} \stackrel{?}{=} I_{3,i} \cdot \left(\frac{\gamma}{g_0^i}\right)^{c_i};$$

and accepts the proof if all the equations hold for all $0 \leq i \leq \nu$.

Protocol 12 (Threshold Proof of Correct Decryption)**.** Let $(p, \mathbb{G}, g_1, g_2)$ be the parameter set for an instance of a threshold M-ElGamal Cryptosystem, T a public key whose private part is shared among n parties P_i with threshold t, each holding a secret share $(x_{i,1}, x_{i,2})$, and let (α, β, γ) be a ciphertext. The P_i may prove to a Verifier $\mathcal{V}$ that $M \in \mathbb{G}$ is the correct decryption of (α, β, γ) in the following way:

1. Each P_i generates uniformly at random $\hat{t}_{i,1}, \hat{t}_{i,2} \in \mathbb{Z}_p$, and partakes in two runs of Protocol 2: first with $\hat{s}_i = \hat{t}_{i,1}$, $g = g_1$ to get the share $(i, t_{i,1})$ and the public value $g_1^{t_{i,1}}$; then with $\hat{s}_i = \hat{t}_{i,2}$, $g = g_2$ to get the share $(i, t_{i,2})$ and the public value $g_2^{t_{i,2}}$.

2. Each P_i computes and broadcasts to the other decrypting parties the share $(i, \alpha^{t_{i,1}} \cdot \beta^{t_{i,2}})$.

3. With t shares each P_i can interpolate and obtain $I_1 = \alpha^{t_1} \cdot \beta^{t_2}$, $I_2 = g_1^{t_1} \cdot g_2^{t_2}$, then compute $c = \mathcal{H}_p(\alpha, \beta, \gamma, T, I_1, I_2)$, $z_{i,1} = t_{i,1} + c \cdot x_{i,1}$, $z_{i,2} = t_{i,2} + c \cdot x_{i,2}$, and broadcasts the share $(i, I_1, I_2, z_{i,1}, z_{i,2})$.

4. With t shares it is possible to interpolate and obtain the complete NIZKP (I_1, I_2, z_1, z_2).

5. $\mathcal{V}$ computes c as above and accepts if the following equations hold in $\mathbb{G}$:

$$\alpha^{z_1} \cdot \beta^{z_2} \stackrel{?}{=} I_1 \cdot \left(\frac{\gamma}{M}\right)^c, \qquad g_1^{z_1} \cdot g_2^{z_2} \stackrel{?}{=} I_2 \cdot T^c.$$

2.10.5 Designated-Verifier Proofs

Designated-Verifier Non-Interactive Zero-Knowledge Proofs (DVNIZKPs [79]) are protocols which retain most of the security properties of a NIZKP, but are not publicly verifiable: only the owner of some secret information (the designated verifier) can check the proof.

In particular the verifier $\mathcal{V}$ holds a key pair $((\mathtt{sk}_\mathcal{V}, \mathtt{pk}_\mathcal{V}))$ and using the public key, the prover $\mathcal{P}$ produces a proof for a statement x, such that only $\mathcal{V}$ is convinced that the statement is true. This is achieved by allowing $\mathcal{V}$ to produce fake but valid DVNIZKPs for any statement, using their private key.

The basic construction for designing DVNIZKP is to consider $x \in \mathcal{L}$ if and only if "$x \in \mathcal{L}$ or $(\mathtt{sk}_\mathcal{V}, \mathtt{pk}_\mathcal{V})$ is a valid secret-public key pair". In this way, the original $\mathcal{P}$ can prove that $x \in \mathcal{L}$, since it does not know the secret key $\mathtt{sk}_\mathcal{V}$ of the verifier. Later, the verifier can produce a proof for any x' (thus even for $x' \notin \mathcal{L}$) since it knows $\mathtt{sk}_\mathcal{V}$ and thus can prove the second branch of the "or".

This property is useful in the context of e-voting to achieve end-to-end verifiability while still preventing the voter from transferring some proofs (and thus preventing coercion through plausible deniability).

2.11 Shoup's Game Hops for Security Proofs

In game-based security proofs, the concept of *game hopping* has been introduced in [128]. It involves a series of interactions, known as *games*, which model the protocol and the adversary. Each game represents a distinct scenario within the security analysis, and transitioning from one game to another is defined as a *game hop*. These hops typically introduce incremental changes to either the protocol or the adversary's capabilities. Each game hop is designed to make the adversary's task increasingly difficult. The sequence begins with the initial game, where the adversary interacts with the real-world protocol. Intermediate games introduce gradual changes to the protocol or adversary, aiming to demonstrate that these changes only negligibly affect the adversary's advantage. In the final game the adversary typically interacts with the ideal-world protocol, where the adversary's success is either infeasible or highly improbable.

However, in constructing such proofs, it is desirable that the changes between successive games are very small, so that analyzing the change is as simple as possible. The paper [128] identifies three types of transitions, that here we briefly restate:

Transitions based on indistinguishability. A minor change is made and, if noticed by the adversary, this would allow them to differentiate be-

tween two distributions that are otherwise indistinguishable, whether statistically or computationally. As an example, we can consider two computationally indistinguishable distributions $\mathcal{D}_1$, $\mathcal{D}_2$. To prove that $\mid \Pr[S_i] - \Pr[S_{i+1}] \mid$ is negligible, we can construct a distinguisher $\mathbb{B}$ so that when given an element drawn from distribution $\mathcal{D}_1$ as input, $\mathbb{B}$ outputs 1 with probability $\Pr[S_i]$, and when given an element drawn from distribution $\mathcal{D}_2$ as input, $\mathbb{B}$ outputs 1 with probability $\Pr[S_{i+1}]$. The indistinguishability assumption then implies that $|\Pr[S_i] - \Pr[S_{i+1}]|$ is negligible. Usually, the two games are designed as a unique hybrid game that takes an auxiliary input. If the auxiliary input is drawn from $\mathcal{D}_1$ the game behaves like Game i, if it is drawn from $\mathcal{D}_2$ the game behaves like Game $i+1$. The distinguisher then runs this hybrid game using its input and outputs 1 if the relevant event occurs.

Transitions based on failure events. Two consecutive games proceed identically unless a certain failure event occurs. In this transition, it is best if the two games are defined on the same underlying probability space (i.e., the only differences between the two games are the rules for computing certain random variables). Saying that the two games proceed identically unless F occurs is equivalent to saying that the events $S_i \vee \neg F \iff S_{i+1} \vee \neg F$ are the same. In this case, we can use the Lemma 1.

Lemma 1 (Difference Lemma [128])**.** *Let A, B and F be three events defined in some probability distribution, and suppose that $A \vee \neg F \iff B \vee \neg F$. Then $\mid \Pr[A] - \Pr[B] \mid \leq \Pr[F]$.*

In order to prove that $\Pr[S_i]$ is negligibly close to $\Pr[S_{i+1}]$, it suffices to prove that $\Pr[F]$ is negligible.

Bridging steps. Purely conceptual step needed to prepare the ground for a transition of one of the above types, in which two consecutive games have the same probability.

2.12 Mix-Nets and Verifiable Encrypted Shuffle

A mix-net [34] is a multi-party protocol which shuffles a set of ciphertexts so that nobody knows the permutation linking the input and output. The

actors of the protocol are called *mix-servers*, basically each one chooses a permutation and the corresponding shuffles (which permute and re-encrypt the set of input ciphertexts) are applied in sequence. If at least one mix-server is honest and chooses a random permutation without revealing it, it is impossible to link the input and output.

Mix-nets are widely used in anonymization protocols and in particular in voting schemes, where they are used to shuffle ballots and unlink the contents of the ballot from the identity of the voter who cast it.

In a mix-net it is problematic if a mix-server does not shuffle correctly, but in particular in e-voting protocols it is usually required that the process is auditable, so it is necessary to use verifiable shuffling protocols, where each mix-server can, after making a shuffle, prove that the shuffle is correct.

A typical solution is the usage of a Honest-Verifier Zero-Knowledge (HVZK) argument: its soundness guarantees that the shuffle is correct, while the zero-knowledge property ensures that the HVZK argument does not leak the permutation, the randomizers or any other information pertaining to the shuffle.

2.12.1 Verifiable Shuffle

A shuffle of ciphertexts $e_1, \ldots, e_n$ is a new set of ciphertexts $E_1, \ldots, E_n$ whose decryption gives the same plaintexts but in permuted order. If the cryptosystem is homomorphic, we may shuffle the ciphertexts $e_1, \ldots, e_n$ by selecting a permutation π, randomizers $r_1, \ldots, r_n$, and setting

$$E_1 = e_{\pi(1)} \cdot \mathcal{E}_{\mathtt{pk}}^{r_1}(1), \ldots, E_n = e_{\pi(n)} \cdot \mathcal{E}_{\mathtt{pk}}^{r_n}(1).$$

If the cryptosystem is semantically secure, publishing $E_1, \ldots, E_n$ reveals nothing about the permutation.

We give an high-level description of the protocol of Groth [69] which provides an efficient honest verifier zero-knowledge argument for the correctness of a shuffle of homomorphic encryptions. The main other solutions are the protocol of Furukawa [62] (based on permutation matrices in the common reference string model, where the mixer commits to a permutation matrix and then argues that the commitment is correct and that the ciphertexts have been shuffled according to this permutation), and the protocol of Neff [106] (based, as Groth, on polynomials being identical under permutation of their roots, the ZKP does not require a common reference string, but relies on

the public key of the cryptosystem specifying a group where the decision Diffie–Hellman assumption holds).

The protocol of Groth builds upon a special-honest verifier zero-knowledge argument for a commitment containing a permutation of a set of known messages. That is, starting from a set of messages $m_1, \ldots, m_n$ and a commitment c, prove the knowledge of a permutation π and a randomizer r such that $c = \mathrm{Com}(m_{\pi(1)}, \ldots, m_{\pi(n)}; r)$. The commitment scheme used is homomorphic and has the root-extraction property (see [115]). The main idea is that, for any permutation pi, the polynomial $p(X) = \prod_{i=1}^{n}(X - m_i)$ is stable under permutation of the roots, i.e. $p(X) = \prod_{i=1}^{n}(X - m_{\pi(i)})$. So the argument proves knowledge of $\mu_1, \ldots, \mu_n$, r so that $c = \mathrm{Com}(\mu_1, \ldots, \mu_n; r)$ and

$$\prod_{i=1}^{n}(X - \mu_i) = \prod_{i=1}^{n}(X - m_i).$$

Since we are working over a field $\mathbb{Z}_q$, this equality implies the existence of a permutation π, such that $\mu_i = m_{\pi(i)}$. To prove the equality of the polynomials it is sufficient to evaluate them in a random point chosen by the verifier, because a polynomial in $\mathbb{Z}_q[X]$ of degree n can have at most n roots, so there is overwhelming probability of failing the test unless the polynomials are indeed equal.

Chapter 3

Vote App Protocol Description

This chapter introduces *Vote App*, starting with a high-level overview of its components in Sections 3.2 and 3.3, followed by a detailed description of the underlying protocol in Sections 3.4 to 3.11.
The content of this chapter is based on the works [97, 96].

3.1 Introduction

Vote App is a mobile internet voting application developed as part of a collaboration between Fondazione Bruno Kessler (FBK[1]) and the Istituto Poligrafico e Zecca dello Stato (IPZS[2]). Its development was motivated by the official recognition of remote electronic voting as an area for experimentation in 2021 [103].

Italian citizens currently have access to two eIDAS-notified[3] electronic identification (eID) schemes, which provide secure access to public services. These schemes meet a High Level of Assurance [54] and are widely adopted. In this work, we assume that existing eID schemes can reliably authenticate voters with sufficient security guarantees.

[1] Bruno Kessler Foundation, see https://www.fbk.eu/en/.

[2] State Mint and Polygraphic Institute, see https://www.ipzs.it/ext/index.html.

[3] eIDAS (Electronic Identification, Authentication and Trust Services) [6, 55] is an EU regulation that establishes a framework for electronic identification and trust services for electronic transactions in the European Single Market. It aims to ensure secure and seamless electronic interactions between businesses, citizens and public authorities in all EU Member States.

Vote selling and coercion have been documented as significant threats to voting systems, often linked to organized crime. Addressing these risks is a key objective of *Vote App*. Specifically, the protocol aims to provide coercion resistance, a property that ensures voters cannot prove how they voted, even if they interact with a coercer during the voting process.

To achieve this, *Vote App* introduces anti-coercion credentials (ACC)[4], inspired by Civitas and ABRTY [35, 7]. These credentials are based on the *fake credential* technique of JCJ [83, 39], which allows voters to generate decoy credentials[5] if they face coercion. These decoy credentials are indistinguishable from valid ones but do not validate the corresponding ballots during tallying.

An improvement of this protocol compared to JCJ, is that the complexity of vote tallying is reduced from quadratic to linear in the number of cast votes. In the event that a voter uses a decoy credential, the coercer is unable to discern the validity of the credential. The distinction between real and decoy credentials emerges only during the tallying process, after the ballots have been verifiably shuffled, thus is still impossible to say whether the voter under attack cast their vote with a valid or decoy credential.

Another problem we try to tackle is a compromise between security and usability. In [7] the private piece of an ACC that distinguishes between a fake and a real one is an element $x \in \mathbb{Z}_p$, which can be written as 20-30 ASCII characters for a value of p that conforms to standard security levels. For security reasons x should not be saved on the voting device, should be manually inserted by the voter, and ideally remembered to better fool a coercer. Of course it is extremely impractical for a human being to remember that many characters, and studies suggest that usability greatly degrades when users have to deal with something more complicated than a PIN of few digits [76, 143].

Therefore our approach is to save the value x masked, and to split the mask in two. The κ least significant digits are a `PIN` that unlocks the private ACC, while the other part is stored encrypted alongside the ACC. In order to vote, the voter inserts the `PIN` which then retrieves the private ACC used to construct the ballot. In order to create a fake ACC, the voter just has to create a ruse `PIN`.

[4] Often called voting credential or just credential, they are composed by a public part and a private part.

[5] The public part of the credential always remains the same, what can be *faked* is the private part.

In addition to addressing coercion resistance, the proposed protocol satisfies all the fundamental requirements (see Section 1.2) for a secure internet voting system.

The real interfaces of *Vote App* are shown in Appendix A.6.

To improve readability, a nomenclature section (see Chapter 8) provides definitions for key terms and notations used throughout.

3.2 Protocol Actors

3.2.1 Authorities

Authorities are entities which have some degree of trust and are managed by a recognized public institution. In *Vote App* there are four authorities:

1. *Registration Teller (RT).* The RTs are entities that have access to a private database and can communicate with the application and the Tabulation Tellers.
2. *Tabulation Teller (TT).* The TTs are entities that have access to a private database and can communicate with the Web Bulletin Board (WBB) and the Registration Tellers.
3. *Electoral Roll (ER).* The ER is an entity which checks the eligibility of a user and assigns a pseudonymous identifier, has access to a private database and communicates with the application. Also, it issues the single-use tokens required to cast ballots.
4. *Ballot Box (BB).* The BBs are entities which verify cast IDs, collect encrypted votes, check their formal correctness, and timestamp them via the WBB.

We consider all the authorities to be trusted, for further discussion on this see Section 3.12.

3.2.2 Services

A service is an entity that is supposed to work reliably according to the specifications. In *Vote App* the followings are identified as services:

1. *Web Bulletin Board (WBB).* The WBB serves as the authoritative record of the election's true state. It ensures the reliable and consistent publication of public information while verifying the authorship of the data. Only authorized entities are permitted to publish information, which is accessible for anyone to read. However, once published, the data cannot be deleted or altered, safeguarding its integrity.

2. *Verification Service (VS).* The VS is an application which helps the voters verify the correctness of the information given by *Vote App* (e.g., given a real or forged private ACC x, computes the corresponding o^x). Its authenticity is certified by some authority, e.g., the Electoral Roll (ER).

3. *Notification Server (NS).* The NS is a service which mediates the communication from the RTs to *Vote App* by sending notifications when some content is ready to be downloaded. It communicates with the RTs, *Vote App*, and the ER.

4. *Digital Identity Provider (DIP).* The DIP is a previously established service that certifies the identity and personal information of voters. It communicates with the ER and *Vote App*.

3.2.3 Users

We identify with *users* all the other actors interacting with the protocol.

1. *Eligible Voter.* An eligible voter is anyone who can cast a vote in the election.

2. *Auditors.* The auditors are parties responsible for checking all publicly verifiable information. Auditors are considered to be Anonymous Users, have access to the WBB page, can download the app, but do not necessarily have a valid digital identity or the right to vote.

3.3 High Level Overview

Vote App voting protocol is divided into five phases (see Figure 3.1, [45]):

1. *Pre-Setup and Setup.* The protocol parameters are chosen, and the public keys are generated (1a and 1b) and published (1c). Given some

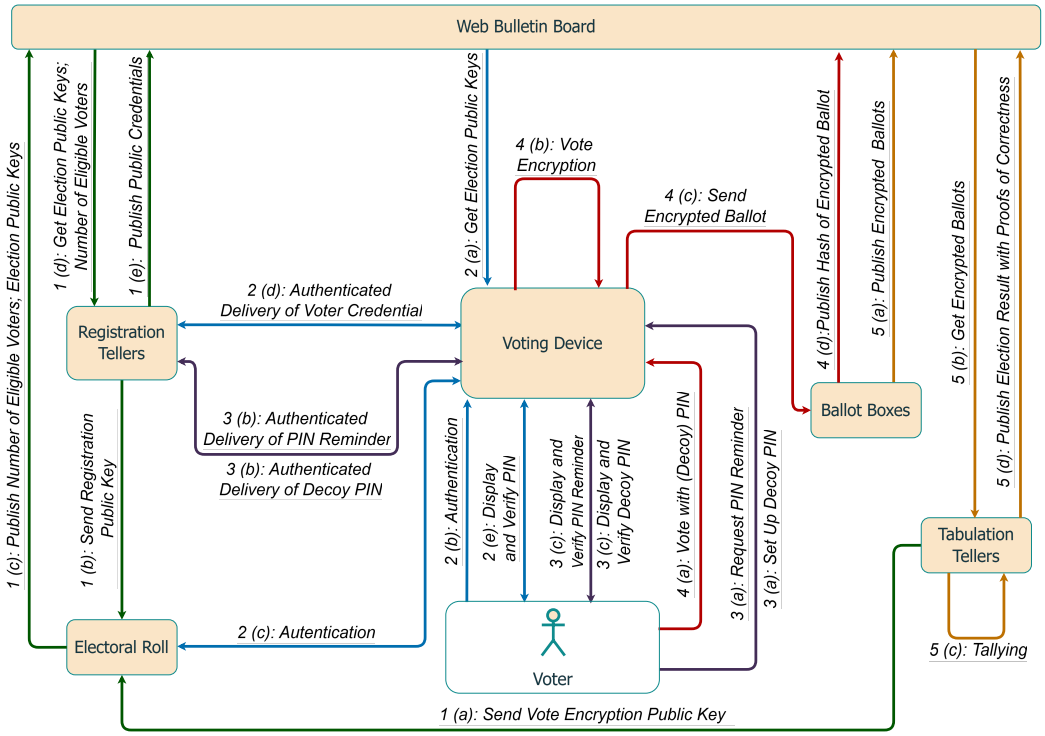

Figure 3.1: Simplified Diagram of *Vote App* protocol. The five phases are represented in different colours: *1-Setup* in green, *2-Enrollment* in blue, *3-PIN and Device Management* in purple, *4-Voting* in red, and *5-Tallying* in orange.

election parameters (1d), the Registration Tellers (RT) cooperatively generate a voting credential for every eligible voter and publish the public parts (1e).

2. *Enrollment.* Using a voting device updated with the current election parameters (2a), each voter authenticates to the voting platform via their eID (2b), and the Electoral Roll (2c) checks their status as eligible voters. Then, the voting device requests to the RTs the voter's private credential, which is delivered after a random waiting period (2d) and shown to the voter as a short PIN. The authorities also send a DVNIZKP, so that voters can verify the validity of the PIN shown by their device (2e).

3. *PIN and Device Management.* Each voter can set up one or more decoy PINs (3a) and verify them with the decoy DVNIZKP (3c). Each voter

can also request to receive again their valid PIN (3a) and/or verify a PIN with the DVNIZKP (3c) (multiple times). Both decoy PINs and reminders (3b) are delivered and displayed exactly as in phase 2 (2d, 2e).

4. *Voting.* Each voter expresses their preferences on their voting device and validates them by inserting a PIN (4a). The device creates an encrypted ballot (4b) and casts it (4c). For confirmation, a hash is published on a Web Bulletin Board (WBB) (4d). Each voter can simulate multiple votes by using their decoy PINs or express their real preference with their valid PIN. Re-voting is allowed; only the last ballot cast with the valid PIN will count in the final tally.

5. *Tallying.* When the voting period ends, the encrypted ballots are released by the Ballot Boxes (5a). Duplicate, malformed and invalid ballots are set aside after being verifiably mixed by the Tabulation Tellers. The remaining ballots are fetched (5b) and counted (5c), and proofs of correct tally execution are published (5d).

As shown in the high-level overview and diagram (see Figure 3.1), the protocol consists of multiple phases that work together to achieve its goals. We now proceed to a detailed mathematical description of these phases, formalizing the operations and interactions introduced earlier.

3.4 Pre-Setup

This phase defines the necessary parameters for the protocol and initializes the required entities, such as authorities and services. This step ensures that the basic elements are in place for the proper execution of the protocol setup.

3.4.1 Cryptographic parameters definition

The following parameters are defined and fixed:

- a cyclic group $\mathbb{G}$ of prime order p in which the DDH problem is hard (see Section 2.2);
- five generators of $\mathbb{G}$: g_0, g_1, g_2, g_3, o;

- a cryptographically secure extensible output hash function H, a cryptographically secure hash function $\mathcal{H}$, and a cryptographically secure hash with outputs in $\mathbb{Z}_p$ $\mathcal{H}_p$ (see Section 2.3);
- the parameters necessary for Groth's protocol [69] for verifiably mix and re-encrypt homomorphic ciphertexts (see Section 2.12).

3.4.2 Definition of Authorities, Services, and Permissions

In this phase, the parameters and key authorities and services involved in the protocol are fixed and defined:

- the number of RT is fixed as $n_{\mathtt{RT}}$;
- the minimum number of RTs which need to collaborate in the registration of a new voter post setup is fixed as $t_{\mathtt{RT}} \leq n_{\mathtt{RT}}$;
- the threshold which defines security and resiliency during setup is fixed as $t'_{\mathtt{RT}} \leq (n_{\mathtt{RT}} + 1)/2$: a minimum of $2t'_{\mathtt{RT}} - 1$ RTs have to collaborate during the setup, at least $n_{\mathtt{RT}} - t'_{\mathtt{RT}}$ are supposed to not collude;
- the minimum and maximum waiting time (in seconds) before sending shares to *Vote App* are fixed as $\tau_{\mathtt{min}}$ and $\tau_{\mathtt{max}}$ respectively;
- the number of TTs is fixed as $n_{\mathtt{TT}}$;
- the minimum number of TTs which need to collaborate in the tallying is fixed as $t_{\mathtt{TT}} \leq n_{\mathtt{TT}}$, at least $n_{\mathtt{TT}} - t_{\mathtt{TT}}$ are supposed to not collude;
- the sequence of the $n_{\mathtt{Ls}}$ electoral lists, each one defined by a distinct number ($1 \leq i \leq n_{\mathtt{Ls}}$), name, and symbol;
- the sequence of the $n_{\mathtt{Ca},i}$ candidates associated to the i-th electoral list, for $1 \leq i \leq n_{\mathtt{Ls}}$, each defined by a distinct number ($1 \leq j \leq n_{\mathtt{Ca},i}$) and name;
- the number of BBs is fixed as $n_{\mathtt{BB}}$, at least one is supposed to be honest;
- the PIN length is defined as $\mathtt{L}_{\mathtt{PIN}}$;
- the passphrase is defined as `psph` with length $\mathtt{L}_{\mathtt{psph}}$ (where the length is the number of words the passphrase is composed of);

- a word list $\texttt{wl}$ is defined with $2^{\texttt{L}_{\texttt{wl}}}$ elements[6];
- the number of voters is defined as $n_{\texttt{V}}$;
- the total number of credentials to generate is defined as $n_{\texttt{ACC}}$, which is strictly bigger that $n_{\texttt{V}}$ to accommodate revocations and re-issues in case of emergencies (e.g., if a private credential has been compromised, or if the voter has lost access to their phone and is unable to log in on a different device);
- the public institutions which control the authorities and services introduced in Section 3.2 are identified according to legal requirements;
- the service provider which manages the NS is identified;
- the requirements that a Verification Service has to satisfy in order to be certified are defined.

It is worth noting that special attention should be paid to the Verification Service, which can be implemented by any party. To mitigate the risks of using malicious or unreliable implementations, we must establish precise requirements to ensure its certification and safeguard user data.

Furthermore, in this phase, the WBB defines the writing permissions for each phase of the protocol, establishing which entities are authorized to publish or modify specific data:

- during setup:
  - $t_{\texttt{RT}}$ RTs which agree on the same data can write the ACC generation public key $\texttt{pk}_{\texttt{ACC}}$ and the list of $n_{\texttt{ACC}}$ public ACCs;
  - the ER can write the election public key $\texttt{pk}_{\texttt{EL}}$, the number of pseudonymous identifiers $n_{\texttt{ACC}}$, and the root of the Merkle tree of the voter-to-id association;
- during the voting phase:
  - each BB_i can write the ballot digests and the associated $\texttt{BallotEmoji}$ and $\texttt{PublicPINEmoji}$, then subsequently the corresponding *cast-as-intended disclosure*.
- during the tallying:

[6] We use BIP32 Italian wordlist which has $\texttt{L}_{\texttt{wl}} = 11$ https://github.com/bitcoin/bips/blob/master/bip-0039/bip-0039-wordlists.md

    - $t_{\texttt{RT}}$ RTs which agree on the same data can write the credential control elements used to distinguish between valid and invalid ACCs;
    - each TT_i must comply with the following rules when writing:
        - write exactly once per tallying step;
        - for the mixing steps, write sequentially in the order of the mixers, from the first to the last;
        - include data accompanied by NIZKPs, ensuring they validate against all previously published data at the time of writing;
    - the ER can write the list of the pseudonymous identifiers associated to eligible voters, considering possible revocations and reissuing;
    - each BB_i can write the content of ballots whose hash was previously published;

- no-one else can write anything;

- no-one can delete or alter previously published data.

At the end of the pre-setup phase, the WBB publishes the identifiers of the system's authorities and services, along with all election-related data, such as candidates, electoral lists, and other pertinent information.

To certify the voter's view of the WBB, its hash can be embedded in *Vote App*. This approach ensures that voters can verify they are interacting with the intended version of the WBB, similar to the mechanism employed in the Swiss Post e-voting system [118].

From now on, even though it is not explicitly written, we will assume that each time a proof is received, its validity will be checked before proceeding with the protocol. Similarly, whenever shares are received, only those that are accompanied by a valid proof will be considered.

3.5 Setup

In the setup phase, the authorities generate the voting credentials and pseudonyms for the voters, along with the election key and their respective cryptographic keys.

3.5.1 Tabulation Tellers Setup

The TTs distributively compute the tallying public key $\mathtt{pk}_{\mathtt{TT}}$ for the M-ElGamal Cryptosystem (see Protocol 3) with parameters $\mathbb{G}, p, g_1, g_2$.

1. Each TT_i for $i = 1 \dots n_{\mathtt{TT}}$ generates uniformly at random $\left(\hat{\xi}_{1,i}, \hat{\xi}_{2,i}\right) \in \mathbb{Z}_p^2$.
2. TT_i, $1 \leq i \leq n_{\mathtt{TT}}$, performs two runs of Protocol 2:
   - with $n = n_{\mathtt{TT}}, t = t_{\mathtt{TT}}, g = g_1, \hat{s}_i = \hat{\xi}_{1,i}$ to retrieve the share $(i, \xi_{1,i})$;
   - with $n = n_{\mathtt{TT}}, t = t_{\mathtt{TT}}, g = g_2, \hat{s}_i = \hat{\xi}_{2,i}$ to retrieve the share $(i, \xi_{2,i})$;

   and broadcast two commitments $g_1^{\hat{\xi}_{1,i}}$ and $g_2^{\hat{\xi}_{2,i}}$ to the randomly chosen values.
3. Anyone, using the values published during the previous step, can compute the tallying public key as:

$$\mathtt{pk}_{\mathtt{TT}} = \prod_{i=1}^{n_{\mathtt{TT}}} g_1^{\hat{\xi}_{1,i}} g_2^{\hat{\xi}_{2,i}} = g_1^{\sum_{i=1}^{n_{\mathtt{TT}}} \hat{\xi}_{1,i}} g_2^{\sum_{i=1}^{n_{\mathtt{TT}}} \hat{\xi}_{2,i}} = g_1^{\xi_1} g_2^{\xi_2}.$$

Eventually, each TT_i owns a $t_{\mathtt{TT}}$-out-of-$n_{\mathtt{TT}}$ share of $\mathtt{sk}_{\mathtt{TT}} = (\xi_1, \xi_2)$ which is $\mathtt{sk}_{\mathtt{TT}_i} = (\xi_{1,i}, \xi_{2,i})$.

The key $\mathtt{pk}_{\mathtt{TT}}$ is sent by each TT_i to the ER.

Note that, by generating the keys in this way, all the Tabulation Tellers participate in the creation of the key pair, but only $t_{\mathtt{TT}}$ are required to recover the secret key, thus transforming an $n_{\mathtt{TT}}$-out-of-$n_{\mathtt{TT}}$ protocol into a $t_{\mathtt{TT}}$-out-of-$n_{\mathtt{TT}}$ protocol for key retrieval.

3.5.2 Registration Tellers Setup

The RTs distributively compute the ACC generation public key $\mathtt{pk}_{\mathtt{ACC}}$ for the M-ElGamal Cryptosystem with parameters $\mathbb{G}, p, g_1, g_2$.

1. Each RT_i for $i = 1 \dots n_{\mathtt{RT}}$ generates uniformly at random $(\hat{\chi}_{1,i}, \hat{\chi}_{2,i}) \in \mathbb{Z}_p^2$.
2. RT_i, $1 \leq i \leq n_{\mathtt{RT}}$, performs two runs of Protocol 2:

- with $n = n_{\mathtt{RT}}, t = t_{\mathtt{RT}}, g = g_1, \hat{s}_i = \hat{\chi}_{1,i}$ to retrieve the share $(i, \chi_{1,i})$;
- with $n = n_{\mathtt{RT}}, t = t_{\mathtt{RT}}, g = g_2, \hat{s}_i = \hat{\chi}_{2,i}$ to retrieve the share $(i, \chi_{2,i})$;

and broadcast two commitments $g_1^{\hat{\chi}_{1,i}}$ and $g_2^{\hat{\chi}_{2,i}}$ to the randomly chosen values.

3. Anyone, using the values published during the previous step, can compute the ACC generation public key as:

$$\mathtt{pk}_{\mathtt{ACC}} = \prod_{i=1}^{n_{\mathtt{RT}}} g_1^{\hat{\chi}_{1,i}} g_2^{\hat{\chi}_{2,i}} = g_1^{\sum_{i=1}^{n_{\mathtt{RT}}} \hat{\chi}_{1,i}} g_2^{\sum_{i=1}^{n_{\mathtt{RT}}} \hat{\chi}_{2,i}} = g_1^{\chi_1} g_2^{\chi_2}.$$

Note that each RT_i owns a $t_{\mathtt{RT}}$-out-of-$n_{\mathtt{RT}}$ share of $\mathtt{sk}_{\mathtt{ACC}} = (\chi_1, \chi_2)$ which is $\mathtt{sk}_{\mathtt{ACC}_i} = (\chi_{1,i}, \chi_{2,i})$.

As before, all the Registration Tellers participate in the creation of the key pair, but only $t_{\mathtt{RT}}$ are required to recover the secret key.

The RTs distributively compute the registration public key $\mathtt{pk}_{\mathtt{RT}}$.

i. Each RT_i for $i = 1 \dots n_{\mathtt{RT}}$ generates uniformly at random $\gamma_i \in \mathbb{Z}_p$.

ii. Each RT_i partakes in a run of Protocol 2 with

$$n = n_{\mathtt{RT}}, t = t_{\mathtt{RT}}, g = g_3, \hat{s}_i = \gamma_i$$

to obtain the shares (i, y_i) and sets $\mathtt{sk}_{\mathtt{RT}_i} = y_i$.

iii. Each RT_i computes and broadcasts its share of the registration public key $\mathtt{pk}_{\mathtt{RT}_i} = g_3^{y_i}$.

iv. Anyone can interpolate in the exponent (see Section 2.7) the values $\mathtt{pk}_{\mathtt{RT}_i}$ published in the previous step and evaluate the polynomial in $x = 0$ to obtain the registration public key $\mathtt{pk}_{\mathtt{RT}} = g_3^y$, where the registration secret key is $\mathtt{sk}_{\mathtt{RT}} = y$.

The keys $\mathtt{pk}_{\mathtt{ACC}}, \mathtt{pk}_{\mathtt{RT}}$ are sent by each RT_i to the ER.

3.5.3 Electoral Roll Setup

The ER defines the election public key $\mathtt{pk}_{\mathtt{EL}}$ which is assembled using the public keys computed by the the authorities and contains:

- the tallying public key $\mathtt{pk}_{\mathtt{TT}}$ (obtained from the TTs during setup, the value is accepted if confirmed by $t_{\mathtt{TT}}$ TTs);
- the registration public key $\mathtt{pk}_{\mathtt{RT}}$ (obtained from the RTs during setup, the value is accepted if confirmed by $t_{\mathtt{RT}}$ RTs);
- the cryptographic parameters $\mathbb{G}, p, g_0, g_1, g_2, g_3, o, H, \mathcal{H}, \mathcal{H}_p$;
- the election identifier $\mathtt{EL}_{\mathtt{ID}}$ (an integer that uniquely identifies the election);
- the ballot parameters, i.e., the number of electoral lists $n_{\mathtt{Ls}}$ and the number of candidates associated to each electoral list $(n_{\mathtt{Ca},i})_{1 \leq i \leq n_{\mathtt{Ls}}}$.

Let $n_{\mathtt{ACC}}$ be the number of credentials generated, where $n_{\mathtt{ACC}} > n_{\mathtt{V}}$ since the $n_{\mathtt{ACC}} - n_{\mathtt{V}}$ extra credentials are used to accommodate for revocations and reissue in case of emergencies, and let $\mathtt{id}$ be the personal data of the eligible voters certified by the Digital Identity Provider which includes $n_{\mathtt{ACC}} - n_{\mathtt{V}}$ extra random strings. The ER defines the list of voters' pseudonymous identifiers in the following way:

1. it creates the couples $(\mathtt{id}, v_{\mathtt{id}})$, where $v_{\mathtt{id}}$ is a randomly chosen integer, distinct for each voter, which will serve as their pseudonymous identifier and privately stores this association;
2. it computes a Merkle tree of this association, where the leaves are the pairs $(\mathtt{id}, v_{\mathtt{id}})$;
3. it publishes on the WBB the integer $n_{\mathtt{ACC}}$ and the root of the Merkle tree.

3.5.4 ACC Generation

The RTs distributively compute $n_{\mathtt{ACC}}$ ACCs (one for each pseudonymous identifier). Each ACC (with index $v_{\mathtt{id}}$ that we omit here for notation simplicity) in the following way:

1. each RT_i generates uniformly at random $\left(\tilde{\xi}_i, ß_i, \rho_i, \alpha_i\right) \in \mathbb{Z}_p^4$;
2. each RT_i sets $\beta_i = \gamma_i + \rho_i$[7] where γ_i is value generates in Item i. of Section 3.5.2, and performs four runs of Protocol 2:

[7] This step is needed not to introduce bias on the generation of it.

(a) with $n = n_{\texttt{RT}}, t = t_{\texttt{RT}}, g = g_1, \hat{s}_i = \tilde{\xi}_i$ to retrieve the share (i, x_i) and commits to it $g_1^{x_i}$[8];

(b) with $n = n_{\texttt{RT}}, t = t_{\texttt{RT}}, g = g_2, \hat{s}_i = ß_i$ to retrieve the share (i, σ_i);

(c) with $n = n_{\texttt{RT}}, t = t_{\texttt{RT}}, g = g_3, \hat{s}_i = \rho_i$ to retrieve the share (i, r_i) and commits to it $g_3^{r_i}$;

(d) with $n = n_{\texttt{RT}}, t = t'_{\texttt{RT}}, g = g_2, \hat{s}_i = \alpha_i$ to retrieve the share (i, a_i) and commits to it $g_2^{a_i}$;

(e) with $n = n_{\texttt{RT}}, t = t'_{\texttt{RT}}, g = g_3, \hat{s}_i = \beta_i$ to retrieve the share (i, b_i) and commits to it $g_3^{b_i}$;

all the commitments are then broadcast;

3. each RT_i interpolates $g_3^{b_i}$ and $g_3^{r_i}$ obtaining g_3^b and g_3^r;

4. each RT_i checks that $g_3^b = g_3^y \cdot g_3^r = \texttt{pk}_{\texttt{RT}} \cdot g_3^r$ and, if successful, computes $\delta_i = a_i \cdot b_i = a_i \cdot (y_i + r_i)$;

5. each RT_i broadcasts the shares (i, r_i) and (i, δ_i), accompanied with the NIZKPs that $\log_{g_2^{a_i}}(g_2^{\delta_i}) = \log_{g_3}(g_3^{b_i})$ and $\log_{g_3^{b_i}}(g_3^{\delta_i}) = \log_{g_2}(g_2^{a_i})$ (see Protocol 7);

6. with at least $t_{\texttt{RT}}$ shares (j, r_j) received (considering only the shares whose accompanying NIZKP are valid), each RT_i can compute the Lagrangian interpolation of the points (see Section 2.7.1) and evaluate the polynomial in $x = 0$ to obtain the value r;

7. similarly, with at least $2t'_{\texttt{RT}} - 1$ shares (j, δ_j), each RT_i can obtain the value $\delta = a \cdot (y + r)$;

8. each RT_i computes and broadcasts the encryptions $E_{\texttt{pk}_{\texttt{ACC}}}[g_1]$ and $E_{\texttt{pk}_{\texttt{ACC}}}[g_3^{x_i}]$, together with a NIZKP that proves knowledge of the exponent x_i (see Protocol 5);

9. with $t_{\texttt{RT}}$ of these encryptions (with valid proofs), each RT_i can interpolate in the exponent and exploit the homomorphic properties of the encryption to obtain $E_{\texttt{pk}_{\texttt{ACC}}}[g_1 \cdot g_3^x]$;

[8] A common practice in multi-party protocols is for each party to commit to randomly generated values before performing any operations on them. This commitment ensures that the values cannot be changed later, which maintains unpredictability and integrity throughout the protocol.

10. each RT_i computes and broadcasts $\left(i, E_{\mathtt{pk}_{\mathtt{ACC}}}[g_1 \cdot g_3^x]^{d_i}\right)$, with $d_i = a_i \cdot \delta^{-1}$, so that they can be interpolated to obtain $E_{\mathtt{pk}_{\mathtt{ACC}}}[g_1 \cdot g_3^x]^d$:

$$\begin{aligned} E_{\mathtt{pk}_{\mathtt{ACC}}}[g_1 \cdot g_3^x]^d &= E_{\mathtt{pk}_{\mathtt{ACC}}}[g_1 \cdot g_3^x]^{\frac{a}{\delta}} = E_{\mathtt{pk}_{\mathtt{ACC}}}[g_1 \cdot g_3^x]^{\frac{a}{a(y+r)}} \\ &= E_{\mathtt{pk}_{\mathtt{ACC}}}[(g_1 \cdot g_3^x)^{\frac{1}{y+r}}] = E_{\mathtt{pk}_{\mathtt{ACC}}}[A]; \end{aligned}$$

11. every RT_i retrieves the value $A = (g_1 \cdot g_3^x)^{\frac{1}{y+r}}$ by threshold decryption (see Section 2.9.1) of $E_{\mathtt{pk}_{\mathtt{ACC}}}[g_1 \cdot g_3^x]^d$;

12. then every RT_i computes and broadcasts the encryption $E^{\tilde{r}_i}_{\mathtt{pk}_{\mathtt{TT}}}[A]$ together with the proof that it encrypts A;

13. each RT_i interpolates the received values to obtain $E_{\mathtt{pk}_{\mathtt{TT}}}[A]$;

14. each RT_i computes $x_i + \sigma_i$ and stores privately the tuple:

$$\begin{aligned} \mathcal{T}_{i,v_{\mathtt{id}}} = (&v_{\mathtt{id}}, \\ &A, r, E_{\mathtt{pk}_{\mathtt{TT}}}[A], E_{\mathtt{pk}_{\mathtt{ACC}}}\left[g_1 \cdot g_3^x\right], \\ &x_i + \sigma_i, \\ &\sigma_i, \\ &\tilde{r}_i, E^{\tilde{r}_i}_{\mathtt{pk}_{\mathtt{TT}}}[A], \Pi_i), \end{aligned}$$

    where Π_i includes all the proofs generated;

15. the tuple $\left(A, r, E_{\mathtt{pk}_{\mathtt{TT}}}[A], E_{\mathtt{pk}_{\mathtt{ACC}}}\left[g_1 \cdot g_3^x\right]\right)$ is called *public ACC* while x_i is the *private ACC share* and σ_i are the shares of the mask used to hide it and from which the `PIN` will be retrieved.

