University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Voting Verification Mechanism for a Distributed Ledger based Remote Electronic Voting System

Fabio Maddaloni
Zurich, Switzerland
Student ID: 15-703-150

Supervisor: Christian Killer, Eder Scheid, Prof. Dr. Burkhard Stiller
Date of Submission: April 21, 2021

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zurich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zurich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Fair, secure and trustworthy voting processes and elections are a cornerstone of any functioning democracy. Caused by the ongoing digitalization and digitization, new ways of voting are emerging. A highly discussed and promising topic is remote electrical voting, which allows users to cast their votes independent of the location and additionally (partly) independent of the used device. The combination of this would bring more flexibility to voters since they must not be present at a poll station at a particular time. For countries already allowing voting by mail, remote electronic voting is an evolution of this known practice.

As with every new technology, also with remote electrical voting, many obstacles must be conquered before it will be secure and reliable enough to be rolled out to the public. Especially in the scenario of electronic voting, cryptographic operations and encryption must be applied to guarantee features like vote secrecy, the resistance against coercion, or prevention against vote selling. Nevertheless, a voter must be confident that the indeed selected voting option is encrypted, transferred to the ballot box, and later counted in the tally. Proving this to the voter is not trivially since most electrical and cryptographic operations performed on data are not comprehensible and not verifiable without proper auxiliary tools.

This work focuses on the verification of the encryption of a selected voting option. The verification allows voters to verify if a ballot contains the chosen voting option or if the voting device tampers the selection before encrypting it. This will enable voters to verify whether their voting setup encrypts the selection trustworthy or if the voting device is cheating and altering the selection. Hence, voting with an unknown, unfamiliar, or not trusted device is possible. The thesis shows how to successfully implement the cast-as-intended property with the help of the challenge-or-cast mechanism into an existing remote electronic voting system. The challenge-or-cast mechanism allows voters to either challenge the encryption of a ballot or cast it. For the verification, a second device is needed such that the encryption can be repeated in an air-gapped environment. Furthermore, the challenge-or-cast approach is compared to other mechanisms having the same goal. Towards the end, the selected method is analyzed and discussed, revealing strengths, weaknesses, chances, and concerns.

Zusammenfassung

Faire, sicherer und vertrauenswürdige Abstimmung und Wahlen sind ein Eckpfeiler von funktionierenden Demkokratien. Aufgrund der Digitalisierung kommen verschiedene neue Abstimmungsprozesse und -möglichkeiten auf. Remote Electrical Voting, was mit "elektronische Fernabstimmung" ins Deutsche übersetzt werden kann, wird in diesem Zusammenhang oft diskutiert und als vielversprechend angeschaut. Dies da es neben der Ortsunabhängigkeit auch unabhängig vom Gerät ist, welches verwendet werden kann. Beides bringt Felxibilität für die Wähler, da diese nicht mehr in ein Wahlbüro gehen müssen, um ihre Stimme abzugeben. Für Länder, welche die Briefwahl bereits verwenden, ist es eine Evolution der bestehenden Möglichkeit.

Wie mit praktisch jeder neuen Technologie gibt es beim Remote Electronic Voting auch zahlreiche Hürden und Hindernisse, welche überwunden werden müssen, bevor die Möglichkeit von der Bevölkerung gebraucht werden kann. Speziell bei der elektronischen Stimmabgabe werden viele kryptografische Verschlüsselungen angewendet, um Eigenschaften wie das Wahlgeheimins zu halten, Nötigung zu verhindern oder de Möglichkeit zum Verkaufen der Wahlstimme einzugrenzen. Trotzdem muss ein Wähler sicher sein, dass die wirklich ausgewählte Wahloption verschlüsselt, verschickt und gezählt wird. Dies zu beweisen ist nicht trivial, da die meisten elektronischen und kryptografischen Operationen ohne Hilfsmittel weder nachvollziehbar noch überprüfbar sind.

Der Fokus dieser Arbeit liegt auf der Verifizierung der Verschlüsselung der ausgewählten Wahloption. Diese Überprüfung erlaubt es einem Wähler/einer Wählerin sicherzustellen, dass die benutzten Geräte vertauenswürdig sind und wirklich die gewünschte Option verschlüsseln. Dies erlaubt die Stimmabgabe mit unfamiliären, unbekannten oder nicht vertrauenswürdigen Geräten. Diese Arbeit zeigt wie die Cast-as-Intended Eigenschaft mit der challenge-or-cast Methode erfolgreich in ein bestehendes Remote Electronic Voting System implementiert werden kann. Die challenge-or-cast Methode erlaubt es Wählern entweder die Verschlüsselung ihrer Stimme zu überprüfen oder der verschlüsselte Stimmzettel einzureichen. Ein zweites unabhängiges Gerät wird verwedent, um die Verschlüsselung zu überprüfen. Im Weiteren wurde die challenge-or-cast Methode mit ähnlichen Ideen verglichen und eine Analyse der Stärken und Schwächen sowie der Möglichkeiten und Hindernissen durchgeführt.

Acknowledgments

I want to express my thankfulness to all people that have supported me in reaching this point. First and foremost, I would like to thank Christian Killer for his excellent support through the work of my thesis and especially for his rewarding feedback, for his enlightening discussions and guidance. The possibility to contact you any time and receiving instant feedback was a tremendous alleviation and highly appreciated. I also want to thank Prof. Dr. Burkhard Stiller for the opportunity to work and research on this fascinating and highly topical subject within the Communication System Group.

Moreover, I would like to thank all the people who supported me on my journey to achieve this important academic milestone.

Contents

Chapter 1

Introduction

Voting, elections, and polls are often applied when multiple people have to decide among a limited number of possibilities belonging to a particular topic. They are a widely used tool used to find consensus between individuals. Voting has many different faces, and its outcomes are of varying importance depending on the topic and the number of people it influences. In democracies, the power to determine legislation and rules is given to the people who are eligible to vote since they (*e.g.*) have reached a certain age or are native to a specific country, canton, or municipality. This results in many people being able to use their voice to vouch for their will and convenience. To ensure that there will be no fraud and make it possible for everybody to use their suffrage, there needs to be fair, secure, and trustworthy voting and processes to be established. Guaranteeing this is arguably the most crucial point of democracy, and as soon as this is not given anymore, this governmental form stops working as intended.

With the ongoing digitalization, new possibilities for voting polls arose. These opportunities range from machines scanning and counting handwritten ballots over computers used for casting votes in specific locations to truly remote electronic voting systems in which the location and device of the voter do not play a role anymore [25, 45, 74]. Since the circumstances between these techniques are different also the challenges, properties, and requirements are different [45]. As soon as the voting truly happens electronically, cryptographic operations are performed on the voting data. For most people, these operations make it impossible to trace the handling of a chosen voting option and the creation of a ballot. Therefore, mechanisms providing surveillance of the voting devices and the overall voting process are needed.

Many different steps and devices influence the surveillance of the voting process. Especially in the remote electronic voting setup, various devices, trustworthy and potentially malfunctioning ones, are part of the process. Since some of the devices, especially those where voters cast their ballot, are often not controlled by a reliable party or government but instead managed and owned by the voters themselves, these devices can be malicious and untrustworthy. In addition to the cryptographic processes in the ballot creation, voter-managed devices pose the problem of an unsupervised, insecure, untrustworthy, and potentially malfunctioning voting setup. This is a contradiction to the intention of secure and trustworthy voting processes that are a cornerstone of democracies.

1.1 Motivation & Description of Work

Resolving the previously described contradiction is one of the critical elements to make remote electronic voting available to the population. A system in which malfunctioning devices are not detected can lead to distorted and fraudulent results. Malfunctioning appliances could, besides other things, alter selected votes before encryption, not deliver ballots to the counting entity, or not count well-formed ballots at all. Every single option can cause tremendous changes in the result if conducted often enough. To prevent this, it is essential that the voting processes are transparent and voters can trust the devices they use to cast a vote. The fact that Switzerland started its (remote) electronic voting project *Vote électronique* in the year 2000 with three pilot cantons and still does not have established a country wide (remote) eVoting system, shows impressively how complex such systems are and how intricate their development is [18]. Thus, the Communication Systems Research Group (CSG) from the Department of Informatics (IfI) at the University of Zurich (UZH) started working on their Remote Electronic Voting system called Provotum in 2018 [59]. Provotum is still under development, and at the time of writing, the CSG is working on version 3.0 of it.

An essential point of electronic voting is the encryption process to create a ballot that a voter can cast. If the voting device alters the selected option at this step, an unselected voting option will be encrypted and counted in the tallying of the election or poll. The overwhelming majority of voters cannot comprehend the encryption, so that they would detect fraud only by looking at an encrypted ballot. Therefore, it is indispensable that there is an easy and straightforward process to verify the encryption that reveals whether an encrypted ballot indeed contains the selected voting option or if the voting device altered the selection before encryption.

This thesis is a contribution to the creation of a trustworthy and secure voting setup with devices not managed by a trusted entity. In detail, it covers the verification of the encryption such that every voter can easily verify if the device used for voting encrypted the indeed selected voting option. This is especially important if a voter does not use a device that they trust. However, even a trusted device can be malfunctioning or malicious due to malware or simple programming errors. Ensuring that the indeed selected option is encrypted and ready to be sent to the ballot box is a tremendously important step towards trustworthy remote electronic voting systems. This mainly because if a counterfeit ballot is cast, all following process steps can be correct and verified, but the result will still be fraudulent. Additionally, this step can only be detected by a voter personally, while other steps can also be verified by the broad public.

1.2 Thesis Outline

This thesis starts with a short introduction of cryptographic methods necessary for this work. Following is a chapter about related work, including the theory of vote verification and mechanisms thereof (Chapter 3). In the chapter called *Design*, the architecture of the chosen verification mechanism is illustrated, and in the following chapter the real

implementation to the Provotum environment is explained. In chapter 6, the comparison, discussion, and evaluation of the implemented solution is stated. The thesis closes with a summary and some words about future work.

Chapter 2

Cryptographic Methods

In this chapter cryptographic methods relevant for this work are shortly described. Namely the methods are the ElGamal encryption theory, zero-knowledge proofs (ZKP) including the Fiat-Shamir heuristic, Σ-protocols and Schnorr proofs, designated verifier proofs, chameleon hashes and trapdoor commitments.

2.1 ElGamal

ElGamal (1985) brought up the asymmetric ElGamal encryption scheme which is based on the difficulty to calculate discrete logarithms [23, 29, 68]. It can be used for key generation and thus also for encryption and decryption of ciphertexts [29, 68].

Domain Parameter

Domain parameters can be shared by numerous users and include p which is a large prime having the nature that $p-1$ is divisible by another prime q [68]. Furthermore, the generator g is defined as $g = r^{(p-1)/q} \pmod{p}$ with r as an element of a finite field [68]. This creates the abelian Group G of order q with generator g [29, 68].

Key Generation

The secret key sk is an integer chosen in the interval $[0, ..., p-1]$ and must be known only by the intended user itself [68]. The inherent public key pk is a modular exponentiation of the generator $pk = g^{sk} \pmod{p}$ [68]. This is formally summarized in equation 2.1.

$$\begin{aligned} sk &= random\ integer\ from\ [0, ..., p-1] \\ pk &= g^s k\ (\mathrm{mod} p) \end{aligned} \tag{2.1}$$

Encryption

The encryption creates the ciphertext pair (c_1, c_2) for the message m [68]. Since the ephemeral key k is chosen randomly from the range $[0, ..., p-1]$ for every encryption round, encrypting the same message will result in different ciphertexts [68]. The message m as well as the resulting ciphertexts are assumed to be in group G [68]. The encryption process is listed in equation 2.2 and based on the book *Cryptography Made Simple (2016)* [68].

$$\begin{aligned} k &= random\ integer\ from\ [0, ..., p-1] \\ c_1 &= g^k \pmod{p} \\ c_2 &= m * pk^k \pmod{p} \end{aligned} \tag{2.2}$$

Decryption

To decrypt the ciphertext pair (c_1, c_2) the secret key sk must be known. Then the decryption can be done according to equation 2.3 [68].

$$\frac{c_2}{{c_1}^{sk}} = \frac{m * pk^k}{g^{sk*k}} = \frac{m * g^{x*k}}{g^{sk*k}} = m \pmod{p} \tag{2.3}$$

2.2 Zero-Knowledge Proofs

In this section, zero-knowledge proofs (ZKP) are explained. Zero-knowledge proofs are proofs which reveal nothing except the validity of the statement in question [27]. Furthermore, the Fiat-Shamir heuristic is stated and interactive as well as non-interactive Schnorr proofs are listed.

Zero-knowledge Proofs were introduced by *Goldwasser et al. (1985)* and have the feature that a prover can proof that he/she has knowledge about something without revealing the secret itself [27]. They defined ZKPs as proofs which do not provide additional knowledge beside the correctness of the assertion in question [27].

According to *De Santis et al. (1987)*, zero-knowledge proofs must satisfy three properties, assuming honest entities included [22, 68]:

1. **Completeness:** The likeliness of successfully proving a true theorem to a honest verifier is overwhelming. If the prover knows the statement, the verifier should accept.

2. **Soundness:** The probability of successfully proving a false theorem is negligible. Here a probabilistic guarantee is provided meaning there is an insignificant possibility of accepting a false statement. If the prover does not know the statement, the likeliness of a verifier accepting the proof should be extraordinary small.

3. **Zero-knowledge:** The only information a proof reveals is the validity of the theorem in question.

Furthermore, ZKPs are only possible if the prover and the verifier are not completely independent [22]. *De Santis et al. (1987)* state that the simplest way to achieve this is when the prover and verifier share a random string [22].

zero-knowledge proofs can be of *interactive* or *non-interactive* type [22]. In an interactive setting, usually Σ-protocols are used where a verifier challenges the prover at the time of action [16,22]. Σ-protocols are a special form of three-move protocol in which a prover first sends a commitment, then a verifier responses with a challenge which will be answered by the prover with an appropriate response [16, 68]. It is assumed that the verifier is honest and sticks with the protocol [68]. With he help of the *Fiat-Shamir heuristic*, an interactive ZKP can become a non-interactive zero knowledge proof (NIZKP) [22, 24].

2.2.1 Fiat-Shamir Heurisitc

The Fiat-Shamir heurisitc can turn an interactive zero-knowledge proof into a non-interactive one [16,24]. Therefore, the challenge of the verifier to the prover gets replaced by the result of a cryptographic hash function hashing certain previously defined parameters [24]. If only the commitment of the prover is hashed in the hashing step, *Bernhard et al. (2012)* speak about the *weak* form while the *strong* form includes the statement to be proven additionally to the commitment [16]. The weak form can lead to unsound proofs since a malicious prover can adaptively select the statements to proof (*i.e.* the malicious prover can generate proofs which do not prove the expected statement but an adaptively chosen one) [16].

Independent of what is included in the hashing process, based on this transformation, a proof can be published and verified whenever needed [24]. It still does not leak any additional information [24].

2.2.2 Schnorr Proof

A Schnorr proof is a ZKP making use of the knowledge about a discrete logarithm [64,68]. Thus it can be used to prove knowledge about a secret value a such that $A = g^a \bmod p$ (*e.g.* a private-key belonging to a public-key or knowledge about the encryption randomness) [28, 30, 64].

The process of an interactive Schnorr proof over a finite filed is based on the Σ-protocol structure known for three-move schemes and illustrated in figure 2.1. In this illustration the knowledge of a is proven to the verifier with the help of the public known parameters *p (large prime)*, *q (large prime divisor of p-1)*, *t (bit length of challenge)* and *g (generator) of the subgroup* G_q *of* $Z_p{}^*$ *of prime order q* [30]. In figures 2.1 and 2.2, the commitment known from the Σ-protocol is denoted as K, the challenge is c and the response is listed as r.

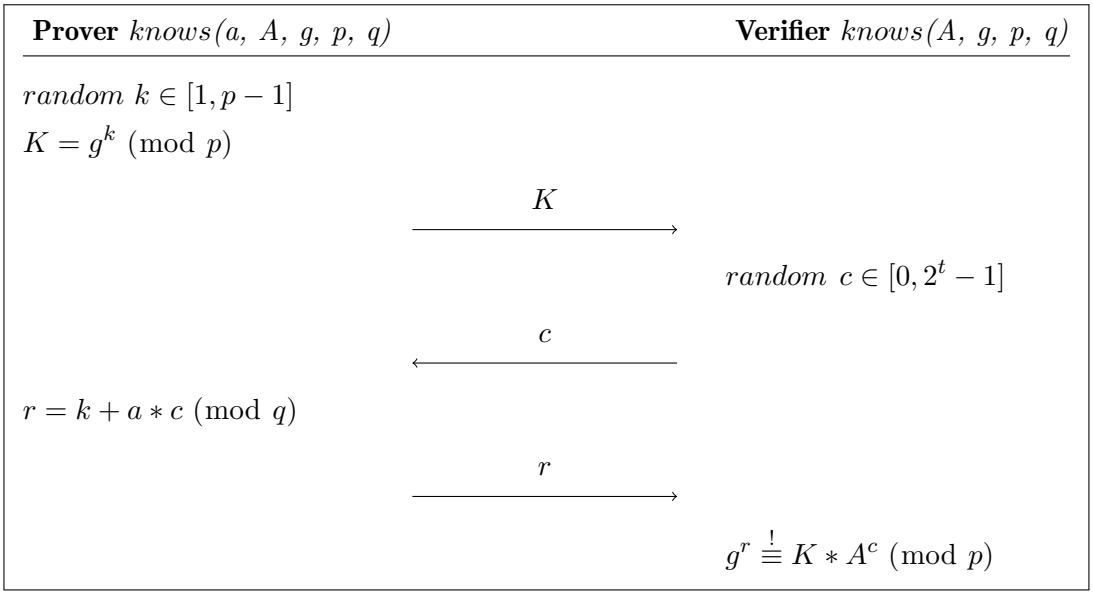

Figure 2.1: Interactive Schnorr proof to prove the knowledge of a [30,68].

To make the Schnorr proof illustrated in figure 2.1 non-interactive, the Fiat-Shamir transformation is applied and thus the challenge c is replaced with a secure cryptographic hash created with hash function H [24, 30]. This is listed in figure 2.2.

Equation 2.4 illustrates the completeness property and thus the knowledge of a [68].

$$\begin{aligned} g^r &\stackrel{!}{\equiv} K * A^c \pmod p \\ g^{k+a*c} &\stackrel{!}{\equiv} g^k * (g^a)^c \pmod p \\ g^{k+a*c} &\stackrel{!}{\equiv} g^{k+a*c} \pmod p \end{aligned} \tag{2.4}$$

2.3 Designated Verifier Proofs

Designated verifier proofs have the feature that only a verifier designated by the prover can verify a proof [29, 36]. *Jakobsson et al. (1996)* state that with three involved parties *Alice (prover)*, *Bob (designated verifier)* and *Cindy (verifier)*, instead of proofing a proof p, Alice proofs the statement "either p is true or I am Bob" [36]. This statement can be created either by the prover or the designated verifier [46]. Thus, the statement can *e.g.* be constructed as a *OR*-combination of a given proof and the knowledge about the certain private key [36, 46].

Under honest assumptions Bob is convinced by the statement and thus that p is true since he knows that Alice is not Bob (expect his private key is known by Alice) [36]. However, if Cindy receives the same statement, she will not be convinced since she cannot determine whether the prover is Bob or not [36]. If Bob forwards the poof, Cindy cannot be convinced of p since Bob can fully authenticate himself such that the second part of the statement holds [36]. If the latter part of the statement holds, the first one must not

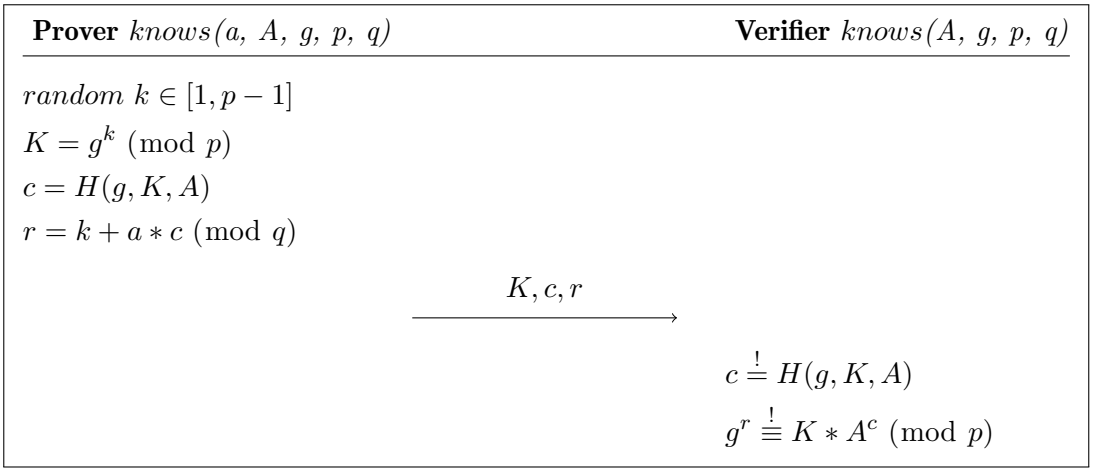

Figure 2.2: Non-Interactive Schnorr proof to prove the knowledge of a [30, 68].

hold and thus can be simulated by the designated verifier [46]. The scheme of *Jakobsson et al. (1996)* makes use of trapdoor commitment schemes (details in section 2.4) which allow Bob to simulate proofs [29, 36].

2.4 Trapdoor Commitments & Chameleon Hashes

Trapdoor commitments (also called chameleon commitments) are proof functions mainly used in interactive settings while chameleon hashes are rather used in non-interactive settings [17,28]. With the help of these schemes, it is possible to create simulated lookalike proofs of input statements such that only a designated verifier can be convinced [17, 39]. Other verifiers cannot distinguish whether the proof is generated with the help of the trapdoor or not and thus cannot be convinced if the asserted pre-image is truly proofed [10, 39].

With the knowledge of a trapdoor, chameleon hash functions can easily find collisions while without knowledge of the trapdoor they look like normal one-way hash functions with characteristics such as pre-image resistance and collision resistance [10, 28, 39, 68]. This means, that with the knowledge of the trapdoor value it is possible to generate collisions for different input values resulting in uncertainty about the value pre-image [28, 39]. Without the knowledge of the trapdoor metrics finding similar results (*i.e.* collisions) is negligible [28, 39]. *Khalili et al. (2019)* defined four characteristics of chameleon hash functions: *(i)* for each chameleon hash function a key pair consisting of hashing key and trapdoor key exists, *(ii)* anyone knowing the hashing key can generate the hash result, *(iii)* anyone knowing the trapdoor key can find collisions in the function's domain and *(iv)* the function's collision resistance remains for everyone without knowledge of the trapdoor key and its value [39]. Especially point *iii* illustrates the problem when the trapdoor information is available to multiple entities since in this case, all of them can create collisions [39].

Chapter 3

Related Work

In this chapter related work and ideas are listed and discussed. It starts with a short overview of voting forms and the history of eVoting and then the focus is laid on vote verification. In section 3.3, the in section 3.2 discussed approaches are summarized.

3.1 Different Forms of Voting & History of eVoting

According to *Krimmer et al. (2007)*, there are three main mediums of voting: *hand*, *paper* and *electronic* [45]. Furthermore, the location can be divided into two categories: *onsite*, where a voter votes at a specific place or *remote* which is (partially) location independent [45]. *Krimmer et al. (2007)* named the *onsite* location also *controlled environment* and consequently the *remote* one is also labeled *uncontrolled environment* [45]. Table 3.1 visualizes the different mediums and forms.

Table 3.1: Voting forms and mediums based on [45]

	Hand based	**Paper based**	**Electronic**
Onsite	In-person	Specific polling place or voting booth	Voting machine
Remote	Not possible	Postal voting	Device with Internet connection

Voting by hand is only possible *onsite*, and thus there is no possibility to vote remotely via this medium [45]. Furthermore, *onsite* hand based voting is only applicable for a limited number of people because otherwise it becomes cumbersome [45].

Using the medium paper as a carrier for votes allows polls in *onsite* and *remote* environments [45]. Thus voting can either be done in the *onsite* environment, where voters insert their ballots in a ballot box at a particular location, or in a *remote* environment, where especially *postal voting* is widely used [45, 72]. According to estimates, in the year 2020, about 90% of ballots counted in Switzerland were handed in via *postal voting* [61].

When voting electronically in an *onsite* environment, voters usually use a voting machine to cast their vote [45]. On the other hand, voting is categorized as *remote electronic voting*

if there is no specific voting location, and thus the completion of a vote can theoretically be done on any electronic device connected to the Internet [45, 72].

Krimmer et al. (2007) also explicitly state that there can be combinations of the previously described mediums [45]. The combination of *postal voting* with *polling places* and *voting machines* is probably the most common, and for example, used in Switzerland or the United States of America [4, 61]

Remote electronic voting has a long journey behind it. Switzerland started its project *Vote électronique* in the year 2000 [18]. In 2004, the first trials in the canton of Geneva were executed [18]. In the following years, further test runs were made, and in 2015 the first system with individual verifiability was introduced [18]. A year later, the Swiss Post system was used for the first time [18]. Back at this time, it was planned that this system would be used for federal voting in 2019 [18]. This never happened due to failings in the individual verifiability [18]. These failures were that severe that the Swiss Post decided that their system will no longer be available starting July 2019 [18].

The situation looks different in Estonia, a country where about 99% of all public services are possible to be done online since late 2020 [2]. In 2005, Estonia was the first country to hold national wide elections, and two years later, in 2007, parliamentary elections were conducted successfully [2, 70]. In the 2019 parliamentary elections, about 247'232 of all valid ballots, which is about 44%, were cast via the Internet [6]. It is also interesting that 2'107 of these votes were handed in from a foreign country what shows the location independence of remote electronic voting nicely [6].

Brazil, Canada, Germany, or the United States of America, use voting machines for electronic voting [45, 62, 72, 74]. Since this technique cannot be realized remotely, this work will not cover it.

3.2 Vote Verification

When voting on paper, the process is (typically) completely transparent and comprehensible [44]. Transparency is given by the voting medium's physical properties and how it is handled [44]. A voter using a permanent pen (which is mandatory in most cases) can be sure that the selected option cannot be changed on the paper neither in an envelop nor in a locked ballot box. However, as soon as electronic devices are included in the voting process, the transparency of applied processes shrinks tremendously since operations (especially encryption steps) executed on data cannot be observed easily and without extra effort [44]. Therefore, new mechanisms and auxiliary equipment are needed.

At the beginning of this section, the main features and properties that an electronic voting system should have are discussed in subsection 3.2.1. Then the focus is laid on verifiability and its mechanisms and schemes starting with subsection 3.2.2. In this work only approaches without specific voting hardware are considered. *Hardware-based verification* or *verifiable optical scanning*, as listed in *Guasch (2016)* are not discussed in detail. For hardware-based verification voters must have access to trusted hardware such as smart cards with embedded keyboards or hardware tokens which run the encryption process [29].

This has the advantage that neither the voting device nor the voting server has knowledge of neither the selected voting option nor the used randomness for the encryption [29]. Verifiable optical scanning is mainly used to automate the counting process of a paper based voting process [29].

3.2.1 Receipt-Freeness, Coercion-Resistance & Privacy

In Switzerland *vote secrecy* must be granted [5]. Vote secrecy defines the fact that nobody is allowed to find out who voted for what by any unlawful means [5]. *Privacy* goes even further since a voter must be able to decide freely whether they go voting or not and also for whom or what they vote [5]. According to *Küsters et al. (2017)*, privacy ensures that nobody expect the voter himself/herself, has knowledge about what he/she voted for by any means [53]. Thus, privacy is a stronger form of vote secrecy since it does not only prohibit *unlawful means* [53]. In literature, vote secrecy and privacy often are used indifferent.

- **Receipt-freeness** ensures that a voting system must not provide a valid receipt of a vote selection and that a voter cannot gain any information he/she can use in any form for vote selling [8, 15, 53].
- **Coercion-resistant** describes the state, that a voter cannot use any information to prove the voting decision to a coercer [37, 50, 53]. In the definition of *Küsters et al. (2010)* a voting protocol can become coercion-resistant in an honest environment if there is a process-step that cannot be conducted by the coercer itself (*e.g.* registration, vote in a voting booth, operations with private security parameters) [9, 50]. Furthermore, there must exist a counter-strategy which a victim can apply to achieve his/her goals without the coercer knowing it (*e.g.* voting multiple times and only count the last ballot) [9, 50]. Coercion-resistance only holds if honest voters not misbehaving on purpose are assumed, and it is a stronger form of receipt-freeness which protects voters actively against vote-buying and coercion [14, 53, 53].
- **Privacy & Vote Secrecy** ensures that no one should find out whether a voter went voting or not and how he/she voted by any means [5, 51, 53]. Hence, for an observer who may control certain steps/parties of the voting process, there is no possibility (under probabilistic polynomial-time assumptions) to distinguish how a voter voted if the voter is honest, meaning that the voter does not reveal the selected voting option on purpose [51, 53].

It is commonly believed and expected that coercion-resistance implies privacy, which is, according to *Küsters et al. (2017)*, not true [53]. The connection between them is subtle, since improving the level of privacy can lead to a lower level of coercion-resistance [53]. However, the privacy level can also be much lower than the level of coercion-resistance [53]. It becomes problematic if a (coerced) voter applying a counter-strategy can hide his/her behavior and vote selection better than with the honest voting process [53]. This occurs if the honest receipt unveils more information than necessary [53].
An example thereof would be a poll where half of the voters must reveal how they voted

and where coerced voters can lie about it [53]. This results in a low privacy level (*i.e.* $\frac{1}{2}$ of the votes are known) but a high coercion-resistance due to the counter-strategy (*i.e.* lying about the selected vote) [53]. This hides the true vote selection in a better way than revealing the vote [53]. If the counter-strategy does not outperform the honest process, a coercion-resistance protocol brings at least the same level of privacy [53].

3.2.2 Verification & Accountability

When voting with an electronic medium (as visible in table 3.1), a voter cannot observe all steps done by an electronic device since operations on data will not be detected by a voter [44]. According to *Benaloh (2017)*, voters using their own devices can be targets of coercion and malware or other risks of viruses and integrity violation [14]. Voters using devices managed by entities of their choosing and trust face fewer problems and threats regarding integrity and malware infections but still face the problem with coercion [14]. To prevent coercion, voters should use devices provided and controlled by the voting authority [14]. But even then, and especially if the voting authority is not trustworthy and fair, voters still cannot be ensured that their vote is encrypted and cast as intended [14]. Thus vote verification is an important part of electronic voting [44]. It allows voters to verify the electronic voting process and therefore ensure that a ballot is not altered by a malfunctioning voting device [44]. *Küsters et al. (2017)* list programming errors and manipulated or malware-infected voting systems, or certain parts of it, as the reason why verifiability is needed in eVoting [53].

- **Verifiability & Verification** allows detection if something went wrong during the voting process [51, 53]. Furthermore, the verification will only succeed if everything worked out correctly and the verifier is assumed to be honest and unbiased, which means that this entity will not accept a voting step if it is not conducted correctly or does not result in the theoretically expected output [51]. Hence, a verification entity accepts the process if it is conducted correctly, honestly, and trustworthy [51].
- **Accountability** is the possibility to detect malfunctioning parties and devices by a user [53]. *Küsters et al. (2017)* list accountability as a stronger form of verifiability since it allows not only the detection of an incorrect state but also what, who, or which device caused it [53]. Furthermore, eVoting systems should be designed for accountability, and not only end-to-end verifiability since the latter one is not sufficient [53].

There is a conflict between *verifiability & verification* and *privacy* (details in 3.2.1) [8]. On one side, a voter should have as much information as needed to verify that the vote was cast correctly [8]. On the other side, privacy, vote-selling, and coercion become a problem if too much information is provided and a voter can persuade another person about a personal voting selection [8].

Verification & verifiability is in literature also referenced as *voting verifiability*, *election verifiability* or *end-to-end (E2E) verifiability* [8, 41, 44]. Based on *Küsters et al. (2017)*, E2E verifiability ensures that the result of a voting round not representing the actually

cast votes is negligible [53]. The definition of *Kiayias et al. (2015)* goes into the same direction since they define E2E verifiability as the ability a voter has to verify that a vote was cast, recorded, and tallied properly in the standard model [40]. *Adida (2006)* stated that end-to-end vote verification does not verify the voting equipment but rather the voting results since it does not check the source code but the output thereof [8]. In general one can say that with vote verification a voter can verify that his/her intentions are (a) properly encrypted by the voting machine, (b) delivered to a voting system unaltered and (c) counted correctly in an evaluation phase [8, 44, 53]. Verifiability is a combination of *Cast-as-Intended*, *Recorded-as-Cast* and *Counted-as-Recorded* [28, 41].

- **Cast-as-Intended** ensures that encrypted ballot indeed contains intended voting selections, and thus a ballot is cast as the voter intended [8, 28, 29, 57]. If this property is met, a corrupt/malfunctioning voting device is unable to cast a vote containing a different voting option without being detected by the voter [29, 58].
- **Recorded-as-Cast** allows voters to verify whether their vote is recorded in the ballot box correctly or not [8, 29].
- **Counted-as-Recorded** ensures that previously received and recorded ballots are counted correctly [8]. Furthermore, it ensures that no accepted ballot is uncounted or unaccepted ones are counted [53].

Furthermore, end-to-end verifiability often gets divided into *individual verifiability* and *universal verifiability* [8, 13, 41, 44].
Individual verifiability ensures that a voter can check that the vote is counted correctly and as intended [13, 26, 44, 69]. Thus, individual verifiability is also a combination of Cast-as-Intended and Recorded-as-Cast [26, 28, 29, 60]. *Kazue et al. (1995)* defined individual verifiability as the possibility a voter has to check if the ballot reached the voting server and was counted correctly [63]. In some literature (*e.g. Kazue et al. (1995)*, *Küsters et al. (2011)* or *Smyth et al. (2015)*) individual verifiability does not cover the Cast-as-Intended property [51, 63, 69]. The disadvantage of individual verifiability is that a voter has to trust other voters that they also verified their vote since a voter cannot check the correct handling of a ballot for others [63].
Universal verifiability ensures the possibility of publicly verifying that the election outcome coincides with the published ballots [28, 41, 44]. This is basically the same as Counted-as-Recorded [28, 41, 44]. It is important that this can be performed by any voter or an interested third party at a later time [63].