Each RT_i sends the public ACC and its progressive index $v_{\mathtt{id}}$ to be written on the WBB in the list $\mathcal{L}_{\mathtt{ACC}}$.

3.5.5 Multiple Concurrent Elections

Frequently, multiple elections (which may have distinct eligibility rules) occur at the same time. In these cases, a distinct ACC is required for every election. The voting protocol allows to create different ACCs which share the same private value x, changing only the public part $\left(A, r, E_{\mathtt{pk}_{\mathtt{TT}}}[A]\right)$. In this way, each voter can still use the same private ACC x for every election, so they can vote without the need to use each time a different PIN. In this context a common PIN improves significantly UI and UX.

To issue distinct ACCs for each election, the RTs first partake in the ACC Generation protocol of Section 3.5.4 to generate the ACC for the first election. Then they perform subsequent rounds of the same protocol of Section 3.5.4 but skipping the steps 2a, 2b, 8, 9, and using in step 10 the value $E_{\mathtt{pk}_{\mathtt{ACC}}}[g_1 \cdot g_3^x]$ previously computed (it can also be retrieved from the bulletin board).

3.6 Enrollment

Let $\mathcal{V}$ be the voter that wants to use *Vote App* to take part in the election.

3.6.1 App Setup

As a first step, $\mathcal{V}$ downloads *Vote App*, installs it on their device, and logs in using their digital identity (see Section 5.3.1).

1. Once proven the eligibility of the voter in the election (see Item 4b of Section 5.3.1.1), *Vote App* receives a `Registration token` (see Item 4c of Section 5.3.1.1);
2. *Vote App* generates uniformly at random the a designated verifier private key $\mathtt{sk}_{\mathtt{DV}} \in \mathbb{Z}_p$ and computes the corresponding public key $\mathtt{pk}_{\mathtt{DV}} = g_2^{\mathtt{sk}_{\mathtt{DV}}}$;
3. *Vote App* generates uniformly at random a bit string of length $\mathtt{L}_{\mathtt{wl}} \cdot \mathtt{L}_{\mathtt{psph}}$, splits it in chunks of $\mathtt{L}_{\mathtt{wl}}$ bits which are used to select a word (interpreting the bit string as an integer) from `wl`, obtaining a passphrase `psph` consisting of $\mathtt{L}_{\mathtt{psph}}$ words;
4. let $\mathtt{mask}_{\mathtt{sk}_{\mathtt{DV}}} = H(\mathtt{psph}, \lceil \log_2(p) \rceil)$, *Vote App* computes the designated verifier key share $\mathtt{share}_{\mathtt{sk}_{\mathtt{DV}}}$ as $\mathtt{share}_{\mathtt{sk}_{\mathtt{DV}}} = \mathtt{mask}_{\mathtt{sk}_{\mathtt{DV}}} \oplus \mathtt{sk}_{\mathtt{DV}}$ (with implicit serialization of $\mathtt{sk}_{\mathtt{DV}}$);
5. *Vote App* sends $\mathtt{share}_{\mathtt{sk}_{\mathtt{DV}}}$, $\mathtt{pk}_{\mathtt{DV}}$ to ER, authorized with `Registration token`;
6. *Vote App* receives a `PIN request token` (see Item 1 of Section 5.3.1.2);
7. *Vote App* generates and saves on the device a random id number $\mathtt{rid}_1$ which is the id that will identify the notifications concerning the PIN and DVNIZKP;

8. using $v_{\texttt{id}}$, *Vote App* retrieves from WBB $\texttt{pk}_{\texttt{EL}}$ and their public ACC $\left(A, r, E_{\texttt{pk}_{\texttt{TT}}}[A], E_{\texttt{pk}_{\texttt{ACC}}}[g_1 \cdot g_3^x]\right)$ from the WBB;
9. then it contacts the RT_i and sends $\texttt{pk}_{\texttt{DV}}$ authorized with the `PIN request token` (see Section 5.3.1.3) and each RT_i sends to *Vote App* $(i, x_i + \sigma_i, E^{\tilde{r}_i}_{\texttt{pk}_{\texttt{TT}}}[A], \Pi_i)$;
10. with $t_{\texttt{RT}}$ tuples, *Vote App* can interpolate the values and compute $x+\sigma$, and $E_{\texttt{pk}_{\texttt{TT}}}[A]$;
11. *Vote App* checks the proofs Π_i and the correctness of the ciphertext interpolation.

Then it saves, encrypted:

- the election public key $\texttt{pk}_{\texttt{EL}}$;
- the designated secret key $\texttt{sk}_{\texttt{DV}}$;
- the designated public key $\texttt{pk}_{\texttt{DV}}$;
- the public ACC $\left(A, r, E_{\texttt{pk}_{\texttt{TT}}}[A], E_{\texttt{pk}_{\texttt{ACC}}}[g_1 \cdot g_3^x]\right)$;
- the masked private ACC $x + \sigma$;
- $n_{\texttt{RT}} - t_{\texttt{RT}} + 1$ copies of the pair $(0, \texttt{mph})$, where the second value is a string of bytes, all with value `FF`, with the same length of the serialization of a scalar in $\mathbb{Z}_p$;
- `dvph`, a string of bytes, all with value `FF`, with the same length of the serialization of a DVNIZKP (see Section 3.6.3);
- $n_{\texttt{RT}} - t_{\texttt{RT}}$ copies of the pair $(0, \texttt{dvphshare})$, where the second value is the same as `mph`;
- the designated verifier key share $\texttt{share}_{\texttt{sk}_{\texttt{DV}}}$;
- the passphrase `psph`.

Vote App shows to the voter $v_{\texttt{id}}$ and `psph`, suggesting to safely back up these values.

Prior to casting their vote, voters are required to wait for the reception of their personal identification number (PIN). In the meantime, from *Vote App* the voter can only access *Vote App* settings, and consult the instructions and tutorials.

3.6.2 DVNIZKP Generation

After receiving $\mathtt{pk}_{\mathtt{DV}}$, each RT_i starts the generation of the designated verifier proof:

1. each RT_i generates uniformly at random $(c'_{i,0}, \varphi_i, z_{i,0}) \in \mathbb{Z}_p^3$ and performs Protocol 2 with
$$n = n_{\mathtt{RT}}, t = t_{\mathtt{RT}}, g = g_2, \hat{s}_i = c'_{i,0}$$
to get and broadcast to the other RTs the shares $(i, c_{i,0})$;

2. with $t_{\mathtt{RT}}$ shares, each RT_i can interpolate and compute c_0, then it computes:
$$I_{i,0} = g_2^{z_{i,0}} \cdot \mathtt{pk}_{\mathtt{DV}}^{-c_0}, I_{i,1} = A^{\varphi_i}, I_{i,2} = g_3^{\varphi_i},$$
and broadcasts to the other RTs the tuple $(i, I_{i,0}, I_{i,1}, I_{i,2})$;

3. with $t_{\mathtt{RT}}$ tuples, each RT_i can interpolate the values and obtain the three values:
$$I_0 = g_2^{z_0} \cdot \mathtt{pk}_{\mathtt{DV}}^{-c_0}, I_1 = A^{\varphi}, I_2 = g_3^{\varphi},$$
then it computes:
$$c = \mathcal{H}_p(A, \mathtt{pk}_{\mathtt{RT}}, I_0, I_1, I_2), c_1 = c - c_0,$$
and finally:
$$z_{i,1} = \varphi_i + c_1 \cdot y_i$$
where $y_i = \mathtt{sk}_{\mathtt{RT}_i}$ is a share of the registration private key corresponding to $\mathtt{pk}_{\mathtt{RT}}$, as defined in Section 3.5.2;

4. each RT_i broadcasts to the other RTs the shares $(i, z_{i,0})$;

5. with $t_{\mathtt{RT}}$ shares, each RT_i can interpolate and compute z_0, then checks that
$$I_0 \stackrel{?}{=} g_2^{z_0} \cdot \mathtt{pk}_{\mathtt{DV}}^{-c_0}$$
and, if all is correct, it then saves the share $(z_{i,1}, z_0, I_0, I_1, I_2, c_0, c_1)$ alongside $\mathcal{T}_{i,v_{\mathtt{id}}}$.

Up until di point, the $\mathtt{PIN}$ and x considered were only the valid ones. From now on we will identify with $\mathtt{PIN}$ and x a generic PIN and private credential, while $\overset{\mathtt{valid}}{\mathtt{PIN}}$ and $\overset{\mathtt{ruse}}{\mathtt{PIN}}$, together with $\overset{\mathtt{valid}}{x}$ and $\overset{\mathtt{ruse}}{x}$, will identify the valid and ruse ones, respectively.

3.6.3 PIN and DVNIZKP Arrival

After a random time (see Section 5.3.1.4), *Vote App* receives a `PIN and DVNIZKP retrieval token` (see Item 8c of Section 5.3.1.4) and uses it to make a request to retrieve the `PIN`[9] and the DVNIZKP from the RTs (see Section 5.3.1.5):

1. each RT_i sends to *Vote App* both their share of the DVNIZKP

   $$(i, z_{i,1}, z_0, I_0, I_1, I_2, c_0, c_1)$$

   and the shares of the mask to retrieve the `PIN` (i, σ_i);

2. having received at least $t_{\texttt{RT}}$ tuples, *Vote App* considers the values

   $$(z_0, I_0, I_1, I_2, c_0, c_1)$$

   sent by trusted RT_i (note that they should all be equal), and, if the check holds, can interpolate the values $(i, z_{i,1})$ to compute z_1;

3. after it has received $t_{\texttt{RT}}$ tuples, *Vote App* can interpolate the values and compute σ which is split as: $\sigma = \hat{\sigma} \cdot 10^{\texttt{L}_{\texttt{PIN}}} + \texttt{PIN}$;

4. *Vote App* computes x from $x + \sigma$ and uses it to compute o^x;

5. *Vote App* shows to the voter:

   - the PIN `PIN`;
   - the visual encoding `PrivatePINEmoji` (for further discussion on the effectiveness of using emoji to represent information see Appendix A.3) which represents $\mathcal{H}(o^x)$ [10];
   - the encoding of o^x (see Appendix A.3.6);
   - the possibility to perform the DVNIZKP;

[9] If this is the first time that this request is made, then $\texttt{PIN} = \overset{\texttt{valid}}{\texttt{PIN}}$. Otherwise $\texttt{PIN} = \overset{\texttt{ruse}}{\texttt{PIN}}$ if this originates from a ruse PIN request (see Section 3.7.3), or $\texttt{PIN} = \overset{\texttt{valid}}{\texttt{PIN}}$ is this originates from a pin re-sending request (see Section 3.7.2) or if it originates from a new device registration (see Section 3.7.4). The same applies for x.

[10] In case of multiple concurrent elections there is a different generator o for each election, so with this approach the voter should see a distinct visual encoding for each election. This deteriorates UX so a preferable approach could be to use directly x instead of o^x. The same argument applies for the next step and every time the visual encoding of o^x is mentioned.

6. *Vote App* updates its storage, substituting:
   - the first `mph` with the serialization of the truncated mask $\sigma - \texttt{PIN}$;
   - for each RT_i that is not considered trusted (see Section 3.12), a pair $(0, \texttt{mph})$ is substituted with the serialization of the share sent by that untrusted RT (i, σ_i);
   - the placeholder `dvph` with the serialization of the DVNIZKP
   $$(z_0, z_1, I_0, I_1, I_2, c_0, c_1);$$
   - for each RT_i that is not considered trusted, as set by $\mathcal{V}$ in Section 3.12, a placeholder pair $(0, \texttt{dvphshare})$ is substituted with the serialization of the share sent by that untrusted RT $(i, z_{i,1})$;

   and encrypting it all;
7. *Vote App* enables all its functionalities.

3.7 PIN and Device Management

3.7.1 PIN Verification

Once the verification feature is enabled, the voter $\mathcal{V}$, whose pseudonym is $v_{\texttt{id}}$, can use *Vote App* to check the correctness of the `PIN` and of its ACC, namely that $\log_A (g_1 g_3^x A^{-r}) = \log_{g_3} \texttt{pk}_{\texttt{RT}}$ as many times as wanted with the following procedure:

1. $\mathcal{V}$ opens *Vote App* using biometric authentication or the device PIN[11];
2. *Vote App* decrypts the stored data, in particular recovers the DVNIZKP $(z_0, z_1, I_0, I_1, I_2, c_0, c_1)$, the designated public key $\texttt{pk}_{\texttt{DV}}$, the masked private ACC $x + \sigma$, the truncated mask $\sigma - \texttt{PIN}$, the public ACC (A, r), and the registration public key $\texttt{pk}_{\texttt{RT}}$ and the generators g_1, g_2, g_3, o from the election public key $\texttt{pk}_{\texttt{EL}}$;
3. after $\mathcal{V}$ has typed in `PIN`, *Vote App* computes:
   $$x = (x + \sigma) - (\sigma - \texttt{PIN}) - \texttt{PIN};$$

[11] Biometric authentication or a device PIN provides faster and more convenient access to the application after the initial eID login, avoiding repeated credential input, reducing reliance on external systems, and enabling offline access while maintaining session security.

4. *Vote App* computes $c = \mathcal{H}_p(A, \mathtt{pk}_{\mathtt{RT}}, I_0, I_1, I_2)$ and checks the correctness of the credential by verifying that the following equations hold:

$$\begin{aligned} c &\stackrel{?}{=} c_0 + c_1, \\ g_2^{z_0} &\stackrel{?}{=} I_0 \cdot \mathtt{pk}_{\mathtt{DV}}^{c_0}, \\ g_3^{z_1} &\stackrel{?}{=} I_2 \cdot \mathtt{pk}_{\mathtt{RT}}^{c_1}, \\ A^{z_1} &\stackrel{?}{=} I_1 \cdot (g_1 \cdot g_3^{x} \cdot A^{-r})^{c_1}; \end{aligned}$$

5. if any of these equalities does not hold, the voter is advised that the verification did not work thus the `PIN` inserted is not compatible with the public part;

6. otherwise, *Vote App* shows to $\mathcal{V}$:

   - the visual encoding `PrivatePINEmoji` which represents $\mathcal{H}(o^x)$ (see Appendix A.3.1);
   - the encoding of o^x (see Appendix A.3.6).

This verification confirms to $\mathcal{V}$ that the ACC received from the RTs is valid.

Moreover we underline that the verification can also be used to check that the PIN $\mathcal{V}$ remembers is correct. In fact, if $\mathcal{V}$ types a different PIN, *Vote App* retrieves a different, and not correct, private credential, and the DVNIZKP will not verify.

3.7.2 PIN Re-Sending

If the voter $\mathcal{V}$ forgets $\overset{\texttt{valid}}{\texttt{PIN}}$, they can ask to receive it again with the following procedure:

1. *Vote App* generates and saves on the device a new random id number $\mathtt{rid}'_1$ as in step 7 of Section 3.6.1;

2. using the `PIN request token` provided by ER (in either step 1 of Section 5.3.1.2, or step 8c of Section 5.3.1.4, or step 7 of Section 3.7.4), *Vote App* retrieves $(i, x_i + \sigma_i, E^{\tilde{r}_i}_{\mathtt{pk}_{\mathtt{TT}}}[A], \Pi_i)$ via the protocol of Section 5.3.1.3 as done in Item 9 of Section 3.6.1;

3. *Vote App* may check the consistency of the received data with the values stored, then updates the stored data:

- substitutes the serialization of the truncated mask and the shares of the untrusted RTs with the pair $(0, \texttt{mph})$;
- substitutes the serialization of the DVNIZKP with the placeholder `dvph`;
- substitutes the serialization of the DVNIZKP shares of the untrusted RTs with the pair $(0, \texttt{dvphshare})$;

as done in Section 3.6.1;

4. *Vote App* disables the voting, PIN management, and PIN verification functionalities;
5. each RT_i, the NS, and *Vote App* proceed with:
   (a) the dispatch and reception of the shares:
   $$(i, \sigma_i), (i, z_{i,1}, z_0, I_0, I_1, I_2, c_0, c_1)$$
   and subsequent reconstruction of `PIN` and DVNIZKP as explained in Section 3.6.3.

3.7.3 Ruse PIN

A voter $\mathcal{V}$ that is, or fears to be, under coercion, can set up a *ruse PIN* $\overset{\texttt{ruse}}{\texttt{PIN}}$, to be used to deceive the coercer, with the following procedure:

1. let J be the set of indexes of the trusted RTs, as set by $\mathcal{V}$ (see Section 3.12), and $J' \subseteq J$ a subset of size $t_{\texttt{RT}} - 1$;
2. *Vote App* decrypts the stored data, in particular recovers the truncated mask $\sigma - \texttt{PIN} = \hat{\sigma} \cdot 10^{\text{L}_{\texttt{PIN}}}$, the masked private ACC $x + \sigma$[12], the shares (i, σ_i), $(i, z_{i,1})$ for $i \notin J$, the designated private key $\texttt{sk}_{\texttt{DV}}$, the public ACC (A, r), and extracts the generators g_1, g_2, g_3 and the registration public key $\texttt{pk}_{\texttt{RT}}$ from the election public key $\texttt{pk}_{\texttt{EL}}$;
3. after $\mathcal{V}$ has typed in $\overset{\texttt{ruse}}{\texttt{PIN}}$, of the same length as `PIN`, *Vote App* computes $\sigma' = \hat{\sigma} \cdot 10^{\text{L}_{\texttt{PIN}}} + \overset{\texttt{ruse}}{\texttt{PIN}}$;
4. *Vote App* uses the points $(0, \sigma')$, $\{(i, \sigma_i)\}_{i \notin J'}$ to interpolate a polynomial $f_{\sigma'}$ of degree $t_{\texttt{RT}} - 1$ and computes the ruse shares $\sigma'_j = f_{\sigma'}(j)$ for $j \in J'$;

[12] Depending on the actions performed previously, the retrieved values can be either valid or decoy.

5. *Vote App* computes a ruse DVNIZKP for $\overset{\texttt{ruse}}{x} = \overset{\texttt{valid}}{x} + \overset{\texttt{valid}}{\texttt{PIN}} - \overset{\texttt{ruse}}{\texttt{PIN}}$ to avail $\overset{\texttt{ruse}}{\texttt{PIN}}$:

   (a) generates $c_1, z_1, \varphi \in \mathbb{Z}_p$ uniformly at random;

   (b) computes:

$$\begin{aligned} I_0 &= g_2^{\varphi}, \\ I_1 &= A^{z_1} \cdot (g_1 \cdot g_3^{\overset{\texttt{ruse}}{x}} \cdot A^{-r})^{-c_1}, \\ I_2 &= g_3^{z_1} \cdot \texttt{pk}_{\texttt{RT}}^{-c_1}, \\ c &= \mathcal{H}_p(A, \texttt{pk}_{\texttt{RT}}, I_0, I_1, I_2), \\ c_0 &= c - c_1, \\ z_0 &= \varphi + c_0 \cdot \texttt{sk}_{\texttt{DV}}; \end{aligned}$$

   (c) *Vote App* uses the points $(0, z_1)$, $\{(i, z_{i,1})\}_{i \notin J'}$ to interpolate a polynomial f_{z_1} of degree $t_{\texttt{RT}} - 1$ and computes the ruse shares $z'_{j,1} = f_{z_1}(j)$ for $j \in J'$;

6. *Vote App* generates and saves on the device a new random id $\texttt{rid}_2$ as in step 7 of Section 3.6.1;

7. using the `PIN request token` provided by ER in step 8c of Section 5.3.1.4, *Vote App* sends to each RT_i a request for the re-sending of the PIN[13] containing, just like in Section 5.3.1.3:

   - the asset $\texttt{rid}_2$;
   - if $i \in J$:
     - the serialization of σ'_i;
     - the serialization of $(z'_{i,1}, z_0, I_0, I_1, I_2, c_0, c_1)$;
   - if $i \notin J$:
     - `mph`, a string of bytes, all with value `FF`, with the same length of the serialization of a scalar in $\mathbb{Z}_p$;
     - `dvph`, a string of bytes, all with value `FF`, with the same length of the serialization of a DVNIZKP;

8. each RT_i checks the validity of the authorization token sent by *Vote App* and identifies the request from the content of the two byte strings:

[13] It is important to note that at this stage, the PIN request is as if it were an initial enrollment or PIN reminder.

- the trusted RTs will see the serialization of a scalar $\sigma'_i \in \mathbb{Z}_p$ and of a DVNIZKP share $(z'_{i,1}, z_0, I_0, I_1, I_2, c_0, c_1)$, so they will proceed as in Section 3.7.2 but using these values instead of the original σ_i and $(z_{i,1}, z_0, I_0, I_1, I_2, c_0, c_1)$ respectively;
- untrusted RTs will see only bytes with value `FF`, so they will proceed exactly as in Section 3.7.2, since they believe it is a request of PIN re-sending.

At this point, *Vote App* returns to its initial state, thus before being able to vote the voter will have to wait for the reception of the ruse PIN. In the meantime, from *Vote App* the voter can only access *Vote App* settings, and consult the instructions and tutorials.

The protocol then proceeds as in Section 3.6.3. Note that the values returned by trusted RTs are the decoy values here created, while untrusted RTs return the valid shares.

There are no limits on the *Ruse PIN* requests.

Note that a ruse PIN request causes the substitution of the original DVNIZKP with the ruse DVNIZKP. This means that $\overset{\texttt{valid}}{\texttt{PIN}}$ will no longer verify the DVNIZKP since in step 3 of Section 3.7.1 *Vote App* will compute $\overset{\texttt{ruse}}{x}$ instead of $\overset{\texttt{valid}}{x}$. So *Vote App* will consider $\overset{\texttt{ruse}}{\texttt{PIN}}$ as correct instead of $\overset{\texttt{valid}}{\texttt{PIN}}$. This does not undermine the ability of $\overset{\texttt{valid}}{\texttt{PIN}}$ to cast a valid vote, since it is only intended to fool a coercer.

3.7.4 Device Management

A voter $\mathcal{V}$ can use multiple devices. To register a new device, the voter downloads *Vote App* from the official app store and:

1. after $\mathcal{V}$'s authentication with its digital identity, *Vote App* retrieves $\mathcal{V}$'s pseudonymous identifier $v_{\texttt{id}}$, $\mathcal{V}$'s designated verifier public key $\texttt{pk}_{\texttt{DV}}$ and $\mathcal{V}$'s designated verifier key share $\texttt{share}_{\texttt{sk}_{\texttt{DV}}}$, together with the `Registration token` (see Item 4c of Section 5.3.1.1)

2. *Vote App* asks for the `psph`;

3. $\mathcal{V}$ retrieves `psph` from their safe backup and types it into *Vote App*;

4. *Vote App* computes (with implicit deserialization of $\mathtt{sk}_{\mathtt{DV}}$) the designated verifier private key $\mathtt{sk}_{\mathtt{DV}} = \mathtt{share}_{\mathtt{sk}_{\mathtt{DV}}} \oplus H(\mathtt{psph}, \lceil \log_2(p) \rceil)$ (see step 4 of Section 3.6.1);

5. *Vote App* checks that $\mathtt{pk}_{\mathtt{DV}} \stackrel{?}{=} g_2^{\mathtt{sk}_{\mathtt{DV}}}$, if this equality does not hold the `psph` is not correct (if $\mathcal{V}$ is not able to retrieve the correct `psph`, they can ask for their credential to be revoked and have a new one issued to them, see Section 3.7.5);

6. *Vote App* sends back $\mathtt{share}_{\mathtt{sk}_{\mathtt{DV}}}$, $\mathtt{pk}_{\mathtt{DV}}$ to ER, authorized with the `Registration token` (see Item 4c of Section 5.3.1.1);

7. the ER compares the received and stored values, if they correspond it sends back the `PIN request token` (see Item 1 of Section 5.3.1.2)

8. *Vote App* proceeds as in the PIN re-sending procedure (Section 3.7.2);

Note that from the point of view of the RTs this procedure is indistinguishable from a PIN re-sending, and that *Vote App* retrieves all the data needed to complete the setup as in Section 3.6.1.

The steps performed up to Item 5 are only required to retrieve $\mathtt{sk}_{\mathtt{DV}}$. While *Vote App* remains accessible and functional even if `psph` is entered incorrectly, the voter would lose the ability to generate decoy credentials. Consequently, the coercion resistance property is compromised, since it would no longer be possible to generate a decoy DVNIZKP to validate the decoy credential.

3.7.5 ACC Revocation

A voter $\mathcal{V}$ who loses access to their private ACC or thinks that it may have been compromised, can ask for its revocation and the issuing of a new credential with the following procedure:

1. the ER marks the $v_{\mathtt{id}}$ previously associated to $\mathcal{V}$ as revoked, and assigns a random unassigned identifier $v'_{\mathtt{id}}$ to $\mathcal{V}$;

2. the ER publishes on the WBB a commitment to the revocation request and the new pseudonymous identifier association;

3. $\mathcal{V}$ can proceed with a new registration as in Section 3.6.1.

Note that up to $n_{\mathtt{ACC}} - n_{\mathtt{V}}$ revocations can be handled by the system.

3.8 Voting

We are considering an election for Italian citizens which vote from abroad. In the Italian electoral law [114], the ballot presents $n_{\texttt{Ls}}$ names of electoral lists (political parties) each associated to a list of $n_{\texttt{Ca},i}$, $1 \leq i \leq n_{\texttt{Ls}}$ candidates. We will denote with $\texttt{Ls}_i$, for $1 \leq i \leq n_{\texttt{Ls}}$ the electoral lists and with $\texttt{Ca}_{i,j}$, for $0 \leq j \leq n_{\texttt{Ca},i}$ the associated candidates, where $n_{\texttt{Ca},i}$ is the number of candidates associated to the electoral list $\texttt{Ls}_i$.

A voter $\mathcal{V}$ can express the following preferences:

- vote for *one* electoral list $\texttt{Ls}_i$;
- vote for *one* electoral list $\texttt{Ls}_i$ and *one* of the candidates associated to $\texttt{Ls}_i$;
- leave a blank ballot.

To simplify the notation, we call $\texttt{Ls}_0$ the choice of leaving a blank ballot (thus $n_{\texttt{Ca},0} = 0$), and $\texttt{Ca}_{i,0}$ the blank candidate, used to encode the choice of voting for a electoral list without voting also for a candidate of that electoral list.

3.8.1 Preference encoding and validation

A voter's preference will be encoded as $(\texttt{Ls}_i, (\texttt{Ca}_{i,j})_{j=0}^{n_{\texttt{Ca},i}})_{i=0}^{n_{\texttt{Ls}}}$, where each $\texttt{Ls}_i$ is equal to 1 if the voter wants to vote for the corresponding electoral list, equal to 0 otherwise. Similarly, each $\texttt{Ca}_{i,j}$ is equal to 1 if the voter wants to vote for the corresponding candidate, equal to 0 otherwise.

Therefore a valid preference is a composite list of integers $(\texttt{Ls}_i, (\texttt{Ca}_{i,j})_{j=0}^{n_{\texttt{Ca},i}})_{i=0}^{n_{\texttt{Ls}}}$ that satisfies the following conditions:

C1: $\texttt{Ls}_i, \texttt{Ca}_{i,j} \in \{0, 1\}$ for every $0 \leq j \leq n_{\texttt{Ca},i}$, $0 \leq i \leq n_{\texttt{Ls}}$;

C2: $\sum_{i=0}^{n_{\texttt{Ls}}} \texttt{Ls}_i = 1$;

C3: for every $0 \leq i \leq n_{\texttt{Ls}}$, $\texttt{Ls}_i = \sum_{j=0}^{n_{\texttt{Ca},i}} \texttt{Ca}_{i,j}$.

In simpler terms, $\mathcal{V}$ can't give more than one vote to an electoral list or a candidate (condition C1), they can vote for at most one electoral list among the ones available (condition C2) and for at most one candidate associated to the previously chosen electoral list (condition C3).

3.8.2 Voting: Preference Acquisition and Encryption

To simplify the notation, let us fix the voter $\mathcal{V}$ with pseudonymous identifier $v_{\mathtt{id}}$, in the following subsections we will omit them from the notation. $\mathcal{V}$ starts the voting process with the following procedure:

1. *Vote App* decrypts the stored data, in particular recovers the truncated mask $\hat{\sigma}\cdot 10^{\mathtt{L_{PIN}}}$, the masked private ACC $\overset{\mathtt{valid}}{x}+\sigma$, the public ACC (A, r), and extracts the generators g_0, g_1, g_2, g_3, o, the election id $\mathtt{EL_{ID}}$, and the tallying public key $\mathtt{pk_{TT}}$ from the election public key $\mathtt{pk_{EL}}$;
2. $\mathcal{V}$ selects an active election in *Vote App*;
3. *Vote App* shows (according to the ballot as defined in Section 3.4.2) the electoral lists (number, name, symbol) plus the option of choosing a blank ballot (with number equal to 0);
4. $\mathcal{V}$ selects an electoral list or the blank option;
5. *Vote App* shows (according to the Ballot as defined in Section 3.4.2) the candidates associated to the chosen electoral list (number and name) plus the option of not choosing a candidate (with number equal to 0)[14];
6. $\mathcal{V}$ selects a candidate or the blank option;
7. let $v = (\mathtt{Ls}_i, (\mathtt{Ca}_{i,j})_{j=0}^{n_{\mathtt{Ca},i}})_{i=0}^{n_{\mathtt{Ls}}}$ be the encoding of the preferences expressed[15] by $\mathcal{V}$, *Vote App* encrypts v as:
$$\bar{\mathcal{E}} = \left(\mathcal{E}_{\mathtt{pk_{TT}}}^{\mathtt{rnd}_i}[\mathtt{Ls}_i], \left(\mathcal{E}_{\mathtt{pk_{TT}}}^{\mathtt{rnd}_{i,j}}[\mathtt{Ca}_{i,j}]\right)_{j=0}^{n_{\mathtt{Ca},i}}\right)_{i=0}^{n_{\mathtt{Ls}}},$$
where $\mathtt{rnd}_i, \mathtt{rnd}_{i,j} \in \mathbb{Z}_p$ are chosen uniformly at random, for $0 \leq j \leq n_{\mathtt{Ca},i}$, $0 \leq i \leq n_{\mathtt{Ls}}$;
8. *Vote App* chooses uniformly at random a byte string $\mathtt{rnd_{seed}}$ of length $\mathtt{Ca_{seed}}$, and derives from it three scalars as:
$$\begin{aligned}\mathtt{rnd}_A &= \mathcal{H}_p(\mathtt{rnd_{seed}}||\mathtt{DS1}),\\ \mathtt{rnd}_{A^r} &= \mathcal{H}_p(\mathtt{rnd_{seed}}||\mathtt{DS2}),\\ \mathtt{rnd}_{o^x} &= \mathcal{H}_p(\mathtt{rnd_{seed}}||\mathtt{DS3}),\end{aligned}$$

[14] If the blank option has been selected in the previous step, *Vote App* automatically selects the blank candidate since it is the only, mandatory choice.

[15] If the blank option has been selected for the electoral list $\mathtt{Ls}_0$, *Vote App* automatically selects the blank candidate $\mathtt{Ca}_{0,0}$ since it is the only, mandatory choice.

where $\texttt{DS1}, \texttt{DS2}, \texttt{DS3}$ are three distinct (fixed) domain separation byte strings;

9. *Vote App* computes the visual encoding `BallotEmoji` (see Appendix A.3.1) which represents:

$$\mathcal{H}\left(\texttt{EL}_{\texttt{ID}}, E^{\texttt{rnd}_A}_{\texttt{pk}_{\texttt{TT}}}\left[A\right], E^{\texttt{rnd}_{A^r}}_{\texttt{pk}_{\texttt{TT}}}\left[A^r\right]\right),$$

10. *Vote App* computes the encryption of the *cast-as-intended values* [40]:
    (a) let $\texttt{Ls} = \sum_{i=0}^{n_{\texttt{Ls}}} i \cdot \texttt{Ls}_i$ (note that this is exactly the index of the electoral list chosen by the voter or 0 if the vote is blank);
    (b) let $\texttt{Ca} = \sum_{i=0}^{n_{\texttt{Ls}}} \sum_{j=0}^{n_{\texttt{Ca},i}} j \cdot \texttt{Ca}_{i,j}$ (note that this is exactly the index of the candidate chosen by the voter or 0 if no preference has been expressed);
    (c) *Vote App* generates uniformly at random two values $0 \leq s_{\texttt{Ls}}, s_{\texttt{Ca}} < 100$[16];
    (d) *Vote App* generates uniformly at random $(\texttt{rnd}_{\texttt{Ls},1}, \texttt{rnd}_{\texttt{Ls},0}, \texttt{rnd}_{\texttt{Ca},1}, \texttt{rnd}_{\texttt{Ca},0}) \in \mathbb{Z}_p^4$ and computes the following encryptions:

$$\begin{aligned} &\mathcal{E}^{\texttt{rnd}_{\texttt{Ls},0}}_{\texttt{pk}_{\texttt{TT}}}[s_{\texttt{Ls}}], \quad \mathcal{E}^{\texttt{rnd}_{\texttt{Ls},1}}_{\texttt{pk}_{\texttt{TT}}}[(s_{\texttt{Ls}} + \texttt{Ls}) \mod 100], \\ &\mathcal{E}^{\texttt{rnd}_{\texttt{Ca},0}}_{\texttt{pk}_{\texttt{TT}}}[s_{\texttt{Ca}}], \quad \mathcal{E}^{\texttt{rnd}_{\texttt{Ca},1}}_{\texttt{pk}_{\texttt{TT}}}[(s_{\texttt{Ca}} + \texttt{Ca}) \mod 100], \end{aligned}$$

    these values will be used by the voter to check that its vote has not been altered (see Item 10);

11. *Vote App* shows to $\mathcal{V}$ the visual encoding `BallotEmoji` and asks for the PIN;

12. after $\mathcal{V}$ has typed in `PIN`, *Vote App* computes:

$$x = (\overset{\texttt{valid}}{x} + \sigma) - (\hat{\sigma} \cdot 10^{\texttt{L}_{\texttt{PIN}}}) - \texttt{PIN},$$

    note that if $\texttt{PIN} = \overset{\texttt{valid}}{\texttt{PIN}}$, then $x = \overset{\texttt{valid}}{x}$, $\mathcal{V}$'s real private ACC;

13. *Vote App* chooses uniformly at random $\texttt{rnd}_{g_3^x} \in \mathbb{Z}_p$ and computes:

$$E^{\texttt{rnd}_{g_3^x}}_{\texttt{pk}_{\texttt{TT}}}\left[g_3^x\right];$$

[16] We chose 100 for usability reasons, since in this case the voter will have to perform two-digits additions which are manageable for most of them (see Appendix A.3). By analyzing the number of candidates and electoral lists presented in a real election, we can assume that the upper-bound for `Ls` and `Ca` is 10.

14. *Vote App* computes o^x, its visual encoding $\texttt{PrivatePINEmoji}\ \mathcal{H}(o^x)$ (see Appendix A.3.1)[17], and its encryption $E^{\texttt{rnd}_{o^x}}_{\texttt{pk}_{\texttt{TT}}}[o^x]$[18];

15. *Vote App* selects uniformly at random $s \in \mathbb{Z}_p$, computes $B = A^s \in \mathbb{G}$;

16. the complete ballot is:

$$\begin{aligned}\mathcal{B} = \Big(&\texttt{ID}, \bar{\mathcal{E}}, B, E^{\texttt{rnd}_A}_{\texttt{pk}_{\texttt{TT}}}[A], E^{\texttt{rnd}_{A^r}}_{\texttt{pk}_{\texttt{TT}}}[A^r],\\ &E^{\texttt{rnd}_{g_3^x}}_{\texttt{pk}_{\texttt{TT}}}[g_3^x], E^{\texttt{rnd}_{o^x}}_{\texttt{pk}_{\texttt{TT}}}[o^x],\\ &\mathcal{E}^{\texttt{rnd}_{\texttt{Ls},0}}_{\texttt{pk}_{\texttt{TT}}}[s_{\texttt{Ls}}], \mathcal{E}^{\texttt{rnd}_{\texttt{Ls},1}}_{\texttt{pk}_{\texttt{TT}}}[(s_{\texttt{Ls}} + \texttt{Ls}) \mod 100],\\ &\mathcal{E}^{\texttt{rnd}_{\texttt{Ca},0}}_{\texttt{pk}_{\texttt{TT}}}[s_{\texttt{Ca}}], \mathcal{E}^{\texttt{rnd}_{\texttt{Ca},1}}_{\texttt{pk}_{\texttt{TT}}}[(s_{\texttt{Ca}} + \texttt{Ca}) \mod 100],\\ &\Pi\Big),\end{aligned}$$

    where Π is a collection of NIZKPs that will be defined in Section 3.8.3;

17. *Vote App* shows to $\mathcal{V}$:

    - the visual encoding $\texttt{BallotEmoji}$;
    - the visual encoding $\texttt{PrivatePINEmoji}$[19].

3.8.3 Voting: Ballot NIZKPs

Each encrypted ballot contains a set of NIZKPs $\Pi = (\Pi_1, \Pi_2, \Pi_3, \Pi_4, \Pi_5, \Pi_6)$ which proves the formal correctness of the ballot itself. Each proof is designed to verify specific conditions for valid ballot encoding, as defined above:

1. Π_1 proves that $\bar{\mathcal{E}}$ encrypts a valid vote, actually this proof is composed by multiple proofs:

[17] If this encoding does not correspond to the one showed in Item 5 of Section 3.6.3, it is a symptom that the wrong PIN has been typed or something is wrong with the device, so the voter should abort the voting process.

[18] In the first version of the protocol, just the value o^x was included in the ballot (Item 16). In that case an attack could be mounted: if the adversary has $x + \sigma$ and $\sigma - \texttt{PIN}$, then, since $\texttt{PIN}$ is short, it would have been possible to link a vote with a private (ruse or valid) ACC. For further discussion on the shortness of $\texttt{PIN}$ and why it does not compromise the security of the system, see Chapter 5.

[19] From this moment to the end of the voting process, both visual encodings should be displayed by *Vote App* without changing from screen to screen, otherwise it could be a symptom that something is wrong with the device, so the voter should abort the voting process.

(a) for each electoral list $\texttt{Ls}_i$ (where $0 \leq i \leq n_{\texttt{Ls}}$), we verify the following:

i. $\Pi_{1,0,i}$ ensures that the selection value for the electoral list, $\texttt{Ls}_i$, is binary (i.e., $\texttt{Ls}_i \in \{0, 1\}$), confirming that an electoral list receives at most one vote;

ii. $\Pi_{1,1,i,j}$ verifies that each candidate selection $\texttt{Ca}_{i,j}$ is also binary (i.e., $\texttt{Ca}_{i,j} \in \{0, 1\}$) for all candidates associated with electoral list $\texttt{Ls}_i$, where $0 \leq j \leq n_{\texttt{Ca},i}$. This ensures each candidate receives at most one vote;

these conditions verify condition C1 in Section 3.8.1;

(b) $\Pi_{1,2}$ confirms that exactly one electoral list is chosen, which satisfies condition C2 of Section 3.8.1;

(c) For each electoral list $\texttt{Ls}_i$ (where $0 \leq i \leq n_{\texttt{Ls}}$), $\Pi_{1,3,i}$ verifies that the number of selected candidates associated with electoral list $\texttt{Ls}_i$ does not exceed the allowed limit. Specifically, this proof checks that $\sum_{j=0}^{n_{\texttt{Ca},i}} \texttt{Ca}_{i,j} = \texttt{Ls}_i$, which corresponds to condition C3 in Section 3.8.1.

2. Π_2, Π_3 prove that the voter knows the plaintexts related to $E^{\texttt{rnd}_A}_{\texttt{pk}_{\text{TT}}}[A]$, and $E^{\texttt{rnd}_{A^r}}_{\texttt{pk}_{\text{TT}}}[A^r]$ respectively;

3. Π_4 proves that the voter knows the plaintext related to the exponential encryption $E^{\texttt{rnd}_{g_3^x}}_{\texttt{pk}_{\text{TT}}}[g_3^x]$ with base g_3 and that $E^{\texttt{rnd}_{o^x}}_{\texttt{pk}_{\text{TT}}}[o^x]$ is the exponential encryption of the same value with base o[20];

4. Π_5 proves that the plaintext of $E^{\texttt{rnd}_A}_{\texttt{pk}_{\text{TT}}}[A]$ is different from 1;

5. Π_6 proves the correctness of the *cast-as-intended values* and is composed by two sub-proofs:

(a) $\Pi_{6,1}$ proves that $\mathcal{E}^{\texttt{rnd}_{\texttt{Ls},1}}_{\texttt{pk}_{\text{TT}}}[(s_{\texttt{Ls}} + \texttt{Ls}) \mod 100]$ encrypts the $\texttt{Ls}$ chosen by the voter and the same $s_{\texttt{Ls}}$ that is encrypted in $\mathcal{E}^{\texttt{rnd}_{\texttt{Ls},0}}_{\texttt{pk}_{\text{TT}}}[s_{\texttt{Ls}}]$;

(b) $\Pi_{6,2}$ proves that $\mathcal{E}^{\texttt{rnd}_{\texttt{Ca},1}}_{\texttt{pk}_{\text{TT}}}[(s_{\texttt{Ca}} + \texttt{Ca}) \mod 100]$ encrypts the $\texttt{Ca}$ chosen by the voter and the same $s_{\texttt{Ca}}$ that is encrypted in $\mathcal{E}^{\texttt{rnd}_{\texttt{Ca},0}}_{\texttt{pk}_{\text{TT}}}[s_{\texttt{Ca}}]$.