Cortier et al. (2015) prove that individual and universal verifiability combined imply end-to-end verifiability under certain assumptions [21]. However, *Küsters et al. (2017)* state that the combination of individual and universal verifiability is neither sufficient nor necessary to achieve end-to-end verifiability [53]. Since these are opposite statements, the assumptions made by *Cortier et al. (2015)* are essential to look at. They assume that ballots cannot be confused what means that there is a no-clash property [21]. The no-clash property ensures that a voting machine cannot provide the same receipt of a ballot to different voters [52]. The possibility to do exactly this is called a *clash attack* [52]. *Küsters et al. (2017)* do not rely on this assumption [53]. According to them, the combination

of individual and universal verifiability is not sufficient if dishonest voters team up with malfunctioning authorities or devices and cast malformed ballots or if so-called *clash attacks* are conducted [53]. Clash attacks are possible when the no-clash property is not assumed, or some entities behave dishonestly and use the applied protocol's weaknesses. The weaknesses are described in the listing below:

- **Devices working together:** Suppose voters do not have to prove that the ballot is of correct form, meaning there is an ability to cast malformed or invalid ballots, voters can fiddle the voting result dramatically [53]. In this case, dishonest voters can *e.g.* cast ballots containing negative or multiple votes of a certain voting option [53]. That will not be detected since only they can individually verify their ballot while it is still possible to tally the ballots stored on the bulletin board in an universal, verifiable manner [53]. This means that malformed ballots counted since they can only be detected by the voters handing them in [53]. However, the overall verifiability is not valid since the malicious ballots are not detected, and thus, the voting result is not achieved correctly [53]. Overall, individual and universal verifiability combined does not correctly work if the voter and the voting device are dishonest and work together [53].
- **Clash attack:** A clash attack is an attack where malfunctioning voting devices or dishonest authorities can cast manipulated ballots since the same receipt is shown to different voters who selected the same voting options [52, 53]. Thus, malfunctioning devices can deliver the same receipt to similar voters who voted for the same options [52]. An example of a clash attack would be a voting system where the ballots are published on a bulletin board without voter names or pseudonyms attached to it [53]. If in this system malfunctioning devices are present, identical ballots can be created if n voters vote for the same options and the same randomness is used to encrypt those [53]. The voting device now only must publish one correct ballot and can manipulate n - 1 ones at the same time [53]. All voters will verify one single valid ballot (*i.e.* individual verifiability since each voter can validate "his/her" ballot) while not realizing that all of them are verifying the same one [53]. Universal verifiability is also met because the voting system does not count more ballots than allowed and it is expected that all ballots in the bulletin board are verified by the voters [53]. Thus, the vote of an honest voter was replaced undetectable [53].

Also, the E2E verifiability definitions of *Kiayias et al (2015)* and *Smyth et al. (2019)* are critical since they do not address the voter's intent (*i.e.* Cast-as-Intended is not ensured) [69]. If Cast-as-Intended is not taken into account, there is no guarantee of correct encryption [53]. Therefore, certain attack vectors are possible what results in individual verifiability, defined as in *Kazue et al. (1995)*, *Küsters et al. (2011)* or *Smyth et al. (2015)*, is not sufficient for E2E verifiability [51, 53, 63, 69].

Küsters et al. (2017) show that there are forms of Helios (details about Helios in section 3.4) where individual and universal verifiability are given but E2E verifiability is not [53]. Other voting systems (*e.g.* sElect - details in section 3.4) provide E2E verifiability but universal verifiability is not met [53]. The combination of individual and universal verifiability is not necessary to achieve E2E verifiability since such a system can provide

E2E verifiability under reasonable assumptions but without being universal verifiable [53]. *Küsters et al. (2016)* formally proved that sElect provides a prudent level of End-to-End verifiability without offering universal verifiability [49,53]. It is extraordinarily risky for an attacker to manipulate votes in sElect due to voting protocol structure where the voting device can determine which mix net server is malfunctioning [49, 53]. This can easily be reported, which reduces the servers' incentives to betray [49, 53]. However, the mix net cannot be publicly verified such that universal verifiability is not given in this case [53].

Overall, *Küsters et al. (2017)* state that E2E verifiability is typically insufficient for practical tasks when applied alone, and it would be best if eVoting systems are designed in regards to accountability [53]. Accountability has the advantage of strengthening the incentives of being honest and not misbehave for all parties involved in the voting process since it is visible which entity misbehaves or is not trustworthy [53]. The feature that makes it impossible to misbehave and deny it later increases the system's robustness [53]. Because it is known which parts are not trustworthy, they can be picked out and excluded from another protocol run [53].

3.2.3 Challenge-or-Cast Verification Scheme

Here the challenge-or-cast verification mechanism is described in detail. The most famous approach of challenge-or-cast makes use of the Benaloh-challenge [48]. The structure of this subchapter and the following three is the same. All start with the description of the Idea, which is followed by an illustration of the process. The subchapters are closed with opportunities and obstacles emerging from the discussed verification scheme.

Idea

The idea of challenge-or-cast verification is that voters either can cast their ballot or challenge the encryption mechanisms to test if the voting client is honest and trustworthy [13, 48]. It is not possible to cast a ballot and challenge the encryption of the same ballot [13, 28, 31, 48]. This characteristic leads to the fact that all ballots counted in the tally phase are never audited [13, 20]. If a cast ballot is also audited, *receipt-freeness* and *coercion-resistance* will not hold [48]. To ensure receipt-freeness and coercion-resistance, a voter needs to restart from the beginning after he/she challenged the encryption, and the voting device must not recognize a restart after challenging a ballot [20, 28, 48]. There is also the possibility of re-encrypting the vote with a newly generated randomness instead of starting from scratch again [13, 28]. Re-encrypting ballots after verification is less preferred since it does not compromise vote secrecy [48]. The process of gaining trust now lies in the number of vote-iterations done by a voter [14, 31, 38, 48]. As previously described, already cast ballots cannot be audited. Trust must be established by revealing correct encryption of ballots encrypted and processes before the final vote iteration, which results in casting the ballot [14, 48].

In this approach, the critical point is that the voting client and the voting system must not know if a voter is challenging the system or casting the ballot when creating the

commitment [14,31,48]. A voting device will always generate an immutable *commitment* of the encrypted ballot before knowing whether this ballot will be cast or challenged [14,31,48]. This commitment includes the encrypted ballot, which consists of the selected option in an encrypted form [14,31]. So the commitment contains the selected options but does not reveal their values resulting in no knowledge about the encrypted voting option [14,31,48]. Suppose the voting device does not include the selected voting option in the commitment. The commitment cannot be changed anymore as soon as the voting device knows whether it will be challenged or not [14,31,48]. Thus a malicious voting system will be caught if an unselected option is included [14,31,48].

If the voting device knows about the voter's challenge-or-cast decision a priori, it can generate a misleading commitment leading to cheating options [14,14]. In theory, a voter can store the commitment somehow (*e.g.* write it down on paper, copy it to a separate window, scan it with a verifier application) without letting the voting device know where the commitment is stored [14,31,48]. Since this commitment is immutable and potentially stored somewhere else, the voting device cannot change it anymore when receiving the voter's intention of challenging or casting the ballot [14,31,48]. Thus, if a voter selects *challenge*, a commitment with a not-intended voting option will be challenged, revealing that the voting device cheated and is not trustworthy [14,31,48]. This makes it very risky to manipulate a vote and, therefore, become busted since an unbiased voter will challenge the encryption in 50% of the vote-iterations [14,48].

As described previously, a voter must have the possibility to interact with the commitment in a way that allows storing it independently of the voting device or at least the voting application. In *Kulyk et al. (2019)*, and in *Patrick Hayes*'s implementation, two systems, voting client and verifier, are used [31,48]. Since there are two different devices, they are physically air-gapped and if implemented properly the devices do not know which other device is used [48]. As a result of this, the verifier-system does not need an active Internet connection at the time of voting [48]. Nevertheless, if only two devices are used, it needs to be assumed that either the voting client or the verifier is trustworthy by nature [48]. If they collaborate in some way, they can manipulate the factors in the same manner what will not be visible to a voter [48].

Challenge-or-cast verification can ensure that a voting client is trustworthy and does not manipulate the selected options [48]. However, above described version does not ensure that the communication between the voting client and the voting system is honest [48].

Process

In figure 3.1, the process of the challenge-or-cast approach is illustrated. The listing below describes the steps in more detail (based on [31] and [48]):

- After a voter successfully logged in, he/she selects the voting options and then initiates the encryption of the chosen voting option. Once the encrypted ballot is created, the voting device constructs the commitment. The commitment must include the encrypted ballot, which contains the encrypted voting selections.

- Once the commitment is created, it will be displayed to the voter. He/she then should store the commitment somewhere independent and out-of-reach of the voting device. Once this is done, he/she can either select *Challenge* (to challenge the encryption and thus the trustworthiness of the device) or *Cast* (to cast the ballot and finalize voting).

- If *Cast* is selected, the encrypted ballot will be sent to the voting system and counted in the tally phase. The voter finished voting.

- If *Challenge* is selected, the voting device must display the used randomness data in a form a voter can use it for further calculations.

- Once the voter received all the randomness data used in the encryption, he/she encrypts the vote on a verification device. This will result in another encrypted ballot, of which again a commitment is created. In theory the two commitments should be exactly the same since the same intended voting options are encrypted, and also the same randomness data is used for the same encryption algorithms.

- The second commitment can be compared to the first one. If they are similar, the voting device is honest since it encrypted the selected voting options and used them in the commitment. The voter gained trust in the voting device and must restart the process to cast a vote. If the commitment is not similar, the voting device or the verifier device is not trustworthy, and a voter should not vote with this device combination. The commitment can be different if the voting device or the verifier application encrypted not the selected voting options, used wrong randomness data or an error occurred.

Opportunities & Obstacles

This verification approach's most significant opportunity is that a voter can challenge the system as often as he/she wants and until enough trust is gained [48]. Hence, the voters can decide by themselves if they trust the used devices or if they want to challenge it once more [48].

One problem the challenge-or-cast approach suffers is that the challenge of a ballot takes extra effort by the voter because, after the challenging process, the voting process needs to be started again [48, 73]. This becomes problematic since the possibility to detect a untrustworthy voting devices is only given when voters challenge the voting device, meaning, willing to take extra effort [48]. If a voter does not challenge the system, he/she will not detect fraud [48].

People may take the extra effort if they do not know a voting system or if they are suspicious about the devices they use [48]. But after multiple successful verifications, the interest in rechallenging the same setup, with all extra effort, may decrease [48]. This potentially results in many people challenging the system in the beginning, but fewer challenges after some votes are already conducted via the system [48]. Since trust is only generated if voters challenge the system this can become problematic [48]. Making the

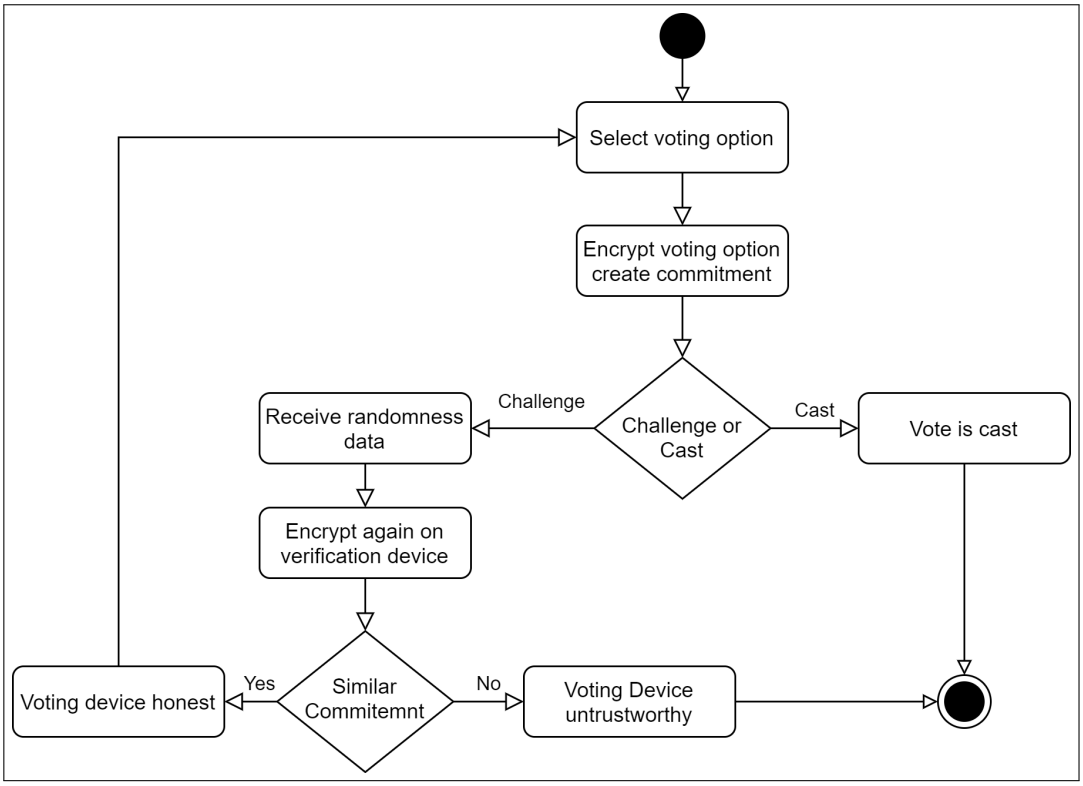

Figure 3.1: The challenge-or-cast process (based on [31])

verification mandatory for every new vote would be problematic since the voting device then knows whether a voter has to challenge the ballot or not [48].

Furthermore, proper vote secrecy depends on the implementation since re-encrypting a challenged vote violates vote secrecy [48]. Thus realizations of this approach should, after the challenge process, always restart the complete voting process [48, 73].

Depending on the implementation two devices are needed for the challenge-or-cast approach [73]. The two devices are preferably in different trust domains, meaning at least one device is trusted [48]. If the voting device and the verification application/device are both untrustworthy, the devices can fake correct encryption and make fraud detection for a user impossible [48, 73]. Thus, one of the devices must be trustworthy and encrypt the selected option honestly or more than two devices must be used [48, 73].

Lastly, it is counter-intuitive to discard a challenged and potentially successful encrypted vote while casting a ballot not challenged and not verified [48]. Gaining trust via previously challenged ballots can confuse voters, especially if there is not enough context and explanation provided [48].

3.2.4 Challenge-and-Cast Verification Scheme

In this subchapter, the challenge-and-cast approach developed by *Sandra Guasch* and *Paz Morillo* in 2017 is described in-depth following the structure introduced in the last subsection.

Idea

Guasch et al. (2017) have proposed a verification mechanism that they named challenge-*and*-cast verification with the goal to avoid some drawbacks of the challenge-*or*-cast verification approach [28,29]. This approach's critical point is that a voter always verifies the encryption and simultaneously generates various simulated receipts [29]. Only the voter can be convinced about the correctness of this proof [29]. People different than the voter cannot distinguish whether the correct proof or a simulated one is shown to them [29]. This verification approach circumvents the fact that a challenged vote should be discarded to ensure vote secrecy and coercion resistance when applying the challenge-or-cast verification [28, 29]. Thus this approach should be more intuitive than the challenge-or-cast version since the votes challenged are also the cast ones [28, 29, 48]. Their approach provides Cast-as-Intended verifiability, and voters can challenge the same encrypted ballots as they will cast later and ensures that voters will not receive a receipt that can be used to sell their vote [28, 29].

The scheme described by *Guasch et al. (2017)* makes use of designated verifier proofs (details in section 2.3) [28, 29]. Designated verifier proofs are proofs in which only the designated, chosen verifier can be convinced [36]. With the challenge-and-cast technique, the designated verifier can simulate proofs about arbitrary statements (*i.e.* unselected voting options) for other verifiers while the designated verifier itself can be convinced of a particular statement (*i.e.* the selected voting option) [28, 29, 36]. Since the voter is the designated verifier while the voting client is the prover and coercers/buyers are other verifiers, only the voter can be convinced that the voting client knows the encryption randomness (since the prover is proving this) [10, 28, 29]. At the same time, the voter can create faked proofs with the help of the voting device and a private trapdoor key such that the coercers/buyers cannot be convinced about a proofs content [10, 28, 29]. Because buyers and coercers cannot be convinced if a proof is counterfeit or not, receipt-freeness and coercion-resistance are achieved [10, 28, 29]. Buyers and coercers can only ensure that one possible voting option is included, but they cannot ensure that the proof consists of the voting option the voter claims [10].

The designated verifier proofs used in this approach make use of trapdoor commitments (also known as chameleon hashes) [28]. A detailed description of the trapdoor commitments can be found in section 2.4. In this setting, it is crucial that the trapdoor information (*i.e.* knowledge about the trapdoor key) is only available and known by the voter (*i.e.* the designated verifier) [28]. He/She can use this information to generate the simulated proofs proving unselected voting options [28].

Process

The process of the challenge-and-cast approach is illustrated in figure 3.2. Furthermore, it is based on *Guasch (2016)* and *Guasch et al. (2017)* [28, 29].

- The challenge-and-cast verification starts similarly to the challenge-or-cast approach. First, a voter authenticates himself/herself successfully, then he/she selects a voting

option on the voting client which will become encrypted, including some randomness value.

- After this, the encrypted ballot and a generated non-interactive zero-knowledge proof of knowledge (NIZKPK) of the used randomness are shown to the voter. In the challenge-or-cast approach, plain values of the used randomness are shown such that a verifier application can recalculate the encryption [48].

- The proof then needs to be verified by the designated verifier (*i.e.* the voter), which only will be successful if the voting device (*i.e.* the prover) is honest, meaning the voting client indeed encrypts the selected value. This can be ensured since the probability of successfully verifying a manipulated proof is negligible for a dishonest device. If a voter disagrees with the proof since it is not valid, an untrustworthy voting device is detected, and it should not be used to vote.

- After the voter agrees on the proof, the same ballot is cast and in a further step published on the public bulletin board. Simultaneously, a voter needs to insert the trapdoor key such that the voting device can create simulated NIZKPs for unselected voting options. These simulated proofs will always succeed in the verification phase.

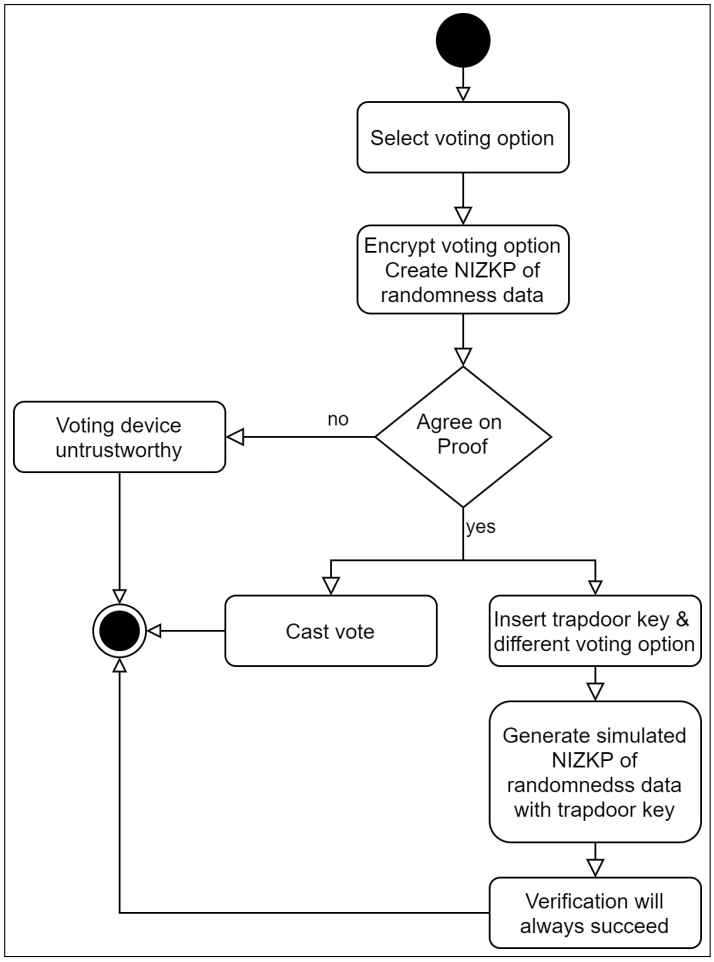

Figure 3.2: The challenge-and-cast process (based on [28, 29])

Using NIZKPKs for the randomness instead of the plain value, and therefore applying the advantage of the possibility to simulate NIZKPKs brings protection against vote selling and coercion [28]. This advantage allows voters to create faked real-looking proofs, representing any voting option [28, 29]. These simulated proofs cannot be used for vote-selling since the buyer/coercer cannot distinguish between a genuine and counterfeit proof [28, 29].

In the paper of *Guasch et al. (2017)*, this process makes use of designated verifier proofs with trapdoor hashes in combination with ElGamal encryption and non-interactive zero-knowledge proofs (cryptographic primitives are described in section 2) [28]. Their calculations are divided into three main stages: *NIZKProve*, *NIZKVerify* and *NIZKSimulate* [28]. Additionally *RSA Full Domain Hash* algorithms are used for the signature scheme, *Chaum-Pedersen* proofs of correct decryption are applied, and a *common reference string* was used in the previously described stages [28].

The theoretical formula they used for collision finding is listed in equation 3.1, where m is the indeed selected message, and m' is the one for which the proof will be simulated [28]. While the trapdoor key used is denoted as tk, the honest proof's randomness is listed as r_{ch} [28].

$$r'_{ch} = (m - m') * tk^{-1} + r_{ch} \tag{3.1}$$

Equation 3.1 results in the randomness r'_{ch} which can be used in the hashing process (equation 3.2) together with the message m' to create the hash value c'_{ch} (c_{ch} analogous) [28]. The value of c_{ch} is used in the proof for message m while c'_{ch} is used in the proof for message m' [28]. This allows simulated proofs with different input messages if the value of c_{ch} and c'_{ch} are the same [28]. In equation 3.2, g is a generator and h an ElGamal public key [28]. All values are in previously defined message and randomness spaces [28].

$$c_{ch} = g^m * h^{r_{ch}} \stackrel{!}{=} g^{m'} * h^{r'_{ch}} = c'_{ch} \tag{3.2}$$

Opportunities & Obstacles

In this approach, verifying the ballot and challenging the voting device's trustworthiness is mandatory since, otherwise, creating simulated proofs is impossible [29]. Furthermore, it does not depend on gaining trust in the voting setup via previously challenged ballots [29]. Trust is generated by challenging the ballots indeed handed in [29]. This is more intuitive regarding the flow of how voting is done and brings less extra effort compared to the challenge-or-cast approach since there is no fresh start needed after challenging [29, 48].

It is essential that simulated and real NIZKPs look exactly the same since otherwise, coercers or buyers can ascertain whether a simulated or valid proof is shown to them [28, 29]. If distinguishing the proofs would be possible, the system would not be coercion-resistant anymore [28, 29]. The same problem occurs if a voter does not know or does not have access to his/her trapdoor key [28, 29]. Suppose a voter does not know the trapdoor

key or does not have access to it, he/she is not able to produce simulated NIKZPs resulting in the problem, that he/she receives a meaningful receipt and the system is not receipt-free and coercion-resistant anymore [28, 29]. Overall this approach is not receipt-free in the sense of there is no receipt, but since all the receipts cannot be distinguished and only the designated verifier can be sure which receipt is the correct one, a single receipt is vacuous [28, 29].

Besides the fact that the trapdoor key must only be known by the voter, it is also important that the voting device has no knowledge about it until the immutable honest proof is generated [28]. This is crucial [28]. Otherwise, the voting device can generate a dishonest proof before the honest one what results in not even the designated verifier being able to distinguish between an honest and simulated proof [28].

In this approach, the administrative effort is slightly higher than with the challenge-or-cast one since the trapdoor key must be handed over to the voter before the voting starts [28, 29]. However, in this approach only one device is needed [29]. Therefore, this device better is a trusted one [29].

3.2.5 Partial-Audit Verification Scheme

In this subsection, the partial-audit verification mechanism brought up by *Cortier et al. (2019)* is described based on the three sections *idea*, *process*, and *opportunities & obstacles* [20].

Idea

This approach's main idea is that each ballot is partially audited and verified before being counted in the tally phase of a voting scheme [20]. This allows voters to cast and verify the same ballot [20]. Therefore, *Cortier et al. (2019)* make use of a shift and a masked vote which both are included in a ballot next to the encrypted selected voting option [20]. The shift is a random value in the range from 0 to the number of possible voting options [20]. The masked vote is the subtraction of the shift from the selected voting option and can therefore be seen as one-time-padded encryption of the chosen voting option [20]. Thus the knowledge of either of these two values does not leak any information about the indeed voting option selected [20].

The scheme uses zero-knowledge proofs to ensure well-formed ballots and, even more importantly, that the addition of the shift and the masked vote result in the selected voting option [20]. Additionally, a voter can check if the decrypted value of the shift or the masked vote corresponds to the value he/she remembers from the encryption process [20]. If these checks succeed, there is no evidence of a malfunctioning voting device [20].

Process

The process of the partial-audit approach is illustrated in figure 3.3. It is based on the work of *Cortier et al. (2019)* [20].

- In the beginning, an authorized voter must select a voting option on the voting device.

- In the next step, the voting device creates the ballot. To do so, the voting device encrypts the previously selected voting option. Furthermore, it encrypts the shift and the masked vote. Lastly, the voting device creates a zero-knowledge proof to prove that the encryption is well-formed and that the sum of the shift and the masked vote corresponds to the selected voting option.

- A voter then checks that the sum of the shift and masked vote truly corresponds to the chosen option. If they match, the voter stores the ballot, shift, and masked vote.

- In the next step, the voter either freely selects 0 or 1 and inserts this into the voting device. The device then recovers the random value used to encrypt the shift or the masked vote. If a voter selected 0, the shift would be decrypted with the help of the random values, while the masked vote will be decrypted if a voter chose 1.

- A voter then checks whether the published value of the shift or the masked vote corresponds to the value he/she remembers from the previous step. If the value is similar, the voting device encrypted its input correctly and can be considered as honest. Otherwise, the voting device probably encrypted and included an unselected value in the ballot.

Opportunities & Obstacles

The biggest opportunity of this approach is that all votes counted in the tally phase are partially audited [20]. Hence, trust is not generated based on experience or auditing mock ballots but on actual auditing of indeed cast ballots [20].

However, this also nicely shows a drawback because the ballots are only partially audited and not thoroughly since only the shift or the masked vote are checked but never the actually selected voting option [20]. Therefore, it is not the encryption of the selected voting option which is verified directly. Hence, a malfunctioning voting device may simply fake the zero-knowledge proof and select the values in a way such that the voter's checks will always succeed. This would result in the problem that a voter audited the ballot and despite successfully validating the checks, the counted vote is not the indeed selected one.

This process is, compared to the challenge-or-cast approach, not counter-intuitive [20]. Furthermore, only one device is needed to cast a ballot [20]. Hence, it may be easier and also faster to use this approach in certain use-cases since no extra effort is needed [20]. However, the voters must perform some modular addition when checking if the shift and the masked vote correspond to the selected voting option [20]. If the broad public is able to execute this without explanation must be taken into account [20].

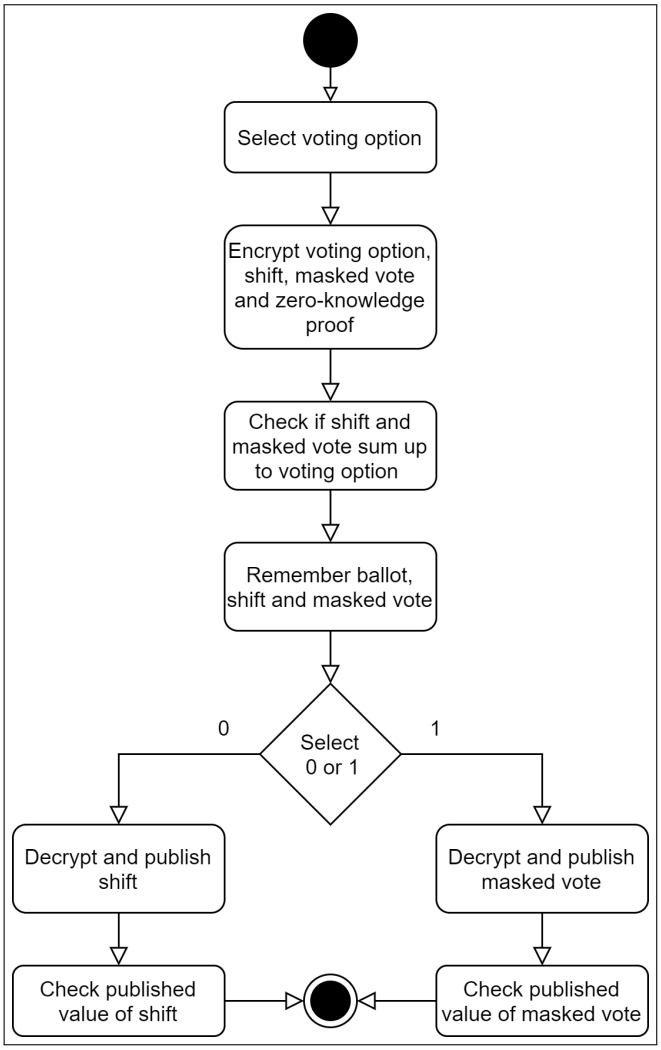

Figure 3.3: The partial-audit process (based on [20])

3.2.6 Code-Based Verification Scheme

In this subsection, the code-based verification mechanism is described in detail. Like the previous subchapters, the idea is stated initially, then the process is illustrated, and the opportunities and obstacles are listed.

Idea

The idea of code-based verification is that a voter can verify the ballot with the help of previously defined codes which can be compared to codes computed by the voting system during the voting process [29, 48, 73]. The codes received on a personalized code sheet represent all possible voting options, and differ from voter to voter, meaning every voter has its own codes for all possible voting options [29, 48, 73]. During voting, a voter can compare the codes displayed by the voting device with the ones listed on the code sheet [29, 48, 73]. If the codes match for the selected option, the voting device

is trustworthy, and the chosen option was encrypted correctly [29, 48, 73]. Thus, not matching codes are an indicator of untrustworthy devices [29, 48, 73].

Therefore, the code-based verification approach's essential point is that every eligible voter is provided with an individualized code sheet before the election starts [28, 29, 48]. The voting device does not know about the codes a voter receives but has to calculate them once the encryption is finished [48, 73]. Therefore, the codes need to be calculated with a deterministic function since they need to match the previously calculated ones [29]. Optimally the receiving of the code sheet is done via a secondary channel [28].

Process

The process illustrated in figure 3.4 and explained in the listing below is based on the work of *Kulyk et al. (2019)* and *Scytl (2017)* [48, 73].

- Before the voting starts, each eligible voter receives a personalized code sheet that contains all relevant codes for a particular vote or election. The codes must be received before the voting starts and generated in the configuration phase of a vote.
- After voting started, an authorized voter selects the intended voting options on the voting device. The voting device then encrypts the vote and displays the check code.
- If the check code is correct, meaning if the on the voting device displayed code matches the code belonging to the selected option listed on the code sheet, the device encrypted the selected voting option honestly. If the codes do not coincide, the voting device is not trustworthy, and it should not be used for voting.
- If the correct code was displayed in the previous step, a voter inserts the confirmation code belonging to the selected option. The voting device checks this code against the known confirmation code, and if these are concurring, it casts the ballot.
- Once the ballot is received at the voting system's servers (*i.e.* the servers or blockchain where the ballot box is located) and if it passes the previous checks, the voter's finalization code belonging to the selected voting option is generated. Then the finalization code is sent back and again displayed to the voter.
- If the displayed finalization code matches the corresponding code on the code sheet, the ballot was received and stored correctly in the ballot box. Thus, voting is finalized. If the codes do not agree, evidence for untrustworthy devices is found. Hence, the voter should not vote with this device combination.

The process above is specialized for countries where only casting a vote once is allowed (*e.g.* France or Switzerland) since it includes a confirmation phase (*i.e.* inserting the confirmation code and receiving the finalization code) [73]. This is needed to protect voters against malfunctioning voting devices [73]. Countries allowing voters to cast multiple ballots and invalidating previously handed-in ones do not need the confirmation phase since voters can cast another ballot if the verification codes are not matching [73].

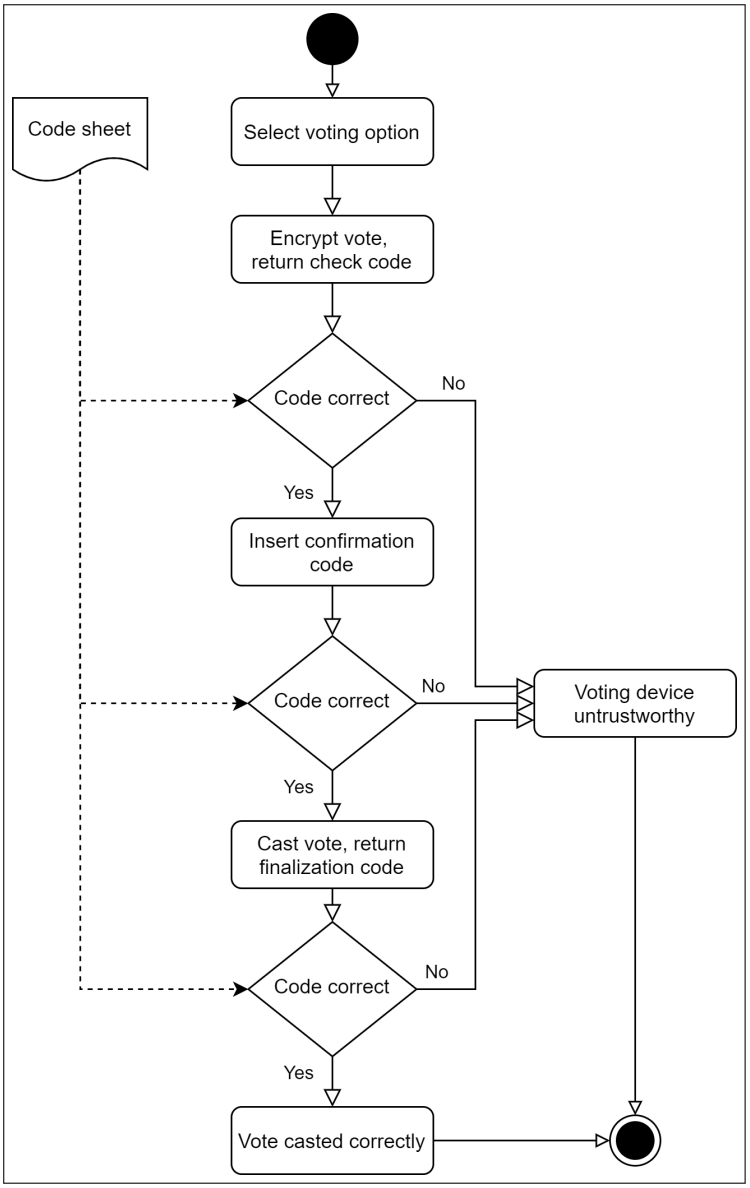

Figure 3.4: The code-based process (based on [48, 73])

Opportunities & Obstacles

The advantage of this approach is that the connection between the voting device and the system running the ballot box is checked with the help of the confirmation and finalization codes [48, 73]. Code-based verification can ensure that the voting client encrypts and forwards the ballot as intended and also that the voting system receives it unchanged [48]. Thus, there is the possibility to combine Cast-as-Intended with Recorded-as-Cast since the voting server returns the finalization code belonging to the received voting option [28, 48, 73]. Furthermore, only ballots with a valid confirmation code are counted [73].

The second advantage is that the challenge of the encryption is mandatory since otherwise, a voter cannot insert the correct confirmation code [48,73]. Thus the potential problem of people getting used to eVoting and therefore challenging the encryption and system less

often is mitigated by design [48].

A drawback of this approach is that the codes should be received on a different channel (*e.g.* via letter or e-mail) than the voting takes place [48]. In Switzerland, this was solved in a way that voters receive their codes via snail mail [3]. Receiving the codes physically on paper is kind of a contradiction to remote electronic voting, especially in a country like Switzerland where postal voting is widely deployed, accepted and used [3,61]. Furthermore, it can also happen that there is no second channel available [28].