[20] This NIZKP ensures that the encrypted private credential x corresponds to the `PIN` inserted by the voter. In fact, the voter verifies that the x encrypted in $E^{\texttt{rnd}_{o^x}}_{\texttt{pk}_{\text{TT}}}[o^x]$ corresponds to the inserted `PIN` via the Verification Service, the `PrivatePINEmoji` and `PublicPINEmoji`. This proof then creates a link between this value and the x encrypted in $E^{\texttt{rnd}_{g_3^x}}_{\texttt{pk}_{\text{TT}}}[g_3^x]$, which will be used to determine, in the tally, if the vote is valid or not.

The Π_i's are constructed as follows:

1. $\Pi_{1,0,i}$ and $\Pi_{1,1,i,j}$, for $0 \leq i \leq n_{\mathtt{Ls}}$, $0 \leq j \leq n_{\mathtt{Ca},i}$ are computed via a disjunctive proof of equality (see Protocol 11) with $\nu = 1$;

2. $\Pi_{1,2}$ is computed via a proof of plaintext equality (see Protocol 9) with the ciphertext homomorphically computed as $\prod_{i=0}^{n_{\mathtt{Ls}}} \mathcal{E}_{\mathtt{pk_{TT}}}^{\mathtt{rnd}_i}[\mathtt{Ls}_i]$, note that the randomness is $\mathtt{rnd}_{\mathtt{Ls}} = \sum_{i=0}^{n_{\mathtt{Ls}}} \mathtt{rnd}_i$;

3. $\Pi_{1,3,i}$, for $0 \leq i \leq n_{\mathtt{Ls}}$ is computed via a proof of plaintext equality (see Protocol 9) between $\mathcal{E}_{\mathtt{pk_{TT}}}^{\mathtt{rnd}_i}[\mathtt{Ls}_i]$ and $\prod_{j=0}^{n_{\mathtt{Ca},i}} \mathcal{E}_{\mathtt{pk_{TT}}}^{\mathtt{rnd}_{i,j}}[\mathtt{Ca}_{i,j}]$ (which is a ciphertext with randomness $\mathtt{rnd}_{\mathtt{Ls}_i} = \sum_{j=0}^{n_{\mathtt{Ca},i}} \mathtt{rnd}_{i,j}$);

4. noting that $A = B^{s^{-1}}$, Π_2 and Π_3 are computed via a proof of plaintext knowledge (see Protocol 5) with parameters:

   (a) Π_2: $\ell = g_1$, $w = g_2$, $h = \mathtt{pk_{TT}}$, $g = B$, and $m = s^{-1}$,

   (b) Π_3: $\ell = g_1$, $w = g_2$, $h = \mathtt{pk_{TT}}$, $g = B$, and $m = rs^{-1}$;

5. Π_4 is computed via a proof of plaintext discrete logarithm equality (see Protocol 10) with parameters:

   (a) $x = x$, $s_1 = \mathtt{rnd}_{g_3^x}$, $s_2 = \mathtt{rnd}_{o^x}$;

6. Π_5 is computed via a proof of inequality of two logarithms (see Protocol 8). Let $(\alpha_1, \beta_1, \gamma_1) = \mathcal{E}_{\mathtt{pk_{TT}}}^{\mathtt{rnd}_A}[A]$ and $t \in \mathbb{Z}_p$, then the parameters are:

   (a) $\ell_2 = \mathtt{pk_{TT}}$, $h_2 = \gamma_1$, $\mu_1 = t \cdot \mathtt{rnd}_A$, $\mu_2 = t$;

   step 2b must be performed for both g_1 and g_2 so we set:

   (a) for the first run: $\ell_1 = g_1$, $h_1 = \alpha_1$,

   (b) for the second run: $\ell_1 = g_2$, $h_1 = \beta_1$.

7. $\Pi_{6,1}$ is a proof that:

$$\frac{\mathcal{E}_{\mathtt{pk_{TT}}}^{\mathtt{rnd}_{\mathtt{Ls},0}}[s_{\mathtt{Ls}}] \cdot \prod_{i=1}^{n_{\mathtt{Ls}}} \mathcal{E}_{\mathtt{pk_{TT}}}^{\mathtt{rnd}_i}[\mathtt{Ls}_i]^i}{\mathcal{E}_{\mathtt{pk_{TT}}}^{\mathtt{rnd}_{\mathtt{Ls},1}}\left[(s_{\mathtt{Ls}} + \mathtt{Ls}) \mod 100\right]}$$

   is the encryption of either 0 or 100;

8. $\Pi_{6,2}$ is a proof that:

$$\frac{\mathcal{E}_{\mathtt{pk_{TT}}}^{\mathtt{rnd_{Ca},0}}[s_{\mathtt{Ca}}] \cdot \prod_{i=1}^{n_{\mathtt{Ls}}} \prod_{j=1}^{n_{\mathtt{Ca},i}} \mathcal{E}_{\mathtt{pk_{TT}}}^{\mathtt{rnd}_{i,j}}[\mathtt{Ca}_{i,j}]^j}{\mathcal{E}_{\mathtt{pk_{TT}}}^{\mathtt{rnd_{Ca},1}}\left[(s_{\mathtt{Ca}} + \mathtt{Ca}) \mod 100\right]}$$

is the encryption of either 0 or 100;

In order to tie together every piece of the encrypted ballot, the NIZKPs should be slightly modified by including in the hash computations of the challenges also the election ID, the encrypted votes, and all the commitment values of the other NIZKPs.

3.8.4 Voting: Casting and Confirmation

Once *Vote App* has completed the computation of the ballot $\mathcal{B}$, it is sent to the Ballot Boxes (see Section 5.3.1.6):

1. if the sending has been authorized, each trusted BB checks the validity of the ballot $\mathcal{B}$ by verifying all its NIZKPs Π;
2. if the ballot is valid, each trusted BB_i sends to the WBB:
   - the ballot digest $\mathcal{H}(\mathcal{B})$;
   - $\mathtt{id}_{\mathcal{B}} = \mathcal{H}\left(\mathtt{EL_{ID}}, E_{\mathtt{pk_{TT}}}^{\mathtt{rnd}_A}[A], E_{\mathtt{pk_{TT}}}^{\mathtt{rnd}_{A^r}}[A^r]\right)$, from which the visual encoding `BallotEmoji` can be computed;
   - $\mathcal{H}(E_{\mathtt{pk_{TT}}}^{\mathtt{rnd}_{o^x}}[o^x])$, from which the visual encoding `PublicPINEmoji` can be computed;
   - $\mathcal{E}_{\mathtt{pk_{TT}}}[2^i]$, the encryption of BB_i's identifier i, alongside a NIZKP that certifies that it is indeed the encryption of 2^i (see Protocol 9);
3. each trusted BB_i stores $\mathcal{B}$ indexed with $\mathtt{id}_{\mathcal{B}}$, waiting for its confirmation;
4. when the WBB receives the data from BB_i, it checks the NIZKP and if all is correct publishes $(\mathcal{H}(\mathcal{B}), \mathtt{id}_{\mathcal{B}}, \mathcal{H}(E_{\mathtt{pk_{TT}}}^{\mathtt{rnd}_{o^x}}[o^x]), \mathcal{E}_{\mathtt{pk_{TT}}}[2^i])$ with a timestamp that marks the moment in which this data has first been received;
5. after a pre-determined buffer period the WBB publishes also the product $\prod_{j \in \hat{B}} \mathcal{E}_{\mathtt{pk_{TT}}}[2^j]$ where $\hat{B}$ is the set of identifiers of the BBs that have sent this same ballot after BB_i, if no other BB has sent it, the WBB publishes the special symbol $\perp$ instead (this signals that at least one BB has failed);

6. once a trusted BB can confirm that the data has been published on the WBB, it sends to *Vote App* a confirmation of successful registration;

7. *Vote App* independently checks that the correct data has been published on the WBB, and if all checks out it displays to $\mathcal{V}$:

   - the three visual encodings `BallotEmoji`, `PrivatePINEmoji` and `PublicPINEmoji`[21];
   - the encoding of the ballot digest $\mathcal{H}(\mathcal{B})$ (see Appendix A.3.6);
   - a message that suggests to check the publication on the WBB, with a warning that the vote has to be confirmed for it to be valid;

8. $\mathcal{V}$ should copy the encoding of $\mathcal{H}(\mathcal{B})$, then proceed with the vote confirmation by pressing the designated button on *Vote App*;

9. *Vote App* shows to the voter:

   - the index of the selected electoral list `Ls`;
   - the *electoral list control code* $s_{\texttt{Ls}}$[22];
   - the *electoral list control sum* $s_{\texttt{Ls}} + \texttt{Ls}$;
   - the index of the selected candidate `Ca`;
   - the *candidate control code* $s_{\texttt{Ca}}$;
   - the *candidate control sum* $s_{\texttt{Ca}} + \texttt{Ca}$;
   - instructions for $\mathcal{V}$ to check that the sums are correct and then to select randomly one between $\{s_{\texttt{Ls}}, (s_{\texttt{Ls}} + \texttt{Ls})\}$ and one between $\{s_{\texttt{Ca}}, (s_{\texttt{Ca}} + \texttt{Ca})\}$;

10. $\mathcal{V}$ should check the correctness of the sums displayed, then perform the two random selections, preferably tossing a coin twice[23];

[21] If both `PrivatePINEmoji` and `PublicPINEmoji` are correct, voters are assured that their device has processed the PIN they inserted, but further checks can be made (see Item 2 of Section 3.8.5).

[22] It is fundamental for the cast-as-intended values to be shown after the ballot has been sent. Otherwise, a malicious voting device could change the vote, compute a different cast-as-intended value that still satisfies the NIZKP, and the voter would never notice that their vote has been changed.

[23] Note that if the sum is correct and if the voter sees the chosen values on the WBB after the voting phase, then this satisfies the cast-as-intended requirement.

11. let $b_{\texttt{Ls}}, b_{\texttt{Ca}} \in \{0,1\}$ be the random choices made by $\mathcal{V}$ (0 indicates that the code has been selected, 1 indicates that the sum has been selected), *Vote App* sends anonymously to the trusted BBs the *cast-as-intended disclosure* as the tuple:

$$(\texttt{EL}_{\texttt{ID}}, E_{\texttt{pk}_{\texttt{TT}}}^{\texttt{rnd}_A}[A], E_{\texttt{pk}_{\texttt{TT}}}^{\texttt{rnd}_{A^r}}[A^r], b_{\texttt{Ls}}, \texttt{rnd}_{\texttt{Ls},b_{\texttt{Ls}}}, b_{\texttt{Ca}}, \texttt{rnd}_{\texttt{Ls},b_{\texttt{Ca}}});$$

12. *Vote App* shows to the voter the time (hour, minutes and seconds) when $\mathcal{V}$ has confirmed the choice of *cast-as-intended values*[24];

13. each trusted BB uses the first three values of the tuple to compute $\texttt{id}_{\mathcal{B}}$ and identify the pertinent ballot, then tries to decrypt the selected *cast-as-intended values*:

    (a) let $(\alpha_{\texttt{Ls}}, \beta_{\texttt{Ls}}, \gamma_{\texttt{Ls}}) = \mathcal{E}_{\texttt{pk}_{\texttt{TT}}}^{\texttt{rnd}_{\texttt{Ls},b_{\texttt{Ls}}}}[(s_{\texttt{Ls}}+b_{\texttt{Ls}}\cdot\texttt{Ls}) \mod 100]$ be the encryption of the *cast-as-intended value* for the electoral list indicated by $b_{\texttt{Ls}}$, extracted from $\mathcal{B}$:

        i. the trusted BB checks that:

$$g_1^{\texttt{rnd}_{\texttt{Ls},b_{\texttt{Ls}}}} \stackrel{?}{=} \alpha_{\texttt{Ls}},$$
$$g_2^{\texttt{rnd}_{\texttt{Ls},b_{\texttt{Ls}}}} \stackrel{?}{=} \beta_{\texttt{Ls}};$$

        ii. if one of the equivalences does not hold, the trusted BB rejects the *cast-as-intended disclosure* as invalid;

        iii. otherwise the trusted BB computes:

$$(s_{\texttt{Ls}} + b_{\texttt{Ls}} \cdot \texttt{Ls}) \mod 100 = \log_{g_0}\left(\gamma_{\texttt{Ls}} \cdot \texttt{pk}_{\texttt{TT}}^{-\texttt{rnd}_{\texttt{Ls},b_{\texttt{Ls}}}}\right);$$ [25]

    (b) let $(\alpha_{\texttt{Ca}}, \beta_{\texttt{Ca}}, \gamma_{\texttt{Ca}}) = \mathcal{E}_{\texttt{pk}_{\texttt{TT}}}^{\texttt{rnd}_{\texttt{Ca},b_{\texttt{Ca}}}}[(s_{\texttt{Ca}} + b_{\texttt{Ca}} \cdot \texttt{Ca}) \mod 100]$ be the encryption of the *cast-as-intended value* for the candidate indicated by $b_{\texttt{Ca}}$, extracted from $\mathcal{B}$:

        i. the trusted BB checks that:

$$g_1^{\texttt{rnd}_{\texttt{Ca},b_{\texttt{Ca}}}} \stackrel{?}{=} \alpha_{\texttt{Ca}},$$
$$g_2^{\texttt{rnd}_{\texttt{Ca},b_{\texttt{Ca}}}} \stackrel{?}{=} \beta_{\texttt{Ca}};$$

[24] If this time does not correspond to the real one, it is a symptom that something is wrong with the device, so the voter should repeat the voting process, preferably with another device.

[25] Note that the exponent should be an integer < 100, so the discrete logarithm can be easily found with a pre-computed table.

ii. if one of the equivalences does not hold the trusted BB rejects the *cast-as-intended disclosure* as invalid;

iii. otherwise the trusted BB computes:

$$(s_{\texttt{Ca}} + b_{\texttt{Ca}} \cdot \texttt{Ca}) \mod 100 = \log_{g_0}\left(\gamma_{\texttt{Ca}} \cdot \texttt{pk}_{\texttt{TT}}^{-\texttt{rnd}_{\texttt{Ca},b_{\texttt{Ca}}}}\right);$$ [26]

14. if the trusted BB has successfully found the values $(s_{\texttt{Ls}} + b_{\texttt{Ls}} \cdot \texttt{Ls}) \mod 100$, $(s_{\texttt{Ca}} + b_{\texttt{Ca}} \cdot \texttt{Ca}) \mod 100$, it saves them and $b_{\texttt{Ls}}$, $b_{\texttt{Ca}}$, $\texttt{rnd}_{\texttt{Ls},b_{\texttt{Ls}}}$, $\texttt{rnd}_{\texttt{Ca},b_{\texttt{Ca}}}$, alongside $\mathcal{B}$ and sends to the WBB:

$$\begin{aligned}&\big(\mathcal{H}(\mathcal{B}),\\ &b_{\texttt{Ls}}, \texttt{rnd}_{\texttt{Ls},b_{\texttt{Ls}}}, (s_{\texttt{Ls}} + b_{\texttt{Ls}} \cdot \texttt{Ls}) \mod 100,\\ &b_{\texttt{Ca}}, \texttt{rnd}_{\texttt{Ca},b_{\texttt{Ca}}}, (s_{\texttt{Ca}} + b_{\texttt{Ca}} \cdot \texttt{Ca}) \mod 100\big);\end{aligned}$$

15. the WBB publishes this data linked to the ballot hash previously published, if multiple BBs send the same data (as they should) the values are published just once, if a BB sends different data then this data is also published;

16. once a trusted BB can confirm that the data has been published on the WBB, it sends to *Vote App* a confirmation of successful registration;

17. *Vote App* independently checks that the correct data has been published on the WBB, and if all check out it displays to $\mathcal{V}$ a confirmation message and a button that brings $\mathcal{V}$ to the "manual verification" page of *Vote App*.

3.8.5 Voting: Verification

The voter $\mathcal{V}$ can manually perform the following verification checks:

1. correct registration of the vote:

   (a) with a web browser, $\mathcal{V}$ searches on the WBB for the ballot digest $\mathcal{H}(\mathcal{B})$ (which *Vote App* has shown in Step 7 of Section 3.8.4, and which is accessible in the manual verification functionalities on *Vote App*);

   (b) on the WBB $\mathcal{V}$ should find associated to $\mathcal{H}(\mathcal{B})$:

[26] Note that the exponent should be an integer < 100, so the discrete logarithm can be easily found with a pre-computed table.

- a timestamp coherent with when $\mathcal{V}$ has actually voted;
- $\texttt{id}_{\mathcal{B}}$, which the browser should display as the visual encoding `BallotEmoji`;
- $\mathcal{H}(E^{\texttt{rnd}_{o^x}}_{\texttt{pk}_{\texttt{TT}}}[o^x])$, which the browser should display as the visual encoding `PublicPINEmoji`;
- the choice of *cast-as-intended value* for the electoral list $b_{\texttt{Ls}}$ and the value itself $(s_{\texttt{Ls}}+b_{\texttt{Ls}}\cdot\texttt{Ls}) \mod 100$, which the browser should display in a clear, human-readable form;
- the choice of *cast-as-intended value* for the candidate $b_{\texttt{Ca}}$ and the value itself $(s_{\texttt{Ca}} + b_{\texttt{Ca}} \cdot \texttt{Ca}) \mod 100$, which the browser should display in a clear, human-readable form;

(c) $\mathcal{V}$ should check that:

- the values displayed on the WBB correspond to the ones displayed on *Vote App*[27];
- the control values are the ones that $\mathcal{V}$ chose in Step 10 of Section 3.8.4;
- the timestamp on the WBB is prior to the time of confirmation of the *cast-as-intended values* showed by *Vote App*;

(d) $\mathcal{V}$ can check that the ballot has been sent to the WBB by at least two BBs by verifying that the WBB has not published the special symbol $\perp$ which signals that just one BB has sent the ballot.

2. correctness of `BallotEmoji`, `PrivatePINEmoji`, `PublicPINEmoji`:

(a) preferably using another device, $\mathcal{V}$ launches a certified VS;

(b) $\mathcal{V}$ enters:

- their pseudonymous identifier $v_{\texttt{id}}$ (which the voter should have safely backed up at the end of registration, see last step of Section 3.6.1, and which is also accessible through *Vote App* settings);
- the encoding of $\texttt{rnd}_{\texttt{seed}}$ which *Vote App* displays in the dedicated "manual verification" page;
- the encoding of o^x, which *Vote App* displays in the dedicated "manual verification" page;
- if there are multiple simultaneous elections, $\mathcal{V}$ should also select the correct one;

[27] For the *cast-as-intended values*, $\mathcal{V}$ should check that the last two digits are identical to the value displayed on *Vote App*, i.e. that they are congruent modulo 100.

(c) the VS uses $v_{\mathtt{id}}$ to fetch from the WBB (or a local copy) the public ACC values $A_{v_{\mathtt{id}}}, r_{v_{\mathtt{id}}}$;

(d) the VS computes the scalars:

$$\begin{aligned}\mathtt{rnd}_A &= \mathcal{H}_p(\mathtt{rnd}_{\mathtt{seed}}||\mathtt{DS1}),\\ \mathtt{rnd}_{A^r} &= \mathcal{H}_p(\mathtt{rnd}_{\mathtt{seed}}||\mathtt{DS2}),\\ \mathtt{rnd}_{o^x} &= \mathcal{H}_p(\mathtt{rnd}_{\mathtt{seed}}||\mathtt{DS3}),\end{aligned}$$

(e) the VS can then compute:

$$\begin{aligned}&\mathtt{id}_{\mathcal{B}} = \mathcal{H}\left(\mathtt{EL}_{\mathtt{ID}}, E_{\mathtt{pk}_{\mathtt{TT}}}^{\mathtt{rnd}_A}\left[A_{v_{\mathtt{id}}}\right], E_{\mathtt{pk}_{\mathtt{TT}}}^{\mathtt{rnd}_{A^r}}\left[A_{v_{\mathtt{id}}}^{r_{v_{\mathtt{id}}}}\right]\right),\\ &\mathcal{H}(o^x),\\ &\mathcal{H}\left(E_{\mathtt{pk}_{\mathtt{TT}}}^{\mathtt{rnd}_{o^x}}\left[o^x\right]\right),\end{aligned}$$

(f) the VS displays the visual encodings which represent these values;

(g) $\mathcal{V}$ should check that these visual encodings correspond to the ones displayed by *Vote App* in the "manual verification" page;

(h) $\mathcal{V}$ should check that the first and third visual encodings correspond to the ones showed by the WBB alongside the ballot.

Optionally, the value of $v_{\mathtt{id}}$ can be included in the encoding of `seed`, so that $\mathcal{V}$ does not have to type it in the VS. Instead the VS should show the extracted $v_{\mathtt{id}}$ and $\mathcal{V}$ should check that it is correct.

3.9 Tallying

To compute the election results, the encrypted ballots are gathered from the BBs and processed as follows:

1. the ER publishes on the WBB the list of the pseudonymous identifiers $v_{\mathtt{id}}$ associated to registered eligible voters, ignoring the ones associated to revoked ACCs;

2. each BB discards all the ballots for which no valid *cast-as-intended disclosure* has been received, then sends to the WBB the remaining ballots;

3. the WBB publishes each ballot $\mathcal{B}$ for which a digest $\mathcal{H}(\mathcal{B})$ has been published during the election period, linking them (multiple copies and other ballots are discarded);

4. for each digest $\mathcal{H}(\mathcal{B})$ published on the WBB for which a ballot $\mathcal{B}$ has not been published, the TTs collaboratively decrypt the encrypted identifiers of the BBs who sent the digest (note that $\prod_{j\in\hat{B}} \mathcal{E}_{\mathtt{pk}_{\mathtt{TT}}}[2^j] = \mathcal{E}_{\mathtt{pk}_{\mathtt{TT}}}[\sum_{j\in\hat{B}} 2^j]$, and the binary representation of this sum reveals which BBs have sent the digest);

5. for each ballot, the election IDs, the proofs, the *cast-as-intended disclosure* are checked and the invalid ballots are discarded, while the remaining are trimmed to the tuple:

$$\left(\bar{\mathcal{E}}, E^{\mathtt{rnd}_A}_{\mathtt{pk}_{\mathtt{TT}}}[A], E^{\mathtt{rnd}_{A^r}}_{\mathtt{pk}_{\mathtt{TT}}}[A^r], E^{\mathtt{rnd}_{g_3^x}}_{\mathtt{pk}_{\mathtt{TT}}}[g_3^x], E^{\mathtt{rnd}_{o^x}}_{\mathtt{pk}_{\mathtt{TT}}}[o^x]\right);$$

6. each TT_i selects uniformly at random $\zeta_i^* \in Z_p$ and partakes in Protocol 2 with $\hat{s}_i = \zeta_i^*$ to retrieve the share (i, z_i^*) and the public value $g_0^{z_i^*}$;

7. for each ballot, each TT_i elevates the last component to z_i^* and broadcasts the resulting ciphertext alongside a NIZKP that the exponent used is the same as $g_0^{z_i^*}$, this proof is extremely similar to the one of Protocol 7;

8. with $t_{\mathtt{TT}}$ shares, it is possible to interpolate and compute $E_{\mathtt{pk}_{\mathtt{TT}}}[o^x]^{z^*}$;

9. then, these values are (threshold) verifiably decrypted by the TTs, so that each ballot can be associated to a *ballot fingerprint* o^{xz^*};

10. the ballot fingerprint of every ballot is confronted to the others to delete duplicates: only the most recent ballot for a given value o^{xz^*} is kept;

11. the remaining ballots are trimmed to the tuple:

$$\left(\bar{\mathcal{E}}, E^{\mathtt{rnd}_A}_{\mathtt{pk}_{\mathtt{TT}}}[A], E^{\mathtt{rnd}_{A^r}}_{\mathtt{pk}_{\mathtt{TT}}}[A^r], E^{\mathtt{rnd}_{g_3^x}}_{\mathtt{pk}_{\mathtt{TT}}}[g_3^x]\right);$$

12. the TTs use the protocol of Groth [69] to verifiably mix and re-encrypt the ballots;

13. the TTs publish the mixed ballots with the proofs;

14. each RT_i, after checking the proofs, retrieves the ballots from the WBB;

15. each RT_i computes $E_{\mathtt{pk}_{\mathtt{TT}}}[A]^{y_i}$ and broadcasts it alongside a NIZKP that the exponent used is the same as $\log_{g_3}(\mathtt{pk}_{\mathtt{RT}_i})$, where $\mathtt{pk}_{\mathtt{RT}_i} = g_3^{y_i}$, (just like in Step 7);

16. the RTs consider only shares with valid NIZKPs, moreover it is checked that the interpolation of the $\mathtt{pk}_{\mathtt{RT}_i}$ gives $\mathtt{pk}_{\mathtt{RT}}$ before continuing;

17. the RTs check the proofs and, with at least $t_{\mathtt{RT}}$ shares, each RT interpolates and compute $E_{\mathtt{pk}_{\mathtt{TT}}}[A]^y$;

18. this value is used to compute:

$$E = E_{\mathtt{pk}_{\mathtt{TT}}}[A]^y \cdot E_{\mathtt{pk}_{\mathtt{TT}}}[A^r] \cdot E_{\mathtt{pk}_{\mathtt{TT}}}[g_3^x]^{-1} \cdot E_{\mathtt{pk}_{\mathtt{TT}}}[g_1]^{-1} = E_{\mathtt{pk}_{\mathtt{TT}}}\left[A^{y+r}\left(g_1 g_3^x\right)^{-1}\right];$$

19. for each ACC, the RTs send the credential control element E to the WBB;

20. each TT_i selects uniformly at random $\zeta_i \in \mathbb{Z}_p$ and partakes in Protocol 2 with $\hat{s}_i = \zeta_i$ to retrieve the share (i, z_i), then computes and broadcasts the couple (i, E^{z_i}) alongside a NIZKP that all three components of E have been elevated to the same exponent[28];

21. with at least $t_{\mathtt{TT}}$ shares, it is possible to interpolate and compute E^z, which, if $A, r, x \overset{\mathtt{valid}}{=} x$ are a valid ACC, is:

$$E_{\mathtt{pk}_{\mathtt{TT}}}\left[A^{y+r}\left(g_1 g_3^x\right)^{-1}\right]^z = E_{\mathtt{pk}_{\mathtt{TT}}}\left[\left(g_1 g_3^x\right)^{\frac{1}{y+r}\cdot(y+r)} \cdot \left(g_1 g_3^x\right)^{-1}\right]^z = E_{\mathtt{pk}_{\mathtt{TT}}}[1_{\mathbb{G}}]^z;$$

22. the TTs collaboratively decrypt E^z, if the decrypted value is not $1_{\mathbb{G}}$ then the ballot is discarded, otherwise it is trimmed to $\left(\bar{\mathcal{E}}, E_{\mathtt{pk}_{\mathtt{TT}}}[A]\right)$;

23. the TTs build, using the data retrieved from the WBB, the list $\mathcal{L}_{\mathtt{ACC}}$ that contains all the $E_{\mathtt{pk}_{\mathtt{TT}}}[A]$ pertaining to legitimate (and non-revoked) voters, and they verifiably mix and re-encrypt them;

24. each TT_i selects uniformly at random $\zeta_i' \in Z_p$ and partakes in Protocol 2 with $\hat{s}_i = \zeta_i'$ to retrieve the share (i, z_i') and the public value $g_0^{z_i'}$;

25. for each $E_{\mathtt{pk}_{\mathtt{TT}}}[A]$ in $\mathcal{L}_{\mathtt{ACC}}$ and each $E_{\mathtt{pk}_{\mathtt{TT}}}[A]$ contained in the remaining ballots, each TT_i elevates it to z_i' and broadcasts the resulting ciphertext alongside a NIZKP that the exponent used is the same as $g_0^{z_i'}$ (just like in Step 7);

26. as before these values are interpolated and (threshold) verifiably decrypted by the TTs, so that each remaining ballot can be associated to a *fingerprint* $A^{z'}$, similarly the fingerprints corresponding to the elements of $\mathcal{L}_{\mathtt{ACC}}$ are assembled in the list $\mathcal{L}'_{\mathtt{ACC}}$;

[28] Again, this proof is extremely similar to the one of Protocol 7.

27. every ballot whose fingerprint cannot be found in $\mathcal{L}'_{\texttt{ACC}}$ is discarded, the others are trimmed to just $\bar{\mathcal{E}}$;

28. all remaining ballots are reduced to a single composite list of ciphertexts exploiting the homomorphic properties of the encryption scheme (see Section 2.9.1):

$$\hat{\mathcal{E}} = \left(\prod \mathcal{E}_{\texttt{pk}_{\texttt{TT}}}[\texttt{Ls}_i], \left(\prod \mathcal{E}_{\texttt{pk}_{\texttt{TT}}}[\texttt{Ca}_{i,j}]\right)_{j=0}^{n_{\texttt{Ca},i}}\right)_{i=0}^{n_{\texttt{Ls}}};$$

29. finally each component of $\hat{\mathcal{E}}$ is (threshold) verifiably decrypted[29] by the TTs, obtaining the final tally with the total number of preferences given to each list and each candidate;

30. the results of each intermediate computation (including all NIZKPs) are published on the bulletin board to allow everyone to check the correctness of the computations and final results.

3.10 Universal Verification

Everyone can check the correctness of the election as follows:

1. reconstruct the filtered list of ballots:
   (a) fetch from WBB the timestamped ballot digests and the *cast-as-intended disclosure* published during the voting phase;
   (b) fetch from the WBB the ballots published at the beginning of tallying;
   (c) discard the ballots which do not have a published timestamped digest;
   (d) discard the ballots which do not have a published valid *cast-as-intended disclosure* (see step 13 of Section 3.8.4);
   (e) discard the ballots with invalid NIZKPs;
   (f) fetch from the WBB the verifiable ballot fingerprints (see step 9 of Section 3.9) and check their correctness;

[29] Note that each encrypted value is upper-bounded by the total number of voters, which is small enough to allow the discrete logarithm computation using a pre-computed table.

(g) keep only the most recent ballot for every given ballot fingerprint o^{xz^*} (see step 10 of Section 3.9);

2. check that the filtered ballots have been correctly mixed, by fetching from the WBB the verifiable mix-re-encryption of the ballots published by the TTs during the tallying, and by verifying the NIZKPs included [69];

3. reconstruct the list of valid ballots:

   (a) fetch from the WBB the values E^z (and the corresponding NIZKPs) published by the RTs during the tallying;

   (b) fetch from the WBB the verifiable decryption of the E^z published by the TTs during the tallying;

   (c) verify the NIZKPs and discard the invalid ballots, i.e. the ones associated to a decrypted value $\neq 1_{\mathbb{G}}$ (see step 22 of Section 3.9);

4. check that the valid public credentials have been correctly mixed:

   (a) fetch from the WBB the verifiable mix-re-encryption of the public credentials published by the TTs during the tallying;

   (b) fetch from the WBB the list of valid pseudonymous ids published by the ER at the start of tallying;

   (c) reconstruct the list $\mathcal{L}_{\texttt{ACC}}$ as in step 23 of Section 3.9;

   (d) verify that the mixed public credentials are indeed a mixed-re-encryption of $\mathcal{L}_{\texttt{ACC}}$ by verifying the NIZKPs included [69];

5. reconstruct the list of legitimate ballots:

   (a) fetch from the WBB the fingerprints associated to the valid ballots, which are published by the TTs during the tally;

   (b) fetch from the WBB the list of fingerprints $\mathcal{L}'_{\texttt{ACC}}$ associated to the valid public ACCs, which are published by the TTs during the tally;

   (c) verify the NIZKPs associated to the fingerprints;

   (d) discard the illegitimate ballots, i.e. those whose fingerprint is not in $\mathcal{L}'_{\texttt{ACC}}$ (see step27 of Section 3.9);

6. check the final tally:

   (a) fetch from the WBB the decrypted tally published by the TTs at the end of the tally;

(b) homomorphically sum all the legitimate ballots;

(c) check that the published tally is indeed the verifiable decryption of the sum of the legitimate ballots, by verifying the associated NIZKPs.

3.11 Optimization for a Referendum

In the case of a Referendum, where a voter has simply to approve or reject a single proposal, the ballot can be simplified.

In this setting, a voter $\mathcal{V}$ can express the following preferences:

- leave a blank ballot ($\texttt{Ls}_0$);
- vote for the approval of the proposal ($\texttt{Ls}_1$);
- vote for the rejection of the proposal ($\texttt{Ls}_2$).

A voter's preference will be encoded as $(\texttt{Ls}_i)_{i=0}^2$, where each $\texttt{Ls}_i$ is equal to 1 if the voter wants to express the corresponding preference, equal to 0 otherwise.

Therefore a valid preference is list of integers $(\texttt{Ls}_i)_{i=0}^2$ that satisfies the following conditions:

1. $\texttt{Ls}_i \in \{0, 1\}$ for every $i \in \{0, 1, 2\}$;
2. $\sum_{i=0}^{2} \texttt{Ls}_i = 1$.

For the *cast-as-intended values*, we need only one code. *Vote App* computes the encryption of the *cast-as-intended values* as:

1. let $\texttt{Ls} = \sum_{i=0}^{2} i \cdot \texttt{Ls}_i$;
2. *Vote App* generates uniformly at random one value $0 \leq s_{\texttt{Ls}} < 10$;
3. *Vote App* generates uniformly at random $\texttt{rnd}_{\texttt{Ls},1}, \texttt{rnd}_{\texttt{Ls},0} \in \mathbb{Z}_p$ and computes the following encryptions:

$$\mathcal{E}_{\texttt{pk}_{\texttt{TT}}}^{\texttt{rnd}_{\texttt{Ls},0}}[s_{\texttt{Ls}}], \quad \mathcal{E}_{\texttt{pk}_{\texttt{TT}}}^{\texttt{rnd}_{\texttt{Ls},1}}[(s_{\texttt{Ls}} + \texttt{Ls}) \mod 10].$$

For the ballot NIZKPs, we can simplify Π_1 and Π_6.

1. Π_1: for each list $\texttt{Ls}_i$ (where $0 \leq i \leq 2$), we verify the following:
   (a) $\Pi_{1,0,i}$ ensures that the selection $\texttt{Ls}_i$ is binary (i.e., $\texttt{Ls}_i \in \{0, 1\}$), confirming that an option receives at most one vote;
   (b) $\Pi_{1,1}$ can be reduced to proving only that $\sum_{i=0}^{2} \texttt{Ls}_i = 1$;
2. Π_6 is composed only by $\Pi_{6,1}$.

These Π_i's are constructed as in Section 3.8.3.

3.12 Trusted Authorities

The default settings consider all RTs and BBs as trusted, but voters can change this setting. Allowing voters to choose the authorities they trust improves transparency and builds confidence in the voting process by giving them more control. This approach recognizes that not all voters trust the same authorities and helps address concerns about possible bias or collusion by letting them exclude those they distrust. It also makes the system more flexible, especially in different jurisdictions where trust in certain authorities may vary.

The voter $\mathcal{V}$ can manage which authorities to trust from *Vote App* settings:

- $\mathcal{V}$ can select between $t_{\texttt{RT}}$ and $n_{\texttt{RT}}$ RTs which will be able to distinguish a *PIN Re-Sending* request (see Section 3.7.2) from a *Ruse PIN* request (see Section 3.7.3): these will be referred as *trusted RTs*;
- $\mathcal{V}$ can select between 2 and $n_{\texttt{BB}}$ BBs which will receive the votes sent from *Vote App* (see Section 3.8): these will be referred as *trusted BBs*;

Note that to make effective a change of the trusted RTs, $\mathcal{V}$ has to make this change before the arrival of the PIN or perform a subsequent *PIN Re-Sending* request (see Section 3.7.2).

3.13 Implementation and Scalability

In the initial version of *Vote App*, the cryptographic primitives were implemented in Python, but we have now transitioned to Rust for improved

efficiency and security. The main mathematical structure underlying our implementation is elliptic curves (see Section 2.6), and we use EdDSA for digital signatures.

We have performed scalability analysis on the Rust version of *Vote App*. This evaluation was done using the test sets summarized in Table 3.1. In all cases, the number of tabulation tellers, registration tellers, and ballot boxes was set to one.

Table 3.1: Test sets used to conduct the scalability analysis.

	Test Set 1	**Test Set 2**	**Test Set 3**	**Test Set 4**
Total number of voters	10	100	10	100
Total number of lists	2	2	9	9
Total number of candidates	4	4	85	85
Total number of confirmed ballots	10	100	10	100

All data were gathered using a laptop DELL Latitude 7450, CPU Ultra 7 165H 16 core, 32GB ram. The results of the analysis are presented in Table 3.2.

Table 3.2: Execution times (in seconds) for different phases across test sets.

Phase	**Test Set 1**	**Test Set 2**	**Test Set 3**	**Test Set 4**
Setup	0.7977	0.5191	0.8074	0.5239
Enrollment	0.9661	9.7157	0.9757	10.1227
Voting	2.5701	2.5383	16.1836	16.5724
Tallying	28.8749	297.0496	162.7687	1646.4811
Tally Verification	6.1276	59.0586	42.2924	382.8495

Chapter 4

Security of the Vote App Protocol

This chapter formalizes the definition of coercion resistance in Section 4.2 and discusses the proof of security of *Vote App*'s protocol in Section 4.3.

4.1 Definition of E-Voting Protocol

To perform the security analysis, we need to formalize a voting protocol. This formalization is deliberately broad, since many existing voting protocols do not fit it exactly. Rather than aiming for an exhaustive classification, this framework provides the necessary structure for the security proofs presented in this chapter.

Definition 18 (Voting Protocol)**.** *A* voting protocol *is defined by the tuple*

$$(\mathsf{Setup}, \mathsf{CredGen}, \mathsf{Vote}, \mathsf{Tally})$$

where:

- $\mathsf{Setup}(\lambda, t_{RT}, n_{RT}, t_{TT}, n_{TT})$*, is a multi party protocol run by the authorities. On input a security parameters* λ*, the number of authorities* (n_{RT}, n_{TT}) *and the thresholds* (t_{RT}, t_{TT}) *it outputs public parameters* $\mathtt{pp}$*, three public keys* $pk_{ACC}, pk_{RT}, pk_{TT}$ *and shares of the private keys* sk_{ACC_i}*,* sk_{RT_i}*,* sk_{TT_i}*, one for each authority.*
- $\mathsf{CredGen}()$ *is a multi party protocol run by the registrars, which takes no input and generates the private credentials shares* $\mathtt{sp}_{\mathtt{ACC}}^{\mathtt{valid}}$ *of the voters, one for each registrar, and outputs the public part* $\mathtt{pp}_{\mathtt{ACC}}$*.*

- $\mathsf{Vote}(\nu, \mathtt{sp}_{\mathtt{ACC},v_{id}})$ *is a protocol which takes a voting choice* ν*, a credential* $\mathtt{sp}_{\mathtt{ACC},v_{id}}$ *and returns a ballot.*
- $\mathsf{Tally}(\mathbb{B}, \mathtt{pp}_{\mathtt{ACC}}, sk_{RT_i}, sk_{TT_i})$ *is a multi party protocol run by at least* (t_{RT}, t_{TT}) *authorities. On input all the cast ballots* $\mathbb{B}$*, the public credentials* $\mathtt{pp}_{\mathtt{ACC}}$ *and at least* (t_{RT}, t_{TT}) *shares of the private keys, it outputs the election result.*

Moreover we define RuseCred:

- $\mathsf{RuseCred}(\overset{\mathtt{ruse}}{\mathtt{PIN}})$ is a protocol run by the voting application and, which take in input the ruse pin $\overset{\mathtt{ruse}}{\mathtt{PIN}}$ of the voter and return a ruse private credential $\overset{\mathtt{ruse}}{\mathtt{sp}}_{\mathtt{ACC}}$.

4.2 Coercion-Resistance

Informally, a voting protocol is coercion-resistant if, even if an adversary can influence the voter before or during the voting process (e.g., by demanding that they vote in a certain way, providing them with special credentials, or monitoring their actions), the voter can still cast their vote freely without the coercer being able to verify it. This suggests that even under coercion, a voter possesses the capacity to cast their intended vote while rendering it indistinguishable from a coerced vote.

4.2.1 Definition of Coercion-Resistance

The most common definition of coercion resistance is by Juels, Catalano and Jakobsson [83]. We say that a voting protocol is coercion-resistant if and only if voters are able to generate some *fake credential* that could be given to an attacker (i.e., a coercer) trying to influence their vote. Votes cast with fake credentials are discarded in the tally phase, thus preserving the ability of the voter to vote with their valid ones.

Let $n_{\mathtt{A}}$ be the number of corrupt voters, $n_{\mathtt{V}}$ be the number of voters and $n_{\mathtt{Ls}}$ be the total number of voting choices $[\mathtt{Ls}_1, \ldots, \mathtt{Ls}_{n_{\mathtt{Ls}}}]$.