The codes must be created before the voting starts [48]. Furthermore, they must be created by a trusted entity, which likely is not in the voter's trust domain, which can become problematic [48]. Basically, a voter must trust the received codes, and hence the entity creating these codes, without having the ability to challenge the trustworthiness of this entity [48].

Furthermore, if the check code and the finalization code are individualized for every voting option, meaning that one of these codes refers to exactly one voting option, the receipt-freeness property is not completely met [29]. This because in combination with the code sheet, a voter can generate a receipt based on the codes he/she receives from the voting system [29].

As conducted in the study of *Kulyk et al. (2019)* and *Guasch (2016)*, one issue was the characteristics of the codes [29, 48]. On one side, certain characteristics (*i.e.* length, complexity, character range) of the codes are crucial regarding security specifications [29, 48]. Still, on the other side, the codes need to be compared easily and inserted effortlessly such that everybody can use the system without difficulty [29, 48].

Lastly, there is more administrative effort than with the challenge-or-cast approach since the second channel, meaning how to deliver the codes, needs to be maintained as well [48].

3.3 Comparison of Verification Mechanisms

In table 3.2, an overview of the *challenge-or-cast*, *challenge-and-cast*, *partial-audit* and *code-based* verification mechanisms is given. The details, process illustrations, and discussions of the mechanisms are handled in sections 3.2.3, 3.2.4, 3.2.5 and 3.2.6. The table is heavily related to these chapters and basically, summarizes them. Hence, the sources of the ideas listed in table 3.2 can be found in the chapter belonging to the examined verification mechanism.

One potential general problem *Kulyk et al. (2019)* list is that the voters gain trust in a system as they become more experienced and thus may not take extra effort to verify a ballot despite it belongs to a new poll [48]. This means that voters gained trust in a system over several elections and not in challenging the current voting setup resulting in potentially higher success rates for malicious devices which cast counterfeit ballots [48]. All verification mechanisms which do not implement mandatory verification may suffer from this issue. However, *Kulyk et al. (2019)* stated that this would be an interesting point for further investigation, and at the time of writing, no result to this question was found [48].

3.4 eVoting Systems & their Verification Mechanisms

As described in chapter 3.1, there are currently some remote electronic voting systems in use or at least in development. In this section, voting systems applying a verification mechanism are shortly discussed.

The Estonian Internet Voting System allows users to verify their votes using a smartphone app [70]. To verify the vote, a voter scans a displayed QR code with a smartphone [70]. The smartphone app sends a specific part of the contained data to the election server and receives the encrypted ballot [70]. The app then encrypts votes for all possible options using the encryption data and compares it to the received one [70]. If there is a match between a simulated and the inherent ballot, the app displays the correct voting option, which a voter can check against his/her voting intention [70]. Furthermore, this system allows voters to vote multiple times to prevent coercion [70]. Allowing this makes it impossible for a coercer to distinguish if a voter indeed voted the last time or if he/she will vote later again [70]. This prevention mechanism is extended such that a voter can even override online submitted votes by voting in person on the voting day [70]. However, if the verified vote is the last one handed in, it will be counted, although the randomness data used for the encryption is revealed previously [70].

In Switzerland, the Swiss Post system developed by Scytl makes use of code-based verification of votes [73]. In Switzerland, the confirmation phase, in which a voter confirms receiving correct return codes (in detail in section 3.2.6), is used because each voter can only cast one ballot per vote [73].

Helios is a web-based electronic voting system which was mainly developed by Ben Adida in 2008 [54]. It is open-audit and uses ElGamal encryption and re-encryption as well as mixnet [54]. Helios makes use of a challenge-or-cast approach, but with weaker coercion-resistance, which is applicable since it is not planned to use Helios in elections with high coercion risks [28, 38, 73]. As stated in *Karayumak et al. (2011)*, in the version of Helios they looked at, a user has to copy the mathematical proofs and used data to a second window on the same machine [38]. This means that the verifier and voting device is not air-gapped in this vote setting. Furthermore, the partial-audit method was developed as an extension to the existing Helios system [20].

sElect is a remote electronic voting system designed for low-risk elections with lightweight and simple structures including basic cryptographic primitives and voting processes [49]. Therefore, it is not constructed to protect voters from coercion or protect them from sophisticated attacks against their private voting devices, but it is rather designed to fight against malfunctioning or manipulated voting servers, programming errors, or untrustworthy authorities [49]. Furthermore, it makes use of return codes and Chaumian mix nets [49]. However, since these mix nets are not universally verifiable, due to the lack of zero-knowledge proofs of correct shuffling, *sElect* is only individually verifiable [53]. Since the voter must check if a personalized nonce is displayed next to the selected voting option once the servers are finished with the tallying phase, it is impossible to check this for other voters [53]. The voting devices are able to detect which mix servers are dishonest, and thus, accountability is met [53].

Table 3.2: Comparison of the previously described vote verification mechanisms

	Challenge-or-cast	**Challenge-and-cast**	**Partial-audit**	**Code-based**
Idea	Voter can challenge or cast the ballot with the help of a verifier application	Voter must challenge and cast the ballot and generate simulated proofs to achieve coercion-resistance	Voter verifies certain values (shift or masked vote), but not the vote directly	Voter must challenge the ballots with the help of return codes
Process	After encryption, a voter can either challenge or cast the ballot and thus verify the voting device	After encryption, a voter must insert the trapdoor key such that coercion resistant simulated proofs can be generated	After encryption, a voter verifies the shift or masked vote to reveal the trustworthiness of the voting device	After encryption, a voter must compare and insert certain predefined codes to continue and cast the vote
Needs	Verifier application	Trapdoor key	Basic knowledge in modular arithmetic	Individualized code sheet
Key Point	Voting does not know whether it will be challenged or not	Voting device can create simulated proofs once it knows the trapdoor key	The voter can verify parts of the ballot which do not reveal the selected voting option	Codes must be created and distributed before the voting starts
Advantages	Every voter can create an own level of trust	Challenging and casting of the same ballot	Partial verification of the counted ballot	Verification of connection between voting device and server included
Disadvantages	Verification is optional, counter-intuitive, need for verifier application	Key management for trapdoor key	Only partial verification, selected voting option not verified directly	Need for second channel and management thereof, management of code creation & distribution
Potential problems	Experienced users may not challenge the voting setup every time	Voting devices knowing the trapdoor key before generating the honest proof can work maliciously	Modular arithmetic needs explanation	Second channel for the distribution of the codes may not be existent or appropriate

Chapter 4

Design

In this chapter, design decisions and the applied architecture of the system are discussed in detail. Furthermore, the decisions made about the verification method as well as how to exchange data are founded. Theoretical models of the verifier and the voter front end are given, and their interaction is explained on a high level. In the last section, trust boundaries present in Provotum are demonstrated.

4.1 The Decision about the Verification Scheme

The decision to apply the challenge-or-cast approach is founded on the following points. During the decision process, the applicability in the Provotum scenario was also taken into account.

- As visible in subsection 3.3, code-based verification has the significant advantage that the connection between the voting device and the back end (*e.g.* voting server, distributed ledger, blockchain) is verified. However, the disadvantages of it, namely the management and distribution of the codes as well as the maintenance of the second channel, are tremendous, especially in regard to the Provotum environment. As in Provotum, at the time of writing, no proper access- and identity-management is established, the creation and distribution of individualized codes would be cumbersome. Simulating all of this was not an option since it would be too far from reality and thus not bring an actual added value in neither the research process nor a potential real live application. Furthermore, a voter must blindly trust the entity generating the codes for the code sheet, what can be problematic.

- The challenge-and-cast approach looked quite promising initially, especially since the handling is not counter-intuitive and easier usage is expected. However, especially the handling (*i.e.* creation and storing) of the trapdoor key is not trivial and not applicable to the current Provotum scenario. Another point against challenge-and-cast is that the paper of *Guasch (2017)* is somewhat inconsistent regarding notation. Examples thereof are in section *Concrete instantiation* where they use

the tuple *(a_1, a_2)* as the commitment and hashing a without defining it before, inconvenient naming for ElGamal cipher text (c_1, c_2) and hash value c_{ch}, missing modular operations in all formulas or mathematically unclear formulations of the common reference string *crs* and the trapdoor key *tk* [28].

- The partial-audit approach described in section 3.2.5 has advantages like more intuitive handling compared to the challenge-or-cast version or verification of the actual cast ballot. However, the disadvantages that the ballot is only verified partially and the need of modular arithmetic calculations conducted by the voter lead to the decision against this approach.

- Challenge-or-cast has drawbacks in regards to the handling (*i.e.* counter-intuitive and optional verification) but also advantages since it can be established without any management of keys or access codes needed. Due to the advantages, challenge-or-cast fits into the Provotum environment. Furthermore, it can be implemented in an air-gapped manner such that the voting device and verifier are independent from each other.

4.2 Provotum's Overall Design

Provotum is a Remote Electronic Voting system based on a distributed ledger using a proof-of-authority method to ensure integrity and immutability of the inserted data [42]. This results in a public permissioned distributed ledger in which only authorized entities can validate blocks, and at the same time, every participant can verify all blocks in the ledger [59]. Provotum, as in version 2.0, used Smart Contracts, Distributed Key Generation (DKG), Homomorphic Encryption and Cooperative Decryption [42]. Furthermore, the ledger creates an immutable audit record and voter-side encryption of the vote enables ballot secrecy [42].

With Provotum 3.0, which is at the time of writing under development, the research focus is laid on receipt-freeness as well as vote verification. Furthermore, Provotum 3.0 makes use of a Substrate distributed ledger, which works as a public bulletin board. This distributed ledger must not make use of smart contracts any more since the protocol is directly implemented in Substrate's runtime.

Provotum's stakeholders, based on the thesis of *Hofmann (2020)*, are listed below [34]:

- **Sealer:** The sealers are distributed location-independent entities running nodes responsible for the validation of blocks of the distributed ledger. Additionally, the sealers are involved in the DKG process since their public-key shares are used for the production of the elections public-key. This key will then be used for the encryption of the votes. Lastly, the sealers will run the tallying together since decrypting the votes is only possible collaboratively.

- **Voting Authority:** The voting authority is the administration entity of the votes since it starts, conducts, and closes the voting process.

- **Voter:** A voter is an eligible citizen participating in the voting.
- **Identity Provider:** The identity provider is a trusted third party allowed to authorize voters to participate in the voting.
- **Public Bulletin Board:** The public bulletin board (PBB) is the Substrate proof-of-authority distributed ledger with the Provotum protocol directly implemented into its runtime. It shares an API that allows interaction with the PBB that transactions containing encrypted ballots can be submitted to the distributed ledger. Furthermore, the API also provides *read* possibilities.
- **Community:** The community is a combination of all entities interested.
- **Randomizer:** The randomizer is an entity implemented to blind voter's ballots to achieve receipt-freeness. Due to the blinding factors that are unknown by the voter, an honest voter cannot reproduce the same ballot such that the producing of receipts is impossible.
- **Verifier:** The verifier is the new part of the system allowing voters to verify their voting setup in terms of trustworthiness. Hence, they can determine if a voting device is trustworthy and not malware-infected. This is done with the challenge-or-cast approach.

4.3 Verifier

In this section, the design for the verifier application is described based on the process illustrated in subsection 3.2.3. First, the theoretical needs and ideas are presented, and then a more technical view is given on what truly is needed.

Since the challenge-or-cast approach's key point is that the voting device must never know if it will be challenged or not before generating the commitment, an air-gapped solution makes sense [48]. In this scenario, air-gapped means that the actual voting and the verification are not done on the same device. Furthermore, the voting device should not know which device is used as a verifier so that they cannot communicate before the voting starts. Hence, it would be best if the verifier device can work without an Internet connection such that a voter can decide by himself/herself whether he/she wants to use the device offline. A smartphone application or a Progressive Web App (PWA) can be used to make this work.

PWAs are basically enriched web applications that take advantages of new features and technologies into account [65]. They combine multiple advantages of native apps with the ones from websites as they adapt to the given device or browser such that they are almost completely platform-independent [47, 65]. This can be achieved due to the *service-worker* possibility, which allows native-app-like functionalities and can work as a proxy [65]. Progressive Web Apps have multiple further advantages such as possible installability, offline usage, search engine optimizations, reduced costs, higher speed, smaller size, and no need for a distribution channel such as Apples App Store or Googles Play Store since it

can be installed directly through the device's browser [47, 65]. Due to this advantages, in 2017, Gartner listed that in 2020 about half of all native apps were replaced by PWAs [47].

Thus, the decision made was to create a PWA because of the offline usage possibility, platform independence, and easy distribution. Not using an electronic device at all would be very unhandy or even impossible since all the encryption calculations must be done by hand in this case.

Since the verifier should not be connected to the Internet, a way of transferring the data without an Internet connection is needed. Furthermore, the voting device should not know anything about the verifier. Therefore, the following possibilities can be applied:

- **Bluetooth:** Sending data via Bluetooth between devices would have been a pretty obvious choice. However, with his technique, the communication between the voting device and the verifier would not be solely in one direction (voting device $\rightarrow$ verifier) but bidirectional due to the connection establishing process. This harms the feature that the voting device should not know which device is used for verification. A further problem is that voters disabling the Internet connection probably also disable Bluetooth connections simultaneously since they select a smartphone's or tablet's flight mode instead of disabling WLAN and cellular Internet solely. Furthermore, both devices need Bluetooth capability.

- **Near Field Communication:** Basically the same as with Bluetooth, but the unknownness of Near Filed Communication (NFC), as well as the even smaller availability, made an application thereof even less practical. Furthermore, the distance between the voting device and verifier is smaller than with Bluetooth what can become unpractical.

- **Typing by Hand:** This approach would make sense since all communication connections can be disabled on the verifier application. However, typing cryptographic codes and hashes by hand is extraordinarily cumbersome.

- **Camera:** Making use of a camera and some scanning software (*e.g.* to scan Quick-Response (QR) codes or plain text) has the same advantage as typing by hand since it does not need electronic connections at all. Furthermore, if appropriately implemented, only one device needs a camera. The drawback of this approach is that the camera needs a resolution sufficient to scan displayed data. Scanning QR-Codes is already prevailing in different application areas what simplifies the usage for the voters due to experience.

- **Sound:** Transmitting data by sound waves would need loudspeakers on one device and a microphone on the other. This approach has drawbacks of privacy in regards to other devices listening at the same time. While it is visible which device scans some data with the camera (due to the position in front of the point of interest), it is not possible to easily recognize whether multiple devices listen to sound at the same time or not. Also, there is potentially more noise compared to scanning with the camera. Encrypting the data before sending via sound would bring extra effort and additional complexity.

Since in Switzerland only 2.8% of the population has no smartphone or tablet, the decision was made to use these devices' camera to scan the data in the form of Quick-Response (QR) codes [1]. Furthermore, a tablet could also be used as a voting device especially since the screen of it is large enough to display the QR-code in a sufficient size. Additionally, this technique would allow a complete offline usage with communication only from the voting device to the verifier and not bidirectional.

The QR-codes will contain the data needed for the verification. The content of the QR-codes, based on *Patrick Hayes* implementation in the programming language Rust, is described in the listing below and illustrated in figure 4.1 [31]:

- **Commitment:** The QR-code for the commitment contains an immutable identifier of the encrypted ballot containing the encryption of the selected voting option. This can be achieved with the usage of a secure cryptographic hash of the encrypted ballot. It is essential that the selected option is included since otherwise, the voting device can change it later, and also that the choice is encrypted because otherwise the voter would receive a receipt of the selected voting option. If the selected option is included in the encrypted ballot, creating the hash of a ballot not containing the selected option would end in a different cryptographic hash. Because an honest verifier does not know the randomness data used for encryption at this point, there is no possibility to generate a receipt or try to figure out the input in polynomial time.

- **Challenge:** The challenge, which will only be displayed if the voter wants to challenge the encryption (details in 3.2.3), contains all used data for encryption and the creation of the ballot. This means, the selected options are sent to the verifier as well as the randomness used for encryption. There is also the possibility of not sending the chosen options to the verifier but letting the user select or insert the voting options again in the verifier. However, especially if polls need text input, such as elections of people, inserting text on the smartphone may be cumbersome for certain people. Thus, it may be more user-friendly to display the received voting option and let a user confirm those. Furthermore, when verifying the same device multiple times to establish a higher trust level, it is easier to confirm the displayed options than inserting them in every round. This probably raises the likeliness of a verification round due to smaller extra effort. By receiving the commitment, the user gets a fully transparent receipt of the selected voting options what results in the fact that the ballot belonging to the received commitment must never be cast. However, ensuring that there is no possibility to challenge and cast the same ballot must be ensured by the voting device. Also, publicly known data (such as public keys or general data) can be sent with the challenge.

Both QR-codes should contain an ID such that the verifier can quickly determine which QR code is scanned and thus also in which step the voter currently is. This can also help to prevent wrong order scanning since the verifier can give a warning if the challenge is scanned without the commitment scanned before.