The definition of coercion-resistance (see Definition 19 and [83]) is based on a game between an adversary $\mathbb{A}$ and a voter who is being coerced. The adversary $\mathbb{A}$ aims to determine which of the following two scenarios the voter has chosen during the voting protocol:

- the voter has revealed a fake credential and cast a ballot;
- the voter has disclosed a valid voting credential and abstained from voting.

If the adversary has perfect knowledge about the intentions of all voters, then coercion is unavoidable. Therefore, coercion-resistance requires that there be some noise or statistical uncertainty in the adversary's view of voting patterns which is natural to expect in a real-world election, since an adversary can obtain only fragmentary knowledge about the likely behavior of voters. To characterize the view of the adversary, we consider a distribution $\mathcal{D}([1,\ldots,n_{\mathtt{V}}] \setminus \mathcal{V}_{n_{\mathtt{A}}}, n_{\mathtt{Ls}})$ of sequences of pairs of the form $(v_{\mathtt{id}}, \mathtt{Ls})$ where $\mathtt{Ls}$ represents a valid voting choice and $v_{\mathtt{id}}$ represents either a free voter $v_{\mathtt{id}} \in [1,\ldots,n_{\mathtt{V}}] \setminus \mathcal{V}_{n_{\mathtt{A}}}$ or a voter with a ruse credential.

In order to prove that the voting protocol described in Chapter 3 satisfies this property, one must prove that $\mathbb{A}$ can correctly identify the voter's behavior with a probability only negligibly better than the simulator $\mathbb{S}$ interacting with an ideal election system. The simulator $\mathbb{S}$ is passive and only receives the final tally of votes from honest voters, along with the count of ballots discarded due to invalid credentials.

The security definition for coercion resistance is the following [83]:

Definition 19. *A voting system is coercion-resistant if for every probabilistic polynomial time adversary $\mathbb{A}$, for all parameters n_{RT}, t_{RT}, n_{TT}, t_{TT}, n_V, n_A, n_{Ls} and for distribution $\mathcal{D}$, there exists a probabilistic polynomial time adversary $\mathbb{S}$ and a negligible function μ such that*

$$\begin{aligned}&|\Pr(\mathsf{Ideal}(\mathbb{S},\lambda,n_V,n_A,n_{Ls},\mathcal{D})=1)\\&\quad-\Pr(\mathsf{Real}(\mathbb{A},\lambda,n_{RT},t_{RT},n_{TT},t_{TT},n_V,n_A,n_{Ls},\mathcal{D})=1)\mid\leq\mu(\lambda)\end{aligned}$$

where Ideal *and* Real *are as defined in Algorithms 4.2.1 and 4.2.2.*

In the Real game, the adversary takes part in the Setup and CredGen protocols (lines 2-3) and chooses the voters it controls $V_{n_{\mathtt{A}}}$ (line 4) and the target for coercion, namely a voter j and their voting intention $\mathtt{Ls}_{\mathsf{target}}$ (line 5). The adversary can guess a bit b (line 9). If $b = 1$, this means that the coerced voter j obeys to the coercer (i.e., does not cast a vote). In the other case, the coerced voter follows the evasion strategy thus votes for $\mathtt{Ls}_{\mathsf{target}}$ (line 12) and gives the ruse credential to the coercer (line 11). After that, each vote is added to the WBB, following the order of $\mathbb{B}$, and the adversary

can see the WBB and add votes (lines 18-22). In the end of the game, the adversary takes part in the Tally (line 23) and outputs its guess (line 24).

In the Ideal game, the adversary can only select the voters it controls V_{n_A} (line 4) and the target for coercion (line 5). Votes are automatically added to the WBB (line 14), and the adversary can only observe the number of ballots and the election result. Then it outputs its guess.

In recent years, Cortier, Gaudry, and Yang gave a new definition of coercion-resistance. In fact, their paper [39] shows that the protocol of JCJ [83] leaks the number of re-votes and the number of votes cast with fake credentials instead of leaking just the number of discarded votes (i.e., the difference between the number of cast votes and the number of valid votes). This comes from the procedure that precedes the tally, where the trustees remove the ballots that should not be counted. Fixing this led to the notion of *cleansing-hiding*, that they applied to form a variant of JCJ voting protocol that they called CHide [39].

The protocol of *Vote App* does not allow to have a cleanse-hiding tallying without introducing computational overhead. However, this does not compromise the security of our protocol in the context of our threat model (see Chapter 6): while the definition of [39] remains theoretically valid, and we plan to modify the protocol in the future to meet this stronger guarantee, we do not consider revealing the number of re-votes a severe attack in practice. In a real election, the volume of submitted votes makes it difficult for an adversary to extract meaningful information from this leakage. Therefore, we prove security using the coercion-resistance definition of Definition 19 [83], which remains suitable for our model.

Algorithm 4.2.2 Ideal

Require: $\mathbb{A}, \lambda, n_{\mathtt{V}}, n_{\mathtt{A}}, n_{\mathtt{Ls}}, \mathcal{D}$

$\mathbb{B} \leftarrow \emptyset$ ▷ Create an empty list of ballots

$\mathtt{pp}, \mathtt{pk}_{\mathtt{ACC}}, \mathtt{pk}_{\mathtt{RT}}, \mathtt{pk}_{\mathtt{TT}}, \{\mathtt{sk}_{\mathtt{ACC}_i}, \mathtt{sk}_{\mathtt{RT}_i}\}^{n_{\mathtt{RT}}}, \{\mathtt{sk}_{\mathtt{TT}_i}\}^{n_{\mathtt{TT}}} \leftarrow \mathsf{Setup}(\lambda, t_{\mathtt{RT}}, n_{\mathtt{RT}}, n_{\mathtt{TT}}, t_{\mathtt{TT}})$ ▷ Run the Setup algorithm

$(\{\mathtt{pp}_{\mathtt{ACC},v_{\mathtt{id}}}, \mathtt{sp}^{\mathtt{valid}}_{\mathtt{ACC},v_{\mathtt{id}}} : v_{\mathtt{id}} \in [1,\ldots,n_{\mathtt{V}}]\}) \leftarrow \mathsf{CredGen}()$ ▷ Register voter

$V_{n_{\mathtt{A}}} \leftarrow \mathbb{A}(\lambda)$ ▷ $\mathbb{A}$ corrupt voters

$(j, \mathtt{Ls}_{\mathsf{target}}) \leftarrow \mathbb{A}()$ ▷ Target voter j who wants to vote for $\mathtt{Ls}_{\mathsf{target}}$

if $| V_{n_{\mathtt{A}}} | \neq n_{\mathtt{A}} \vee j \notin [1,\ldots,n_{\mathtt{V}}] \setminus V_{n_{\mathtt{A}}} \vee \mathtt{Ls}_{\mathsf{target}} \notin [1,\ldots,n_{\mathtt{Ls}}] \bigcup \{\phi\}$ **then**

  return 0 ▷ If the selection is invalid return 0, ϕ is the choice to abstain

end if

$b \xleftarrow{\$} \{0,1\}$ ▷ Coin is flipped

if b=0 **then** ▷ Voter j evades coercion

  $\mathbb{B} \leftarrow \mathbb{B} \bigcup \{\mathsf{Vote}(\mathtt{Ls}_{\mathtt{target}}, \mathtt{sp}_{\mathtt{ACC},j})\}$ ▷ Voter j votes

end if

$\tilde{c} \leftarrow \mathtt{sp}_{\mathtt{ACC},j}$

$\mathbb{B} \leftarrow \mathcal{D}([1,\ldots,n_{\mathtt{V}}] \setminus V_{n_{\mathtt{A}}}, n_{\mathtt{Ls}})$ ▷ Draw votes for honest voters

$\mathbb{A}(\tilde{c}, \{\mathtt{sp}^{\mathtt{valid}}_{\mathtt{ACC},v_{\mathtt{id}}}\}_{v_{\mathtt{id}} \in V_{n_{\mathtt{A}}}})$ ▷ $\mathbb{A}$ learns $\tilde{c}$ and the private credentials of corrupt voters

$\mathbb{B} \leftarrow \mathbb{B} \bigcup \{(a, \mathtt{Ls}) \mid a \in V_{n_{\mathtt{A}}}, \mathtt{Ls} \in [1,\ldots,n_{\mathtt{Ls}}]\}$

$X, \Pi \leftarrow Ideal\mathsf{Tally}(\mathbb{B}, \mathtt{pp}_{\mathtt{ACC}}, \mathtt{sk}_{\mathtt{RT}_i}, \mathtt{sk}_{\mathtt{TT}_i})$

$b' \leftarrow \mathbb{A}(X)$

if $b' == b$ **then**

  return 1

else

  return 0

end if

4.3 Proof of Security

In this section we prove the security of the protocol presented in Chapter 3.

Security for cryptographic protocols is typically defined as an attack game played between an adversary $\mathbb{A}$ and some benign entity, which is usually called challenger $\mathbb{C}$. Both $\mathbb{A}$ and $\mathbb{C}$ are probabilistic processes that communicate with each other, and so we can model the game as a probability space.

Theorem 1. The voting protocol described in Chapter 3 satisfies the coer-

cion resistance property, as defined in Definition 19, in the Random Oracle Model, under the *q-SDH-II* and *SDDHI-II* (see Section 2.2) assumptions.

To prove the security of the protocol, we model the proof as a sequence of games, using Shoup's methodology (see Section 2.11 and [128]).

The proof is similar to the one presented in [8]. We start by defining the attack game (`Game0`) with respect to a polynomial time adversary $\mathbb{A}$, then we will define the other games (`Game1` - `Game6`) outlining the differences with `Game0`.

Let S_i be the probability that the adversary wins `Gamei` (i.e., $b' = b$ in line 25 of Algorithm 4.2.1 and in `Game0`, `Phase5`). For all i, we show that $\mid S_{i+1} - S_i \mid$ is negligible, which proves that $\mid S_6 - S_0 \mid$ is also negligible.

In the proof, we assume that $\mathbb{C}$ (and $\mathbb{A}$) can authenticate to the Electoral Roll on behalf of the honest (resp. corrupted) voters. Since we want to prove the coercion-resistance property, the confirmation and verification of the vote is out of scope, so the corresponding steps are omitted from the proof.

`Game0`. This is the real game in which the adversary $\mathbb{A}$ interacts with the real protocol Real (see Algorithm 4.2.1).

Setup. The challenger $\mathbb{C}$ runs the Setup of the protocol, in particular it:

- picks randomly five generators g_0, g_1, g_2, g_3, o of a group $\mathbb{G}$ of prime order p;
- picks randomly $\xi_1, \xi_2, y \in \mathbb{Z}_p$ and computes $\mathtt{pk}_{\mathtt{TT}} = g_1^{\xi_1} g_2^{\xi_2}$ and $\mathtt{pk}_{\mathtt{RT}} = g_3^{y}$;

Phase1. $\mathbb{C}$ uses y to generate the credentials for the voters $\{(\mathtt{pp}_{\mathtt{ACC},v_{\mathtt{id}}}, \mathtt{sp}^{\mathtt{valid}}_{\mathtt{ACC},v_{\mathtt{id}}})\}_{v_{\mathtt{id}}}$, using y and three randomly generated values $r_{v_{\mathtt{id}}}, x^{\mathtt{valid}}_{v_{\mathtt{id}}}, \sigma_{v_{\mathtt{id}}} \in \mathbb{Z}_p$ with $v_{\mathtt{id}} \in n_{\mathtt{V}}$. These are saved in a list $\mathcal{L}_{\mathtt{ACC}}$.

Phase2. $\mathbb{A}$ selects the set $V_{n_{\mathtt{A}}}$ of $n_{\mathtt{A}}$ voters to corrupt together with a target voter j who wants to vote for $\mathtt{Ls}_{\mathsf{target}}$. If any choice is invalid, the simulation is aborted.

Challenge. Pick randomly $b \in \{0, 1\}$. If $b = 1$ then $\mathbb{C}$ gives $\mathbb{A}$ the real credential $(\mathtt{pp}_{\mathtt{ACC}}, \mathtt{sp}^{\mathtt{valid}}_{\mathtt{ACC}})$ for voter j, otherwise it gives $\mathbb{A}$ $(\mathtt{pp}_{\mathtt{ACC}}, \mathtt{sp}^{\mathtt{ruse}}_{\mathtt{ACC}})$ with $\mathtt{sp}^{\mathtt{ruse}}_{\mathtt{ACC}}$ derived from a random PIN. Then $\mathbb{C}$ gives to $\mathbb{A}$ all the other generated credentials.

Phase3. $\mathbb{C}$ votes on behalf of the honest voters, generating also the proofs. $\mathbb{A}$ votes on behalf of the corrupted voters.

Phase4. $\mathbb{C}$ performs the tally. This is feasible since $\mathbb{C}$ knows ξ_1, ξ_2, y.

Phase5. $\mathbb{A}$ outputs a bit b'.

We have that $\Pr[S_0] = \Pr[\mathsf{Real} = 1]$.

`Game1.` In this game we modify the operations of $\mathbb{C}$ during Phase3.

Phase3. $\mathbb{C}$ generates a DDH quadruple (g_1, g_2, T_1, T_2) by picking $\tau \in \mathbb{Z}_p$ random and setting $T_i = g_i^{\tau}$, $i = 1, 2$. Then it uses this quadruple to cast the votes of honest voters with new public key $T' = T_1^{\xi_1} T_2^{\xi_2}$. Since the used quadruple is DDH, then the ciphertext has the correct distribution thus

$$\Pr[S_0] = \Pr[S_1].$$

`Game2.` In this game $\mathbb{C}$ extracts the encrypted credentials from the proofs associated to all the ballots cast by $\mathbb{A}$ in Phase3. We suppose that the ballots are cast sequentially so that we can rewind $\mathbb{A}$ to extract the credentials. If the extractor fails, $\mathbb{C}$ outputs $\perp$ and aborts.

By the Difference Lemma (see Lemma 1 and [128]) we have that

$$| \Pr[S_1] - \Pr[S_2] | \leq \Pr[F] \leq \varepsilon$$

where F is the failure event and ε is the (negligible) knowledge error of the proof system.

`Game3.` In this game $\mathbb{C}$, after extracting the credentials, outputs $\perp$ and aborts if one of them is a valid credential that has not been issued by $\mathbb{C}$ or if it contains a valid private credential belonging to a honest voter.

In the first case it means that $\mathbb{A}$ broke the *q-SDH-II* problem[1], in the second case it means that $\mathbb{A}$ broke the discrete logarithm problem and thus the *SDDHI-II* problem. Let F be the union of the two failure events:

$$| \Pr[S_2] - \Pr[S_3] | \leq \Pr[F] \leq \varepsilon'$$

[1] In this case, q is the number of credential issued in Phase1.

where $\varepsilon' = n_{\mathtt{A}}\varepsilon_1 + \varepsilon_2$ with ε_1 the advantage of an adversary to break the *q-SDH-II* problem[2] and ε_2 the advantage of an adversary to break the *SDDHI-II* problem.

`Game4.` In this game, $\mathbb{C}$ cast the votes of honest voters in Phase3 using a random quadruple instead of a DDH one. Under the DDH assumption, those are indistinguishable thus

$$| \Pr[S_3] - \Pr[S_4] | \leq \varepsilon''$$

where ε'' is the advantage of a distinguisher to distinguish between a DDH quadruple and a random one.

Since the Modified Exponential ElGamal is semantically secure under the DDH assumption, this does not give any information about the votes cast by the honest voters.

`Game5.` In this game if $\mathbb{A}$ cast a vote with a fake credential received when $b = 0$, then $\mathbb{C}$ during the mixing phase, will modify the content of the vote to an encryption of $\mathtt{Ls}_{\mathsf{target}}$ and prove the correctness of the shuffle [11]. $\mathbb{A}$ will not be able to distinguish this change since the Modified Exponential ElGamal is semantically secure. Thus

$$| \Pr[S_4] - \Pr[S_5] | \leq \varepsilon'''$$

where ε''' is the advantage of a DDH distinguisher against the semantic security of the Modified Exponential ElGamal cryptosystem.

`Game6.` In this game $\mathbb{C}$ always gives the valid credential to $\mathbb{A}$ in the Challenge phase even if $b = 0$ (this does not allow $\mathbb{A}$ to cast multiple valid votes for $\mathtt{Ls}_{\mathtt{target}}$). $\mathbb{A}$ cannot distinguish this change otherwise it will be able to break the *SDDHI-II* assumption thus

$$| \Pr[S_5] - \Pr[S_6] | \leq \varepsilon''''$$

where ε'''' is the advantage of an adversary to break the *SDDHI-II* problem.

[2] We require the factor $n_{\mathtt{A}}$ since the adversary does not have access to y thus has to guess which among the $n_{\mathtt{A}}$ credentials used by $\mathbb{A}$ to cast a vote is the valid one.

In this game $\mathbb{A}$ always obtains the valid credential and can see only the final tally of votes. This means that $\Pr[S_6] = \Pr[\mathsf{Ideal} = 1]$.

To conclude we can say that

$$| \Pr[\mathsf{Real} = 1] - \Pr[\mathsf{Ideal} = 1] | = | \Pr[S_0] - \Pr[S_6] | \leq \varepsilon + \varepsilon' + \varepsilon'' + \varepsilon''' + \varepsilon''''$$

which is negligible thus the voting protocol of Chapter 3 satisfies coercion-resistance in the random oracle model under the *q-SDH-II* and *SDDHI-II* assumptions.

Algorithm 4.2.1 Real

Require: $\mathbb{A}, \lambda, n_{\mathtt{RT}}, t_{\mathtt{RT}}, n_{\mathtt{TT}}, t_{\mathtt{TT}}, n_{\mathtt{V}}, n_{\mathtt{A}}, n_{\mathtt{Ls}}, \mathcal{D}$

$\mathbb{B} \leftarrow \emptyset$ ▷ Create an empty list of ballots
$\mathtt{pp}, \mathtt{pk}_{\mathtt{ACC}}, \mathtt{pk}_{\mathtt{RT}}, \mathtt{pk}_{\mathtt{TT}}, \{\mathtt{sk}_{\mathtt{ACC}_i}, \mathtt{sk}_{\mathtt{RT}_i}\}^{n_{\mathtt{RT}}}, \{\mathtt{sk}_{\mathtt{TT}_i}\}^{n_{\mathtt{TT}}} \leftarrow \mathsf{Setup}(\lambda, t_{\mathtt{RT}}, n_{\mathtt{RT}}, n_{\mathtt{TT}}, t_{\mathtt{TT}})$ ▷ Run the Setup algorithm
$(\{\mathtt{pp}_{\mathtt{ACC},v_{\mathtt{id}}}, \mathtt{sp}^{\mathtt{valid}}_{\mathtt{ACC},v_{\mathtt{id}}} : v_{\mathtt{id}} \in [1,\ldots,n_{\mathtt{V}}]\}) \leftarrow \mathsf{CredGen}()$ ▷ Register voter
$V_{n_{\mathtt{A}}} \leftarrow \mathbb{A}(\mathtt{pp}_{\mathtt{ACC},v_{\mathtt{id}}})$ ▷ A corrupt voters and gets their public credential
$(j, \mathtt{Ls}_{\mathsf{target}}) \leftarrow \mathbb{A}(\{\mathtt{sp}_{\mathtt{ACC},v_{\mathtt{id}}} : v_{\mathtt{id}} \in V_{n_{\mathtt{A}}}\})$ ▷ Target voter j who wants to vote for $\mathtt{Ls}_{\mathsf{target}}$
if $\mid V_{n_{\mathtt{A}}} \mid \neq n_{\mathtt{A}} \vee j \notin [1,\ldots,n_{\mathtt{V}}] \setminus V_{n_{\mathtt{A}}} \vee \mathtt{Ls}_{\mathsf{target}} \notin [1,\ldots,n_{\mathtt{Ls}}] \bigcup \{\phi\}$ **then**
    return 0 ▷ If the selection is invalid return 0, ϕ is the choice to abstain
end if
$b \xleftarrow{\$} \{0,1\}$ ▷ Coin is flipped
if b=0 **then** ▷ Voter j evades coercion
    $\tilde{c} \leftarrow \mathsf{RuseCred}(\overset{\mathtt{ruse}}{\mathtt{PIN}})$ ▷ The $\overset{\mathtt{ruse}}{\mathtt{PIN}}$ is chosen randomly
    $\mathbb{B} \leftarrow \mathbb{B} \bigcup \{\mathsf{Vote}(\mathtt{Ls}_{\mathsf{target}}, \mathtt{sp}_{\mathtt{ACC},j})\}$ ▷ Voter j votes
else
    $\tilde{c} \leftarrow \mathtt{sp}_{\mathtt{ACC},j}$ ▷ Voter submits to coercion
end if
$\mathbb{B} \leftarrow \mathcal{D}([1,\ldots,n_{\mathtt{V}}] \setminus V_{n_{\mathtt{A}}}, n_{\mathtt{Ls}})$ ▷ Draw votes for honest voters
$\mathbb{A}(\tilde{c})$ ▷ $\mathbb{A}$ learns $\tilde{c}$
for $(v_{\mathtt{id}}, \mathtt{Ls}) \in \mathbb{B}$ **do**
    $M \leftarrow \mathbb{A}(\mathbb{B})$ ▷ Ballots are cast and $\mathbb{A}$ can see the WBB and add votes
    $\mathbb{B} \leftarrow \mathbb{B} \bigcup \{m \in M \mid m \text{ valid}\}$
    $\mathbb{B} \leftarrow \mathbb{B} \bigcup \{\mathsf{Vote}(\mathtt{Ls}, \mathtt{sp}_{\mathtt{ACC},v_{\mathtt{id}}})\}$
end for
$X, \Pi \leftarrow \mathsf{Tally}(\mathbb{B}, \mathtt{pp}_{\mathtt{ACC}}, \mathtt{sk}_{\mathtt{RT}_i}, \mathtt{sk}_{\mathtt{TT}_i})$
$b' \leftarrow \mathbb{A}(X, \Pi)$
if $b' == b$ **then**
    return 1
else
    return 0
end if

Chapter 5

Authorization Tokens

This chapter provides some background on OAuth in Section 5.1 and on the Commitment Access Token in Section 5.2. Section 5.3 dives deeper into token management within *Vote App*.
The content of this chapter is based on the work [133].

5.1 OAuth 2.0

OAuth 2.0 is an open standard authorization protocol designed to provide secure, delegated access to resources across different platforms without the need to share user credentials [71]. It is widely employed in various applications, such as social media integrations, payment gateways, and other services requiring third-party access.

The key roles in the OAuth 2.0 ecosystem are:

- *Resource Owner:* the user or entity owning the protected resources;
- *Client:* the application requesting access to the Resource Owner's data;
- *Authorization Server:* the server responsible for authenticating the resource owner and issuing access tokens;
- *Resource Server:* the server hosting the protected resources.

The OAuth 2.0 framework enables resource owners to grant limited access to their resources, hosted on a resource server, to third-party applications (clients). The client redirects the resource owner through the user

agent into the authorization server, where the resource owner performs the authentication. After the successful authentication of the resource owner, the authorization server issues an access token which clients use to access the resource owner's data in the resource server.

Access Tokens. Access tokens are commonly bearer tokens, meaning that access to the resource is granted as long as the entity possessing (or *bearing*) the token is authorized. However, this characteristic makes them vulnerable to various attacks. As a result, OAuth security best practices [95] recommend the following:

- *Token Binding:* the authorization server should constrain tokens to their intended sender. This is typically achieved by the authorization server by binding the token to a public key held by the sender and forcing them to prove its possession by digital signature. Two methods for sender-constrained access tokens are mutual TLS (mTLS) [29] and Demonstrating Proof-of-Possession (DPoP) [28].
- *Client Authentication:* the client should always be strongly authenticated, e.g., by using signed JSON Web Tokens [82].
- *Authorization Code Protection:* to prevent code injection or theft, Proof Key for Code Exchange (PKCE) [122] should be employed.
- *Privilege Restriction:* access tokens should limit the privileges granted to the holder to the minimum necessary by specifying the precise scope, audience, and resource claims.

Authorization Code Grant with PKCE. The Authorization Code Grant is a secure flow designed for server-side applications. In this flow, the client first obtains an authorization code through user consent. This code is then exchanged for an access token, which grants access to protected resources. To further enhance security, the Proof Key for Code Exchange (PKCE) [122] protocol is employed. PKCE mitigates interception attacks by introducing two key elements: a code verifier, which is a random string generated by the client, and a code challenge, which is the hash of the code verifier sent with the authorization request. When exchanging the authorization code for an access token, the client must send the original code verifier. Subsequently, the authorization server performs a verification process on the code challenge. This verification process involves the hashing of the code verifier

and a comparison of the hashed code verifier to the previously received challenge. This process ensures that only the client that initiated the request can exchange the authorization code, effectively preventing attackers from using intercepted authorization codes to obtain access tokens.

OAuth 2.0 Token Exchange. Token exchange, as defined in [81], is a protocol that allows clients to exchange one security token for another. This mechanism is part of the OAuth 2.0 framework and enables scenarios where an initial token needs to be exchanged for a different type of token that is more suitable for a specific use case or target resource. The client includes the original token, its type, the desired token type, and optional parameters such as audience or scope in the token exchange request. The authorization server validates the request, enforces policies, and issues a new token tailored to the client's needs, enhancing security and interoperability in distributed systems.

5.1.1 Applications to E-Voting

The OAuth 2.0 framework can be used to enhance e-voting systems with regards to:

- *authentication and authorization*, by ensuring that voters can access the system securely while protecting their identities;
- *role management*, by assigning roles such as voter, administrator, or auditor using token-based claims;
- *third-party auditing*, by enabling external auditors to access data without exposing sensitive credentials.

The OAuth 2.0 Token Exchange could be used to issue specialized tokens for different roles. For instance, a voter might exchange an initial authentication token for a token specific to casting a ballot, with limited scope and validity, ensuring role-based access control and minimizing risks of token misuse.

5.2 Commitment Access Token

There are cases in which:

- the resource owner needs to perform a single, high-value operation;
- the authorization server must remain unaware of the exact details of the operation but is responsible for ensuring the correct audience and scope, as well as enforcing rate limits for requests within that scope, without delegating access control to the client;
- the resource server must verify that the resource owner was authenticated at the time of authorizing the specific access.

To meet these constraints, we propose the Commitment Access Token (`CAT`). The `CAT` authorization flow is designed for a scenario in which an OAuth client holds a message m and requires an access token linked to m. The flow must ensure anonymity: the authorization server should not learn m, and the resource server should not link m to the identity of the client. Our proposal is the following:

1. to send the message m to a resource server, the client generates a random $r \in \{0,1\}^{\lambda}$, computes a commitment to m

$$c \leftarrow \texttt{Commit}(m, r) = H(m||r)$$

   and sends it to the authorization server;
2. the authorization server authenticates the client and provides an access token - a signature σ of the commitment c - to the client;
3. the client requests the resource server to process the message m, by providing the signature σ and the opening r of the commitment c;
4. the resource server accepts the message m if both the signature and the commitment are valid.

The flow is shown in Figure 5.1, with the hash-based commitment scheme as in Section 2.4.1.

Holder binding. In some scenarios, ensuring holder binding to the access token is unnecessary. Since the commitment is binding and the authorization server has approved the access, the identity of the client accessing the resource server may be irrelevant. This decoupling enhances the client's anonymity, making it particularly valuable for sensitive operations, such as ballot casting.

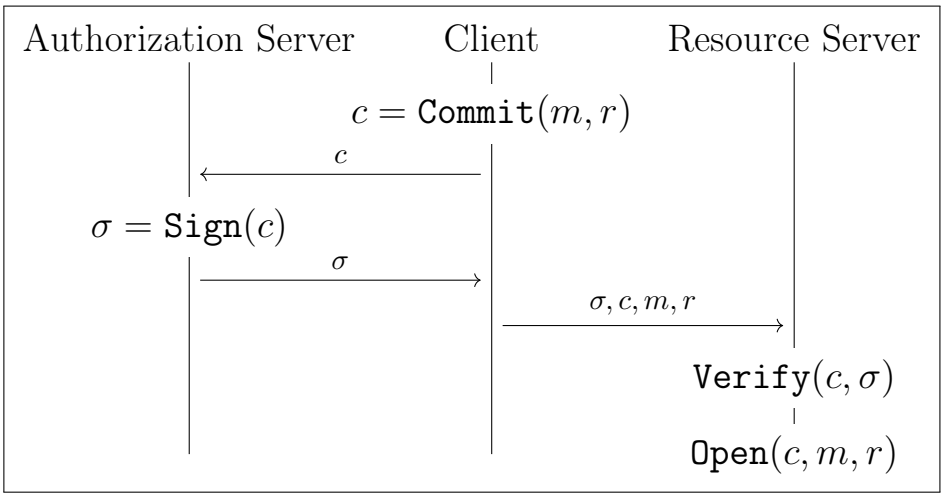

Figure 5.1: Commitment Access Token (CAT).

5.3 Integration of OAuth 2.0 and CAT into Vote App

As part of the registration process, each instance of *Vote App* registers with the Electoral Roll (ER) as a separate, confidential OAuth client. The Registration Tellers and the Ballot Boxes act as OAuth resource servers, providing secure access to voting-related resources. The e-ID Provider fulfills the role of an OpenID Provider, enabling authentication using digital identities. The ER acts as an OAuth authorization server, overseeing the distribution of anti-coercion credentials and voting tokens.

As detailed in Section 3.8, in order to vote, the voter must type in a five digits PIN. Usability requirements (e.g., CoE 1 [43]) strongly suggest the PIN must be short, particularly if used seldom and delivered some time before usage. As pointed out in [53], a short PIN may be brute-forced: a coercer briefly in control of a voter's device could try multiple PINs, succeeding with non-negligible probability, then deprive the voter of the device. A rate-limiting measure on ballot casting is therefore advisable, but it must preserve all other requirements. In particular, anonymity (CoE 26) requires the Ballot Box (BB) does not authenticate the voter, and the BBs must not be able to distinguish votes cast with the same PIN (i.e., credential). An extract of the CoE requirements and how we enforce them whitin *Vote App* is presented in Table 5.1.

Table 5.1: Analysis of requirements and their corresponding measures within the OAuth 2.0 framework.

Requirement	Measure
CoE 1: Usability	Short PIN
CoE 8: Grant access after authenticating with the right to vote	OAuth
CoE 10: No undue influence	`Commitment Access Token`, ruse PIN, re-voting
CoE 26: Vote anonimity - unlinkability of the unsealed vote and voter	`Commitment Access Token`, pseudonymous ids

To counter the brute-forcing attack while staying consistent with the CoE's recommendations, we propose to use the Commitment Access Token (`CAT` , Figure 5.1). In particular:

- the authorization server (ER):
  - enforces rate limits on a user level, ensuring that each voter is restricted in their ability to submit ballots;
  - does not access the payload of the authorization request, preserving the confidentiality of the ballot content;
- the resource servers (BBs):
  - accept and process only ballots that have been properly authorized;
  - do not learn the identity of the voter who generated the payload and obtained authorization, ensuring unlinkability between the voter and their ballot.

Since the BBs cannot do rate limiting, the authorization server must limit the number of cast attempts by a client without revealing the identity of the voter to the BBs: the `CAT` precisely allows anonymous authentication, thus considerably limiting the impact of brute-force attacks. This countermeasure also mitigates against the *leaky duplicate removal* attack of [53], since it

becomes more difficult to cast a valid (duplicate) vote. By combining these properties, the CAT provides a powerful tool for balancing security and privacy, ensuring anonymous and authenticated casting without compromising usability.

5.3.1 Protocol Integrations

The following subsections provide an overview of tokens and token management within *Vote App*'s protocol.

As already discussed, the subscript $v_{\texttt{id}}$ is omitted for clarity.

5.3.1.1 App Inizialization

Let $\mathcal{V}$ be a voter that wants to use Vote App to take part in the election. $\mathcal{V}$ has to download *Vote App* from the official app store, to guarantee its integrity, and install it on their device.

Once *Vote App* is installed on their device, $\mathcal{V}$ can set it up and register to the election system:

1. $\mathcal{V}$ opens *Vote App* which automatically checks the integrity of the device (app origin, device not rooted);
2. *Vote App* fetches from WBB: $\texttt{pk}_{\texttt{EL}}$, the Ballot, and the identities of ER, RTs, BBs, NS;
3. *Vote App* generates a key pair $(\texttt{AtPK}, \texttt{AtSK})$ for $\mathcal{S}$, and saves AtSK safely in the keystore;
4. the ER initiates the login via a identity provider, linked to *Vote App*:
   (a) in a browser custom tab the voter logs into the identity provider and gives consent for sharing personal data;
   (b) after a successful login the identity provider sends to the ER the authenticated personal data;
   (c) ER sends a `Registration token` to *Vote App*;
   (d) if it is a second registration on a new device (and this can be inferred if a $v_{\texttt{id}}$ has already been associated to $\mathcal{V}$'s info), then ER sends also $\mathcal{V}$'s pseudonymous identity $v_{\texttt{id}}$, $\mathcal{V}$'s designated verifier public key $\texttt{pk}_{\texttt{DV}}$ and $\mathcal{V}$'s designated verifier key share $\texttt{share}_{\texttt{sk}_{\texttt{DV}}}$;

5. *Vote App* sends `AtPK` to ER, which saves it associated to $\mathcal{V}$'s info.

5.3.1.2 Registration token

The `Registration token` is sent by *Vote App* to the ER, to authenticate the sending of the $\texttt{share}_{\texttt{sk}_{\texttt{DV}}}$ and of the $\texttt{pk}_{\texttt{DV}}$. Upon receival of the `Registration token`, the ER starts the OAuth token exchange protocol to exchange it with two tokens: the `PIN request token` and the `Notification Registration token`.

1. The ER saves $\texttt{share}_{\texttt{sk}_{\texttt{DV}}}$, $\texttt{pk}_{\texttt{DV}}$ associated to $\mathcal{V}$'s info and generates two authorization tokens for $\mathcal{V}$'s *Vote App*:
   - a `PIN request token` for each RT, each token containing $\mathcal{V}$'s pseudonymous identifier $v_{\texttt{id}}$, $\mathcal{V}$'s designated verifier public key $\texttt{pk}_{\texttt{DV}}$, and the election identifier for which $\mathcal{V}$ is an eligible voter $\texttt{EL}_{\texttt{ID}}$: this type of token will be used by $\mathcal{V}$'s *Vote App* to register, request PIN re-sending, set up a ruse PIN, set up a new device;
   - a `Notification Registration token` for the NS, containing $v_{\texttt{id}}$;

   and sends them back to *Vote App*.
2. *Vote App* immediately uses the `Notification Registration token` to register with the NS.

5.3.1.3 PIN request token

Authorized with the `PIN request token` provided by ER (which contain $v_{\texttt{id}}$ and $\texttt{pk}_{\texttt{DV}}$), *Vote App* sends to each RT_i a registration request containing:

- the asset id $\texttt{rid}_1$;
- `mph`, a string of bytes, all with value `FF`, with the same length of the serialization of a scalar in $\mathbb{Z}_p$;
- `dvph`, a string of bytes, all with value `FF`, with the same length of the serialization of a DVNIZKP;

Each RT_i checks the validity of the token sent by *Vote App* and identifies:

- either a registration request from the fact that they do not have a designated public key associated to $v_{\mathtt{id}}$ yet;
- or identifies the request from the content of the two byte strings: if they contain only bytes with value `FF` it is a request of PIN re-sending, otherwise it is a ruse PIN request.

Subsequently:

1. each RT_i uses $v_{\mathtt{id}}$ to identify the correct $\mathcal{T}_{i,v_{\mathtt{id}}}$ corresponding to $\mathcal{V}$;
2. each RT_i randomly chooses $\tau_{\mathtt{min}} \leq \tau_{i,\mathtt{rid}_1} < \tau_{\mathtt{max}}$, one integer that represent the number of seconds that RT_i will wait before sending the shares of the PIN and DVNIZKP;
3. each RT_i sends to *Vote App* $(i, x_i + \sigma_i, E^{\tilde{r}_i}_{\mathtt{pk}_{\mathrm{TT}}}[A], \Pi_i)$ and marks this time as $\tau_{i,0}$.

5.3.1.4 Actual PIN delivery

After $\tau_{i,\mathtt{rid}_1}$ seconds (this random waiting time has been determined in step 2 of Section 5.3.1.3), each RT_i starts the procedure to deliver the token required to retrieve the PIN and the DVNIZKP:

1. each RT_i notifies to NS that its share of the asset with id $\mathtt{rid}_1$ for the pseudonymous $v_{\mathtt{id}}$ is ready;
2. the NS maintains for each voter's pseudonymous identifier $v_{\mathtt{id}}$ a table with the notifications received from the RTs organized by the asset identifiers $\mathtt{rid}_*$;
3. when for an asset $\mathtt{rid}_*$ the NS has received at least $t_{\mathtt{RT}}$ notifications, it notifies *Vote App* associated to $v_{\mathtt{id}}$ that the asset is ready to be retrieved;
4. when *Vote App* receives from the NS a notification that an asset $\mathtt{rid}_*$ is ready, first it checks whether $\mathtt{rid}_* \stackrel{?}{=} \mathtt{rid}_1$: in case that $\mathtt{rid}_* \neq \mathtt{rid}_1$ the notification is ignored. *Vote App* in fact does not accept the response, as it may not have been what was expected (this is to ensure that if someone else sends irrelevant data without the correct identifier, the request is discarded);

5. if $\texttt{rid}_* = \texttt{rid}_1$, *Vote App* shows on the device a notification that new data has been received;

6. when the voter opens *Vote App*, they are requested to log-in via an identity provider;

7. *Vote App* contacts the ER sending $\texttt{rid}_1$;

8. the ER initiates the login via a identity provider, linked to *Vote App*:

   (a) in a browser custom tab the voter logs into the identity provider and gives consent for sharing personal data;

   (b) after a successful login the identity provider sends to the ER the authenticated personal data;

   (c) the ER generates two authorization tokens for $\mathcal{V}$'s *Vote App*:

   - a `PIN request token` for each RT, containing $v_{\texttt{id}}$, $\texttt{pk}_{\texttt{DV}}$, $\texttt{EL}_{\texttt{ID}}$, which will be used to (i) request the valid PIN for the first time on a device, (ii) for a PIN remainder, or (iii) to set up a ruse PIN;
   - a `PIN and DVNIZKP retrieval token` for each RT, each token containing $v_{\texttt{id}}$, $\texttt{rid}_1$: this type of token will be used by $\mathcal{V}$'s *Vote App* to retrieve the mask shares from the RTs and the DVNIZKP shares;

   (d) ER sends the tokens to *Vote App*.

5.3.1.5 PIN and DVNIZKP retrieval token

Upon *Vote App*'s request authenticated via `PIN and DVNIZKP retrieval token`:

1. each RT_i checks if at least $\tau_{i,\texttt{rid}_1}$ seconds have been elapsed from $\tau_{i,0}$, if it is the case then each RT_i sends to *Vote App* both their share of the DVNIZKP:

   $$(i, z_{i,1}, z_0, I_0, I_1, I_2, c_0, c_1)$$

   and the shares of the mask to retrieve the $\overset{\texttt{valid}}{\texttt{PIN}}$ (i, σ_i), otherwise the RT_i notes that the request has arrived, but does not respond yet (*Vote App* will re-iterate this request subsequently until it is successful).

5.3.1.6 Casting token (CAT)

The Commitment Access Token (`CAT`) is used during the voting phase to secure casting. Once *Vote App* has completed the computation of the ballot $\mathcal{B}$, it is sent to the ballot boxes with the following procedure:

1. *Vote App* retrieves `AtPK` from its storage;
2. *Vote App* chooses uniformly at random a byte string $\texttt{rnd}_{\texttt{comm}}$ and computes the ballot commitment:

$$\texttt{comm}_{\mathcal{B}} = \mathcal{H}(\mathcal{B}, \texttt{rnd}_{\texttt{comm}}); \tag{5.1}$$

3. using `AtSK`, *Vote App* signs and sends to the ER a request for a `Casting token` tied to the commitment $\texttt{comm}_{\mathcal{B}}$;
4. the ER verifies the validity of the request using `AtPK`, and if $\mathcal{V}$ satisfies the policies on vote casting (e.g. has not voted too recently or too frequently), the ER sends to *Vote App* an anonymous `Casting token` tied to $\texttt{comm}_{\mathcal{B}}$ (with limited time validity) for each BB;
5. *Vote App* sends anonymously to each trusted BB (as defined by $\mathcal{V}$ in Section 3.12) the ballot $\mathcal{B}$ and the randomness $\texttt{rnd}_{\texttt{comm}}$, authenticated with the appropriate `Casting token`;
6. each trusted BB checks the authentication of the received ballot by checking the validity of the `Casting token` (which contains $\texttt{comm}_{\mathcal{B}}$):
   - checks that the token is a valid, not already used, `Casting token` for that BB, properly issued by the ER;
   - checks that the token is not expired (time validity);
   - checks that the Equation (5.1) holds.

If all the checks hold, the ballot is accepted by the BBs.

Chapter 6

Threat Modeling Process

In this chapter, we propose a novel methodology for performing a threat analysis on e-voting systems in Section 6.2, and we summarize the results of its application to assess the risks posed by a coercer to *Vote App* protocol in Section 6.3.
The content of this chapter is based on the work [98] which was awarded Best Paper at Crisis 2024: 19th International Conference on Risks and Security of Internet and Systems [45].

6.1 Introduction

The threat modeling process [127, 111] is a well-established approach to verifying the security properties of a system. It provides the tools to identify potential security and privacy threats, assess the risk, and determine possible mitigation strategies. In the context of remote voting, coercion stands out as one of the most significant and complex threats to model.

Typically, the threat modeling process is divided into five phases:

1. *System Characterization:* this phase involves understanding the system's architecture, assets, data flows, and trust boundaries in order to establish the scope and the context for the model.

2. *Threat Modeling:* this phase enumerates the potential threats and vulnerabilities by analyzing the system's assets and their interactions with identified attackers.

3. *Risk Analysis:* this phase quantifies the risk of the identified threats, providing a priority for the process of risk mitigation.

4. *Mitigation Planning:* this phase involves designing measures to address the prioritized threats.

5. *Validation and Maintenance:* in the final phase, the implemented mitigations are verified and the treat model is maintained, in order to ensure that it remains up to date as the system evolves.

Numerous methodologies have been proposed for performing a threat modeling process. While many of these approaches focus primarily on security, e.g., STRIDE [126] and PASTA [139], others specifically address privacy concerns, e.g., LINDDUN [94]. In the area of information security, many organizations have also introduced various risk assessment frameworks, including NIST SP 800-30 [80] and ISO/IEC 27005 [78]. Despite differences in their methodologies, these frameworks share a common perspective: risk is defined as an unanticipated event that could harm business assets, whether tangible (e.g., physical infrastructure) or intangible (e.g., services provided by the organization).

In the context of remote voting, [119] proposes a multi-criteria evaluation method to analyze the vulnerabilities and risks associated with physical and remote voting systems, while [113] introduces a threat tree as a tool for risk assessment in e-voting.

6.2 Proposed Methodology for the Threat Analysis of an E-Voting System

In this Section, we present a novel methodology for performing a threat analysis on e-voting systems.

The System Characterization lays the groundwork for understanding the system architecture, its components, and the data flow, which are crucial for identifying potential vulnerabilities and threats in the subsequent phases. It also helps to scope the analysis by making security and trust assumptions about the system components.

The first step is to identify the assets, which are any valuable elements that need to be protected from potential threats. This may include data or services that, if compromised, could lead to adverse consequences. It is

also useful to identify, for each asset, the entities involved in the protocol that could access it and when they could do so. It should be noted that this characterization is contingent upon the precise protocol under analysis.

Regarding the security and trust assumptions to be used in the analysis, some elements could be considered out of scope and therefore not included in the model[1]:

- the integrity and reliability of the authentication of voters through their digital identities and the supporting infrastructure and communication channels;
- the integrity and reliability of the public key infrastructure that verifies the identities behind public keys and reliably distributes the keys to all parties;
- the security of the cryptography used: encryption is secure, signatures are unforgeable, and zero-knowledge proofs are sound and complete;
- all entities have access to a reasonably reliable clock that provides a common time;
- the integrity and reliability of the store used by voters to download the voting application.

6.2.1 Threat Categorization: STRIDE and LINDDUN for E-Voting

Currently, there is no standard framework for assessing threats to an e-voting system, therefore we adapt the STRIDE [126] methodology and the LINDDUN [94] framework to the e-voting scenario, as recently proposed also by [56].

STRIDE [126] is a framework developed by Microsoft to categorize security threats into six distinct types: Spoofing (SP), Tampering (TA), Repudiation (RE), Information Disclosure (ID), Denial of Service (DS), and Elevation of Privilege (EP). This framework is particularly well-suited to analyze IT infrastructures with extensive attack surfaces, such as Internet voting systems. However, when applied to e-voting systems, the *Information Disclosure* category alone is insufficient to capture all potential privacy-related threats that

[1]Note that these assumptions are also consistent with a security assessment of a public service infrastructure with trusted entities, and accessible via a mobile application.

voters or the system may face. To address this limitation, it is necessary to complement STRIDE with the LINDDUN framework, which is specifically designed for privacy threats.

LINDDUN [94] is a privacy-focused framework that categorizes threats into seven distinct types: Linking (LN), Identifying (IF), Non-Repudiation (NR), Detecting (DT), Data Disclosure (DD), Unawareness and Unintervenability (UU), and Non-Compliance (NC). It is designed to model risks arising from the connections established between individuals and the data they generate. In the context of e-voting systems, the *linking* and *identifying* categories are particularly significant.

These threats are highly relevant to e-voting systems, both remote and physical, as they involve two critical requirements:

1. voters must be authenticated to verify their eligibility and uniquely identified to prevent double voting;

2. the voting choices must remain completely unlinkable from the voter's identities throughout the tallying process and in the published results, ensuring voter privacy and preventing any potential loss of confidentiality.

In this work, we reformulate the STRIDE [126] and LINDDUN [94] threat categories, adapting them specifically to the e-voting context.

- *Spoofing (SP):* pretend to be a trustworthy entity to gain unauthorized access to sensitive voting or voter's data. This could mean impersonate a voter to vote for a different candidate (e.g., by gaining access to their device), or an authority service to intercept a voter's request for the voting credential.

- *Tampering (TA):* maliciously change or modify data stored or in transit. This can be performed on any of the communication channels or by compromising one of the web servers or the voters' devices.

- *Repudiation (RE):* perform prohibited operations without leaving traces by violating the verifiability of the e-voting system. More precisely, a repudiation attack manages to conceal evidence that a forbidden action has been performed or that some task has not been executed correctly; while the verifiability required from an e-voting systems prescribes that it is possible to check the correct execution of the protocol.

- *Information disclosure (ID):* read data stored or in transit without the necessary permissions. This results in leaking sensitive data, e.g., by compromising one of the entities or listening on communication channels.
- *Denial of Service (DS):* deny access to resources, such as by making a web server temporarily unavailable or unusable.
- *Elevation of privilege (EP):* gain privileged access to resources in order to gain unauthorized access to information or to compromise a system.
- *Linking (LN):* associate data items or voter actions to learn more about a voter or groups of voters. For instance, by associating a voting credential with the identity of a voter.
- *Identifying (IF):* learn the identity of a voter, for instance in case of compromising a voter's device.
- *Non-Repudiation (NR):* being able to attribute an action to a voter, for instance in case of an over-the-shoulder attack. This is the core threat category for coercion scenarios.
- *Detecting (DT):* deduce the involvement of a voter through observation, e.g., while listening on the communication channel between the voter's device and the web servers. This threat category is also crucial in coercion scenarios.
- *Data Disclosure (DD):* excessively collect, store, process, or share voters' personal data.
- *Unawareness and Unintervenability (UU):* when individuals are not sufficiently informed, involved, or empowered concerning processing of their personal data.
- *Non-Compliance (NC):* when the system deviates from legislation, regulation, or from standards and best practices.

Unawareness and Unintervenability and *Non-Compliance* are considered out of scope for our threat analysis methodology. In fact, awareness is raised before voting begins through channels that lie outside the scope of the protocol in use, and compliance needs to be assessed against regulations and guidelines applicable to each concrete instance. Indeed, compliance with individual frameworks has been the subject of research for individual systems [112].

6.2.2 Attackers

Attackers are entities that are motivated to find and exploit vulnerabilities to achieve malicious goals according to their abilities and opportunities. We identify the most relevant attackers for an e-voting system, with their capabilities:

1. *Coercer*: this attacker aims to manipulate the outcome of an election by forcing voters to vote in a specific way or abstain from voting. They can, for instance, observe the voter and request recordings of their actions.

2. *Network Attacker*: this attacker has the ability to snoop and tamper with all Internet traffic, albeit with limited capabilities of DOS.

3. *Internal Attacker*: this attacker can corrupt and control some trusted entities. They can, for instance, corrupt some of the entities responsible for tallying ballots.

4. *Device Cracker*: this attacker compromises the device or devices that the voter uses to vote and perform verifications checks.

5. *Curious Authority*: this attacker can snoop but not tamper with everything managed by the authority responsible for voter authentication.

In our proposed methodology one should analyze, for each attacker, which threats it could pose to the identified assets according to the attack vectors at their disposal and the disruptions that enable further threats.

Beyond active adversaries, it is also crucial to consider the consequences of catastrophic events that could compromise the integrity or availability of the voting process. To analyze such scenarios, we model them under the name of a hypothetical adversary, *Disaster*, although it does not represent an intentional attacker in the traditional sense.

6.2.3 Rating the Risk of Threats

The OWASP Risk Rating Methodology [110] is an approach to quantify the risk of security threats in order to make informed decisions. It evaluates the risk as the product of *likelihood* – i.e., how likely/easily a vulnerability is discovered and exploited by an attacker – and *impact* – i.e., material and

non-material damage, such as the loss of data integrity or the reputation damage.

Another contribution of our work is the adaptation of the OWASP risk-rating methodology to the e-voting scenario. As mentioned before, our focus is to guarantee coercion resistance, thus we tailored it on this scenario. Future works will expand this adaptation to include all the other identified attackers.

The likelihood factors proposed by OWASP provide a robust basis for assessing general security risks. However, in the context of coercion in e-voting, these factors need to be adapted to take into account the unique dynamics of this scenario.

6.2.3.1 Likelihood

To tailor the OWASP Risk Rating methodology to e-voting, considering the coercion scenario, we propose the following *likelihood factors* (scale 1 to 9) [110]:

- *Skill Level.* How technically skilled is a coercer, in particular, the lower it is, the easier it is to attack the voter. We consider a value of (2) if an asset is disclosed by physical observation or by simply requesting it from the voter; (4) if the coercer needs to access a voter's device; (6) if the coercer is able to access a communication channel to/from the voter's devices; (8) if the attacker can perform combined attacks (e.g., access the communication channel and over-the-shoulder attack).

- *Motive.* How motivated is the coercer to carry out the attack. In particular, if the attack enables voter monitoring and control, we consider the motivation to be high. We propose a rating of (2) if the monitoring is not specifically on ballot casting and has low probability of being successful; (4) if the monitoring is on ballot casting but succeeds with low probability; (6) if the attacker knows that a ruse PIN has been requested; (8) if the attack would give some direct control over voting capabilities. Intermediate values should be used to adapt for higher success probabilities or monitoring that allows the coercer to detect coercion evasion strategies.

- *Opportunity.* What resources and opportunities are needed to perform the attack. We consider a value of (3) if the coercer needs to tamper with an external software supposed to be secure (e.g., a credential manager used by the voter); (4) if the attacker needs to know when

the voter device will communicate (in order to attack); (5) if the attacker has to be able to attack a voter in two or more different time frames; (6) if the attacked asset can be retrieved by simply observing the voter; (7) if the attacked asset is visible on the device during one of the operations necessary to cast a vote.

- *Size.* How large is the group of voters under coercion. We propose a rating of (1) if it is between 10% and 20% of the electorate; (2) if it is between 20% and 30%; and so on until (9) if it is between 90% and 100%.

- *Ease of Exploit.* How easy is it for this group of threat agents to actually exploit this vulnerability. In this case, we followed the OWASP rating: (1) theoretical; (3) difficult; (5) easy; (9) automated tools available.

- *Evasion of Attack Detection*[2]. How likely is an exploit to be detected. We propose a rating of (3) if the voter is promptly alerted or the attack is easily noticeable (e.g., they realize to have lost the voting device); (6) the voter may not realize to have been attacked if proper care has not been used (e.g., by not leveraging optional security mechanisms such as verification steps); (8) the attack can easily go unnoticed (e.g., in case a coercer passively listens on communication channels).

Since our main focus is the coercer, we choose to do not consider *Ease of Discovery* (i.e., how easy an attacker discovers this vulnerability) and *Awareness* (i.e., how well known is this vulnerability to the attacker) since these aspects are more related to the individual's personal characteristics, making them difficult to quantify. Intermediate values (such as a motive level of 3) are assigned by comparison among similar attacks (or vulnerable assets), and lastly by using the values from the OWASP methodology. The overall likelihood of each identified threat is computed as the average value of all the considered factors.

While we followed and adapted OWASP criteria and scales for likelihood factors, the traditional impact factors used to assess risk are insufficient for the e-voting context. These factors were primarily developed for scenarios involving cyberattacks on corporate IT infrastructures and focus mainly on financial damage. While these considerations are relevant, they do not adequately cover the consequences of attacks on the aggregated properties described in Section 1.2.1.

[2]This is named Intrusion Detection by OWASP. We decided to modify its name to make it easier to understand.

To address this limitation, we propose a new approach that measures the impact of successful attacks based on their effect on the aggregated e-voting properties outlined in Section 1.2.1 (see Table 1.2).

6.2.3.2 Impact

To evaluate the impact of a coercion attack on an e-voting system we provide, below, the new *impact factors* (scale 1 to 9) and how to estimate their value.

- *Functional and Security Requirements.* An eligible voter cannot cast a vote, but manages to solve the issue (1); votes can be modified or canceled but the issue can be solved (3); a few eligible voters are unable to vote or votes can be modified, a few ineligible voters can vote (5); multiple eligible voters are unable to vote, multiple ineligible voters can vote, several votes can be modified, results are disclosed ahead of time (7); several voters are unable to conclude the voting process (9).
- *Vote Freedom.* Evasion strategies unlikely to be detected (1); evasion strategies detected if the voter does not follow the recommendations (2); evasion strategies of a specific voter detected before voting phases (3); knowledge that a specific voter cast a vote[3] (5); knowledge of content but not validity of a vote cast by a specific voter (6); knowledge of validity but not content of a vote cast by a specific voter (7); knowledge of content and validity of a vote cast by a specific voter (9).
- *Verifiability.* A single verification does not work, the issue can be detected and immediately solved (1); a single verification does not work, the issue can be detected and solved but not immediately (3); a single verification is missing or does not work (4); a single verification has been falsified and this goes unnoticed (6); tally proofs are missing, some voters can cast multiple valid votes unnoticed (7); tally proofs have been falsified and this goes unnoticed, or many voters can cast multiple valid votes unnoticed (9).

We consider an oversimplification to compute an overall impact as the average of the impact factors, since the three aggregated properties are damaged differently by different attackers. Therefore we propose to use a weighted average, giving more consideration to the aggregated properties affected by more attacks with non-zero impact value.

[3]Excluding if the attacker tells the voter to do so.

6.2.3.3 Risk

For each threat, we compute its associated *risk* separately for each aggregated property. The risk value of the attack is the multiplication of its likelihood by its impact. We divide the risk into 6 levels according to a set of thresholds in line with the intervals from OWASP: the risk is *None* if it is exactly 0, *Very Low* if it is below 3.5, *Low* if it is less than 10.0, *Medium* if it is below 24.0, *High* if it is less than 47.0, and *Critical* if it is 47.0 or greater.

Finally, we propose to aggregate the results by dividing the attacks according to the phase during which they can be performed, thus giving some insight into which parts of the system are more vulnerable.

6.3 Application to Vote App

In this Section, we apply the threat modeling methodology developed in Section 6.2 to evaluate the security of *Vote App* described in Chapter 3 and illustrated in Figure 3.1. For the risk analysis, we will focus specifically on the Coercer. This approach will allow us to assess our methodology and to identify, categorize, and analyze potential security threats to *Vote App*.

The work behind this evaluation is extensive (we identified 164 assets, and the table used for the threat modeling spans 462 rows), thus here, due to space limitations, we will only present a part of it. For the full details, please refer to [36].

Protocol Phases. As outlined in Section 3.3, we report here the phases in which *Vote App* voting protocol is divided: (1) setup; (2) enrollment; (3) PIN and device management; (4) voting and (5) tallying.

6.3.1 System Characterization

Protocol Assumptions. An untappable channel is generally required to issue the valid voting private credential, so that the coercer cannot intercept this communication and learn the real voting credential. The suggested implementation for this channel is typically the physical presence of the voter, which is difficult to reconcile with voters residing abroad. We try to soften this requirement by making the following assumptions:

- *A1 Surveillance gaps.* The coercer cannot continuously physically oversee the voter - in other words, there will be wide surveillance gaps in which the voter is free to act.

- *A2 A threshold of honest RT.* There is at least one trusted RT that does not collude with the coercer. More precisely, at least one trusted RT is required to construct a voting credential, and at least one trusted RT is required to be available to respond to fake PIN requests without informing a colluding coercer. If there are $n_{\mathtt{RT}}$ RTs and a threshold of $t_{\mathtt{RT}}$ RTs is required to build a voting credential from the shares given by each RT, then we require that at least $n_{\mathtt{RT}} - t_{\mathtt{RT}} + 1$ RTs are trusted, and all have to cooperate with a voter employing an evasion strategy. For the sake of simplicity, in the high-level description we assume $t_{\mathtt{RT}} = n_{\mathtt{RT}}$, so that one trusted RT is enough.

We also make the assumptions below to guarantee the security and functionality properties of the e-voting solution, in addition to *A1* and *A2*.

- *A3 Strong Web Bulletin Board.* We assume the integrity of the Web Bulletin Board. The data contained on the WBB is assumed to be stored and backed up (and replicated) using a write once, read many model. This implies that everyone has access to the same (updated) version.

- *A4 Honest Electoral Roll.* We assume the ER operates correctly, authorizing all eligible voters and no ineligible voters.

- *A5 Reliable eID.* The login to the digital identity provider is out of scope of the analysis. We assume that the communication channel with the identity provider is secure and tamper-proof, ensuring that a successful login provides the relying party with accurate and tamper-proof personally identifiable information. Additionally, the user is properly informed about the relying party receiving their personally identifiable information, and logins are restricted to the intended relying party, making them unusable for any other audience.

- *A6 Robust PKI.* We assume that the public-key infrastructure, that enables mutual TLS between servers, correctly checks the identities behind public keys and distributes keys reliably to all parties.

- *A7 Safe Crypto.* Encryption is secure, signatures are unforgeable, ZKPs are sound and complete.

- *A8 Synchronized clock.* All parties can access a fairly reliable clock, giving a common time. In particular, this is to ensure all ballots received by BBs are consistently timestamped.

- *A9 At least one honest BB.* At least one must not delete received ballots while publishing their hash digest.

- *A10 Safe app store.* The origin of the voting app is controlled and cannot be tampered with, the app store cannot be taken down indefinitely.

These assumptions define the basic context for our analysis and system design.

Threats and Attack Vectors. In order to analyze the security of the protocol, we determined a list of threats that could disrupt our e-voting solution and we singled out the related attackers from the classification reported in [116]. Table 6.1 lists the principal threats that we identified, together with the attack vectors that could be exploited by the previously considered attackers.

Table 6.1: Main threats and attack vectors considered in our threat analysis.

Threat	**Attack Vectors**		
DOS	PKI/channel DOS	server/device failure	forced abstention
leak	server/device/app compromise	malware exfiltration from storage	forced disclosure, over-the-shoulder, social engineering
loss	storage failure	ransomware on storage	
tampering	channel tampering	server/device/app compromise	malware on server/device
(continuous/partial) snooping	channel sniffing	pre-TLS monitoring	server/device/app compromise

Attackers. As discussed before, we suppose that the coercer is the primary adversary in our threat analysis. The coercer can surveil the communication channels between the voting device and the other entities, sniffing encrypted traffic, but their monitoring and surveillance capabilities are limited to discontinuous time frames, with the goal of ensuring that the voter comply with the coercer's instructions. We model this monitoring ability with three attack vectors:

- *over-the-shoulder*: the attacker can observe the voter while it interacts with the voting device;
- *social engineering*: the attacker can deceive and manipulate voters to obtain information or influence their actions;
- *channel sniffing*: the attacker can observe the encrypted exchange of messages between *Vote App* and the ER[4], BB, RT and NS.

While the coercer poses a significant threat, other adversaries also play a critical role in compromising the security of the voting process. Below we describe the attack vectors of the others, previously identified (see Section 6.2.2), attackers.

- *Network Attacker.* Channel DOS, channel sniffing and channel tampering between all the entities.
- *Internal Attacker.* Malware exfiltration from storage of RT, TT, NS, BB; malware on RT, TT, BB, NS; ransomware on storage of RT, TT, NS, BB; RT, TT, NS, BB compromise; RT, TT, NS, BB storage failure; RT, TT failure; RT, TT, NS, BB pre-TLS monitoring.
- *Device Cracker.* Channel DOS (*Vote App* - ER, BB, NS, RT, VS); channel sniffing (*Vote App* - ER, BB, NS, RT, VS); channel tampering (*Vote App* - ER, BB, NS, RT, VS); malware exfiltration from *Vote App* storage; malware on *Vote App*, VS; ransomware on storage of *Vote App*; *Vote App*, VS failure; *Vote App* storage failure; *Vote App*, VS compromise; *Vote App*, VS pre-TLS monitoring.
- *Curious Authority.* channel sniffing (ER - *Vote App*, NS, WBB); ER pre-TLS monitoring; malware exfiltration from ER storage.

[4]All the entities and acronyms are introduced in Section 3.2 and Chapter 8.

- *Disaster.* ransomware on storage of *Vote App*, NS, RT, TT, BB, ER; *Vote App*, VS, RT, TT, BB, ER failure; *Vote App*, RT, TT, BB, ER storage failure.

While *Disaster* is not a real adversary, modeling it as such allows us to assess the resilience of the system under extreme conditions.

Assets. The list of all assets of the protocol under analysis is available at [36]. Table 6.2 presents an excerpt for the *PIN*, the *Vote*, and the *communication channel* between the voter device and the ballot box. This includes, together with a short description, the identification of both the entities involved in the protocol (see Figure 3.1) that could access the asset and of the phase in which this can be done.

Table 6.2: An excerpt of identified assets. The *Access* column lists the entities that can access the asset, in the phases indicated between parentheses.

Asset	Description	Access
PIN	Code that unlocks a voting credential	Voting Device (2, 3, 4), Voter (2, 3, 4)
Vote	Preference expressed by the voter	Voting Device (4), Voter (4)
VD-BB channel	Communication channel between the Voting Device (VD) and the Ballot Box (BB)	Voting Device (4), Ballot Box (4)

6.3.2 Threat Modeling

Following the methodology, in this second phase we start by considering, for each asset identified in Section 6.3.1, the consequences of the threats it could face. Afterwards, for each identified threat, we categorize it with the adapted STRIDE and LINDDUN frameworks (see Section 6.2.1). The result of this categorization is presented in Table 6.3, which outlines the number of compromised assets.

Subsequently, we refined our analysis by marking in which phases the threat could be posed and which attack vectors could be used to exploit

Table 6.3: Number of assets impacted by STRIDE and LINDDUN threats.

	STRIDE						**LINDDUN**				
	SP	**TA**	**RE**	**ID**	**DS**	**EP**	**LN**	**IF**	**NR**	**DT**	**DD**
Assets	14	114	8	100	80	14	10	6	33	20	4

it. Finally, we identified the disruption caused by successful attacks, and whether they generate other threats. Table 6.4 presents an excerpt of the results, the full analysis can be found at [36].

For example, consider the asset *PIN* and the *Coercer* attacker (see line 1 of Table 6.4). A *leak* of the PIN, i.e., when the coercer learns it, may lead to a coercion attack. In fact, if the valid PIN is known, the attacker may force the voter to cast a vote complying with the given instructions and, while observing them voting, determine whether the voter is actually casting a valid vote. This threat can be categorized by STRIDE as an *information disclosure* (ID), and by LINDDUN as *non-repudiation* (NR) because the coercer could determine whether their instructions have been followed.

The phases in which the voting device displays the PIN to the voter are the enrollment (phase 2), and whenever a reminder is requested in the pin and device management (phase 3). Note also that, when the voter sets up a *ruse PIN* (phase 3), the voting device displays it exactly as in the previous cases. Finally, the PIN is visible on the voting device for a short period of time when it is typed in during voting (phase 4). Therefore the coercer can learn the PIN at these times with an *over-the-shoulder* attack. However, since the coercer cannot continuously monitor the voter, and a ruse PIN is always displayed exactly as the valid PIN, the attacker is not sure of the validity of the PIN learned. The PIN could also be leaked through social engineering techniques, such as pretending to be the official voting authority support service, but even in this case there is a possibility that actually a ruse PIN is leaked instead of the valid PIN.

Note that compromising the voting device (e.g., with a malicious app or malware) could also lead to a PIN leak. However, this attack vector is not among the capabilities of the coercer (but can be performed by the *device cracker*).

Additionally, to give an intuition of how each attacker threatens the e-

voting system, we summarize in Tables 6.5 to 6.10 which type of threats each attacker can pose in each phase.

Table 6.4: Excerpt of Threat Analysis. The *Cat.* column categorizes the attack according to STRIDE and LINDDUN methodologies, listing all relevant types.

Asset	Threat	Cat.	Phases	Attack Vectors	Disruptions
PIN	leak	ID, NR	2, 3, 4	over-the-shoulder, app compromise, social engineering	coercion attack[1]
Vote	tampering	TA, DS	4	app compromise, malware on app	tampering of encrypted ballot[2]
VD-BB channel	DOS	DS	4	channel DOS	inability to cast a vote
VD-BB channel	snooping	DT	4	channel sniffing	coercion attack[3]

[1] Only if the attacker can reliably establish the validity of the PIN.
[2] If the voter performs the individual verifiability checks, the tampering is detected.
[3] If the attacker instructed the voter to abstain, it may detect a defiant ballot casting.

6.3.3 Risk Analysis

For the coercer, from the analysis emerges that there are a total of 44 successful attacks. However, only 3 have a non-zero impact value for *Functional and Security Requirements*; 40 have non-zero impact for *Vote Freedom*, and 3 for *Verifiability*. Following the methodology presented in Section 6.2.3, we assign for each aggregated property the weights of 0.068 (i.e., $\frac{3}{44}$), 0.909 ($\frac{40}{44}$), and 0.068 ($\frac{3}{44}$) respectively.

In this risk analysis of the coercer, we choose not to consider the *Size*

Table 6.5: Assessment of the phases impacted by STRIDE and LINDDUN threats posed by the Coercer: ● indicates that the phase is impacted while ■ signifies that the phase is not impacted by the corresponding threat.

	STRIDE						LINDDUN				
Phase	**SP**	**TA**	**RE**	**ID**	**DS**	**EP**	**LN**	**IF**	**NR**	**DT**	**DD**
1	■	■	■	■	■	■	■	■	■	■	■
2	■	●	■	●	●	■	●	●	■	●	■
3	■	●	■	●	●	■	●	●	●	●	■
4	■	●	■	●	●	■	●	●	●	●	■
5	■	■	■	■	■	■	■	■	■	■	■

factor, as it depends to much on the context in which the election is carried out.

To give an example of the risk analysis, let us consider once again the threat "PIN leak" (see line 1 of Table 6.11). Regarding the skill level, we consider it to be low but not null, so we assigned a value of 2. The motive is high, but we scale the value down to a 7 since the observed PIN can be a ruse one. For the opportunity, some access or resources are required so it is a 7, and the exploit is easy, so it is a 5. Regarding the attack detection, if the coercer asks directly for the PIN obviously the voter is aware of it. Meanwhile, an over-the-shoulder attack is sneakier. Considering that voting operations are sensitive, so they are not usually performed in public spaces, we mediate the two cases with an average of 3. Finally, the leak of a PIN does not impact *Verifiability* or *Functional and Security Requirements*, while for *Vote Freedom* the worst case is that with an over-the-shoulder attack the coercer detects a vote casting (impact 5), but considering all the other cases, we scaled it down to a 4. This example, together with the snooping of the channel between the voter device and the ballot box, are presented in Table 6.11.

For each threat, we compute its associated risk separately for each aggregated property by multiplying the likelihood of an attack and its impact. To continue the previous example, PIN leak has a likelihood of 5.8, and a

Table 6.6: Assessment of the phases impacted by STRIDE and LINDDUN threats posed by the Network Attacker: ● indicates that the phase is impacted while ■ signifies that the phase is not impacted by the corresponding threat.

	STRIDE						LINDDUN				
Phase	**SP**	**TA**	**RE**	**ID**	**DS**	**EP**	**LN**	**IF**	**NR**	**DT**	**DD**
1	■	●	■	■	●	■	■	■	■	■	■
2	■	●	■	●	●	■	■	■	●	●	■
3	■	●	■	●	●	■	■	■	●	●	■
4	■	●	■	■	●	■	●	■	■	●	■
5	■	●	■	■	●	■	■	■	■	■	■

risk value of 23.2 (Medium) for *Vote Freedom*, while the other two are not impacted, hence are zero. This gives a total risk value of 21.2 (Medium).

In Table 6.12, we report the results obtained by analyzing the coercer and the overall risk. The numbers in the tables indicate how many successful threats with the selected risk level are posed in the corresponding phase (e.g., the coercer poses 1 threat to the property *Functional and Security Requirements* in phase 3 with risk level *Very Low*). It is important to note that some attacks may affect more than one phase and more than one property. Therefore, the total number of threats is not simply the sum but we account for the overlapping influence of certain threats on multiple phases and properties by ignoring intersections.

Table 6.7: Assessment of the phases impacted by STRIDE and LINDDUN threats posed by the Internal Attacker: ● indicates that the phase is impacted while ■ signifies that the phase is not impacted by the corresponding threat.

	STRIDE						LINDDUN				
Phase	**SP**	**TA**	**RE**	**ID**	**DS**	**EP**	**LN**	**IF**	**NR**	**DT**	**DD**
1	■	●	■	●	●	■	■	■	■	●	■
2	■	●	■	●	●	■	■	■	●	●	■
3	■	●	■	●	●	■	■	■	●	●	■
4	■	●	■	●	●	●	●	■	●	●	■
5	■	●	■	●	●	■	■	■	■	■	■

Table 6.8: Assessment of the phases impacted by STRIDE and LINDDUN threats posed by the Device Cracker: ● indicates that the phase is impacted while ■ signifies that the phase is not impacted by the corresponding threat.

	STRIDE						LINDDUN				
Phase	**SP**	**TA**	**RE**	**ID**	**DS**	**EP**	**LN**	**IF**	**NR**	**DT**	**DD**
1	■	●	■	●	●	■	●	●	■	■	●
2	●	●	●	●	●	●	●	●	●	●	●
3	■	●	●	●	●	■	●	●	●	●	●
4	●	●	●	●	●	●	●	●	●	●	●
5	■	●	■	●	●	●	●	●	●	■	●

Table 6.9: Assessment of the phases impacted by STRIDE and LINDDUN threats posed by the Curious Authority: ● indicates that the phase is impacted while ■ signifies that the phase is not impacted by the corresponding threat.

	STRIDE						LINDDUN				
Phase	**SP**	**TA**	**RE**	**ID**	**DS**	**EP**	**LN**	**IF**	**NR**	**DT**	**DD**
1	■	■	■	●	■	■	●	●	■	■	●
2	■	■	■	●	■	■	●	●	■	●	●
3	■	■	■	●	■	■	●	●	■	●	●
4	■	■	■	●	■	■	●	●	●	●	●
5	■	■	■	●	■	■	●	●	■	■	●

Table 6.10: Assessment of the phases impacted by STRIDE and LINDDUN threats posed by the Disaster: ● indicates that the phase is impacted while ■ signifies that the phase is not impacted by the corresponding threat.

	STRIDE						LINDDUN				
Phase	**SP**	**TA**	**RE**	**ID**	**DS**	**EP**	**LN**	**IF**	**NR**	**DT**	**DD**
1	■	■	■	■	●	■	■	■	■	■	■
2	■	●	■	■	●	■	■	■	■	■	■
3	■	●	■	■	●	■	■	■	●	■	■
4	●	●	■	■	●	●	■	■	■	■	■
5	■	●	■	■	●	■	■	■	■	■	■

Table 6.11: Example of Likelihood and Impact Assessment.
Legend: S = Skill Level, M = Motive, O = Opportunity, E = Ease of Exploit, D = Evasion of Attack Detection, R = Functional and Security Requirements, F = Vote Freedom, V = Verifiability.

	Likelihood					**Impact**		
Threat	**S**	**M**	**O**	**E**	**D**	**R**	**F**	**V**
PIN leak	2	7	7	5	3	0	4	0
VD-BB channel snooping	5	4	4	6	8	0	4	0

Table 6.12: Phase-by-phase (**P**) breakdown of successful attacks posed by the *Coercer* along with associated risks (see **Legend** below), and both total (**T**) and per phase average (**A**) risk.
Legend: ●= Very Low Risk, ●= Low Risk, ●= Medium Risk, ●= High Risk, ●= Critical Risk, **A**= Average Risk.

	Functional and Security Requirements						**Vote Freedom**						**Verifiability**						**Total**					
P	●	●	●	●	●	**A**	●	●	●	●	●	**A**	●	●	●	●	●	**A**	●	●	●	●	●	**A**
1	0	0	0	0	0		0	0	0	0	0		0	0	0	0	0		0	0	0	0	0	
2	0	0	0	0	0		0	**5**	**5**	0	0	●	0	0	**3**	0	0	●	**1**	**7**	**3**	0	0	●
3	**1**	0	0	0	0	●	0	**8**	**10**	0	0	●	0	0	**3**	0	0	●	**1**	**11**	**7**	0	0	●
4	**2**	0	0	0	0	●	0	0	**20**	**4**	0	●	0	0	**1**	0	0	●	**1**	**1**	**19**	**4**	0	●
5	0	0	0	0	0		0	0	0	0	0		0	0	0	0	0		0	0	0	0	0	
T	**3**	0	0	0	0	●	0	**8**	**27**	**4**	0	●	0	0	**3**	0	0	●	**1**	**11**	**24**	**4**	0	●

6.3.4 Mitigation Planning

Here we discuss the results reported in Table 6.12, giving some possible mitigations.

Vote Freedom. This is the most impacted property, as it should be expected given the nature of the attacker we are considering. The highest risks come from attacks during the voting phase which threaten the vote expressed by the voter. In fact, the vote is a critical asset, and its leak has a high impact even if the ballot itself is not valid. The reason behind this classification is that despite the possibility to employ evasion strategies to disguise the validity of a vote, it is not certain that these strategies have been employed, or whether they have been correctly executed. To have additional guarantees on their correct adoption, one of the strategies is to make all verification mechanisms mandatory in order to complete the voting process. However, this could reduce the usability of the service. It is therefore important to understand the trade-off between security and usability that works best in each specific context.

In general, this property may be impacted when the voter fails to follow an evasion strategy or the coercer detects defiance by chance. The lesson learned is that to contrast a coercer it is vital to deploy evasion strategies correctly, which requires great care and good instructions.

To improve the voting experience and reduce risks, the protocol offers features such as PIN reminders for forgotten credentials, the ability to cast multiple ballots to correct errors, and the ability to revoke a compromised voting credential and re-register if devices are lost or stolen. In addition, *Vote App* includes a comprehensive and accessible guide that instructs voters on necessary security measures throughout the voting process.

Functional and Security Requirements. The risks on this property derive from attacks where the coercer takes away the voter's voting device. Since it is sufficient to re-register with another device to solve the problem, the risk is very low. The other threats do not have an impact on this property.

Verifiability. To explain the attacks that impact *Verifiability* we need to recall a few details on the registration phase. When the voter registers to the e-voting system for the first time, they are supposed to safely back-up some information (e.g., via a credential manager), which is needed to verify the

security and correctness of the initialization of further voting devices. If the coercer manages to destroy or corrupt this backup, then the voter will not be able to perform these verifications. The impact is medium (value 4) for all these attacks, the likelihood is also medium (around 4), so the risk level is medium.

Chapter 7

Amun Voting Protocol

This chapter introduces the Amun voting protocol, starting with a high-level overview in Section 7.1 and then describing the protocol itself in Section 7.2. A brief discussion of its security is given in Section 7.3.
The content of this chapter is based on the work [99], which extends the two-candidates protocol of [134].

7.1 Introduction

This chapter presents the extension of the protocol [134] to handle an election with N voters, where each one expresses P preferences among M candidates (obviously $P < M$). The basic idea is that every voter owns M voting tokens (*v-tokens*): P are valid, the others are a decoy, but only the voter knows which is which. When voting, voters express their preferences assigning the valid *v-tokens* to the chosen candidates and the decoy ones to the others.

Compared with the two-candidates protocol [134], this generalization forsakes the blockchain infrastructure in favor of a more classical web bulletin board and introduces an additional authority, that is required in order to properly mask the multiple valid and decoy tokens in each ballot, so that the system remains secure even if one authority is corrupt.

The protocol allows for re-voting: before tallying duplicate ballots (i.e. with the same *v-tokens* ignoring their order) are discarded, keeping only the most recent. After the voting phase, when counting the votes, the decoy *v-tokens* do not contribute to the tally, so only valid *v-tokens* are counted. The whole process is publicly auditable and fully verifiable, and preserves

privacy as long as at most one authority is corrupt.

The protocol is divided into four phases:

- **Setup.** Three authorities, knowing a list of eligible voters, generate the values for the creation of both the *v-tokens* and the masks associated to the candidates. These masks guarantee the voters' privacy, and prevent early tallying.
- **Registration Phase.** In this phase, the three authorities engage in a 5-step protocol (see Figure 7.1) to create M indistinguishable *v-tokens* (P are valid and $M-P$ are a decoy) employing masking and shuffling so that at the end the authorities will not be able to identify which tokens are valid. The voter can check the validity of these *v-tokens* thanks to DVNIZKPs issued by the authorities. These proofs are worthless for a coercer because the voter can forge them.

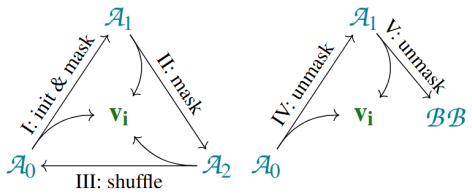

Figure 7.1: The main steps of the ballot generation procedure, which correspond to steps 3-7 of the Registration Phase as described in Section 7.2.2.

- **Voting Phase.** During this phase the voter express their preferences by assigning each of their M *v-tokens* to the candidates. All *v-tokens* of a voter must be assigned together, each to a distinct candidate. After the *v-tokens* have been assigned, the voter gets a transcript that reports the assignment of the *v-tokens* to the candidates. This transcript is worthless for a coercer since the *v-tokens* are indistinguishable. Here we assume that every candidate receives at least one legitimate vote (with a valid *v-token*), otherwise it is trivial to discern the validity of some tokens from the election results.
- **Tallying.** The *v-tokens* are processed (see Figure 7.2), removing the candidate masks, which allows to count the number of valid and decoy tokens assigned to each candidate. The results and the intermediate computations are published, alongside a set of NIZKPs that allow anyone to check that the results are correct and there has not been

any tampering. Every voter can also check, by examining the bulletin board, that their *v-tokens* have been cast and counted correctly.

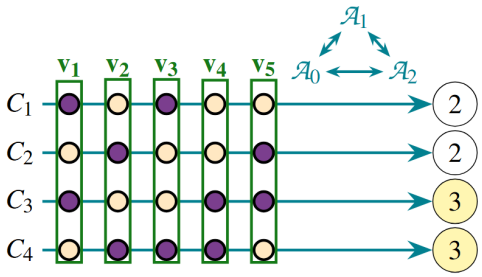

Figure 7.2: Example of voting and tallying. Each voter has two valid tokens, represented by the purple circle, and two decoy tokens, represented by the yellow one. After the tallying it is revealed that candidates C_3 and C_4 are elected having received more preferences (3) with respect to the other two candidates (who received only 2).

Notation. We use the following notation for the indexes: $[n] = \{i \in \mathbb{N} : 1 \leq i \leq n\}$, $(t_j)_{j\in[m]} = (t_1, \ldots, t_m)$.

7.2 Protocol Description

The key components involved in the protocol are:

1. a finite set of voters $V = \{v_i\}_{i\in[N]}$ (where v_i is a pseudonymous id), with $N \in \mathbb{N}$ the number of eligible voters;

2. a finite set of candidates $C = \{c_\ell\}_{\ell\in[M]}$ with $M \in \mathbb{N}$ the number of candidates;

3. three trusted authorities[1] $\mathcal{A}_0$, $\mathcal{A}_1$, and $\mathcal{A}_2$;

4. one ballot b_i (comprising M *v-tokens*) for every $i \in [N]$, i.e. one for each eligible voter.

[1]We use a weak concept of trust here, since the conduct of these authorities can be checked by voters.

Throughout the protocol we implicitly assume that every public value (including a description of the key components presented above) are published in the web bulletin board.

7.2.1 Setup

The authority $\mathcal{A}_0$ selects and publishes:

1. a secure group $\mathbb{G}$ of prime order p in which the DDH assumption holds, with a generator $g \in \mathbb{G}$ (see Section 2.2);
2. a commitment scheme `Commit` to be used to commit to the values computed before publishing them, in order to improve security (see Section 2.4).

Then $\mathcal{A}_0$ performs the following operations:

1. chooses uniformly at random two values k and λ in $\mathbb{Z}_p^*$. $\mathcal{A}_0$ knows that the *v-tokens* computed using k are valid, while the ones computed using λ are decoys, but this information is kept secret;
2. chooses uniformly at random $N \cdot M$ distinct values $\bar{z}_{i,\ell} \in \mathbb{Z}_p^*$, with $i \in [N]$, $\ell \in [M]$;
3. finally, $\mathcal{A}_0$ commits to the values g^k, g^λ, and, for every $i \in [N]$, it commits to $\left(v_i, (g^{\bar{z}_{i,\ell}})_{\ell \in [M]}\right)$.