As soon as the voter scanned the commitment QR code with the verifier application, the verification begins. The scanned commitment must somehow be visible for the voter

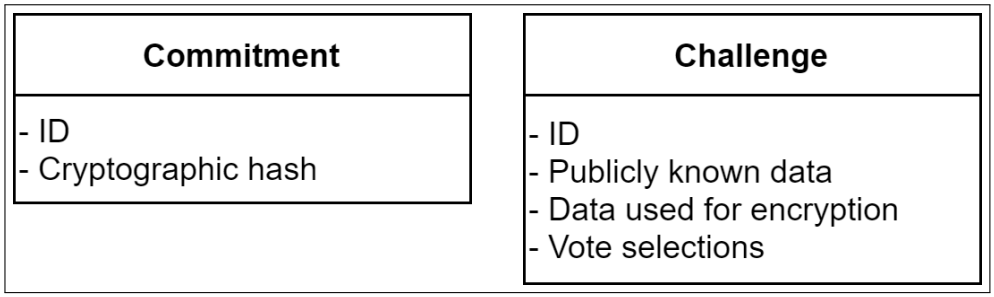

Figure 4.1: The theoretical content of the QR-codes used for the commitment and the challenge (based on [31]).

such that he/she can compare it to the one shown on the voting device. Comparing a cryptographic hash by eye is not done effortlessly. Thus when displaying the hash to the user, an easily readable format (*e.g.* space after every second character such that tuples emerge) should be chosen. Furthermore, items like cryptographic hash icons can be added to simplify the comparison (details in chapter 5). However, as soon as some color combinations are used, comparison solely on icons can become impossible for color-blind people. Lastly, the hash (and if used the hash icon) must always be visible during the verification process such that small exchanges of them are easily visible.

When the voter scanned the challenge, he/she needs to confirm the selected voting options. If a voter does not confirm the selected voting options, the verification process must be stopped immediately. This because there could be a malfunctioning voting device sending unselected voting options. If the user confirms the shown selections, the verifier encrypts the options with the same encryption algorithms and also the same encryption data. If everything was conducted honestly, this should result in the same encrypted ballot, which can be hashed by the same cryptographic hash function to create the commitment. Once the second commitment is created, the previously received one and the freshly computed one should be displayed to the voter next to each other. Here the same displaying style should be used as before, and maybe even an alert should appear if the two hashes are not similar. However, only displaying that the hashes are similar (or not) without displaying the underlying data is insufficient since then a voter cannot compare the commitments by himself/herself.

Independent of the result of the comparison, the user should be redirected to the same landing page as if starting the verification the first time. This indicates to the user that the verifier application does not know whether the voter verified a vote before or not.

4.4 Voter Front End

In this section, the additions needed for the verification steps are provided. The exact implementation and the adjustment done to Provotum's voter front end are described in chapter 5.

As stated at the beginning of section 4.3, the voter front end running on the voting device and the verifier must not know about each other in terms of which device is used for which

step. However, it is necessary that the two entities are aligned in terms of needed actions, data format, and exchange interface. This means it is unlikely that a different verifier application not designed for a certain voter front end can be used for verification.

The voter front end must store specific data during its encryption process. Namely, all the information contained in the QR-codes (figure 4.1) must be recorded during encryption and ballot generation. However, it is important that there is no way an honest voter can reach the data when not selecting *challenge* due to the receipt-freeness property.

When no challenge-or-cast verification mechanism is implemented, the voter could cast a vote directly after selecting the desired properties. However, as the verification gets added, the voter must confirm the selected options such that the voting device knows when to create the commitment. Before this, the encrypted ballot must be created such that the commitment (*i.e.* cryptographic hash of the ballot) can be prepared as well. The selection of an appropriate secure hash function is a trade-off between usability and security. If the hash is too long, it becomes cumbersome to compare them, while it becomes risky if a collision is found too quickly or for too many pre-images the same hash exists. However, the ballot is already encrypted for the hash creating, meaning there is no data leak when using a weaker hash function.

As soon as the hashing is completed, the hash should be presented to the voter. This must be done in the same way as in the verifier such that the comparison effort is minimized. This means if spaces are inserted after every second character of the displayed hash and a hash icon is used, it must be similarly displayed in the voting system.

While the commitment is shown to the voter, he/she must have the possibility to either select *challenge* or *cast*. If *cast* is selected, the previously encrypted ballot gets sent to the voting servers and if it passes all tests there, it will be put in the ballot box such that it will be counted in the tallying phase. At the same time, the stored data used during encryption gets discarded unrecoverable. If the user selects *challenge*, the information used for encryption is shown. Here it must be ensured that the ballot cannot be cast as well. There must be no way of doing both, challenge and cast, at the same time or consecutively.

After a successful round of verification, the voter should be returned to the first seen landing page to illustrate that the voter front end does not know the number of verifications done previously. This simply because the voter front end is not allowed to know how many times a voter verified the device since it could make probabilistic assumptions whether it will be challenged again or not.

4.5 Trust Boundaries of the Provotum System

A *trust boundary* is an edge where a user's level of trust in an application, a system, or hardware changes [35]. The trust boundaries are mainly existent since specific data is exchanged between different devices, probably not in the voter's custody [35]. Therefore, the level of trust a voter expresses towards an entity differs, resulting in various

trust boundaries [35]. In Provotum, four different trust boundaries are present, namely *Distributed Ledger*, *Voting Device*, *Randomizer*, and *Verifier Device*.

The trust boundaries are illustrated in figure 4.2 as light grey boxes with dotted borders and explained in the listing below. The white boxes represent the software part included in the Provotum system, while the arrows amongst them are an abstract representation of data exchanged between the entities. This also indicates the one-way connection and data exchange between the voting front end and the verifier application. Hence the impossibility of a verifier transferring data to the voting software is shown.

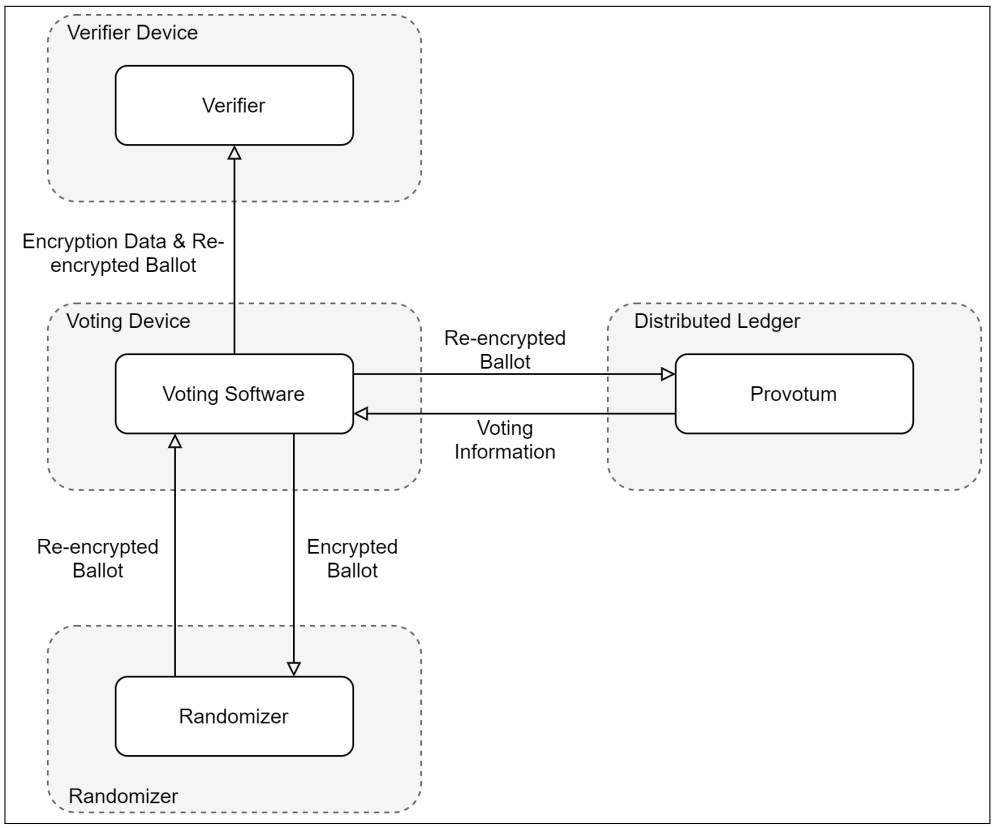

Figure 4.2: Illustration of the trust boundaries in Provotum including the verifier application.

- **Distributed Ledger** The Distributed Ledger trust boundary contains the Provotum chain, which is distributed over sealers participating in the current vote. Hence, the trust is not centralized on one entity but distributed over the sealers. This reveals classical blockchain advantages like no need to trust a single entity, immutability, and coercion-resistance.
- **Voting Device** The voting device can be in the voter's trust domain but can also be outside of it. This weakens the restriction of using a voting device that is trusted. If the voting software is not running on a device not trusted by the voter, the trust domain *voting device* is not trustworthy.
- **Randomizer** The Randomizer can be operated by different entities which are not in control of the voter. Hence, the voter may not trust this entity resulting in a different level of trust for this entity compared to others.

- **Verifier Device** The voter should theoretically trust the verifier device. Therefore, it is the only device trusted by the voter. However, it is also possible to use unknown and potentially untrustworthy devices as a verifier.

As visible in figure 4.2, the verifier and the voting software are not in the same trust boundary. The rigid separation of these trust domains allows the usage of devices that the voter may not trust. Since the verifier is used to verify the voting front end's honesty and trustworthiness, especially the latter mentioned can be running in an untrustworthy environment. This allows voters to use devices that are not controlled by a trustworthy organization or not their property. However, it is also possible that the verifier is running in an untrustworthy trust area. In this scenario, multiple devices must be used as a verifier, and the verification process must be conducted with every verifier for the same input data. A voter can trust the most occurring device assuming the majority is honest and not malfunctioning.

Chapter 5

Implementation

The implementation consists of two major parts: the additions made to the voter front end originally created by Alexander Hofmann and an all-new verifier application. The additions to the voter front end are described in section 5.1 and contain the creation of QR-codes containing the commitment and the data stored during encryption. The verifier application is explained in section 5.2.

The detailed process is visible in figure 5.1. It is also visible that the verifier application must not be used for a successful handing-in of a ballot. As in figure 4.2 illustrated, the voting system and the verifier application have different trust domains and thus a trust boundary between them exist. With regards to figure 5.1, it becomes visible the encryption and commitment creation is done on different devices and thus also in different trust domains.

5.1 Addition to the Voter Front End

In this section, the additions to the existing React front end will be explained. The starting point, meaning the front end developed by Alexander Hofmann, can be found in a private GitHub repository, precisely: `https://github.com/provotum/voter` (Accessed: 13.03.2021)

As described in section 4.4, when applying the challenge-or-cast verification mechanism, the voter must confirm the selection such that the encryption process can start. To achieve this, an additional step must be implemented such that the creation of the ballot is not bound to the cast mechanism. Therefore, the initial encryption process is outsourced to an own function (*i.e.* `export const createEncryptedBallot = (vote, keyring) => async (dispatch) => {...}` in *voter/src/redux/action.js*). Consequently, there are two additional functions called *castBallot* and *challengeBallot* to hand-in a ballot and to challenge the encryption. This minor change allowed the division between the encryption of a ballot and the casting or challenging thereof. Furthermore, it brings the possibility to create the commitment hash of the encrypted ballot. In Provotum, the encryption of a ballot is based on the encryption of the given answer to a certain voting subject (*i.e.* the

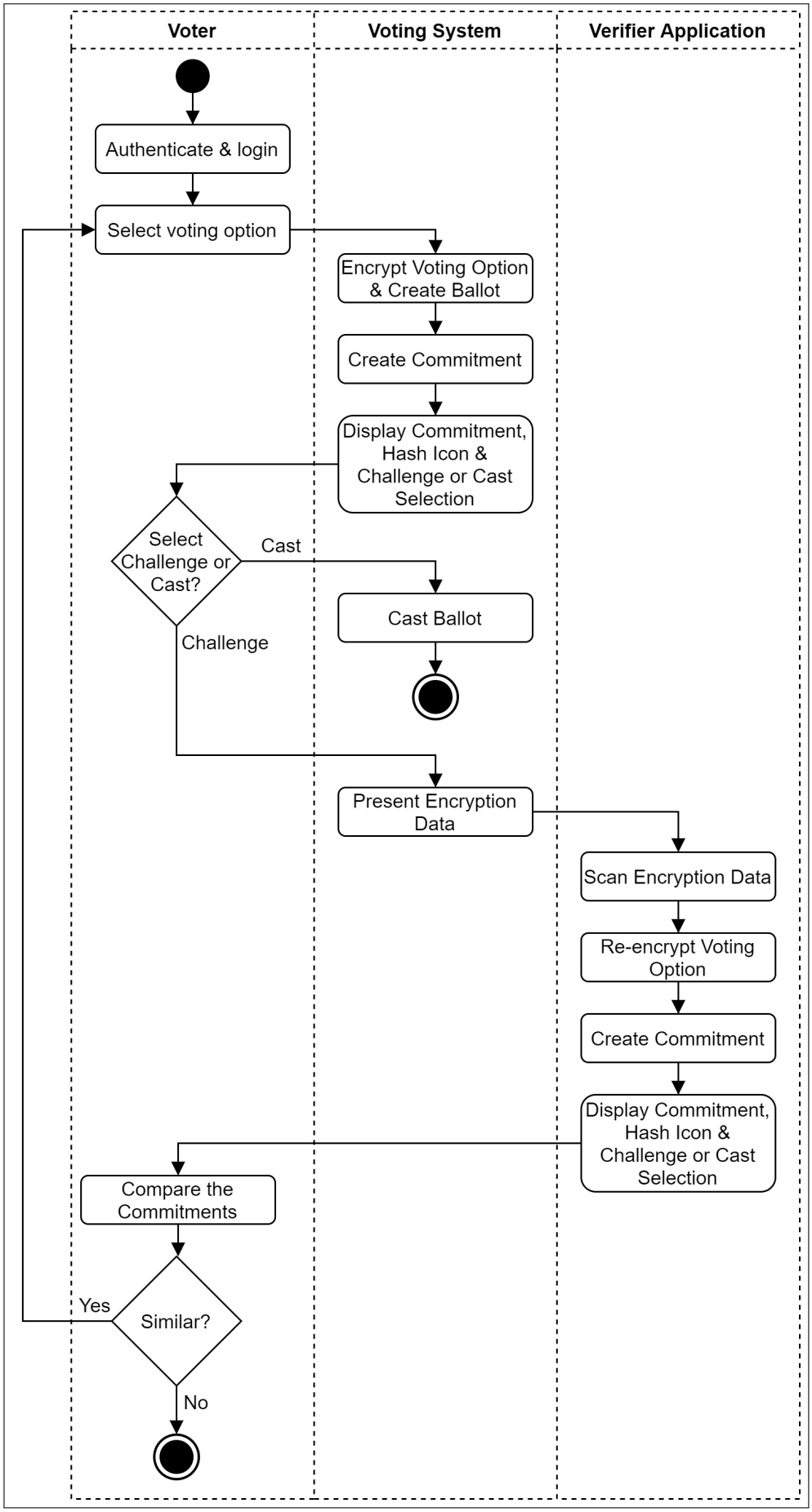

Figure 5.1: The Challenge-Or-Cast process in detail

answer to a specific voting question). Hence the handed-in ballot is a combination of the encrypted ones for every vote subject.

The voter front end must store specific data during encryption such that this can be transferred to the verifier with the help of the QR-codes. The data used to verify the encryption in Provotum is listed below:

- `publicKey` contains the `publicKeyH` as well as the parameters `g` (generator), `p` (prime number) and `q` (prime number).
- `voterPublicKey.h` is the voter's specific public key.
- `answerBin` representing the selected answer in binary form (*i.e 1* for *YES* and *0* if the user selected *NO*).
- `Nonce` is a random integer in the range of the parameter `q` defined in the publicKey's parameters.
- `reEncryptedBallot` represents the ballot after it is randomized with the help of the randomizer.
- `reEncryptionProof` is the proof used to witness that the re-encryption by the randomizer is done correctly.

Figure 5.2 illustrates Provotum specific adaptions to the content the QR-code must contain according to figure 4.1. The commitment includes the actual voting questions in plain text since they must be transferred to the verifier device as well. The `publicKey` and the `voterPublicKey.h`, summarized as *General Data* in figure 5.2, are similar for all voting subjects. Hence, they will not alter during the encryption of different subjects. On the other side, the remaining variables are different for every subject, meaning if multiple voting subjects are included, multiple (*e.g*) *answerBin* are included in the QR-codes. This means, the points *Vote Selection* and *Data used for encryption* in figure 5.2 are included multiple times when multiple voting questions are present.

The above-listed data is stored in JavaScript objects during the encryption process because this allows easy generating of structures and dependencies. Furthermore, it will enable straightforward parsing to the *JSON* format to generate the commitment and the QR-codes. To simplify the exchange between different React views, the data is stored in a *Redux* store.

The hash of the ballot (*i.e.* the commitment) is created with a SHA-256 hash of the JSON-element *preparedBallotsForHash* as visible in listing 5.1. Creating the hash in an isolated step allows simple security enhancements since exchanging the SHA-256 algorithm with a more secure one does not influence other parts of the code. The JSON-element used contains the *subjectID* (*i.e.* the ID of the vote question, represented as *sID*), the *encrypted ballot* (*i.e.* the encrypted ballot, which can be accessed through the *encryptedBallotArchive[index]*) and the *cipherToSubstrate* (*i.e.* the data sent to substrate and thus revealed on the bulletin board, represented as *cTS*)). An explanation of why this is enough is given in section 5.2.

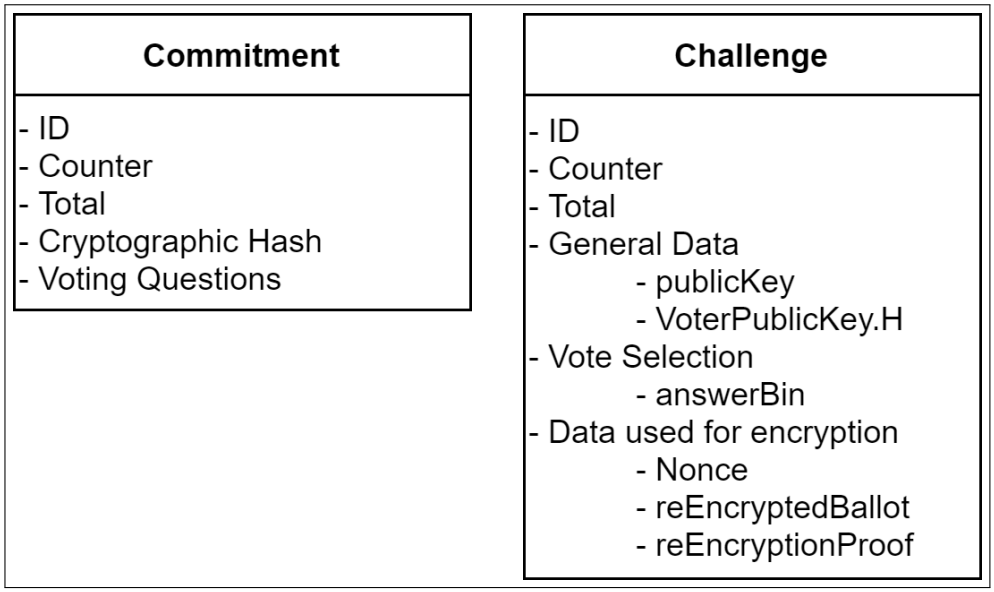

Figure 5.2: The content of the QR-codes used for the commitment and the challenge in Provotum.

Once the cryptographic hash is created (*i.e.* after the voter selected *Encrypt Ballot* on the voting device and thus confirmed the selected voting options), the commitment must be shown to the voter immediately. This is done in *voter/src/router/ScreenBallot.jsx*, where the cryptographic hash is fetched from the Redux store. For the commitment, this is shown in listing 5.2.

```
1 var preparedBallotsForHash = encryptedBallots.map(([sID, cTS], index) => {
2     return [sID, cTS, encryptedBallotArchive[index]]
3 });
4
5 const ballotHash = sha256(JSON.stringify(preparedBallotsForHash)).toString();
```

Listing 5.1: Creation of the commitment

To create the QR-code the npm-package *qrcode.react* (`https://www.npmjs.com/package/qrcode.react`, accessed: 16.03.2021) is used. The hash icon is created with the npm-package *@emeraldpay/hashicon-react* (`https://www.npmjs.com/package/@emeraldpay/hashicon-react`, accessed: 16.03.2021). Nevertheless, the cryptographic hash is also displayed in plain text with blanks after every second character such that it is possible to verify the characters of the code one by one easily.

The raw data of the `qrData` used as data input for the QR-code (line 6 of listing 5.2) is shown in listing 5.3. It contains the *id*, the ballot hash *BH*, a *Counter*, and a *Total* to ensure all QR-codes are received. Furthermore the *VotingQuestions* containing the *subjectID* as well as the plain text voting question are also added. The plaintext questions with the inherent voting question ID are already included in this QR-code due to easier handling of the received data in the verifier. When parsing the challenge QR-codes, the data can be allocated to a subject ID already received in the commitment.

```
<h1>Commitment</h1>
<div>
    {qrCodeReady &&
        <div className="qrFlexBox">
            <div className="qrCode">
                <QRCode value={qrData} size={600} includeMargin={true} />
            </div>
            <div className="item">
                <div className="cardDivSmall">
                    <h3>Commitment</h3>
                    <div>{ballotsHash.match(/.{1,2}/g).join(' ')}</div>
                    <div className="centerHorizontally">
                        <Hashicon value="ballotsHash.match(/.{1,2}/g).join(' ')" />
                    </div>
                    <div className="buttonDiv">
                        ...
                    </div>
                </div>
            </div>
        </div>
    }
</div>
```

Listing 5.2: HTML code for QR-code creation

The screen presented to the voter after the ballot creation phase is shown in figure 5.3. As it is visible, at this point, the user can either select *cast* or *challenge* but not both. Furthermore, a voter should scan the QR-code independent of the next step. A pop-up message to summon the user to scan the commitment is excluded on purpose since it is an extra effort for an experienced user to close this message. Allowing the selection between *cast* or *challenge* only after the user scanned the commitment is technically impossible since the voting device has no knowledge about the scanning progress. If the voter selects *cast*, the ballot gets cast to the bulletin board and, if all proofs are correct, gets counted in the tally. If the voter selects *challenge*, the previously stored encryption data must be presented to the voter such that it is possible to scan it with the verifier.

In theory, the challenge's content is similar to the one of the commitment shown in listing 5.3. The *id* is exchanged with *Challenge* and there is no ballot hash included but a *Key* property. This key contains either the value *GeneralData*, as visible in listing 5.4, or the *subjectID*, as visible in the code snippets 5.5 and 5.6. The verifier assorts the received data according to this key. The *Counter* is needed for the challenge since the data needed for verification exceeds the maximum size of a QR-code.

A QR-Code with an error correction level low and $177 * 177$ modules can maximally contain 4296 alphanumeric characters [7]. Additionally, huge QR-codes are less handy to scan since it needs a better camera resolution as well as a display with higher resolution to display them. Thus, the challenge data is divided into several QR-codes, where the total amount of QR-codes is denoted with the *Total* property and the current QR-code is labeled with the *Counter*. In the current implementation, the first QR-code (*i.e.* the QR-code with *Counter 0* - listing 5.4) is always used for the general data. Each subject-

```
{
    "id":"Commitment",
    "BH":"3a965d1d44ef07c16904e9e09014100a27c3eee5e03a952f1065e5ea0dba1579",
    "Counter":0,
    "Total":1,
    "VotingQuestions":{
        "0x8d00badf7491ac322cd7969ab37e9b44632ba5fa7d47868fa113a128b6d40291":"Popular
        ↪  initiative 'For responsible businesses - protecting human rights and the
        ↪  environment'",
        "0xd386cf8378309431a880a92e0fc59339c331b802d7fa2409b90994a8689f16cc":"Popular
        ↪  initiative 'For a ban on financing war material manufacturers'"
    }
}
```

Listing 5.3: Commitment QR-code content

specific question needs two additional QR-codes (*i.e.* the QR-codes with *Counter 1* & *2* - listing 5.5 & 5.6) containing all the voting question specific information. As listed above, this is the *answerBin*, *Nonce*, *reEncryptedBallot* and the *reEncryptionProof*. In the listings 5.4 to 5.6, long cryptographic hashes are shortened, which is indicated by ellipses, due to readability reasons.

```
{
    "id":"Challenge",
    "Key":"GeneralData",
    "Counter":0,
    "Total":5,
    "publicKey":{
        "h":"cb882ce4f7...",
        "parameters":{
            "p":"fffffffff...",
            "g":"02",
            "q":"7fffffffff..."
        }
    },
    "voterPublicKeyH":"6ae48794de..."
}
```

Listing 5.4: Challenge QR-code content 1

The QR-codes for the challenge are displayed in the same manner as the QR-code for the commitment, as visible in figure 5.4. The difference is that every second one of the challenge QR-codes is shown in a round-robin fashion. This rotation will be stopped once the voter selected *Back to Start* since the voting device has again no knowledge whether the voter scanned all QR-codes or not. This means that for a total of 5 QR-codes, the voter must, with an optimal scanning success rate, at least scan QR codes for 5 seconds. Besides that, the commitment and the cryptographic hash icon must be displayed all the time such that a voter can recognize potential changes immediately.

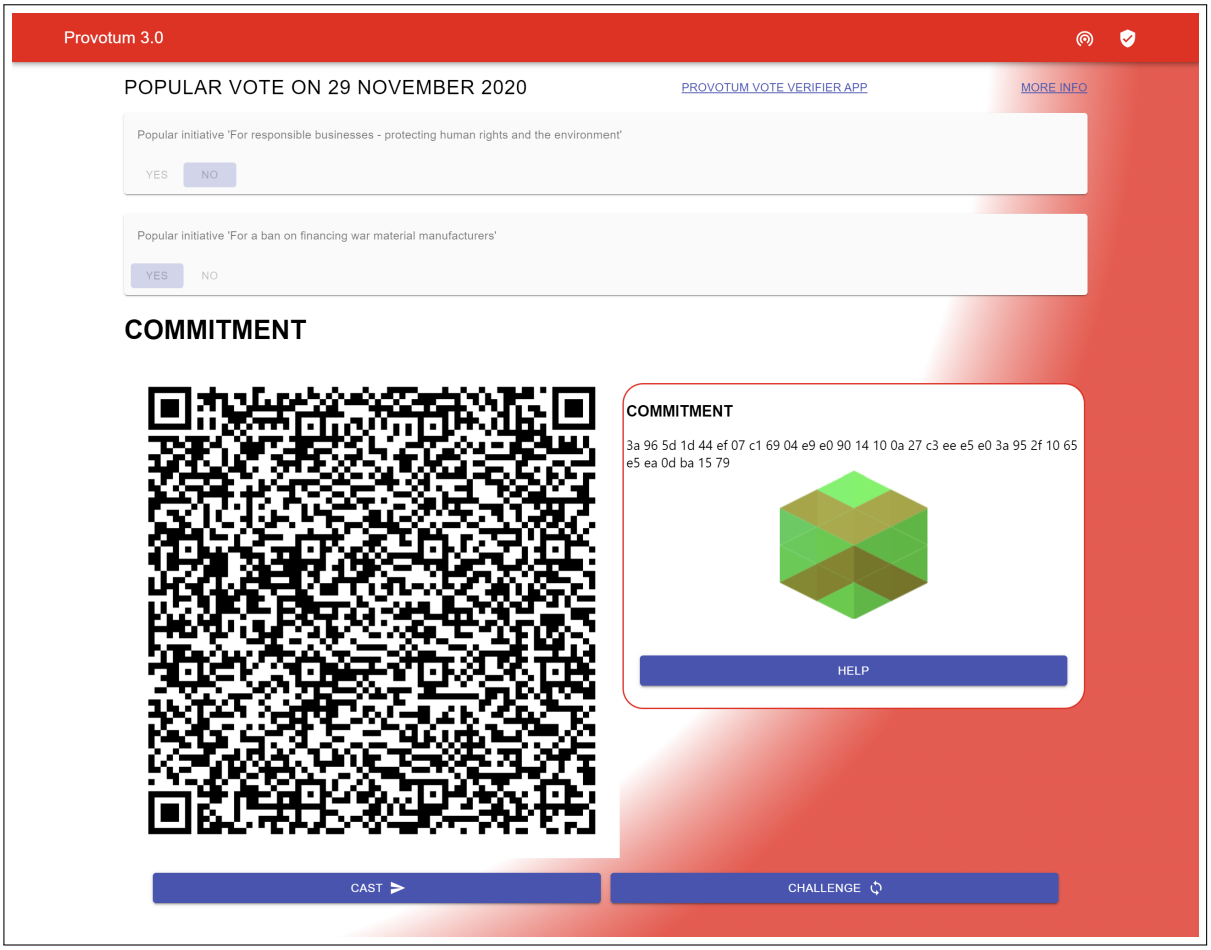

Figure 5.3: The commitment is shown to the voter in the voting front end

5.2 Provotum-Vote-Verifier-App

As described in the *Design* chapter 4, the challenge-or-cast approach with the help of QR-codes for the data exchange was selected. Furthermore, the verifier application must be able to run offline, which can be achieved by a Progressive Web App. There should be no knowledge of what device is challenging which voting device.

The PWA is created with the `create-react-app cra-template-pwa-typescript` template, which can be found on the *Create React App* website `https://create-react-app.dev/docs/making-a-progressive-web-app/` (Accessed: 14. March 2021). This template provides offline and cache-first behaviour, what enables offline usage, out of the box [71]. Furthermore, Google's Material-UI is used for styling of the application and Redux as store.

The app itself is divided into an *intro*, *scanning*, *confirmation*, and *result* page. These are accompanied with a *404 - Not Found* page for invalid URLs despite this page should never be reached if installing the verifier as an application on the smartphone. This simply because then the address line is inaccessible, and there are no routes towards an invalid URL in the application.

```
{
    "id":"Challenge",
    "Key":"0x8d00badf7491ac322cd7969ab37e9b44632ba5fa7d47868fa113a128b6d40291",
    "Counter":1,
    "Total":5,
    "answerBin":0,
    "Nonce":"3df6999519...",
    "reEncryptedBallot":{
        "c":"7094363b10...",
        "d":"a78b74de67..."
    }
}
```

Listing 5.5: Challenge QR-code content 2

```
{
    "id":"Challenge",
    "Key":"0x8d00badf7491ac322cd7969ab37e9b44632ba5fa7d47868fa113a128b6d40291",
    "Counter":2,
    "Total":5,
    "reEncryptionProof":{
        "ePrime":{
            "c":"770e7752c7...",
            "d":"2545a2c685...e"
        },
        "t2":"e1a1d01700...",
        "c1":"7f0127be2b...",
        "c2":"fed841d420...",
        "beta":"5bc5291701...",
        "s2":"9914178faa..."
    }
}
```

Listing 5.6: Challenge QR-code content 3

The app has *help* buttons on every page where the content alone is not sufficient as instruction, such that the user can gather additional information if wanted. The additional information is shown as an overlay view. The help button is visible in figure 5.5 and how it looks when the help view is open is shown in figure 5.6. The text will adapt to the current process step that a voter is facing. It was designed that way since a user probably does not need any additional information when he/she has done the verification already more than once. Thus, for an experienced user, it would be cumbersome if additional help pops up automatically every time and needs to be closed before it is possible to continue.

For the scanner, the *react-qr-reader* component is used (Accessed: 14.03.2021 via `https://www.npmjs.com/package/react-qr-reader`). Furthermore, an own overlay is made such that the users know where the scanning area is located. This is visible in screen 5.7, which is accessible once the voter selected *verify* on the *intro* screen. As soon as the voter scanned the commitment successfully, the QR-code will be parsed, the content stored to the Redux store and the screen shown in figure 5.8 will be displayed. At this point, the voter can compare the cryptographic hash and the hash icon thereof with the

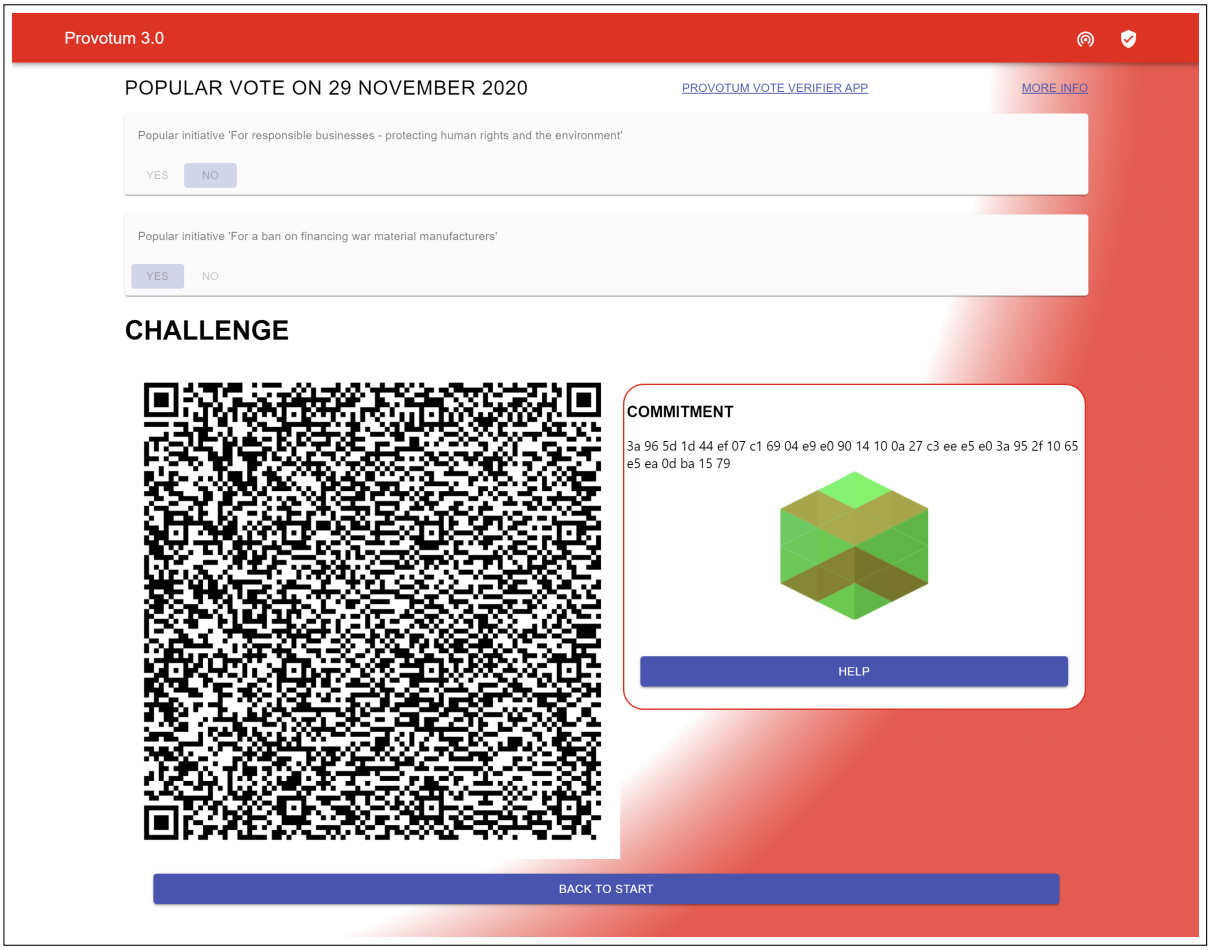

Figure 5.4: The challenge is shown to the voter in the voting front end

corresponding ones shown on the voting front end (figure 5.3).