An honest authority $\mathcal{A}_0$ is supposed to keep private all the values $\bar{z}_{i,\ell}$, k, λ.

The authority $\mathcal{A}_1$ performs the following operations:

1. chooses uniformly at random M distinct values $\alpha'_\ell \in \mathbb{Z}_p^*$, with $\ell \in [M]$, these will be the first half of the candidates' masks;
2. chooses uniformly at random N distinct values $x'_i \in \mathbb{Z}_p^*$, with $i \in [N]$;
3. chooses uniformly at random two sets of $N \cdot M$ distinct values $z'_{i,\ell}, y'_{i,\ell} \in \mathbb{Z}_p^*$, with $i \in [N], \ell \in [M]$;
4. finally, $\mathcal{A}_1$ commits to the values $g^{\alpha'_\ell}$, $\forall \ell \in [M]$, and for every $i \in [N]$ it commits to the tuple $\left(v_i, g^{x'_i}, (g^{z'_{i,\ell}})_{\ell \in [M]}, (g^{y'_{i,\ell}})_{\ell \in [M]}\right)$.

An honest authority $\mathcal{A}_1$ is supposed to keep private the values α'_ℓ, x'_i, $z'_{i,\ell}$, $y'_{i,\ell}$.
The authority $\mathcal{A}_2$ performs the following operations:

1. chooses uniformly at random M distinct values $\alpha''_\ell \in \mathbb{Z}_p^*$, with $\ell \in [M]$, these will be the second half of the candidates' masks;
2. chooses uniformly at random N distinct values $x''_i \in \mathbb{Z}_p^*$, with $i \in [N]$;
3. chooses uniformly at random $N \cdot M$ distinct values $y''_{i,\ell} \in \mathbb{Z}_p^*$, with $i \in [N]$, $\ell \in [M]$;
4. Finally $\mathcal{A}_2$ commits to the values $g^{\alpha''_\ell}$, $\forall \ell \in [M]$, and for every $i \in [N]$ it commits to the tuple $\left(v_i, g^{x''_i}, (g^{y''_{i,\ell}})_{\ell \in [M]}\right)$.

An honest authority $\mathcal{A}_2$ is supposed to keep all values private α''_ℓ, x''_i, $y''_{i,\ell}$.

Once all the commitments have been published, the authorities can decommit the values:

- $\mathcal{A}_0$ publishes the decommitments for the values g^k, g^λ, alongside all the tuples $\left(v_i, (g^{\bar{z}_{i,\ell}})_{\ell \in [M]}\right) \forall i \in [N]$;
- $\mathcal{A}_1$ publishes the decommitments for the values $g^{\alpha'_\ell} \forall \ell \in [M]$, and the tuples $\left(v_i, g^{x'_i}, (g^{z'_{i,\ell}})_{\ell \in [M]}, (g^{y'_{i,\ell}})_{\ell \in [M]}\right) \forall i \in [N]$;
- $\mathcal{A}_2$ publishes the decommitments for the values $g^{\alpha''_\ell} \forall \ell \in [M]$, and the tuples $\left(v_i, g^{x''_i}, (g^{y''_{i,\ell}})_{\ell \in [M]}\right) \forall i \in [N]$.

All these published values are accompanied by NIZKPs which prove that the authority who published them knows the corresponding secret exponents. These NIZKPs can be constructed using the Schnorr protocol and the Fiat-Shamir transformation just like in Section 2.10.

To simplify notation we introduce some definitions for aggregate values for all $i \in [N]$ and $\ell \in [M]$:

$$x_i = x'_i + x''_i, \qquad \alpha_\ell = \alpha'_\ell \cdot \alpha''_\ell,$$
$$z_{i,\ell} = \bar{z}_{i,\ell} \cdot z'_{i,\ell}, \qquad y_{i,\ell} = y'_{i,\ell} \cdot y''_{i,\ell}.$$

7.2.2 Registration Phase

For every pseudonymous id $v_i \in V$ the following steps are performed[2]:

1. let Alice be the person associated to the pseudonymous id v_i, note that the authorities do not need to know this association. They go in a safe and controlled environment where they are identified and authenticated as the eligible and not yet registered pseudonymous id v_i. In this environment they can interact with all three authorities without fear of eavesdropping or interference.

2. Alice creates a signing key-pair (s_i, K_i), a designated verifier key-pair (e_i, D_i), and gives K_i, D_i to the authorities proving the knowledge of s_i (e.g. by signing a challenge message), and of e_i via the classical Schnorr zk-proof of discrete logarithm (which includes the challenge message among the public values). The authorities associate K_i, D_i to v_i in their respective voters lists.

3. $\mathcal{A}_0$ performs the following steps:

   (a) $\mathcal{A}_0$ chooses, for every $i \in [N]$, a random subset $V_i \subset [M]$ with cardinality is exactly P, then sets:

   $$\sigma_{i,\ell} = \begin{cases} k & \iff \ell \in V_i \\ \lambda & \iff \ell \notin V_i \end{cases}$$

   i.e. the random choice of the V_i determines which tokens will be valid and which a decoy;

   (b) $\mathcal{A}_0$ takes the (publicly available) values $g^{x'_i}$ and $g^{x''_i}$ and creates the step 0 of the ballot $\bar{b}_{0,i} = (\bar{b}_{0,i,l})_{\ell \in [M]}$ where, $\forall \ell \in [M]$:

   $$\bar{b}_{0,i,l} = \left(g^{\sigma_{i,\ell}} \cdot g^{x'_i} \cdot g^{x''_i}\right)^{\bar{z}_{i,\ell}} = g^{\bar{z}_{i,\ell}(\sigma_{i,\ell} + x_i)};$$

   (c) $\mathcal{A}_0$ sends to $\mathcal{A}_1$ the initial ballot $\bar{b}_{0,i}$ and sends to Alice $\bar{b}_{0,i}$ and V_i;

   (d) $\mathcal{A}_0$ proves the correctness of its computations (see Protocol 7):

[2] In this phase, we will always refer to the designated-verifier version of the non-interactive zero-knowledge proof used.

i. $\mathcal{A}_0$ proves that the $g^{\bar{z}_{i,\ell}\sigma_{i,\ell}}$ are correct (using $\sigma_{i,\ell} = k$ or $\sigma_{i,\ell} = \lambda$) with:

$$\begin{aligned} &\rho = k, && g_1 = g, && h_1 = g^k, \\ &g_2 = g^{\bar{z}_{i,\ell}}, && h_2 = g^{\bar{z}_{i,\ell}k}, && \forall \ell \in V_i, \\ &\rho = \lambda, && g_1 = g, && h_1 = g^\lambda, \\ &g_2 = g^{\bar{z}_{i,\ell}}, && h_2 = g^{\bar{z}_{i,\ell}\lambda} && \forall \ell \in [M] \setminus V_i, \end{aligned}$$

ii. then $\mathcal{A}_0$ proves that the $\bar{b}_{0,i,l}$ are correct using for all $\ell \in [M]$:

$$\begin{aligned} &\rho = \bar{z}_{i,\ell}, && g_1 = g, && h_1 = g^{\bar{z}_{i,\ell}}, \\ &g_2 = g^{\sigma_{i,\ell}} \cdot g^{x'_i} \cdot g^{x''_i}, && h_2 = \bar{b}_{0,i,l}. \end{aligned}$$

4. $\mathcal{A}_1$ computes the step 1 of the ballot $\bar{b}_{1,i} = (\bar{b}_{1,i,l})_{\ell \in [M]}$ where:

$$\bar{b}_{1,i,l} = \left(\bar{b}_{0,i,l}\right)^{z'_{i,\ell}} = g^{z_{i,\ell}(\sigma_{i,\ell} + x_i)} \qquad \forall \ell \in [M]$$

and sends it to Alice and to $\mathcal{A}_2$. Then $\mathcal{A}_1$ proves that the $\bar{b}_{1,i,l}$ are correct with (see Protocol 7):

$$\begin{aligned} &\rho = z'_{i,\ell}, && g_1 = g, && h_1 = g^{z'_{i,\ell}}, \\ &g_2 = \bar{b}_{0,i,l}, && h_2 = \bar{b}_{1,i,l} && \forall \ell \in [M]. \end{aligned}$$

5. $\mathcal{A}_2$ chooses uniformly at random a permutation $\pi_i \in \mathrm{Sym}([M])$ and computes the step 2 of the ballot $\bar{b}_{2,i} = (\bar{b}_{2,i,l})_{\ell \in [M]}$ where, $\forall \ell \in [M]$:

$$\bar{b}_{2,i,l} = \left(\bar{b}_{1,i,l}\right)^{y''_{i,\pi_i^{-1}(\ell)}} = g^{z_{i,\ell} y''_{i,\pi_i^{-1}(\ell)} (\sigma_{i,\ell} + x_i)}$$

and sends it to Alice and to $\mathcal{A}_0$, π_i is sent to Alice and $\mathcal{A}_1$. Then $\mathcal{A}_2$ proves that the $\bar{b}_{2,i,l}$ are correct with (see Protocol 7):

$$\begin{aligned} &\rho = y''_{i,\pi_i^{-1}(\ell)}, && g_1 = g, && h_1 = g^{y''_{i,\pi_i^{-1}(\ell)}}, \\ &g_2 = \bar{b}_{1,i,l}, && h_2 = \bar{b}_{2,i,l} && \forall \ell \in [M]. \end{aligned}$$

6. $\mathcal{A}_0$ computes the step 3 of the ballot $\bar{b}_{3,i} = (\bar{b}_{3,i,l})_{\ell \in [M]}$ where, $\forall \ell \in [M]$:

$$\bar{b}_{3,i,l} = \left(\bar{b}_{2,i,l}\right)^{\frac{1}{\bar{z}_{i,\ell}}} = g^{z'_{i,\ell} y''_{i,\pi_i^{-1}(\ell)} (\sigma_{i,\ell} + x_i)}$$

and sends it to Alice and to $\mathcal{A}_1$. Then $\mathcal{A}_0$ proves that the $\bar{b}_{3,i,l}$ are correct with (see Protocol 7):

$$\begin{aligned} &\rho = \frac{1}{\bar{z}_{i,\ell}}, && g_1 = g^{\bar{z}_{i,\ell}}, && h_1 = g, \\ &g_2 = \bar{b}_{2,i,l}, && h_2 = \bar{b}_{3,i,l} && \forall \ell \in [M]. \end{aligned}$$

7. $\mathcal{A}_1$ computes the final ballot $b_i = (b_{i,\ell})_{\ell \in [M]}$, with:

$$b_{i,\ell} = \left(\bar{b}_{3,i,\pi_i(\ell)}\right)^{\frac{y'_{i,\ell}}{z'_{i,\pi_i(\ell)}}} = g^{y_{i,\ell}(\sigma_{i,\pi_i(\ell)}+x_i)} \forall \ell \in [M]$$

and sends it to Alice and publishes on the web bulletin board the pair (K_i, b_i). Then $\mathcal{A}_1$ proves that the $b_{i,\ell}$ are correct with (see Protocol 7):

$$\rho = \frac{y'_{i,\ell}}{z'_{i,\pi_i(\ell)}}, \qquad g_1 = g^{z'_{i,\pi_i(\ell)}}, \qquad h_1 = g^{y'_{i,\ell}},$$
$$g_2 = \bar{b}_{3,i,\pi_i(\ell)}, \qquad h_2 = b_{i,\ell} \qquad \forall \ell \in [M].$$

Note that Alice, thanks to the proofs and the knowledge of the intermediate values, knows which ones are a valid token (the ones with $\sigma_{i,\ell} = k$), but thanks to the random choices of V_i and π_i the authorities cannot distinguish the tokens unless they collude. Effectively, the DVNIZKPs prove to Alice that the ballot has been created by $\mathcal{A}_0$ with the correct number of valid and decoy tokens, and that it has been correctly shuffled by $\mathcal{A}_2$. Moreover the properties of the DVNIZKP allow Alice to forge their transcript, making them useless for proving the validity of a token. In fact, since Alice is in a protected environment, they can manipulate the received data without being able to prove or disprove any manipulation.So, given that they know e_i, they can forge a proof that states the presumed validity of any P of the M tokens, making any proof worthless to a coercer.

7.2.3 Voting Phase

Voters assign their valid tokens to preferred candidates and decoy tokens to the others. Each voter signs this assignment with their private key and publishes it on the web bulletin board for verification. After voting, duplicate, incomplete, and forged ballots are filtered out (relying on public data only).

7.2.4 Tallying

Once the voting phase is over, the tallying can start.

In order to count the votes, the authorities have to process the tokens received by each candidate, substituting the *voter's masks* $y_{i,\ell}$ with the appropriate *candidate mask* α_ℓ. Suppose that $T \leq N$ participants voted. Without

loss of generality, we can assume that only the participants with index $i \in [T]$ voted, while the remaining $N - T$ abstained from voting.

For every $i \in [T]$, let $\phi_i : [M] \longrightarrow [M]$ be the bijective map that associates to each candidate index ℓ the index of the token $b_{i,\phi_i(\ell)}$ that the voter associated to v_i sent to the candidate C_ℓ. Then, for every $i \in [T], \ell \in [M]$, the authorities process the token $b_{i,\phi_i(\ell)}$ by performing the following steps:

1. $\mathcal{A}_1$ computes and publishes the preliminary vote $\bar{t}_{\ell,i}$ as:

$$\bar{t}_{\ell,i} = \left(b_{i,\phi_i(\ell)}\right)^{\frac{\alpha'_\ell}{y'_{i,\phi_i(\ell)}}} = g^{\alpha'_\ell y''_{i,\phi_i(\ell)}(\sigma_{i,\pi_i(\phi_i(\ell))}+x_i)},$$

   and proves that $\bar{t}_{\ell,i}$ is correct with (see Protocol 7):

$$\rho = \frac{\alpha'_\ell}{y'_{i,\phi_i(\ell)}}, \qquad g_1 = g^{y'_{i,\phi_i(\ell)}}, \qquad h_1 = g^{\alpha'_\ell},$$
$$g_2 = b_{i,\phi_i(\ell)}, \qquad h_2 = \bar{t}_{\ell,i}.$$

2. $\mathcal{A}_2$ then computes and publishes the final vote $t_{\ell,i}$:

$$t_{\ell,i} = (\bar{t}_{\ell,i})^{\frac{\alpha''_\ell}{y''_{i,\phi_i(\ell)}}} = g^{\alpha_\ell(\sigma_{i,\pi_i(\phi_i(\ell))}+x_i)},$$

   and proves that $t_{\ell,i}$ is correct with (see Protocol 7):

$$\rho = \frac{\alpha''_\ell}{y''_{i,\phi_i(\ell)}}, \qquad g_1 = g^{y''_{i,\phi_i(\ell)}}, \qquad h_1 = g^{\alpha''_\ell},$$
$$g_2 = b_{\ell,i}, \qquad h_2 = t_{\ell,i}.$$

Once that all final votes have been computed, the actual tallying is performed.

Let R_ℓ be the number of valid tokens given to the ℓ-th candidate (i.e. the number of preferences received by said candidate), and let F_ℓ be the number of decoy tokens given to the ℓ-th candidate. Clearly $T = R_\ell + F_\ell \quad \forall \ell \in [M]$. The count R_ℓ can be computed with the following steps:

1. Both $\mathcal{A}_1$ and $\mathcal{A}_2$ can compute g^{α_ℓ} (as $(g^{\alpha''_\ell})^{\alpha'_\ell}$ and $(g^{\alpha'_\ell})^{\alpha''_\ell}$ respectively). $\mathcal{A}_1$ can prove the correctness of this value via a proof of equality of two discrete logarithms (see Protocol 7). $\mathcal{A}_2$ can prove the correctness of this value via a proof of equality of two discrete logarithms (see Protocol 7). In practice, each authority may publish half of the values.

2. $\mathcal{A}_0$ computes and publishes $g^{\alpha_\ell k} = (g^{\alpha_\ell})^k$ and $g^{\alpha_\ell \lambda} = (g^{\alpha_\ell})^\lambda$. Then $\mathcal{A}_0$ proves that $g^{\alpha_\ell k}$ and that $g^{\alpha_\ell \lambda}$ are correct via two proofs of equality of two discrete logarithms (see Protocol 7).

3. $\mathcal{A}_1$ computes $\sum_{i=1}^{T} x'_i$, and publishes $g^{\alpha_\ell \sum_{i=1}^{T} x'_i}$. Then $\mathcal{A}_1$ proves that $g^{\alpha_\ell \sum_{i=1}^{T} x'_i}$ is correct via a proof of equality of two discrete logarithms (see Protocol 7). This can be done since any observer can autonomously compute the value $g^{\sum_{i=1}^{T} x'_i} = \prod_{i=1}^{T} g^{x'_i}$.

4. Similarly, $\mathcal{A}_2$ computes $\sum_{i=1}^{T} x''_i$ and publishes $g^{\alpha_\ell \sum_{i=1}^{T} x''_i}$. Then $\mathcal{A}_2$ proves that $g^{\alpha_\ell \sum_{i=1}^{T} x''_i}$ is correct via a proof of equality of two discrete logarithms (see Protocol 7), again, knowing that $g^{\sum_{i=1}^{T} x''_i} = \prod_{i=1}^{T} g^{x''_i}$.

5. Given that any observer can compute the value:

$$g^{\alpha_\ell (\sum_{i=1}^{T} x_i + R_\ell k + F_\ell \lambda)} = \prod_{i=1}^{T} t_{\ell,i},$$

and that:

$$g^{\alpha_\ell \sum_{i=1}^{T} x_i} = g^{\alpha_\ell \sum_{i=1}^{T} (x'_i + x''_i)} = g^{\alpha_\ell \sum_{i=1}^{T} x'_i} \cdot g^{\alpha_\ell \sum_{i=1}^{T} x''_i},$$

then anyone can compute:

$$\begin{aligned} \mathfrak{T} &= \left(g^{\alpha_\ell \sum_{i=1}^{T} x_i}\right)^{-1} \cdot g^{\alpha_\ell (\sum_{i=1}^{T} x_i + R_\ell k + F_\ell \lambda)} \\ &= \left(g^{\alpha_\ell k}\right)^{R_\ell} \cdot \left(g^{\alpha_\ell \lambda}\right)^{F_\ell}. \end{aligned}$$

6. R_ℓ and F_ℓ can now be computed by brute force, giving the number of preferences received by the ℓ-th candidate.

Example 1. *Let $T = 3$, then the possible values of $\mathfrak{T}$ are:*

$$\mathfrak{T} = \begin{cases} \left(g^{\alpha_\ell k}\right)^0 \cdot \left(g^{\alpha_\ell \lambda}\right)^3 & \text{if} \quad R_\ell = 0, \quad F_\ell = 3, \\ \left(g^{\alpha_\ell k}\right)^1 \cdot \left(g^{\alpha_\ell \lambda}\right)^2 & \text{if} \quad R_\ell = 1, \quad F_\ell = 2, \\ \left(g^{\alpha_\ell k}\right)^2 \cdot \left(g^{\alpha_\ell \lambda}\right)^1 & \text{if} \quad R_\ell = 2, \quad F_\ell = 1, \\ \left(g^{\alpha_\ell k}\right)^3 \cdot \left(g^{\alpha_\ell \lambda}\right)^0 & \text{if} \quad R_\ell = 3, \quad F_\ell = 0. \end{cases}$$

Since $g^{\alpha_\ell k}$ and $g^{\alpha_\ell \lambda}$ are publicly available, all these possible values can be computed by anyone, identifying the values of R_ℓ and F_ℓ.

Given a positive integer $T \in \mathbb{N}$, it is possible to represent it in $T + 1$ ways as a sum of two non-negative integers. Given that the number of valid and decoy votes must sum up to the number of actual voters T, it follows that the number of possible values for $\mathfrak{T}$ is $T + 1$, so the effort of computing R_ℓ and F_ℓ is linear in the number of actual votes.

7.3 Security Analysis

The goal is to prove that an adversary cannot distinguish between valid and decoy *v-tokens* and guess how voters cast their preferences. Since election results are obviously public, we have to avoid some trivial cases in which the adversary can deduce the votes by simply observing the results.

Therefore we assume that the adversary controls one authority and all but two voters, and that these two voters express distinct preferences. In particular, we let the adversary select two distinct sets of preferences, then we randomly assign to each of the two uncorrupted voters one set of these sets of preferences. The adversary wins the security game if it guesses correctly which voter expressed which set of preferences, i.e. guesses the random assignment.

7.3.1 Security Model

The security of the protocol will be proven in terms of vote indistinguishability (VI), as detailed in Definition 22. We will assume the presence of a malicious authority, so the simulator in the proof will take on the roles of the two honest authorities and of the two voters that the adversary does not control.

To simplify our analysis we assume that the adversary-controlled authority does not intentionally fail decommitments or (DV)NIZKPs, so the protocol does not abort. This is a reasonable assumption considering the application context, however it is not necessary to attain security. In fact, if the adversary wins the security game with non-negligible advantage, then it must run the protocol smoothly with non-negligible probability (since it outputs its guess only once the protocol has correctly terminated).

The Amun protocol protects against coercers that wish to sway elections towards specific candidates, but is not very effective against randomization and forced abstention. In this simplified model, we use a slightly weaker

adaptation of the definition of coercion resistance given in [83]:

Definition 20 (Vote-Coercion Resistance)**.** *Let $\mathcal{A}$ be a coercer, V_c the set of coerced voters, and $(C_{i,1}, \ldots, C_{i,P})$ the choices that $\mathcal{A}$ wants to impose to the voter corresponding to $v_i \in V_c$. Let Ψ_1 be the scenario in which $\mathcal{A}$ has access only to the final tally. Let Ψ_2 be the scenario in which $\mathcal{A}$ has access to the whole Bulletin Board, and can see all the actions performed by the voters in V_c, with the exception of the ones carried out in a protected environment (or through an untappable channel). A voting protocol is* Vote-Coercion Resistant *if the probability of $\mathcal{A}$ detecting that a voter in V_c has not followed its instruction is the same in Ψ_1 and Ψ_2.*

Definition 21 (Security Game)**.** *The security game for the election protocol proceeds as follows:*

- ***Init.*** *The adversary $\mathcal{A}$ chooses the authority and the $N-2$ voters that it will control (i.e. the adversary knows which are the valid and decoy* v-tokens *of these voters). The remaining two voters are called* free voters*. The challenger $\mathcal{C}$ takes the role of the other authorities and the free voters.*
- ***Phase 0.*** *$\mathcal{A}$ and $\mathcal{C}$ run the* Setup *and* Registration *phases of the protocol, interacting as needed.*
- ***Phase 1.*** *The adversary votes with some or all of the voters it controls.*
- ***Challenge.*** *The challenge phase is articulated as follows:*
  1. *$\mathcal{A}$ selects two distinct sets of preferences $\tilde{P}_0 \neq \tilde{P}_1$, with $\tilde{P}_i \subset [M]$, $\#\tilde{P}_i = P$ for $i = 0, 1$, and sends them to $\mathcal{C}$;*
  2. *$\mathcal{C}$ flips a random coin $\mu \in \{0, 1\}$ to determine which preference set the first free voter will use, i.e. $P_1 = \tilde{P}_\mu$, setting also $P_2 = \tilde{P}_{\mu \oplus 1}$;*
  3. *$\mathcal{C}$ constructs two random ballot assignment maps $\tilde{\phi}_1, \tilde{\phi}_2 : [M] \longrightarrow [M]$ such that $\tilde{\phi}_i(\ell)$ refers to a valid token if and only if $\ell \in P_i$, for $i = 1, 2$;*
  4. *finally, $\mathcal{C}$ votes by sending to the candidate C_ℓ, $\forall \ell \in [M]$, the $\tilde{\phi}_1(\ell)$-th and $\tilde{\phi}_2(\ell)$-th tokens of the first and second free voter respectively.*
- ***Phase 2.*** *The adversary votes with some or all of the voters it controls.*

- ***Phase 3.*** *$\mathcal{A}$ and $\mathcal{C}$ run the* Tallying *phase of the protocol, and the election result is published.*
- ***Guess.*** *The adversary outputs a guess μ' of the coin flip that randomly assigned the voting preferences of the two free voters.*

$\mathcal{A}$ wins if $\mu' = \mu$.

Definition 22 (Vote Indistinguishability)**.** *An E-Voting Protocol with security parameter θ is VI-secure if, for every probabilistic polynomial-time adversary $\mathcal{A}$ that outputs a guess μ' of the coin flip μ (as described in the security game of Definition 21), there exists a negligible function η such that $\mathbb{P}[\mu' = \mu] \leq \frac{1}{2} + \eta(\theta)$.*

In the following theorem we prove our voting protocol VI-secure under the DDH assumption in the security game defined above. The proof is extremely similar to the one presented in [134], and published in [99].

Theorem 2. In the random oracle model, if the DDH assumption holds, then the protocol of Section 7.2 is VI-secure, as per Definition 22.

Proof. Suppose there exists a polynomial time adversary $\mathcal{A}$, that can attack the scheme with advantage ε. We claim that a simulator $\mathcal{S}$ can be built to play the decisional DH game with advantage $\frac{\varepsilon}{2}$. The simulator controls the random oracle that defines the hash function H, and starts by taking in a DDH challenge:

$$(g, A = g^a, B = g^b, \Xi),$$

with $\Xi = g^{ab}$ or $\Xi = R = g^{\xi}$.

First we consider the case in which the adversary controls $\mathcal{A}_0$, the simulation proceeds as follows.

- **Init**. The adversary chooses the $N - 2$ voters to control. Without loss of generality we assume that the two free voters are associated to v_1 and v_2.
- **Setup**. $\mathcal{S}$ chooses uniformly at random in $\mathbb{Z}_p^*$ the values $\tilde{x}_i$, $\tilde{\alpha}_\ell$, $\tilde{y}_{i,\ell}$, and $\tilde{z}_{i,\ell}$ for all $i \in [2]$, $\ell \in [M]$, and implicitly sets for all $i \in [2]$, $\ell \in [M]$:

$$x''_i = \tilde{x}_i + (-1)^i b, \qquad \alpha'_\ell = a \cdot \tilde{\alpha}_\ell,$$
$$y'_{i,\ell} = a \cdot \tilde{y}_{i,\ell}, \qquad z'_{i,\ell} = a \cdot \tilde{y}_{i,\ell}.$$

$\mathcal{S}$ chooses the other values for authorities $\mathcal{A}_1$ and $\mathcal{A}_2$ following the protocol.

In the improbable case that $a = 0$, the DDH problem is trivially solvable ($g^a = g^{ab} = 1$). If $a \neq 0$, since a and b come from an uniform distribution, then also these implicit values are uniformly distributed, so the choices of the simulator are indistinguishable from a real protocol execution.

Note that $\mathcal{S}$ can compute all the values $g^{x''_i}$, $g^{\alpha'_\ell}$, $g^{y'_{i,\ell}}$, $g^{z'_{i,\ell}}$, either normally (when the parameter has been explicitly chosen) or as follows:

$$g^{x''_i} = g^{\tilde{x}_i} \cdot B^{(-1)^i}, \qquad g^{\alpha'_\ell} = A^{\tilde{\alpha}_\ell},$$
$$g^{y'_{i,\ell}} = A^{\tilde{y}_{i,\ell}}, \qquad g^{z'_{i,\ell}} = A^{\tilde{z}_{i,\ell}}.$$

for all $i \in [2]$, $\ell \in [M]$. Therefore, $\mathcal{S}$ can simulate the setup phase, exploiting the RO to simulate the NIZKPs for x''_i, α'_ℓ, $y'_{i,\ell}$ $z'_{i,\ell}$ for $i \in [2]$, $\ell \in [M]$.

- **Registration Phase**. For the voters associated to v_i with $3 \leq i \leq N$, $\mathcal{S}$ can simulate this phase following the protocol normally (since all relevant parameters have been explicitly chosen), while for $i \in [2]$ $\mathcal{S}$ does the following:

  1. $\mathcal{A}$ computes the initial step of the ballot $\bar{b}_{0,i}$ on behalf of $\mathcal{A}_0$ and proves its correctness with the appropriate DVNIZKPs. By rewinding $\mathcal{A}$ and exploiting the control of the random oracle, $\mathcal{S}$ is able to extract from the DVNIZKPs the values of k, λ, and $\bar{z}_{i,\ell}$ for all $\ell \in [M]$ (see [134]). Moreover, since $\mathcal{A}_0$ communicates the set of indexes of valid tokens V_i to the voter associated to v_i (that is controlled by the simulator), $\mathcal{S}$ can reconstruct the values of the $\sigma_{i,\ell}$ for all $\ell \in [M]$.
  2. $\mathcal{S}$ computes step 1 of the ballot $\bar{b}_{1,i} = (\bar{b}_{1,i,l})_{\ell \in [M]}$ as:

$$\begin{aligned} \bar{b}_{1,i,l} &= A^{\bar{z}_{i,\ell}\tilde{z}_{i,\ell}(\sigma_{i,\ell}+x'_i+\tilde{x}_i)} \cdot \Xi^{\bar{z}_{i,\ell}\tilde{z}_{i,\ell}(-1)^i} \\ &\stackrel{*}{=} g^{z_{i,\ell}(\sigma_{i,\ell}+x_i)} \quad \forall \ell \in [M] \end{aligned} \tag{7.1}$$

  where $\stackrel{*}{=}$ of Equation (7.1) holds iff $\Xi = g^{ab}$ in the DDH challenge. Since it controls the voter associated to v_i, $\mathcal{S}$ can forge the DVNIZKPs exploiting the value e_i. In order to hide from $\mathcal{A}$

which tokens are valid, these DVNIZKPs are forged using random values.

3. $\mathcal{S}$ can perform step 2 on behalf of $\mathcal{A}_2$ normally, then $\mathcal{A}$ computes step 3 on behalf of $\mathcal{A}_0$ and proves its correctness.
4. Finally $\mathcal{S}$ computes the final ballot $b_i = (b_{i,\ell})_{\ell \in [M]}$ as:

$$\begin{aligned} b_{i,\ell} &= A^{\tilde{y}_{i,\ell} y''_{i,\ell} (\sigma_{i,\pi_i(\ell)} + x'_i + \tilde{x}_i)} \cdot \Xi^{\tilde{y}_{i,\ell} y''_{i,\ell} (-1)^i} \\ &\overset{*}{=} g^{y_{i,\ell}(\sigma_{i,\pi_i(\ell)} + x_i)} \end{aligned} \tag{7.2}$$

where again $\overset{*}{=}$ of Equation (7.2) holds if and only if $\Xi = g^{ab}$ in the DDH challenge.

- **Voting**: Phases 1, 2, and the Challenge are performed as in Definition 21.

- **Tallying**. Without loss of generality, suppose that only the v_i with $i \in [T]$ have voted. For $\ell \in [M]$, $\mathcal{S}$ carries on with the simulation as follows:

1. $\mathcal{S}$ computes the preliminary and final votes on behalf of $\mathcal{A}_1$ and $\mathcal{A}_2$ following the protocol without problems. In fact, for $i \in [2]$, we have:

$$\frac{\alpha'_\ell}{y'_{i,\tilde{\phi}_i(\ell)}} = \frac{a\tilde{\alpha}_\ell}{a\tilde{y}_{i,\tilde{\phi}_i(\ell)}} = \frac{\tilde{\alpha}_\ell}{\tilde{y}_{i,\tilde{\phi}_i(\ell)}} \qquad \forall \ell \in [M],$$

and these values are known to $\mathcal{S}$.

2. $\mathcal{S}$ computes and publishes the values $g^{\alpha_\ell} = A^{\tilde{\alpha}_\ell \alpha''_\ell}$ $\forall \ell \in [M]$, and simulates the proofs of correctness.
3. Finally note that $\mathcal{S}$ can compute:

$$\sum_{i=1}^{T} x''_i = \tilde{x}_1 - b + \tilde{x}_2 + b + \sum_{i=3}^{T} x''_i = \tilde{x}_1 + \tilde{x}_2 + \sum_{i=3}^{T} x''_i,$$

so for the rest of the tallying phase $\mathcal{S}$ can follow the protocol.

- **Guess.** Eventually $\mathcal{A}$ will output a guess μ' of the coin flip performed by $\mathcal{S}$ during the Challenge. $\mathcal{S}$ then outputs 0 to guess that $\Xi = g^{ab}$ if $\mu' = \mu$, otherwise it outputs 1 to indicate that Ξ is a random group element $R \in \mathbb{G}$.

The case in which the adversary controls $\mathcal{A}_1$ and the case in which the adversary controls $\mathcal{A}_2$, proceed similarly. If $\mathcal{A}_1$ is corrupted, the main difference is that $\mathcal{S}$ implicitly sets:

$$\alpha''_\ell = a \cdot \tilde{\alpha}_\ell, \qquad y''_{i,\ell} = a \cdot \tilde{y}_{i,\ell}, \qquad \bar{z}_{i,\ell} = a \cdot \tilde{y}_{i,\ell},$$

while α'_ℓ, $y'_{i,\ell}$, $z'_{i,\ell}$ are chosen normally.

If $\mathcal{A}_2$ is corrupted, the main difference is that $\mathcal{S}$ implicitly sets:

$$x'_i = \tilde{x}_i + (-1)^i b,$$

while x''_i is chosen normally.

Essentially, in all three cases when Ξ is not random the simulator $\mathcal{S}$ gives a perfect simulation. This means that the advantage is preserved, so it holds that:

$$\mathbb{P}[\mathcal{S}(g, A, B, \Xi = g^{ab}) = 0] = \frac{1}{2} + \varepsilon.$$

On the contrary, when Ξ is a random element $R \in \mathbb{G}$, every token and vote belonging to the free voters becomes independent from the values that would have been computed by following the protocol (since they are simulated using the random value R), so $\mathcal{A}$ can gain no information about the votes from them, while the tally is always correct. Since the security game is structured in such a way that the tally and the tokens of the other voters (i.e. the values where Ξ is not used in the computation by $\mathcal{S}$) do not give any information about the coin flip μ, we have that:

$$\mathbb{P}[\mathcal{S}(g, A, B, \Xi = R) = 0] = \frac{1}{2}.$$

Therefore, $\mathcal{S}$ can play the DDH game with non-negligible advantage $\frac{\varepsilon}{2}$. □

We can now prove that the protocol is coercion resistant, as per Definition 20.

Proposition 1 (Vote-Coercion Resistance)**.** *In the random oracle model, if the DDH assumption holds, then the protocol is vote-coercion resistant, as per Definition 20.*

Proof. In order to comply with the coercer's request, a voter associated to $v_i \in V_c$ has to assign the valid tokens to $(C_{i,1}, \ldots, C_{i,P})$. Since the Registration Phase is performed in a protected environment, only the voter associated

to v_i knows which tokens are valid, and cannot give a meaningful proof of this fact to $\mathcal{A}$ as discussed at the end of the registrar phase (Section 7.2.2).

Thanks to Theorem 2, the protocol has vote-indistinguishability and the only way to determine if a vote expresses a specific choice is to distinguish valid and decoy tokens. Since $\mathcal{A}$ cannot do so, all the information that can be gained from the votes is given by the final tally. This means exactly that the probability of $\mathcal{A}$ detecting that a voter in V_c has not followed its instruction is the same in Ψ_1 and Ψ_2. □

7.4 Final Remarks

Efficiency and scalability. The computational and resource cost of our protocol scales linearly in $N \cdot M$, where M is the number of candidates and N is the number of voters managed by a triplet of authorities. In particular:

- In the setup phase the size of the published values is:

  $$2 \cdot (2MN + N + M + 1) \cdot (2|\mathbb{G}| + |\mathbb{Z}_p|) + N \cdot |v|,$$

  where $|v|$ is the size of a pseudonymous identifier. $\mathcal{A}_0$ stores $2 + NM$ secret elements of $\mathbb{Z}_p$, $\mathcal{A}_1$ stores $2NM + N + M$ secret elements of $\mathbb{Z}_p$, $\mathcal{A}_2$ stores $NM + N + M$ secret elements of $\mathbb{Z}_p$. Every authority also stores the N identifiers.

- In the registrar phase each of the N voters receive data of size:

  $$23M \cdot |\mathbb{G}| + 24M \cdot |\mathbb{Z}_p| + |v|,$$

  and have to store also the designated and signing key-pairs which have additional size $|\mathbb{G}| + |\mathbb{Z}_p| + |K| + |s|$ ($|K|$ and $|s|$ are respectively the size of the public and secret signing keys).

  The size of the data published on the web bulletin board in this phase is:

  $$N \cdot (M \cdot |\mathbb{G}| + |K| + |v|).$$

- In the voting phase the size of the data published on the web bulletin board is:

  $$(T + \texttt{revote}) \cdot (|v| + |\texttt{sig}| + M \cdot |C|),$$

  where T is the number of voters that cast a valid ballot, `revote` is the number of duplicate votes, $|\texttt{sig}|$ is the size of the signature, $|C|$ is the size of a candidate's identifier.

- In the tallying phase the size of the data published on the web bulletin board is:
$$M \cdot \big[(6T + 15) \cdot |\mathbb{G}| + (2T + 5) \cdot |\mathbb{Z}_p|\big].$$
The effort required by an observer to compute the results of the election is:
$$\begin{aligned} M \cdot \big[&(2T + 5) \cdot \texttt{hash} + (8T + 9) \cdot \texttt{mul} \\ &+ (10T + 21) \cdot \texttt{exp} + (5T + 10) \cdot \texttt{check}\big], \end{aligned}$$
where `hash` denotes the cost of computing the hash digest on 6 elements of $\mathbb{G}$, `mul` denotes the cost of the group operation (multiplication) in $\mathbb{G}$, `exp` denotes the cost of the scalar operation (exponentiation) in $\mathbb{G}$, `check` denotes the cost of comparing two elements of $\mathbb{G}$.
Once the results have been published, to check them we save a computational effort of $\big[M \cdot (T - 1)\big] \cdot (\texttt{mul} + 2\texttt{exp} + \texttt{check})$, since we do not have to re-compute R_ℓ and F_ℓ.

Unexpressed preferences. Many election systems allow voters to cast a blank ballot or to leave some of the P possible preferences unexpressed. This feature can be easily added to the protocol presented here by simply adding P *dummy* candidates that represent blank choices.

Chapter 8

Conclusions

Mathematics plays a fundamental role in the design of secure electronic voting protocols, providing the tools necessary to ensure integrity, verifiability, and coercion resistance. In this thesis, we have explored how mathematical techniques contribute to improving the security of internet voting protocols, with a particular focus on coercion-resistance.

By combining cryptographic techniques with usability, this work aims to improve not only the theoretical security of e-voting systems, but also their real-world applicability. The threat modeling methodology developed helps identify and mitigate risks specific to e-voting, particularly those related to coercion in remote voting scenarios. These contributions lay the foundation for the safe use of e-voting, strengthen the democratic integrity of digital elections, and pave the way for further research in the field.

However, there are several medium- and long-term challenges that need to be addressed to make internet voting solutions not only technically sound but also widely trusted for use in real democratic processes. Regarding *Vote App*, an important direction for future work is to investigate the scalability of the proposed protocol to ensure that it can efficiently handle large elections. Optimizing computational costs is also critical to making the system more practical. In addition, strengthening coercion resistance by proving security under the stronger definition of [39] rather than the classical JCJ framework is an important step forward. The threat modeling methodology developed improves our understanding of coercion-related threats, but further analysis is needed to assess the risks posed by other types of adversaries and their potential impact on the overall security of the system.

We tested the usability of *Vote App* and the results were very promising.

A critical future direction would be to implement a simulation of a real-world coercion scenario. Such an experiment would provide further assessment of whether the protocol remains effective under real-world conditions. However, developing this test presents significant challenges, as ethical considerations must be taken into account to avoid distressing participants while still collecting meaningful data.

Ultimately, the goal of this thesis is to contribute to the research on e-voting solutions that can be safely adopted in real-world scenarios, so that in the future, online voting can become a standard and trusted option, similar to other digital services we use today.

Bibliography

[1] *Ace Project - Encyclopaedia*. https://aceproject.org/ace-en/focus/e-voting/e-voting-auditing/mobile_browsing/onePag. Accessed: 2021-11-19.

[2] Claudia Z. Acemyan et al. "Usability of Voter Verifiable, End-to-end Voting Systems: Baseline Data for Helios, Prêt à Voter, and Scantegrity II". In: *USENIX Journal of Election Technology and Systems (JETS)* 2.3 (2014), pp. 26–56.

[3] Ben Adida. "Helios: Web-based Open-Audit Voting." In: *USENIX security symposium*. Vol. 17. 2008, pp. 335–348.

[4] Ben Adida and C Andrew Neff. "Ballot Casting Assurance." In: *EVT* 6 (2006), p. 7.

[5] Ben Adida et al. "Electing a university president using open-audit voting: Analysis of real-world use of Helios". In: *EVT/WOTE* 9.10 (2009).

[6] Agenzia per l'Italia Digitale. *eIDAS - Identità digitale europea*. Accessed: 2024-02-06. 2024. URL: https://www.agid.gov.it/it/piattaforme/eidas.

[7] Roberto Araújo and Jacques Traoré. "A practical coercion resistant voting scheme revisited". In: *International Conference on E-Voting and Identity*. Springer. 2013, pp. 193–209.

[8] Roberto Araújo et al. "Towards practical and secure coercion-resistant electronic elections". In: *Cryptology and Network Security: 9th International Conference, CANS 2010, Kuala Lumpur, Malaysia, December 12-14, 2010. Proceedings 9*. Springer. 2010, pp. 278–297.

[9] Michael Backes, Martin Gagné, and Malte Skoruppa. "Using mobile device communication to strengthen e-voting protocols". In: *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*. 2013, pp. 237–242.

[10] *Balsamiq*. https://balsamiq.com/. (Visited on 02/11/2025).

[11] Mihir Bellare and Phillip Rogaway. "Random oracles are practical: A paradigm for designing efficient protocols". In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. 1993, pp. 62–73.

[12] Josh Benaloh. *ElectionGuard*. Microsoft Research, 2020. URL: https://github.com/microsoft/electionguard/releases/download/v0.95.0/ElectionGuard_Specification_v095.pdf (visited on 02/11/2025).

[13] Josh Benaloh. "Simple Verifiable Elections." In: *EVT* 6 (2006), pp. 5–5.

[14] Josh Benaloh et al. *End-to-end verifiability*. Apr. 2015. arXiv: 1504.03778 [cs.CR].

[15] David Bernhard, Oksana Kulyk, and Melanie Volkamer. "Security proofs for participation privacy, receipt-freeness and ballot privacy for the helios voting scheme". In: *Proceedings of the 12th International Conference on Availability, Reliability and Security*. 2017, pp. 1–10.

[16] David Bernhard et al. "Adapting Helios for provable ballot privacy". In: *European Symposium on Research in Computer Security*. Springer. 2011, pp. 335–354.

[17] Daniel J Bernstein and Tanja Lange. "Faster addition and doubling on elliptic curves". In: *Advances in Cryptology–ASIACRYPT 2007: 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007. Proceedings 13*. Springer. 2007, pp. 29–50.

[18] Daniel J Bernstein et al. "High-speed high-security signatures". In: *Journal of cryptographic engineering* 2.2 (2012), pp. 77–89.

[19] Dan Boneh and Xavier Boyen. "Short signatures without random oracles". In: *International conference on the theory and applications of cryptographic techniques*. Springer. 2004, pp. 56–73.

[20] Dan Boneh and Victor Shoup. "A graduate course in applied cryptography". In: *Version 0.6* (2023). URL: https://toc.cryptobook.us/ (visited on 02/11/2025).

[21] Gilles Brassard, David Chaum, and Claude Crépeau. "Minimum disclosure proofs of knowledge". In: *Journal of computer and system sciences* 37.2 (1988), pp. 156–189.

[22] Ian Brightwell et al. "An overview of the iVote 2015 voting system". In: *New South Wales Electoral Commission, Australia, Scytl Secure Electronic Voting, Spain* (2015).

[23] Simone Brunello et al. "E-Voting with Confidence: Usability Challenges in Manual Integrity Checks". In: *To appear in International Conference on Human-Computer Interaction*. Springer. 2025.

[24] Jurlind Budurushi et al. "An investigation into the usability of electronic voting systems for complex elections". In: *Annals of Telecommunications* 71 (Apr. 2016). DOI: 10.1007/s12243-016-0510-2.

[25] Jan Camenisch and Anna Lysyanskaya. "Signature schemes and anonymous credentials from bilinear maps". In: *Annual international cryptology conference*. Springer. 2004, pp. 56–72.

[26] Jan Camenisch et al. "How to win the clonewars: efficient periodic n-times anonymous authentication". In: *Proceedings of the 13th ACM conference on Computer and communications security*. 2006, pp. 201–210.

[27] Jan L Camenisch, Jean-Marc Piveteau, and Markus A Stadler. "Blind signatures based on the discrete logarithm problem". In: *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer. 1994, pp. 428–432.

[28] B. Campbell, D. Lodderstedt, and T. Bradley. *OAuth 2.0 Demonstration of Proof-of-Possession at the Application Layer (DPoP)*. IETF Internet-Draft, Work in Progress. Oct. 2024. URL: https://datatracker.ietf.org/doc/html/draft-ietf-oauth-dpop (visited on 02/11/2025).

[29] D. Campbell. *Mutual TLS Profiles for OAuth Clients*. RFC 8705, Internet Engineering Task Force (IETF). Feb. 2020. URL: https://www.rfc-editor.org/rfc/rfc8705 (visited on 02/11/2025).

[30] *Canva*. https://canva.com/. (Visited on 02/11/2025).

[31] Pyrros Chaidos et al. “BeleniosRF: A non-interactive receipt-free electronic voting scheme”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 1614–1625.

[32] David Chaum. “Blind signatures for untraceable payments”. In: *Advances in cryptology*. Springer. 1983, pp. 199–203.

[33] David Chaum et al. “VoteXX: A solution to improper influence in voter-verifiable elections”. In: (2022).

[34] David L Chaum. “Untraceable electronic mail, return addresses, and digital pseudonyms”. In: *Communications of the ACM* 24.2 (1981), pp. 84–90.

[35] Michael R Clarkson, Stephen Chong, and Andrew C Myers. “Civitas: Toward a secure voting system”. In: *2008 IEEE Symposium on Security and Privacy (sp 2008)*. IEEE. 2008, pp. 354–368.

[36] *Complementary material of Threat Modeling Process*. https://docs.google.com/spreadsheets/d/1Usey6e7hAp2xXE59ae8LLuSfGOnNkmT5s3O7IoJRDr8/edit?usp=sharing. Accessed: 2025-02-26.

[37] Véronique Cortier, Alicia Filipiak, and Joseph Lallemand. “BeleniosVS: Secrecy and verifiability against a corrupted voting device”. In: *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. IEEE. 2019, pp. 367–36714.

[38] Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. “Belenios: a simple private and verifiable electronic voting system”. In: *Foundations of Security, Protocols, and Equational Reasoning*. Springer, 2019, pp. 214–238.

[39] Véronique Cortier, Pierrick Gaudry, and Quentin Yang. “Is the JCJ voting system really coercion-resistant?” In: *2024 IEEE 37th Computer Security Foundations Symposium (CSF)*. IEEE. 2024, pp. 186–200.

[40] Véronique Cortier et al. “A simple alternative to Benaloh challenge for the cast-as-intended property in Helios/Belenios”. In: (2019).

[41] Véronique Cortier et al. “Distributed ElGamal á la Pedersen: application to Helios”. In: *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*. 2013, pp. 131–142.

[42] Véronique Cortier et al. "Election verifiability for helios under weaker trust assumptions". In: *European Symposium on Research in Computer Security*. Springer. 2014, pp. 327–344.

[43] *Recommendation CM/Rec(2017)5 of the Committee of Ministers to member States on standards for e-voting*. URL: https://search.coe.int/cm/Pages/result_details.aspx?ObjectID=0900001680726f6f (visited on 10/22/2021).

[44] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. "Proofs of partial knowledge and simplified design of witness hiding protocols". In: *Annual International Cryptology Conference*. Springer. 1994, pp. 174–187.

[45] *Crisis 2024: 19th International Conference on Risks and Security of Internet and Systems*. https://crisis2024.univ-gustave-eiffel.fr/. (Visited on 02/11/2025).

[46] Laura Cristiano, Riccardo Longo, and Chiara Spadafora. "Click and Cast - Assessing the Usability of Vote App". In: *E-Vote-ID 2024*. Bonn: Gesellschaft für Informatik, 2024, pp. 67–84. DOI: 10.18420/e-vote-id2024_05.

[47] Laura Cristiano and Chiara Spadafora. "Enhancing Usability in E-Voting Systems: Balancing Security and Human Factors with the HC3 Framework". In: *HCI International 2024 Posters. 26th International Conference on Human-Computer Interaction, Proceedings, Part VI*. CCIS Springer Nature, 2024. DOI: 978-3-031-61966-3.

[48] Edouard Cuvelier, Olivier Pereira, and Thomas Peters. "Election verifiability or ballot privacy: Do we need to choose?" In: *European Symposium on Research in Computer Security*. Springer. 2013, pp. 481–498.

[49] Aditya Damodaran et al. "Hyperion: transparent end-to-end verifiable voting with coercion mitigation". In: *Cryptology ePrint Archive* (2024).

[50] Denise Demirel, Jeroen Van De Graaf, and Roberto Samarone dos Santos Araújo. "Improving Helios with Everlasting Privacy Towards the Public." In: *EVT/WOTE* 12 (2012).

[51] Harold Edwards. "A normal form for elliptic curves". In: *Bulletin of the American mathematical society* 44.3 (2007), pp. 393–422.

[52] Alex Escala et al. "Universal cast-as-intended verifiability". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2016, pp. 233–250.

[53] Ehsan Estaji et al. "Revisiting practical and usable coercion-resistant remote e-voting". In: *Electronic Voting: 5th International Joint Conference, E-Vote-ID 2020, Bregenz, Austria, October 6–9, 2020, Proceedings 5*. Springer. 2020, pp. 50–66.

[54] *Commission Implementing Regulation (EU) 2015/1502*. URL: http://data.europa.eu/eli/reg_impl/2015/1502/oj (visited on 05/14/2022).

[55] European Commission. *Overview of pre-notified and notified eID schemes under eIDAS*. Accessed: 2024-02-06. 2024. URL: https://ec.europa.eu/digital-building-blocks/wikis/display/EIDCOMMUNITY/Overview+of+pre-notified+and+notified+eID+schemes+under+eIDAS.

[56] Júlio César Leitão Albuquerque de Farias et al. "Approach based on STPA extended with STRIDE and LINDDUN, and blockchain to develop a mission-critical e-voting system". In: *Journal of Information Security and Applications* 81 (2024), p. 103715. ISSN: 2214-2126. DOI: 10.1016/j.jisa.2024.103715.

[57] Paul Feldman. "A practical scheme for non-interactive verifiable secret sharing". In: *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*. IEEE. 1987, pp. 427–438.

[58] Terence Fenn and Jason Hobbs. "Applying user journey design to resolve complex design problems". In: *University of Johannesburg Institutional Repository*. Sept. 2013. DOI: 10.13140/2.1.4704.6402.

[59] *Figma*. https://www.figma.com/. (Visited on 02/11/2025).

[60] Bryan Ford et al. "Persistent Personal Names for Globally Connected Mobile Devices". In: *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Nov. 2006.

[61] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. "A practical secret voting scheme for large scale elections". In: *International Workshop on the Theory and Application of Cryptographic Techniques*. Springer. 1992, pp. 244–251.

[62] Jun Furukawa. "Efficient and verifiable shuffling and shuffle-decryption". In: *IEICE transactions on fundamentals of electronics, communications and computer sciences* 88.1 (2005), pp. 172–188.

[63] Pierrick Gaudry. "Some ZK security proofs for Belenios". In: (2017). URL: https://inria.hal.science/hal-01576379.

[64] Riccardo Germenia et al. "Automating Compliance for Improving TLS Security Postures: An Assessment of Public Administration Endpoints". In: *Proceedings of the 21st International Conference on Security and Cryptography-Volume 1: SECRYPT*. SciTePress. 2024, pp. 450–458.

[65] Sarah Gibbons. *Service Blueprints: Definition*. 2017. URL: https://www.nngroup.com/articles/service-blueprints-definition/ (visited on 08/27/2017).

[66] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to play any mental game, or a completeness theorem for protocols with honest majority". In: *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. 2019, pp. 307–328.

[67] Michael T. Goodrich et al. "Loud and Clear: Human-Verifiable Authentication Based on Audio". In: *Proceedings of the International Conference on Distributed Computing (ICDCS)*. 2006.

[68] Kristen K. Greene, Michael D. Byrne, and Sarah P. Everett. "A Comparison of Usability Between Voting Methods". In: *USENIX Workshop on Accurate Electronic Voting Technology*. 2006. URL: https://api.semanticscholar.org/CorpusID:13641141.

[69] Jens Groth. "A verifiable secret shuffle of homomorphic encryptions". In: *Journal of Cryptology* 23.4 (2010), pp. 546–579.

[70] Feng Hao et al. "End-to-End Verifiable E-Voting Trial for Polling Station Voting". In: *IEEE Security & Privacy* 18.6 (2020), pp. 6–13. DOI: 10.1109/MSEC.2020.3002728.

[71] Dick Hardt. *The OAuth 2.0 Authorization Framework*. Best Current Practice. IETF RFC 6749, Oct. 2012. URL: https://datatracker.ietf.org/doc/rfc6749/ (visited on 04/07/2023).

[72] Sven Heiberg et al. "Improving the verifiability of the Estonian internet voting scheme". In: *International Joint Conference on Electronic Voting*. Springer. 2016, pp. 92–107.

[73] Paul. S Herrnson et al. "The Importance of Usability Testing of Voting Systems". In: *2006 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT 06)*. Vancouver, B.C.: USENIX Association, Aug. 2006. URL: https://www.usenix.org/conference/evt-06/importance-usability-testing-voting-systems.

[74] Muhammed Zakir Hossain, Fabiha Enam, and Saraj Farhana. "Service Blueprint a Tool for Enhancing Service Quality in Restaurant Business". In: *American Journal of Industrial and Business Management* 07 (July 2017), pp. 919–926. DOI: 10.4236/ajibm.2017.77065.

[75] J. Huh and S. Egelman. "On the Memorability of System-generated PINs: Can Chunking Help?" In: *Proceedings of the Symposium on Usable Privacy and Security (SOUPS) Workshop*. 2014. URL: https://cups.cs.cmu.edu/soups/2014/workshops/papers/chunking_huh_11.pdf.

[76] Jun Ho Huh et al. "On the Memorability of System-generated {PINs}: Can Chunking Help?" In: *eleventh symposium on usable privacy and security (SOUPS 2015)*. 2015, pp. 197–209.

[77] *International Organization for Standardization. ISO 9241-11:2018, Ergonomics of human-system interaction. Part 11: Usability: Definitions and Concepts*. 2018. URL: https://www.iso.org/standard/63500.html (visited on 02/11/2025).

[78] International Organization for Standardization. *Information technology — Security techniques — Information security risk management*. Accessed: 11/02/2025. 2022. URL: https://www.iso.org/standard/75281.html.

[79] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. "Designated verifier proofs and their applications". In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1996, pp. 143–154.

[80] Joint Task Force Transformation Initiative. *Guide for Conducting Risk Assessments*. Tech. rep. 800-30 Rev. 1. National Institute of Standards and Technology, Sept. 2012. URL: https://csrc.nist.gov/pubs/sp/800/30/r1/final (visited on 02/11/2025).

[81] M Jones et al. *RFC 8693: OAuth 2.0 Token Exchange*. 2020. (Visited on 02/11/2025).

[82] M. Jones. *JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens*. RFC 9068, Internet Engineering Task Force (IETF). Aug. 2021. URL: https://www.rfc-editor.org/rfc/rfc9068 (visited on 02/11/2025).

[83] Ari Juels, Dario Catalano, and Markus Jakobsson. "Coercion-Resistant Electronic Elections". In: *Towards Trustworthy Elections*. Vol. 6000. LNCS. Springer, 2010, pp. 37–63.

[84] Plinio Thomaz Aquino Junior and Lucia Vilela Leite Filgueiras. "User modeling with personas". In: *Proceedings of the 2005 Latin American conference on Human-computer interaction*. 2005, pp. 277–282.

[85] Kate Kaplan. *User Journeys vs. User Flows*. 2023. URL: https://www.nngroup.com/articles/user-journeys-vs-user-flows/ (visited on 04/16/2023).

[86] Ambarish Karole, Nitesh Saxena, and Nicolas Christin. "A comparative usability evaluation of traditional password managers". In: *Information Security and Cryptology-ICISC 2010: 13th International Conference, Seoul, Korea, December 1-3, 2010, Revised Selected Papers 13*. Springer. 2011, pp. 233–251.

[87] Aggelos Kiayias et al. "On the security properties of e-voting bulletin boards". In: *International Conference on Security and Cryptography for Networks*. Springer. 2018, pp. 505–523.

[88] Oksana Kulyk et al. "German voters' attitudes towards voting online with a verifiable system". In: *Lecture Notes in Computer Science, LNCS, volume 13412*. Financial Cryptography and Data Security. FC 2022 International Workshop (May 6, 2022). Springer Link, 2022.

[89] 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*. 2010, pp. 526–535.

[90] Ralf Küsters et al. "Ordinos: a verifiable tally-hiding e-voting system". In: *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2020, pp. 216–235.

[91] Sharon Laskowski et al. *Improving the Usability and Accessibility of Voting Systems and Products*. 2004. URL: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=150478 (visited on 02/11/2025).

[92] Nathan Licht et al. "To i-vote or not to i-vote: Drivers and barriers to the implementation of internet voting". In: *Electronic Voting: 6th International Joint Conference, E-Vote-ID 2021, Virtual Event, October 5–8, 2021, Proceedings 6*. Springer. 2021, pp. 91–105.

[93] Y.-H. Lin et al. "SPATE: Small-group PKI-less Authenticated Trust Establishment". In: *Proceedings of the 7th ACM/USENIX International Conference on Mobile Systems, Applications, and Services (MobiSys)*. ACM, June 2009, pp. 1–14.

[94] *LINDDUN privacy threat types*. Accessed: 2024-05-03. URL: https://linddun.org/threat-types.

[95] D. Lodderstedt, J. Bradley, and B. Campbell. *OAuth 2.0 Security Best Current Practice*. IETF Internet-Draft, Work in Progress. Oct. 2024. URL: https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics (visited on 02/11/2025).

[96] Riccardo Longo, Chiara Spadafora, and Francesco Antonio Marino. "Vote App: Verifiable and Coercion-Resistant I-Voting". In: *De Cifris Koine* 1 (Apr. 2024), pp. 107–109. ISSN: 3034-9796. DOI: 10.69091/koine/vol-1-W14. URL: https://doi.org/10.69091/koine/vol-1-W14.

[97] Riccardo Longo et al. "Adaptation of an i-voting scheme to Italian Elections for Citizens Abroad". In: *University of Tartu Press* (2022). DOI: 10.15157/DISS/027. URL: https://dspace.ut.ee/handle/10062/84325.

[98] Riccardo Longo et al. "Modeling and Assessing Coercion Threats in Electronic Voting". In: *Crisis 2024: 19th International Conference on Risks and Security of Internet and Systems*. Springer, 2024.

[99] Riccardo Longo. and Chiara Spadafora. "Amun: Securing E-Voting Against Over-the-Shoulder Coercion". In: *Proceedings of the 21st International Conference on Security and Cryptography - SECRYPT*. INSTICC. SciTePress, 2024, pp. 508–517. ISBN: 978-989-758-709-2. DOI: 10.5220/0012786800003767.

[100] Karola Marky et al. "How to Assess the Usability Metrics of E-Voting Schemes". In: *Financial Cryptography Workshops*. 2019. URL: https://api.semanticscholar.org/CorpusID:86497210.

[101] Karola Marky et al. "How to assess the usability metrics of e-voting schemes". In: *Financial Cryptography and Data Security: FC 2019 International Workshops, VOTING and WTSC, St. Kitts, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23*. Springer. 2020, pp. 257–271.

[102] John McCarthy et al. "The Bird is the Word: A Usability Evaluation of Emojis in Passwords". In: *Proceedings of the 32nd International BCS Human Computer Interaction Conference (HCI)*. ACM, 2018. DOI: 10.1145/3152771.3152773.

[103] Ministero dell'Interno della Republica Italiana. *Decreto sperimentazione voto elettronico 9.07.2021*. URL: https://www.interno.gov.it/sites/default/files/2021-07/decreto_ministro_su_sperimentazione_voto_elettronico_9.7.2021.pdf (visited on 02/11/2025).

[104] Tal Moran and Moni Naor. "Receipt-free universally-verifiable voting with everlasting privacy". In: *Annual International Cryptology Conference*. Springer. 2006, pp. 373–392.

[105] Johannes Müller and Tomasz Truderung. "CAISED: A Protocol for Cast-as-Intended Verifiability with a Second Device". In: *International Joint Conference on Electronic Voting*. Springer. 2023, pp. 123–139.

[106] C Andrew Neff. "Verifiable mixing (shuffling) of ElGamal pairs". In: *VHTi Technical Document, VoteHere, Inc* (2003), p. 112.

[107] Jakob Nielsen. *Thinking Aloud: The #1 Usability Tool*. 2012. URL: https://www.nngroup.com/articles/thinking-aloud-the-1-usability-tool/ (visited on 01/15/2012).

[108] NIST. *Voluntary Voting System Guidelines Overview - NIST*. URL: https://www.nist.gov/system/files/documents/2017/05/24/vvsgvol12.pdf (visited on 04/11/2025).

[109] NSW Electoral Commission. *iVote and the 2021 NSW Local Government elections*. Accessed: 2025-01-15. 2021. URL: https://elections.nsw.gov.au/about-us/media-centre/news-and-media-releases/ivote-and-2021-nsw-local-government-elections.

[110] OWASP. *Risk Rating Methodology*. URL: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology (visited on 04/27/2024).

[111] OWASP. *Threat Modeling Process*. URL: https://owasp.org/www-community/Threat_Modeling_Process (visited on 04/27/2024).

[112] Luis Panizo Alonso et al. "E-Voting System Evaluation Based on The Council of Europe Recommendations: Helios Voting". In: *IEEE Transactions on Emerging Topics in Computing* 9.1 (2021), pp. 161–173. DOI: 10.1109/TETC.2018.2881891.

[113] Harold Pardue, Alec Yasinsac, and Jeffrey Landry. "Towards internet voting security: A threat tree for risk assessment". In: *2010 Fifth International Conference on Risks and Security of Internet and Systems (CRiSIS)*. IEEE. 2010, pp. 1–7.

[114] Parlamento Italiano. *Norme per l' esercizio del diritto di voto dei cittadini italiani residenti all' estero*. https://web.camera.it/parlam/leggi/01459l.htm. Accessed: 2022-05-15.

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

[116] Marco Pernpruner et al. "The Good, the Bad and the (Not So) Ugly of Out-of-Band Authentication with eID Cards and Push Notifications: Design, Formal and Risk Analysis". In: *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*. 2020.

[117] Stefan Popoveniuc et al. "Performance Requirements for End-to-End Verifiable Elections". In: *2010 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 10)*. Washington, DC: USENIX Association, Aug. 2010. URL: https://www.usenix.org/conference/evtwote-10/performance-requirements-end-end-verifiable-elections.

[118] Swiss Post. *Testare il voto online*. Accessed: 3-Feb-2025. 2025. URL: https://demo.evoting.ch/it.

[119] Alguliyev Rasim and Yusifov Farhad. "Multi-criteria evaluation of electronic voting system security threats". In: *Cybersecurity issues* 3 (27) (2018), pp. 16–21.

[120] Ronald L Rivest, Adi Shamir, and Leonard Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[121] Peter YA 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 20*. Springer. 2016, pp. 176–192.

[122] N. Sakimura, J. Bradley, and N. Agarwal. *Proof Key for Code Exchange by OAuth Public Clients*. RFC 7636, Internet Engineering Task Force (IETF). Sept. 2015. URL: https://tools.ietf.org/html/rfc7636 (visited on 02/11/2025).

[123] Kazue Sako and Joe Kilian. "Receipt-free mix-type voting scheme". In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1995, pp. 393–403.

[124] Kristine Salamonsen. "A Security Analysis of the Helios Voting Protocol and Application to the Norwegian County Election". MA thesis. Institutt for matematiske fag, 2014.

[125] Roberto Samarone dos Santos Araújo. "On remote and voter-verifiable voting". PhD thesis. Technische Universität, 2008.

[126] Adam Shostack. *Threat modeling: Designing for security*. John Wiley & Sons, 2014. ISBN: 978-1-118-80999-0.

[127] Adam Shostack. *Threat modeling: Designing for security*. John Wiley & Sons, 2014.

[128] Victor Shoup. "Sequences of games: a tool for taming complexity in security proofs". In: *cryptology eprint archive* (2004).

[129] Melissa A. Smith, Samuel S. Monfort, and Eric J. Blumberg. "Improving Voter Experience Through User Testing and Iterative Design". In: *Journal of Usability Studies* 10.4 (2015), pp. 116–128.

[130] Ben Smyth. "Mind the Gap: Individual-and universal-verifiability plus cast-as-intended don't yield verifiable voting systems." In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 1054.

[131] Ben Smyth et al. "Towards automatic analysis of election verifiability properties". In: *Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security*. Springer. 2010, pp. 146–163.

[132] Morten L. Sondahl et al. *Usability Evaluation of Passwords with Multiple Words: A Case Study on Long Textual Passwords*. https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2562795. 2019. (Visited on 02/11/2025).

[133] C. Spadafora et al. *Coercion-resistant i-voting with short PIN and OAuth 2.0*. E-Vote-ID 2023. 2023. DOI: 10.18420/e-vote-id2023_04.

[134] Chiara Spadafora, Riccardo Longo, and Massimiliano Sala. "A coercion-resistant blockchain-based e-voting protocol with receipts". In: *Advances in Mathematics of Communications* 17.2 (2023), pp. 500–521.

[135] Blomkvist. Stefan. "Persona - an Overview". In: *Theoretical perspectives in Human-Computer Interaction*. 2002. URL: https://api.semanticscholar.org/CorpusID:13518131.

[136] *Swiss Post: Swiss Post Voting System*. URL: https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/tree/master/System (visited on 04/11/2025).

[137] Alexander H Trechsel, Vasyl V Kucherenko, and Frederico Ferreira Da Silva. *Potential and challenges of e-voting in the European Union*. Tech. rep. European University Institute, 2016.

[138] Meltem Sönmez Turan et al. *NIST SP 800-90B Recommendation for the Entropy Sources Used for Random Bit Generation*. NIST, Jan. 2018. DOI: 10.6028/NIST.SP.800-90B. (Visited on 02/11/2025).

[139] Tony UcedaVelez and Marco M Morana. *Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis*. 2015.

[140] Melanie Volkamer, Oliver Spycher, and Eric Dubuis. "Measures to Establish Trust in Internet Voting". In: *Proceedings of ICEGOV '11*. 5th International Conference on Theory and Practice of Electronic Governance, ICEGOV '11'. ACM Digital Library, 2011.

[141] Hong Wang, Yuqing Zhang, and Dengguo Feng. "Short threshold signature schemes without random oracles". In: *International Conference on Cryptology in India*. Springer. 2005, pp. 297–310.

[142] Xin Wang. "Personas in the user interface design". In: *University of Calgary, Alberta, Canada* (2014).

[143] Jeff Yan et al. "Password memorability and security: Empirical results". In: *IEEE Security & privacy* 2.5 (2004), pp. 25–31.

[144] Yurong Yao and Lisa Murphy. "Remote electronic voting systems: An exploration of voters' perceptions and intention to use". In: *EJIS* 16 (Apr. 2007), pp. 106–120. DOI: 10.1057/palgrave.ejis.3000672.

[145] Marie-Laure Zollinger et al. "User Experience Design for E-Voting: How mental models align with security mechanisms". In: *E-Vote-ID 2019 TalTech Proceedings* (2019). DOI: 10.13140/RG.2.2.27007.15527.

Nomenclature for Vote App

$\mathcal{S}$	Digital signature scheme
$\mathcal{E}^r_{\mathtt{pk}}[v]$	Encryption of a message v under the (threshold) modified exponential ElGamal cryptosystem, using randomness r and public key $\mathtt{pk}$
$\mathbb{G}$	Group of prime order p
$\mathcal{H}$	Hash function
$\mathcal{H}_p$	Hash function with outputs in $\mathbb{Z}_p$
H	Extensible output hash function
$\mathbb{Z}_p$	Group of residues modulo a prime p
$E^r_{\mathtt{pk}}[v]$	Encryption of a message v under the (threshold) modified ElGamal cryptosystem, using randomness r and public key $\mathtt{pk}$
g_0	Generator of $\mathbb{G}$. It is used in the modified exponential ElGamal cryptosystem to encrypt the message
g_1	Generator of $\mathbb{G}$. It is used for the public key of the modified (exponential) ElGamal cryptosystem
g_2	Generator of $\mathbb{G}$. It is used for the public key of the modified (exponential) ElGamal cryptosystem
g_3	Generator of $\mathbb{G}$. It is used to construct the public voting credential and the public key of the RTs
o	Generator of $\mathbb{G}$. It is used in relation with the private credential
p	Prime number, $p = ord(\mathbb{G})$
$\mathtt{EL}_{\mathtt{ID}}$	Unique identifier of the election
$\mathtt{pk}_{\mathtt{EL}}$	Election public key
$n_{\mathtt{BB}}$	Number of Ballot Boxes
$n_{\mathtt{TT}}$	Number of Tabulation Tellers
$\mathtt{pk}_{\mathtt{ACC}}$	Public key of the Registration Tellers used to generate Anti-Coercion Credentials
$\mathtt{pk}_{\mathtt{RT}}$	Public key of the Registration Tellers
$\mathtt{pk}_{\mathtt{TT}}$	Public key of the Tabulation Tellers used to encrypt the votes
$n_{\mathtt{RT}}$	Number of Registration Tellers
$t_{\mathtt{RT}}$	Threshold for the Registration Tellers
$\mathtt{sk}_{\mathtt{RT}} = y$	Secret key of the Registration Tellers, where the share of RT_i is $\mathtt{sk}_{\mathtt{RT}_i} = y_i$
$t_{\mathtt{TT}}$	Threshold for the Tabulation Tellers
ACC	Anti-Coercion Credential, or voting credential
BB	Ballot Boxes
DIP	Digital Identity Provider
ER	Electoral Roll
NS	Notification Server
RT	Registration Tellers
TT	Tabulation Tellers
VS	Verification Service
WBB	Web Bulletin Board
$\mathtt{Ca}_{i,j}$	Identifier for the j-th candidate associated to the i-th electoral list
$\mathtt{pk}_{\mathtt{DV}}, \mathtt{sk}_{\mathtt{DV}}$	Public and private key pair for the designated verifier
$\mathtt{id}$	Personal data of the eligible voter
$\mathtt{Ls}_i$	Identifier for the i-th electoral list
$n_{\mathtt{Ca},i}$	Number of candidates associated to the i-th electoral list
$n_{\mathtt{Ls}}$	Number of electoral lists
$n_{\mathtt{V}}$	Number of Voters
$\mathcal{V}$	Voter
$v_{\mathtt{id}}$	Pseudonymous identifier of the Voter
$\left(A, r, E_{\mathtt{pk}_{\mathtt{TT}}}[A], E_{\mathtt{pk}_{\mathtt{ACC}}}\left[g_1 \cdot g_3^x\right]\right)$	Public ACC of the Voter
$\mathtt{L}_{\mathtt{PIN}}$	Length of the $\mathtt{PIN}$

$n_{\texttt{ACC}}$	Number of Anti-Coercion Credentials created, where $n_{\texttt{ACC}} > n_{\texttt{V}}$ to accommodate revocations and re-issues in case of emergencies
$\overset{\texttt{valid}}{\texttt{PIN}}$	Valid PIN of voter $v_{\texttt{id}}$
$\texttt{PIN}$	Identifier for a general PIN, which could be either valid or ruse
$\overset{\texttt{ruse}}{\texttt{PIN}}$	Last ruse PIN set up by voter $v_{\texttt{id}}$
σ	Mask used to hide the private ACC and from which the PIN is derived
$\overset{\texttt{ruse}}{x}$	Valid private ACC of the voter
$\overset{\texttt{valid}}{x}$	Valid private ACC of the voter
x	Identifier for a general private ACC, which could be either valid or ruse
$\texttt{L}_{\texttt{psph}}$	Length of the passphrase psph
$\texttt{L}_{\texttt{wl}}$	Length of the wordlist wl
$\texttt{psph}$	Passphrase to retrieve $\texttt{sk}_{\texttt{DV}}$ in case of change of voting device, with length $\texttt{L}_{\texttt{psph}}$
$\tau_{\texttt{max}}$	Maximum waiting time
$\tau_{\texttt{min}}$	Minimum waiting time
$\texttt{wl}$	Wordlist of length $\texttt{L}_{\texttt{wl}}$

Appendix A

On the Usability of Vote App

This chapter presents insights from the usability research conducted on *Vote App*. After introducing the methodology used to develop the app in Appendix A.2, we discuss two usability studies: one evaluating the effectiveness of two hash digest encodings in Appendix A.3 and the other assessing the usability of *Vote App* in Appendix A.4. Appendix A.6 presents the interfaces of *Vote App*.
The content of this chapter is based on the work [47, 46, 23].

A.1 Introduction

In the development of an e-voting system, security is undoubtedly crucial but neglecting user research can greatly impact its acceptance and adoption: voters must *believe* that the e-voting system is secure and they must *trust* it. By developing a user-friendly application that is also highly secure, we can maximize the adoption of e-voting technology and safeguard the integrity of the electoral process.

To place the user at the center of protocol development, we employed our newly developed framework, the *Human-Centered approach in Cryptography and Cybersecurity (HC3)* (see Appendix A.2). This is an iterative 7-step framework that integrates user research with cryptography and cybersecurity considerations, ensuring that both aspects evolve together.

After the development of *Vote App*, we evaluated its usability to not only test the application itself but also assess the effectiveness of the HC3 framework in balancing security and usability. To achieve this, we conducted

a usability test (see Appendix A.4, [47, 46]) composed by a pilot test to refine the methodology and a main test. Furthermore, we conducted another usability test to assess the emoji visual encoding mechanism for comparing hash digests (see Appendix A.3 and Section 3.8.5, [23]).

Related Works. User acceptance of e-voting systems depends on both security and usability [140], with several studies emphasizing the role of human factors during system design [88]. Lack of usability can significantly affect voter success rates, as seen in [2], where only 58% of participants completed the voting process.

Despite the well-known trade-off between usability and security, few studies have involved users early in the design process. The paper [129] highlights the benefits of iterative usability testing, while [92, 144] identify trust and perception as key factors for the system adoption. Verifiable e-voting systems present further usability challenges, which are explored in [101]. In addition, [137] provides technical recommendations based on real-world deployments. Given the sensitivity of e-voting, usability testing with real users is critical [145, 73].

The usability/security trade-off extends to cryptographic mechanisms, especially hash digest verification. Traditional encodings, such as hexadecimal or base-64, are complex and error-prone, leading researchers to explore alternative representations. Studies have proposed word-based [60], sentence-based [67], and image-based [93] encodings, with significant improvements in usability [86]. While no studies have looked specifically at emoji for hash digests, research on authentication systems suggests that they improve memorability and user satisfaction [102]. Emoji-based PINs, as examined in [132], demonstrate high usability, quick input, and user preference over alphanumeric passwords.

A.1.1 Usability metrics

According to ISO 9241-11 [77], usability is defined as the *effectiveness*, *efficiency* and *satisfaction* with which users accomplish specific objectives within particular environments. The National Institute of Standards and Technology (NIST) recommends these three metrics as suitable for assessing e-voting systems [91].

Effectiveness is defined as the completeness and accuracy with which a user can achieve a specific task [100]. In electronic voting, effectiveness is the

measure of voters' ability to successfully cast their vote for their intended candidate without encountering errors [24, 68]. To determine effectiveness, the examiner can observe participants while they perform the task, use visual recording, or ask participants to self-report their progress. Additionally, Thinking Aloud [107] can be employed. This method allows participants to vocalize their thoughts as they engage with the system [107]. Error rates are also an effective way for verifying effectiveness, especially in the case of e-voting, as they are linked to the voter's intention and the actual outcome [100].

Efficiency assesses whether participants achieve their goals without using excessive resources, striking a balance between effort and time [100, 68]. For e-voting, it is represented by the time voters spend casting their vote and the time required by the user to complete verification steps [100, 24].

Satisfaction refers to how happy and at ease users feel during their interaction with a system or process [24, 68]. Satisfaction is the only subjective usability metric recommended by NIST. In e-voting, satisfaction reflects how pleased or fulfilled voters feel while participating in the election [24]. Satisfaction can be measured with standard methods like the System Usability Scale (SUS) questionnaire [68] or it can also be measured using a non-standardized questionnaire developed by the examiner, tailored to the specific purpose of the study [100].

The SUS [68] is a 10-item survey designed to collect subjective usability assessments from participants in a usability test, evaluating aspects like efficiency, intuitiveness, ease of use, and satisfaction. Participants express their agreement or disagreement with each statement using a five-point Likert scale: *strongly disagree* (1), *disagree* (2), *neutral* (3), *agree* (4) and *strongly agree* (5) [70].

A.2 The HC3 Framework

The *Human-Centered approach in Cryptography and Cybersecurity (HC3)* [47] framework was developed to prioritize user needs at every stage of the e-voting system design and implementation process. It comprises seven iterative steps:

1. *State-of-the-Art Analysis.* This step involves reviewing existing research on e-voting systems usability and collecting socio-demographic

data on potential users. Based on this information, Personas (see Appendix A.2.1) are defined to represent different user archetypes.

2. *Identification of Requirements.* Usability and accessibility requirements are identified and aligned with the essential security properties an e-voting protocol must satisfy.

3. *Design and Specifications.* The cryptographic protocol serves as a basis to define the User Flow (see Appendix A.2.1) and to define the User Journeys (see Appendix A.2.1) tailored to each Persona.

4. *Refinement and Implementation.* Service Blueprints are developed for every possible voting flow (see Appendix A.2.1), integrating cybersecurity requirements with the usability considerations established in earlier steps.

5. *Initial Graphical Rendering.* Multiple graphical designs of the interfaces are explored to visualize the solution.

6. *Testing.* Comprehensive evaluations of the e-voting system are conducted with potential users, carefully chosen to reflect the defined Personas. These tests ensure the solution is both secure and user-friendly.

7. *Continuous Evaluation and Iteration.* Based on the results obtained, the process may restart from the *Design and Specifications* phase, enabling refinements such as:

   (a) updating usability methods, including revising Personas and enhancing the graphical interface to better meet user needs;

   (b) improving the cryptographic protocol to simplify critical steps for users without compromising security.

The data obtained from applying the *HC3 framework* in the development of *Vote App* is provided in Appendix A.2.1 and [47].

A.2.1 Application to Vote App

A.2.1.1 User Flow

As defined by [85], the User Flow represents a series of actions that outline the optimal sequence of steps required to complete a task using a product or

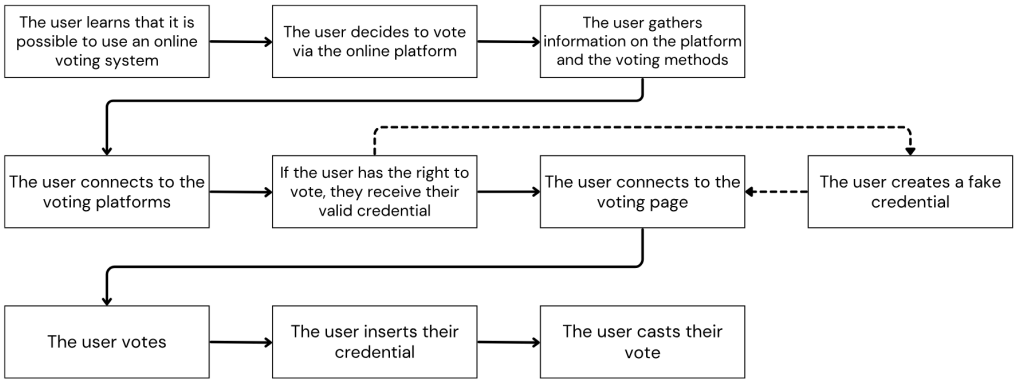

Figure A.1: User Flow for *Vote App*. The dashed lines show the flow for obtaining fake credentials.

a service. In Figure A.1[1] the User Flow for *Vote App* is presented.

A.2.1.2 Personas

Personas are a usability technique that uses the creation of fictional characters to represent potential user types of a product or service [84]. They are designed based on the data collected during the early stages of the design process [135] and their definition helps the project team gain a deeper understanding of the end users by creating a tangible representation that embodies consistent and reliable insights into the user groups [142]. Personas are typically defined in a narrative way, with each one assigned a unique name and biographical details. This is followed by a description of their interests, aspirations, attitudes toward the core theme of the solution being tested, and their interactions with technology. In Figure A.2 we provide a shortened version of the Personas we created.

A.2.1.3 User Journey

The User Journey illustrates the path that each user takes while interacting with a product or service [58]. Typically, a User Journey is defined as follows:

[1] All the illustrations have been made using *Canva* [30].

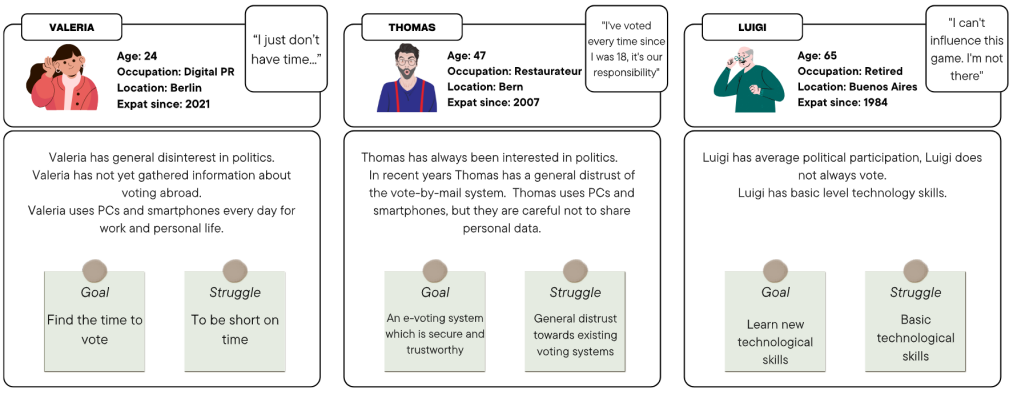

Figure A.2: Personas created to reflect potential users of the e-voting system.

- the first two lines define the actions the user must perform to use the solution under analysis;
- the next two lines focus on the subjective perception of each persona: their *thoughts* and their *feelings*;
- the last line presents the potential improvements that could be made based on each persona's negative thoughts.

In Figure A.3, the User Journey of *Valeria* (see Figure A.2) is presented.

A.2.1.4 Service Blueprint

The Service Blueprint [65] illustrates the relationships between various service components that are directly linked to a specific User Flow, and it highlights the distinction between what users experience *frontstage* and the operational and support processes *backstage* when interacting with a product or service [74].

The key elements to define a Service Blueprint are [65]:

- *User Actions:* the steps the end user must take;
- *Frontstage Actions:* what occurs directly in the user's line of sight;
- *Backstage Actions:* what happens behind the scenes to support the Frontstage Actions.

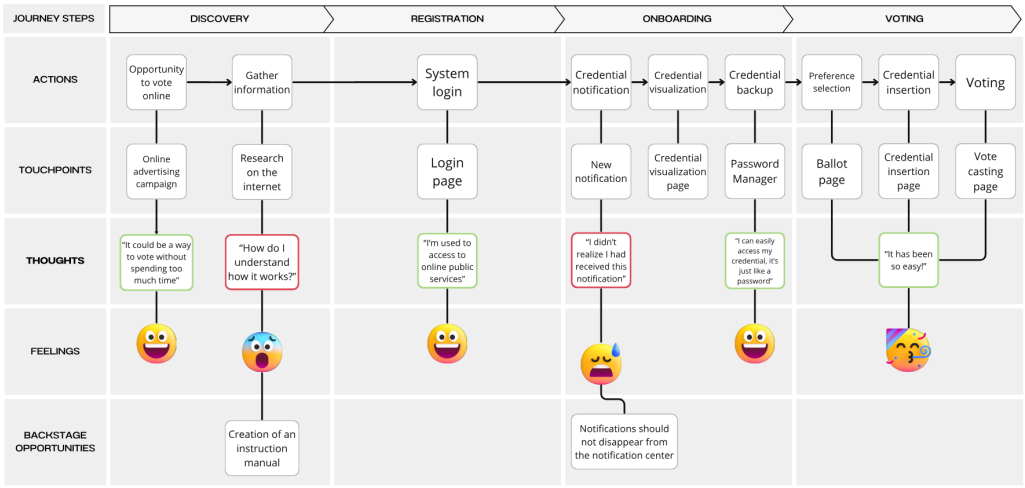

Figure A.3: Valeria's User Journey. Positive thoughts are highlighted in green, negative thoughts in red, and emotions are represented using emojis.

In Figure A.4, the Service Blueprint for both possible flows (with and without fake credentials) for *Valeria*, is presented.

A.2.1.5 Mockup and Prototype

Before developing the prototype of *Vote App*, we designed two mockups: a first, static, realized with Balsamiq [10], allowed us to reproduce low-fidelity interfaces focusing on structure and content, providing a first visual representation of the e-voting protocol. Then, to deliver a more interactive experience with animation and dynamic overlays, a new interactive mockup was made with Figma [59].

A.3 Usability Challenges in Manual Integrity Checks

This section presents a the usability study [23] we conducted to evaluate two encoding schemes for hash digest verification, along with the results obtained.

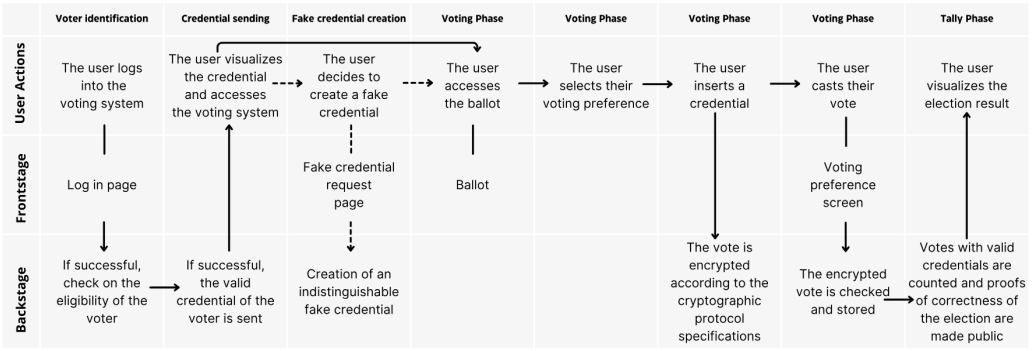

Figure A.4: Service Blueprint of an e-voting system based on the *fake credentials* mechanism. The Service Blueprint considers both the flows, with and without the employment of fake credentials.

A.3.1 Scope

Comparing hash digests to certify the integrity of data is a widely used approach in various applications, such as verifying file downloads (like operating system images), since comparing hashes is a lightweight and efficient mechanism for verification. While this is effective for automated machine checks, recalling and distinguishing long bit strings or their hexadecimal representation is particularly not user-friendly. In some cases it is important to allow users to manually perform integrity checks in order to have some protection against faults or malware. Therefore, our goal is to make manual verifications usable and as straightforward as possible, enabling users to easily and independently perform integrity checks. In particular, we would like a usable and effective system when integrated in a smartphone application.

A.3.2 Requirements

Our proposed solution is to encode the hash digest into an alphabet of visual representations, that we call *glyphs*. In order to achieve our goal, this alphabet and the encoding must satisfy the following requirements:

A. *Ease of glyph distinguishability:* users must be able to easily distinguish between all glyphs.

B. *Ease of sequence distinguishability:* users must be able to distinguish

between two distinct sequences of glyphs. If a sequence comprises too many glyphs, then it becomes too difficult to distinguish between sequences; therefore, a sequence should be as short as possible, and not exceed 7 glyphs [75].

C. *Collision resistance:* in order to detect anomalies, it is necessary that there are enough distinct sequences, so that it is very unlikely that two distinct hash digests are represented with the same sequence of glyphs. We set the minimum threshold at 25-bit collision resistance.

D. *Concise representation:* screen real estate is at a premium on mobile applications, so delivering the same information while minimizing space is a desirable distinguishing factor.

E. *Ease of recall:* the user must be able to recall a previously seen sequence of glyphs without excessive effort. Glyphs for which semantics associations are easy may be aided by the dual use of visual and lexical memory.

F. *Ease of encoding:* if the number of distinct glyphs is a power of 2, it is much simpler to encode a hash digest into a sequence of glyphs.

G. *Unicode encoding:* if the chosen glyphs can already be represented using a known and widely available set of characters, implementation will be easier, less error-prone, and require less data to download.

H. *Properness in context*: the chosen set of glyphs should not be mistaken for an application error.

A.3.3 Our Proposals for Visual Encodings

We recall that to achieve 25-bit collision resistance, it is required to have 50 bits of information. The number of glyphs required to obtain 50 bits of information is $\frac{50}{log_2(\text{alphabet size})}$ where log_2(alphabet size) indicates the number of bits of information each symbol of the alphabet provides. Thus, combining with requirement B, this means that there should be at least 142 different individual glyphs in the chosen alphabet; the more unique glyphs are available, the fewer glyphs have to be used in a sequence to obtain the same level of security.

Let $H(m, l)$ be a hash function, where m is the input to be hashed and l the required length of the digest. Given an ordered alphabet of glyphs, we can check the integrity of a value Ω with a sequence computed as follows:

1. let n_e be the number of glyphs in the sequence, let b be the bits of information given by each glyph;
2. compute $\hat{\Omega} = H(\Omega, b \cdot n_e)$, where the length of the digest $b \cdot n_e$ is expressed in number of bits;
3. split $\hat{\Omega}$ in strings of b bits each: $\hat{\Omega} = (\hat{\Omega}_1, \ldots, \hat{\Omega}_{n_e})$;
4. interpret each $\hat{\Omega}_i$ as an integer j_i, the i-th glyph on the final sequence is the j_i-th glyph of the alphabet.

In the context of this research, the two most promising glyph alternatives that we identified are:

- *Emoji:* From all the available emoji in Unicode, we selected a subset of 2^{10} (1024) following requirements A, E, H. A digest of 50 bits can be straightforwardly encoded as a string of 5 emoji, satisfying requirement B.
- *Words*: ideally a collection of 2^{13} (8192) words should be selected from the native language dictionary of test subjects (i.e., Italian), following requirements A, D, E, H. A digest of 52 bits could then be straightforwardly encoded as a string of 4 words, satisfying requirement B. For the test, we approximated the word collection by using the word list of BIP39[2] which has only 2048 elements, thus we actually encoded only 44 bits instead of 52.

Test Implementation. The test backend was implemented in Python 3 with Flask, with an end-point for each screen type. The frontend was built with React in JavaScript, while an Nginx proxy was set up in front to secure communication with TLS and provide resource caching. The configuration of HTTPS was validated with TLSAssistant [64], balancing security with compatibility with a wide range of mobile devices. The system has been deployed on a virtual machine within the Microsoft Azure cloud infrastructure.

A.3.4 Test Organization

We first conducted a pilot test at the Fondazione Bruno Kessler (FBK[3]), starting with participants from FBK's Center for Cybersecurity and later

[2] https://github.com/bitcoin/bips/blob/master/bip-0039/italian.txt
[3] Bruno Kessler Foundation, see https://www.fbk.eu/en/.

expanding to other departments. The main test was then conducted remotely, allowing a broader audience to participate via a publicly available access link.

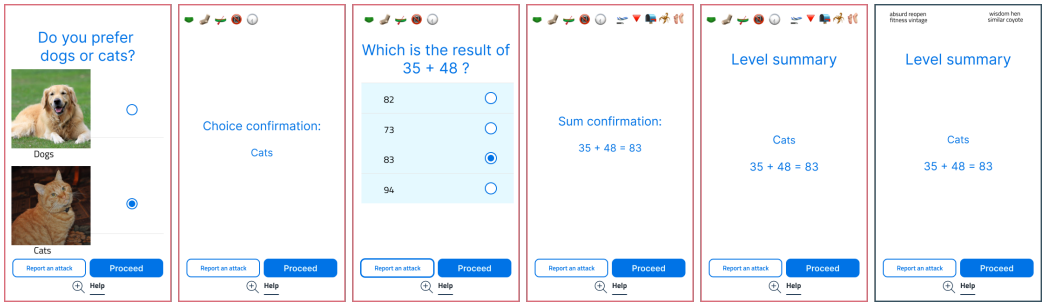

Figure A.5: Level overview. Screens from left to right: (i) question, (ii) choice confirmation with first emoji sequence, (iii) sum question, (iv) sum confirmation with second emoji sequence, and (v) level summary for both emoji and (vi) words encodings.

Scenario. The test consisted of a multi-level quiz, with each level containing questions related to personal tastes and simple arithmetic exercises together with a set of emoji or words displayed on the quiz interfaces[4] (Figure A.5). Participants could encounter simulated cybersecurity attacks during the quiz. These attacks caused two types of anomalies: *encoding anomalies* and *non-encoding anomalies*. If the set of emoji or words changed before completing the level (encoding anomaly), it indicated an ongoing attack, and participants were instructed to report the change. Non-encoding anomalies that an attack could cause were (i) the correct answer was absent from the possible options, (ii) an incorrect answer appeared on the choice confirmation screen; and, also in these cases, participants were asked to report it (Figure A.6). The objective was to assess participants' ability to detect changes in the set of emoji or words (visual integrity checks), and to compare it with their detection of alterations to the quiz questions or answers.

Gamification. Each quiz consisted of sixteen levels, with the first six mandatory and the remaining ten optional. To encourage participants to complete as many levels as possible, we awarded 100 points for each level

[4]All the interfaces have been created with Figma [59].

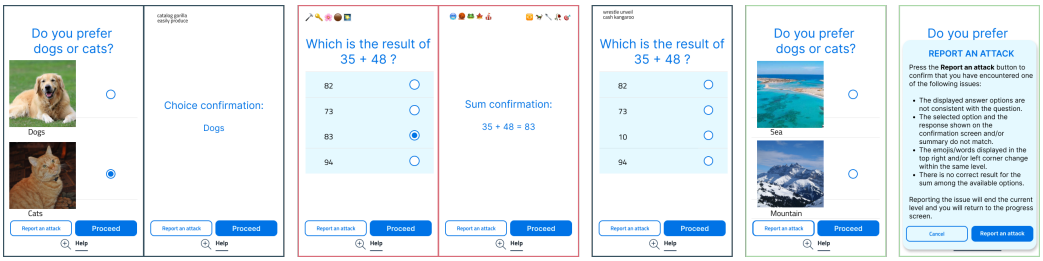

Figure A.6: From left to right: (i) a non-encoding anomaly (inconsistency between the selected and confirmed answer), (ii) an encoding anomaly (the sequence of emoji changes between two screens within the same level), (iii) a non-encoding anomaly (no correct result exists for the sum), (iv) a non-encoding anomaly (inconsistency between the question and the answers), and (v) the overlay to report attacks.

completed correctly. A level was considered correct if participants reported errors when present, or continued when no errors were present. Participants could track their progress on a leaderboard displayed at the end of each level. The leaderboard contained pre-generated scores carefully selected to create a sense of competition.

Anomaly Distribution. For each test we randomly selected which levels would present an anomaly according to a chosen distribution. Additionally, we discarded and re-generated tests that would have in the first six levels no anomalies or more than 3, and that would have in total less than 5 anomalies or more than 8.

Feedback. At the end of the quiz, participants were asked a set of final questions to assess their perception of the cybersecurity attacks they observed during the experience. Specifically, we asked them how often they encountered each type of anomaly and how often they reported it. This was intended to understand their perception of the changes and whether they reported them. If participants reported not having flagged an anomaly, they were asked to explain why.

Data analysis. We did not systematically analyze the data from the pilot because it was an internal test designed to refine the methodology. Instead,

we conducted a structured analysis for the main test, focusing on what we defined as *finished tests*, i.e., those in which participants completed all six mandatory levels. We created queries based on the relevant data and then aggregated the results in Google Sheets, categorizing anomalies as either *encoding anomaly* (changes in the visual encoding) or *non-encoding anomaly* (inconsistent or altered responses). We further segmented the data by type, distinguishing between emoji-based and word-based representations.

A.3.5 Results

The results of the pilot study are mostly related to improvements of the test itself and the user interface. Therefore, we will report only the results of the main test.

A total of 137 participants took part in the main test and completed at least all the six mandatory levels. Among them, 72 participants completed a test where the encoding was represented by a sequence of five emoji, while for the remaining 65 participants the encoding was represented by a sequence of four words. This design allowed for a comparative analysis of the two conditions. The demographic analysis showed that most participants belonged to the 20-34 age group, with an average age of approximately 27 years.

Of the 137 participants, 60 concluded the test after completing the six mandatory levels, while 77 attempted at least one bonus level.

Effectiveness. Effectiveness is measured as the percentage of times that a correct action has been performed. We define the percentage of false positives as the proportion of cases in which a user reported an anomaly when none was present.

- Figure A.7 shows that the number of false positives is very low, less than 3% for both encoding types, though slightly higher for emoji encoding. From a security perspective, false positives are not a concern, even if they may impact user experience.
- Figure A.8 illustrates how effectiveness varies across levels, classified by encoding type. Each level contains only one type of anomaly, and levels 1 and 3 never contain anomalies due to the way they were generated. There is no clear trend in effectiveness as levels progress, meaning that we cannot attribute any effects to fatigue or a desire to finish the test. Looking only at participants who completed at least the first bonus

level, the overall trend remains essentially unchanged, although effectiveness improves slightly, perhaps due to increased motivation. The difference in effectiveness between word and emoji encoding is statistically significant only at levels 7, 10, and 14.

- In Figure A.9, the effectiveness of *encoding anomalies* is measured as the percentage of times that the user reported an anomaly when there actually was an encoding anomaly in the screen. It shows very similar total effectiveness, even identical when the entire encoding sequence is changed. When change is limited to individual elements, words seem more effective, probably because their sequences are shorter. Interestingly, emoji encodings are quite more effective when the sequence is mirrored.

- Figure A.10 reports the difference between the number of anomalies a person reported and the number of anomalies they actually encountered. We define the distance as $d(a, b) = b - a$, representing how many participants (*count*) had a given distance value. If this difference is 0, it means that the participant reported exactly the number of anomalies they encountered. A positive value (right side of the graph) indicates that they reported more errors than were actually present, while a negative value (left side) means that they missed some anomalies. Ideally, the graph should be as narrow as possible, indicating that participants' performance was close to perfect. We can see that the graphs are actually quite narrow with a slight deviation to the left. Although the distribution is a bit different, the average of emoji and words encoding is almost the same. Results indicate that participants perceived more emoji changes than actually occurred, but their perception of emoji anomalies was closer to reality than for words. This suggests that emoji encoding is generally more effective but also more prone to false positives.

In the majority of the cases, we distinguish between tests that use emoji or word encoding, while *TOT* shows the same analysis without distinguishing between encoding types.

Overall effectiveness. The overall effectiveness results show that 19 participants made no errors, demonstrating a perfect understanding of the task. The majority, 113 participants, achieved an effectiveness between 80% and 99%, indicating a generally high level of accuracy with minor errors. Only 5 participants had an accuracy between 60% and 79%. These results show

that both emoji and words seem effective encoding methods and confirm that the test was well designed and clearly explained, as the vast majority of participants performed well.

Efficiency. Analyzing the time spent on each page of the test, we noticed that there were a number of very high values indicating that people abandoned the test for a while and resumed it at a (much) later time. Therefore, we filtered out the extreme values in order to obtain significant results. After this sanitization, the test showed a high level of efficiency, with participants spending around 14 seconds per page and an average total duration of the test of 24 minutes, in line with expectations.

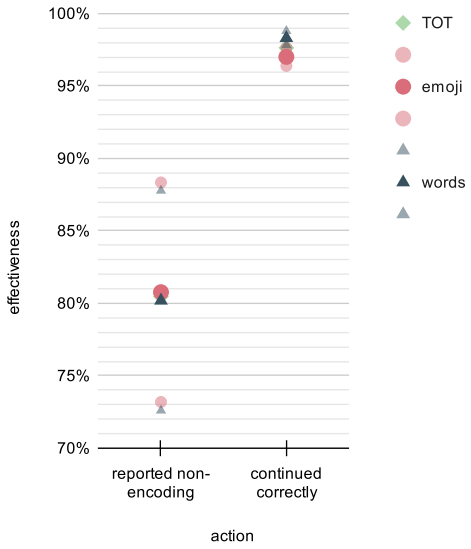

Figure A.7: Effectiveness in reporting non-encoding anomalies and in continuing when there are no anomalies, divided by type of encoding. The extremes of the 95% confidence intervals are displayed in lighter colors.

A.3.6 Human Base64

To make verifications easier for voters, we also modified the classic Base64 encoding (B64) to prevent confusion and errors by removing:

- letters which have similar lower and upper cases (keeping only one case): `Oo, Ss, Cc, Ww, Uu, Kk, Vv, Xx, Zz`;

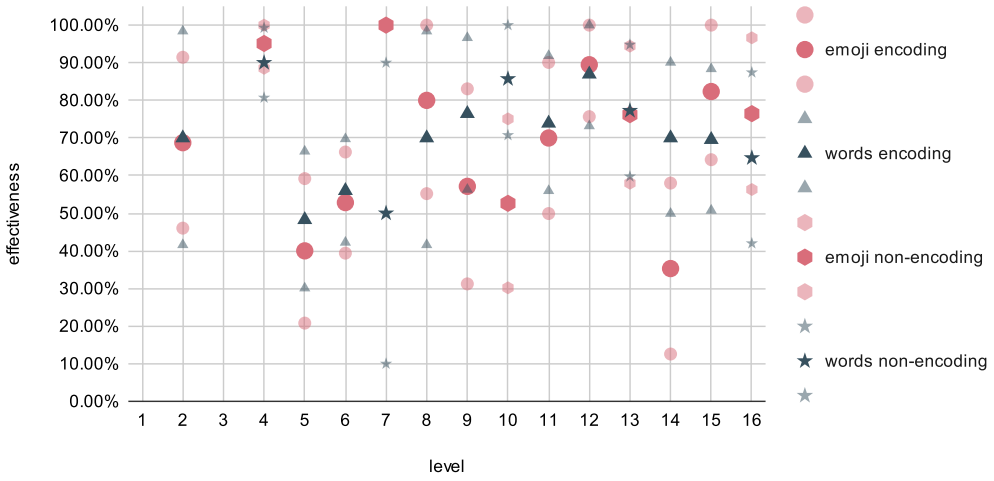

Figure A.8: Effectiveness level by level and divided by type of encoding of: (i) reporting an emoji or word encoding anomaly; (ii) reporting a non-encoding anomaly. The extremes of the 95% confidence intervals are displayed in lighter colors.

- different characters whose graphic rendering is too similar: `I1il`, `OO`, `5S`.

Then we integrated the alphabet by adding characters with the following features:

- easy to recognize;
- easy to pronounce, avoiding characters with names too similar;
- easy to type on modern keyboards (either phisical or touchscreen);
- not too visually similar to other characters.

To fulfill these requirements we also avoided to use two characters which are symmetric (`/\`, `<>`, `()`, `[]`, `{}`). Among the parentheses we only used] because it is the one that in handwriting is both distinguishable and easy to write. We added the symbol € even if it transcends ASCII, because it is easier to recognize, pronounce, and type with respect to the alternatives (~ is difficult to type on Italian keyboards, ^ does not have a widespread name in Italian).

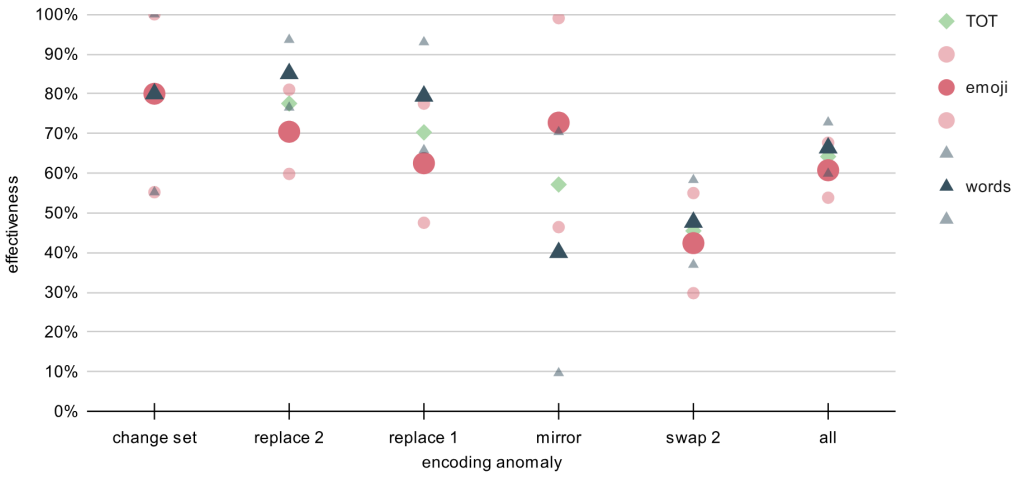

Figure A.9: Effectiveness in reporting encoding anomalies, by type of anomaly and divided by type of encoding. The extremes of the 95% confidence intervals are displayed in lighter colors.
Legend: *change set* refers to an entire changed sequence; *replace 1* or *2* refers to replacing 1 or 2 emoji/words in the set; *mirror* refers to the same sequence but in reverse order; *swap 2* refers to the same sequence with two emoji/words swapped; *all* refers to the totality of anomalies without subdivision.

As result, we defined a new encoding which is a variant of the classical Base64 encoding for byte strings, where characters which may be easily confused have been substituted by other symbols, using also non-ASCII characters. Table A.1 presents the conversion table from the classical Base64 to our new encoding that we call *Human Base64*.

In addition, we also identified the characters that are most prone to being miswritten or confused with others. To address this, we developed a substitution table that automates the correction of user inputs, available in Table A.2.

A.4 Usability Test of Vote App

After the development of *Vote App*, we conducted two usability tests [46]: a pilot test to refine the methodology and a main test to actually test its

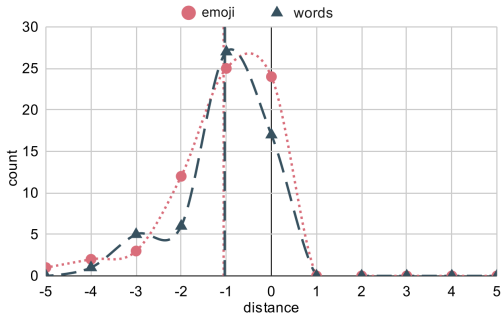

Figure A.10: Distributions of the distances $d(t, r)$ between the number of anomalies present in the *test* (t) and the number of anomalies *reported* (r). The vertical lines mark the averages.

Table A.1: Base64 to *Human Base64* conversion table. In the first row there are the characters removed from the B64 alphabet, in the second row there are the new characters in *Human Base64* that substitute them.

1	5	0	w	u	k	l	z	v	I	O	S	X	C
!	#	&	*	-	@	?	%	<	:	]	$	€	"

usability. Each participant was assigned one of two tasks: either to cast a vote or to create and use a fake credential to vote.

As with the visual-encoding usability test of Appendix A.3, we do not report the results of the pilot study because the main test provided the most relevant findings. The main purpose of the pilot was to refine the application and the test itself, improving its overall quality.

The main test was conducted during the 2023 European Researchers' Night, an EU initiative launched in 2005, that aims to bridge the gap between science and the general public. The event spanned from 5 pm to midnight, providing enough time to conduct multiple test sessions.

The research question (RQ) guiding the evaluation was: *Is the* Vote App *voting system usable by the general public in terms of effectiveness, efficiency, and user satisfaction?*

To address this, following the ISO 9241-11 [77] definition of usability and its three metrics (see Appendix A.1.1), we derive these three hypotheses:

Table A.2: Auto-correction of characters for *Human Base64*.

O	0	S	5	C	w	u	k	v	X	z	′′
o	o	s	s	c	W	U	K	V	x	Z	"

_	>	\	|	I	l	1	;	[	(	)	{	}
-	<	/	/	/	/	/	:	]	]	]	]	]	]

- *Effectiveness Hypothesis (H1):* *Vote App* will be highly effective, ensuring a complete and accurate voting process, free of user errors.
- *Efficiency Hypothesis (H2):* Using *Vote App* will be highly efficient, requiring minimal time and effort from users.
- *Satisfaction Hypothesis (H3):* Users will report high satisfaction with *Vote App*, based on their feedback on the voting experience.

H1 was tested by tracking the number of successful task completions and the number of attempts required; H2 was tested by measuring the time spent to complete the task and H3 was tested employing the System Usability Scale (SUS) questionnaire.

A.4.1 Test organization

During each test session, participants had to complete a task, specifically to cast a vote in a fictional election. After completing the voting process, participants were invited to check that their vote was recorded and to verify the accuracy of the information received through *Vote App*. These features were available in the main test through the Verification Service and the WBB (see Section 3.8.5) and explained in two posters. The first one provided a simplified explanation of the cryptography behind *Vote App*, specifically focusing on the creation and verification of voting credentials, the second one contained a step-by-step guide of *Vote App*.

Fictional Election. To speed up the voting process, the election was a referendum. Six different questions were created for participants to choose from (e.g., Q: “Do you prefer cream-flavored or fruit-flavored ice cream?”).

Scenario. Scenarios were not included to allow participation by multiple individuals simultaneously in a noisy environment. Each voter was given complete freedom to build their own scenario, choosing independently how many times to vote, whether to verify the PIN, and whether to set and vote with a ruse PIN. Almost 75% of the participants tested the anti-coercion functionalities.

User Authentication. The focus of the test was not on the authentication, so this step was simulated. We decided to simulate it instead of excluding it because it was necessary in order to correctly assign voting credentials. To authenticate, voters were required to choose a pseudonymous identity from a pre-generated set of options. Each identity was represented by a drawn portrait of an animal, fictional first and last names, and a code for logging into *Vote App*.

Equipment. The tests have been conducted using Android smartphones provided by the Center, with *Vote App* already installed and opened on the login page.

Test Session. We conducted multiple test sessions, each one involving from 3 to 5 participants, with approximate length between 10 to 15 minutes.

1. *Introduction.* We offered the participants a brief overview of the project. Subsequently, we presented the test instructions and we recommended to maintain a good flow of comments while interacting with the system, adhering to the Thinking Aloud (see Appendix A.1.1).
2. *Election Context.* We gave to the participating group a list of referendum questions, from which they could choose which one to vote on.
3. *Authentication.* Each participant simulated the authentication step as explained before.
4. *PIN reception. Vote App* provided to each participant a PIN which they could write down.
5. *Voting.* Each participant accessed the ballot and cast their vote and they were also able to view the registration of the vote on the WBB.
6. *Re-Voting, Ruse PIN and Verification.* Participants could set up a ruse PIN, re-vote with any PIN, verify a PIN.

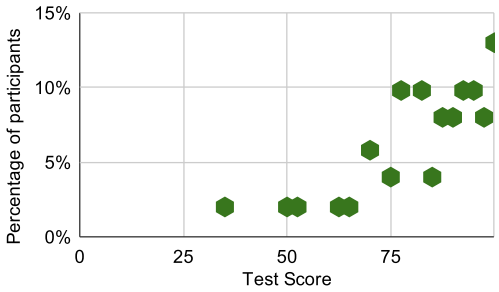

Figure A.11: SUS Results.

7. *Tabulation.* Upon completion of the voting phase by all the participants, we started the tabulation and displayed the results on the WBB.

8. *Post-Test Questionnaire.* At the end of each test session, we asked the participant to fill out the System Usability Scale (SUS) questionnaire.

A.4.2 Results

We performed an in-depth analysis of the data gathered which included the evaluation of the three usability metrics established by ISO 9241-11 [77]. Additionally, we reviewed the feedback provided by participants during their interactions with *Vote App*, aiming to gain valuable insights into their user experience and identify areas for improvement.

Participants. Approximately 80 people participated in the main test. Around 50% held a Bachelor's or Master's degree, 25% had no university degree, 7% held a PhD, and 18% had lower educational qualifications. Half of the participants were born between 1997 and 2003, with the rest spanning older and younger age groups. Students made up 67% of the sample, while 33% were employed.

A.4.2.1 Quantitative Results

Effectiveness (H1). All the participants managed to successfully complete the voting process and send the vote to the WBB. From the data gathered, we can state that participants were able to understand and use the verification service and to handle the ruse PIN request with lower error rates with respect to the pilot.

Efficiency (H2). All the participants managed to complete the test in 10 minutes, which was the time we expected for each test session to last.

Satisfaction (H3). In total, 50 out of the 80 participants in our demo compiled the SUS questionnaire. The total score was 84.9/100, with scores ranging from 35 to 100 out of 100 (see Figure A.11).

A.4.2.2 Qualitative Results

Here we present a brief summary of the qualitative data acquired via the Thinking Aloud method and our observations on the participants behaviors.

Positive Feedback. Based on participants' behavior, we observed that everyone wrote down their PIN, indicating the importance they attached to it, and the difficulty of remembering it. When we asked the participants who tested the anti-coercion functionalities to provide feedback on the ruse PIN, they did not have any negative or positive feedback. This suggests that we made a step towards improving explanations on *Vote App*, by communicating more efficiently the anti-coercion functionalities. Moreover multiple participants expressed positive feedback on emojis, stating that they were an interesting introduction and did not cause confusion. Moreover, their use in the voting process was well understood. The WBB introduction was well-received and participants found it to be useful and easy to use.

Negative Feedback. Some participants did not understand that the PIN could be retrieved if forgotten. Another participant mentioned that it was not clear that the PIN had to be entered each time voting. In terms of the use of emojis in general, one participant found them confusing. Regarding control numbers, participants thought that they were not well-explained and that it was unclear what they were for. Participants also questioned why they had to select one value, instead of letting the app doing it automatically.

Other suggestions. Some participants were confused by the usage of emojis to visualize the PIN and believed that the PIN was represented by them. The reason behind that is because the emojis of "Private PIN Emojis" were centrally placed on the PIN reception interface.

A.5 Conclusions

The usability test on emoji and word-based hash encodings demonstrated that both approaches were effective and efficient for secure voting applications. The findings indicated that font size had a notable impact on word-based encodings, with medium sizes favoring the detection of anomalies, while larger font sizes worked better for emoji-based encodings. Though performance decreased with age, emoji-based encodings outperformed words for older participants, suggesting that both methods can be used effectively across different demographics. The study also revealed that factors such as participant motivation had little effect on performance, indicating the robustness of both encoding methods. Future work could involve exploring additional encoding techniques and comparing them under various conditions.

The usability study of *Vote App* highlighted the significance of an intuitive design for coercion-resistant e-voting systems. Key factors such as clear instructions and the ability to view the registration of the vote on the web bulletin board contributed to its good usability. However, the balance between providing enough information and avoiding overwhelming the users proved challenging, as some participants found the detailed explanations essential, while others felt they were excessive. Despite these mixed opinions, the study revealed high user satisfaction, effective task completion, and quick voting processes. However, further testing of the anti-coercion features is required. Future research will focus on expanding the participant sample, refining the instructions, and improving the anti-coercion functionality.

A.6 Vote App

In the following subsections, the interfaces of *Vote App* are presented, divided by phase. All the graphics have been created with Figma [59].

A.6.1 Enrollment

Figure A.12 shows the enrollment steps of *Vote App*.

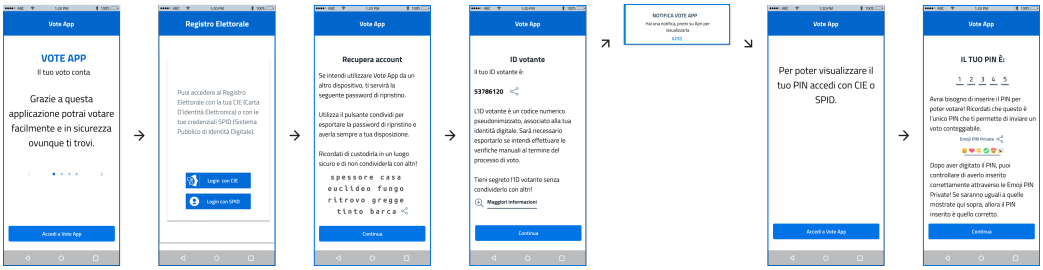

Figure A.12: Voting flow of *Vote App*.

A.6.2 PIN and Device Management

Figure A.13 shows how to set up a ruse PIN, verify the PIN and ask for the re-sending of the valid PIN using *Vote App*.

A.6.3 Voting

Figure A.14 shows how to vote using *Vote App*.

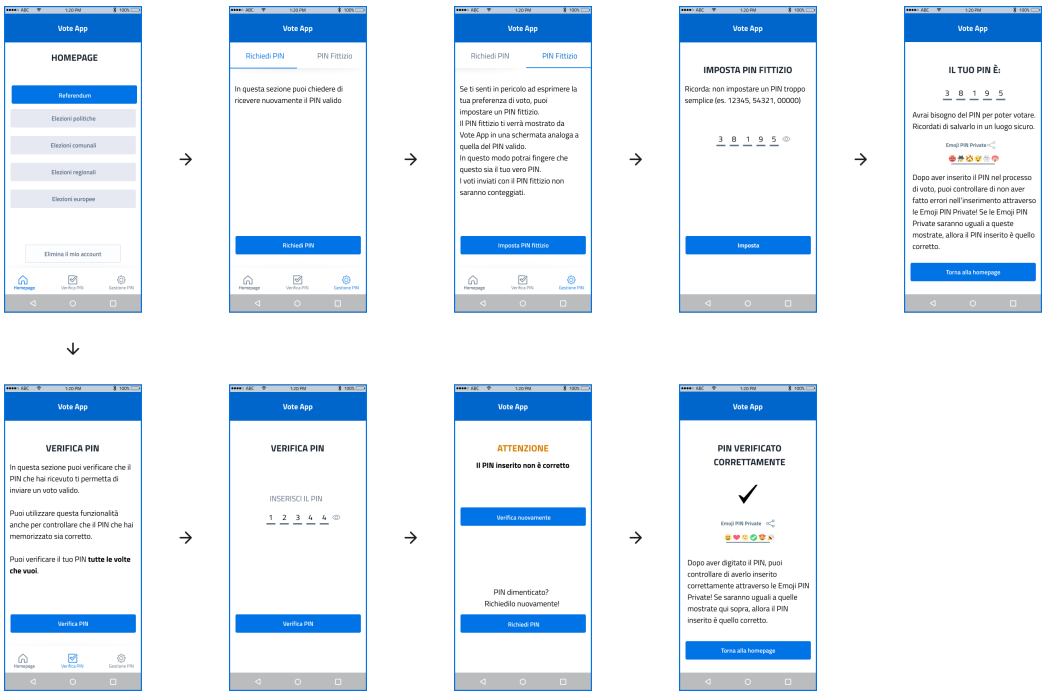

Figure A.13: Voting flow of *Vote App*.

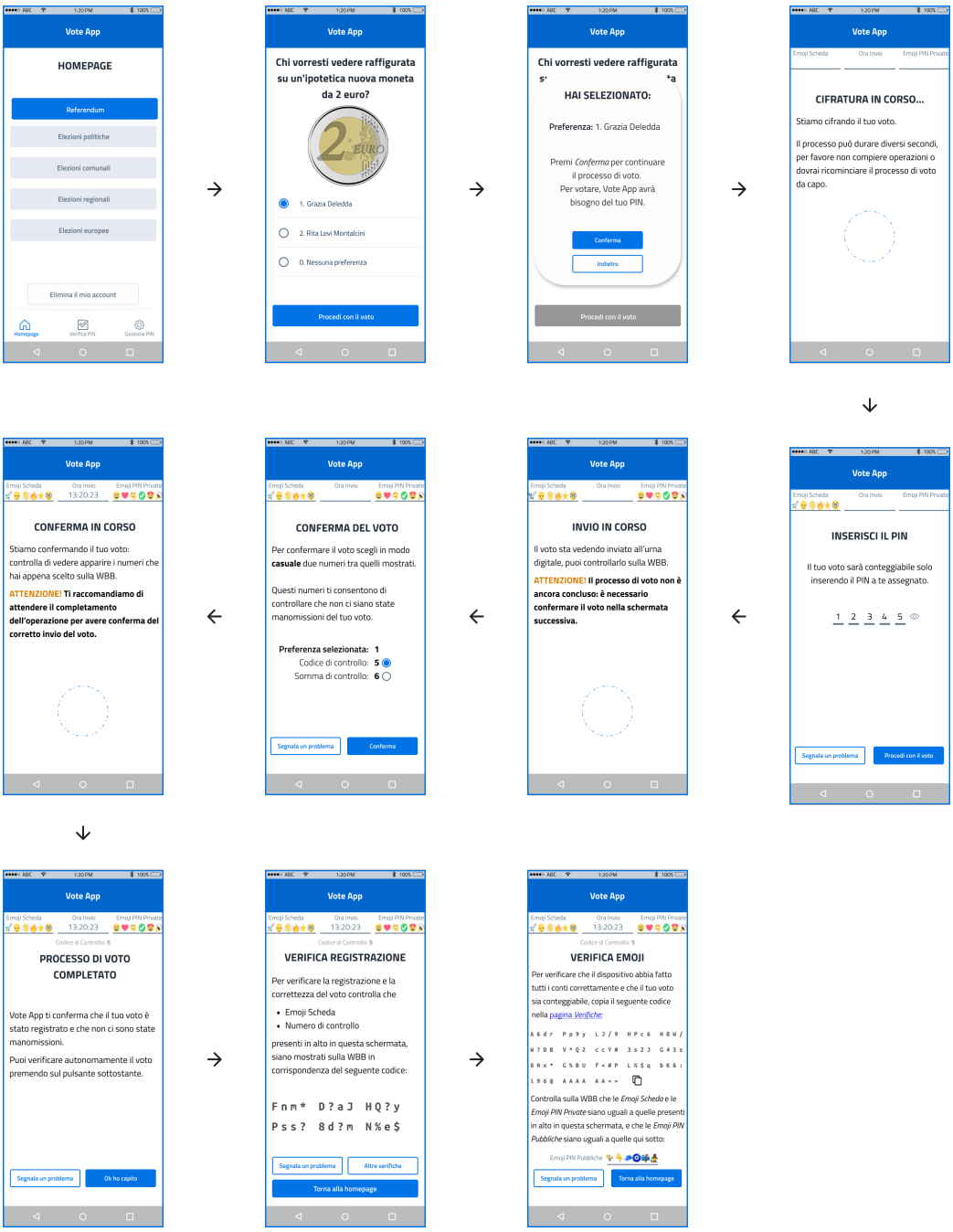

Figure A.14: Voting flow of *Vote App*.