Similar to the voting front end, the user must either select *cast* or *challenge*. If the user selects *cast*, an information is shown, that he/she must select *cast* on the voting device as well since otherwise, the ballot will not be cast. Selecting *challenge* will start the verification process. Thus, the scanner will open again, and once the user selected *challenge* on the voting device, he/she must scan all QR-codes displayed by the front end. The scanning progress is indicated next above the scanner, and once all QR-codes are scanned successfully, the application will continue automatically. For simplicity, a user cannot select which QR-code to scan but has to scan them in the given order. This means, that if one QR-code could not be scanned initially, a user has to wait until the missed QR-code is displayed again. This occurrence is visible in figure 5.9 where the QR-codes with *Counter 0*, *1* and *3* are already scanned while the ones with *Counter 2* and *4* are not done yet.

Scanning a QR-code not of the correct form, scanning the commitment after the challenge, or scanning the challenge before the commitment will result in error messages as visible in figure 5.10. The error messages are individual to match the fault as close as possible.

Once the challenge is scanned successfully, the application will automatically proceed to the *confirmation* step. In this step, the user must confirm if the displayed selections

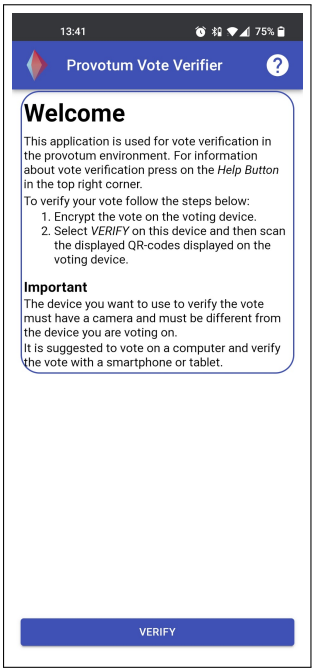

Figure 5.5: Verifier intro page

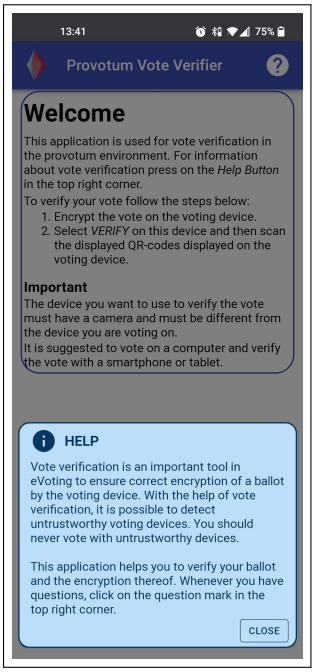

Figure 5.6: Verifier intro page with help

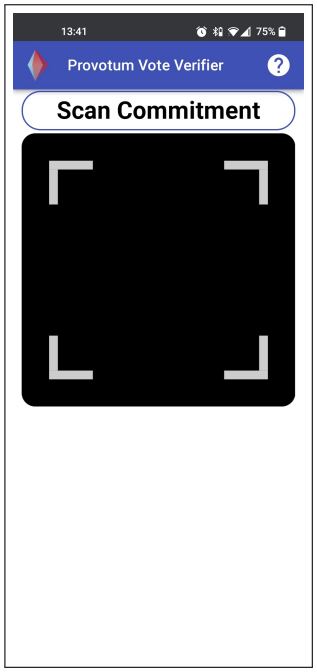

Figure 5.7: Commitment scanner

represent the ones he/she chose during voting on the voting device. If confirmed, the vote gets encrypted on the verifier device, and the cryptographic hash gets created such that it can be compared against the one shown on the voting device. The voter can compare the received commitment with the one calculated on the verifier device, as they are displayed next to each other. For simplicity, the verifier application shows the success depending on the result (visible in figures 5.12 and 5.13). Furthermore, a voter should compare the commitment displayed on the voting device with the calculated hash. However, when this is already done in the step illustrated in figure 5.8 and no change in either the cryptographic hash or the hash icon is detected during the process, comparing to the voting device is already done.

The encryption and verification on the verifier device are done with the same cryptographic library used in the voter front end since the verification needs the same parameters. The node module used can be found here: `https://www.npmjs.com/package/@hoal/evote-crypto-ts` (accessed: 14.03.2021). The complete encryption and verification process is shown in listing 5.7. This function takes the array `votingQuestions`, the `publicKey` and the `voterPublicKeyH` as input and returns the encrypted ballots, and a boolean `verifies`, which indicate if all votes are verified or not. The `votingQuestions` array contains all the data received via QR-code.

The verification is also done on a voting-question level. Hence an encrypted ballot is created based on the `encrypt(...)` function implemented in the library *@hoal/evote-crypto-ts*. This encrypted value is then compared against the value of the re-encrypted ballot, meaning the randomized ballot. The verification is only successful if the values of `encryptedBallot` and `reEncryptedBallot` match each other. While the voting device encrypts the vote, then randomizes and re-encrypts it with the help of the randomizer and

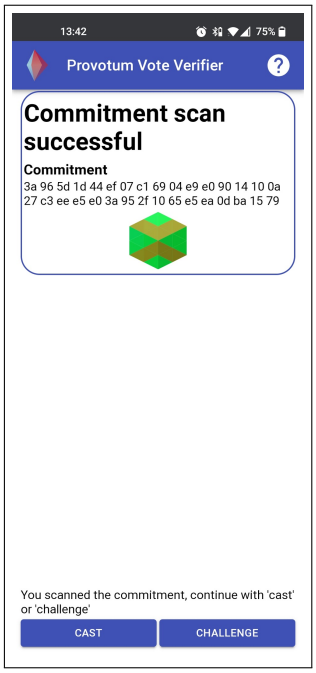

Figure 5.8: Successful commitment scan

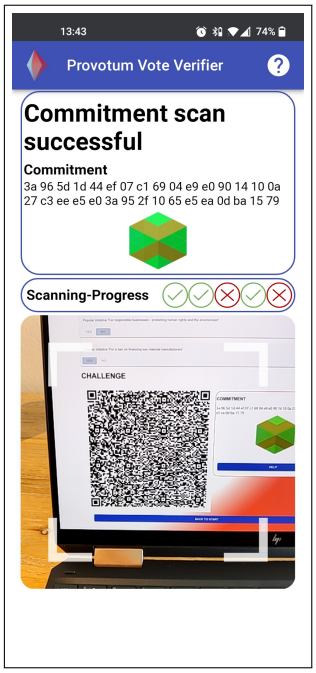

Figure 5.9: Scanning of the challenge

Figure 5.10: Scanning error

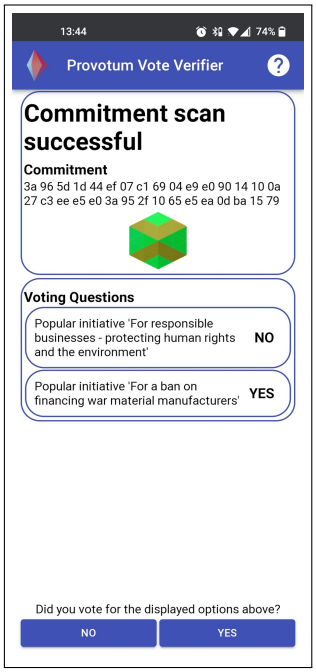

Figure 5.11: Confirmation of selection

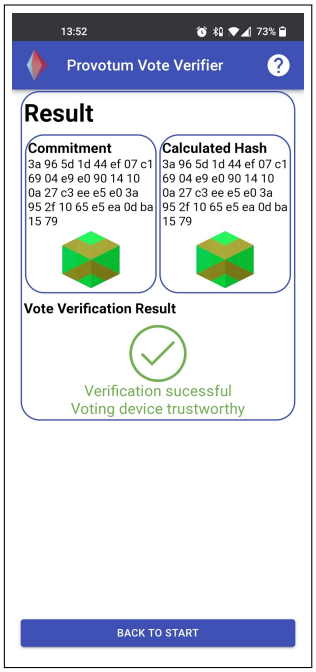

Figure 5.12: Successful verification

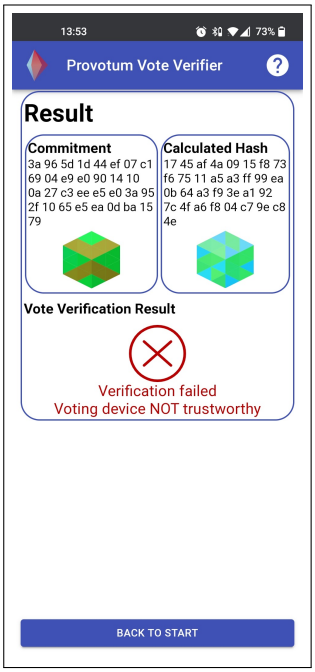

Figure 5.13: Unsuccessful verification

then compares the re-encryption with the initially encrypted ballot, the verifier compares the received re-encrypted ballot with the self-encrypted ballot. Since this verification will only succeed if the encrypted Ballot created on the verifier and the corresponding one created on the voting device are similar, it can be verified that the encryption was honest and also that the re-encryption matches the encrypted Ballot. Suppose the verification is successful `verifies` will be *true*.

The value of `allVerified` (listing 5.7) will only become *true* if, in all verification rounds, the value of `verifies` is *true*. Once one value is *false*, it will be *false* for all remaining rounds of encryption.

With this approach, the randomizer gets verified indirectly since if the randomizer returned wrong values, they will be detected at this point. This allows the verification of the randomizer without receiving random values used for the re-encryption or communicating to the randomizer during the verification process, what allows the verifier to stay completely offline while verifying not only the voting device but also the randomizer.

The creation of the calculated hash, which must be similar to the received commitment to verify the voting setup's trustworthiness, must be calculated with the same cryptographic hash function. Since a SHA-256 hash was used on the voter front end, this must also be used in the verifier. Furthermore, the same input in the same format must be hashed but with the difference that the ballot encrypted on the verifier must be used. If this is different, the verification of the re-encrypted ballot fails and it will result in a different cryptographic hash.

```
function CreateEncryptedBallot(votingQuestions: Array<any>, publicKey:
↪  ElGamalPublicKey, voterPublicKeyH: BN) {
  var allVerified: any = null
  var verifies: Boolean = false

  const encryptedBallots: Array<any> = []

  Object.entries(votingQuestions).forEach(([key, value]) => {
    if (value.answerBin === undefined) {
      return
    }
    const encryptedVote: Array<any> = []

    const encryptedBallot = encrypt(value.answerBin, publicKey, value.nonce);

    var verifies = verifyReEncryptionProof(
      value.reEncryptionProof,
      value.reEncryptedBallot,
      encryptedBallot,
      publicKey,
      voterPublicKeyH,
    );

    allVerified = allVerified !== null ? (verifies && allVerified ? (true) : (false))
    ↪  : (verifies)

    const cipherToSubstrate = {
      c: bnToHex(value.reEncryptedBallot.c),
      d: bnToHex(value.reEncryptedBallot.d),
    };

    encryptedVote.push(key);
    encryptedVote.push(cipherToSubstrate);
    encryptedVote.push(encryptedBallot);
    encryptedBallots.push(encryptedVote);
  })

  verifies = allVerified !== null ? (allVerified) : (false);
  return [encryptedBallots, verifies]
}
```

Listing 5.7: Encryption process for the verifier

Chapter 6

Comparison, Discussion & Evaluation

In this chapter, the analysis, comparison discussion and evaluation of the implemented verification mechanism are delineated. Furthermore, an analysis including security and handling aspects is conducted in subchapter 6.1. The comparison showing the additional value generated for Provotum listed in section 6.2. The chapter closes with a comparison of selected other implementations using the challenge-or-cast verification approach in section 6.3.

6.1 Analysis, Discussion & Evaluation

The analysis, discussion and evaluation of the verification approach can be looked at from different viewpoints. First, a technical perspective is taken, then a process-related one, and lastly, an environmental considering the periphery and situation the Provotum system runs in. In table 6.1, an overview of various threats, including a STRIDE classification is provided. The mitigation strategies against those threats are summarized in table 6.2. The ID refers to the inherent threat, meaning threat *T1* can be mitigated with mitigation strategy *M1*. The problems and mitigation strategies are discussed in further detail in sections 6.1.1 to 6.1.3.

The in table 6.1 introduced STRIDE approach was brought up by Loren Kohnfelder and Praerit Garg in 1999 [43]. The name STRIDE stands for and is an abbreviation of Spoofing, Tampering, Repudiation, Information Disclosure, Denial-of-Service, and Elevation of Privilege [67]. A short description of the threats based on the book *Threat Modeling: Designing for Security* is listed below [67]:

- **Spoofing** describes the threat of pretending to be somebody or something else.
- **Tampering** is the threat of altering source code, input data or process flows.
- **Repudiation** pictures the state where some entity claims that something was not in its responsibility or that something was not done by this entity.

Table 6.1: Summary of the security threats including STRIDE categorization

ID	Title	Threat	STRIDE
T1	Untrustworthy voting setup	The voting device is not trusted. Therefore, voting with it is risky.	Spoofing, Tampering
T2	Insecure connection	Insecure connections such as *HTTP* allow man-in-the-middle attacks.	Spoofing, Tampering
T3	Cross-Site-Scripting	Classic cross-site-scripting attacks to get sensible data.	Tampering
T4	Malicious service workers	The injection of malicious service workers can be achieved on the *localhost* development environment and with the help of social engineering.	Tampering
T5	Entry point to local data	The PWA, especially when installed, can be used by attackers to reach data stored on the verifier device.	Tampering
T6	Denial-of-Service attacks	DoS attacks can target entities providing the verification application such that voters cannot verify their ballots. In combination with malfunctioning voting devices this can facilitate counterfeit ballots.	Denial-of-Service
T7	Coercion-resistance	Voters can become victims of coercion.	Information Disclosure
T8	Deployment of the verifier	The deployment of the verifier can be problematic, if it is only done by a malicious entity.	Tampering, Elevation of Privilege, Repudiation
T9	Leak sensitive data	Attackers try to receive sensitive data.	Tampering, Information Disclosure
T10	Collusion between devices	Voting device and verifier arrange the creation of counterfeit ballots.	Tampering, Information Disclosure
T11	Social engineering attacks	Attackers influence voters not to use the verifier and at the same time provide malicious voting devices.	Tampering

- **Information Disclosure** reveals information to entities which are not allowed to see it.

- **Denial-of-Service** threats attack resources needed to provide specific services such that some services are not reachable.

- **Elevation of Privilege** attacks allow an entity to do something they are not allowed to do.

What holds true for all aspects is that with the introduction of the second device, the verifier, the encryption of the ballot, and thus also the trustworthiness can be challenged. Therefore, it is not mandatory to trust the voting device blindly but challenge it to establish trust on experience (threat *T1* in table 6.1). This introduction mitigates the threat of voting with a malfunctioning or dishonest voting device (mitigation strategy *M1* in table 6.2). It is classified as *Spoofing* and *Tampering* since specific devices or parts of the system can be manipulated, malware could be inserted or devices could fake their identity.

Table 6.2: Summary of the mitigation strategies according to the threats defined in table 6.1

ID	Mitigation
M1	Due to the challenge-and-cast setup, two or more devices can be used. This allows to challenge the voting device which can be caught cheating this way.
M2	PWAs enforce the use of *HTTPS*. Furthermore, there is no communication from the verifier application with a external server containing sensitive data.
M3	Since the data received by the verifier is not sensitive, the intention of an attack is not truly given.
M4	This occurs on *localhost* only. Hence, not using *localhost* for verification and an official version of the verifier instead mitigates this threat.
M5	PWAs run in a browser sandbox even when they are installed on the device. Furthermore, service workers only have access to the cache storage and not to the local or session storage.
M6	There are three main methods against this threat which are preferably combined: *(i)* DoS mitigation tools should be applied at the entity providing the verifier application, *(ii)* the users should install the PWA on their device such that offline usage is possible, and *(iii)* the verifier application should be hosted by multiple entities.
M7	This is only partly mitigated. An honest voter will not receive a receipt, but a dishonest voter can generate a receipt by reading out data at runtime. Furthermore, a coercer observing a voter will see the same information as a voter. This could be fought by allowing to cast multiple ballots and only count the last one.
M8	Multiple deployments or deployment by a Trusted Third Party can mitigate this.
M9	This is mitigated since the verifier does not handle sensitive data since the received data is never cast. Furthermore, offline usage can help against this threat.
M10	This theoretically is possible, but since the voting device and the verifier must communicate, it becomes problematic when using the verifier offline. The unawareness about which device is used as a verifier complicates communication even more.
M11	This can be mitigated technically and processual when making verification mandatory. With challenge-or-cast, this is not reachable. Therefore, active campaigns motivating voters to use the verifier should be applied.

If the user uses a dishonest verifier, he/she will likely never cast a ballot, although the voting device is encrypting the ballot correctly. Having an untrustworthy voting device can simply be detected when exchanging the verifier application. If the verification then fails, the voting device is dishonest with a high probability.

It is essential to state that this technique does not ensure that there is never an unselected voting option reaching the tallying phase since it depends on the user's decision whether to challenge the device or not. Suppose the device, respectively, the ballot it creates, is not challenged. In that case, there is no hedge against casting a counterfeit ballot since it is technically and procedurally impossible to detect a malicious one. Therefore, it is the voter's responsibility to challenge and verify their voting setup before casting a valid ballot. Motivating users to verify their voting setup every time they use it is crucial.

6.1.1 Technical

From a technical perspective, a PWA is basically enriched web applications. Therefore, attack scenarios and vectors known for web applications are also applicable for PWAs [65]. However, since PWAs use new technologies and features, some attack vectors can be

mitigated before becoming a threat. As described in section 4.3, *service-workers* are a crucial point of a PWA since they allow app-like functionalities. These service-workers only work via an *HTTPS* connection [65]. This type of connection is important since otherwise man-in-the-middle attacks targeting the service-workers can occur [65]. With a *HTTPS* connection it can be ensured, that the service-workers are not tampered on the way from the hosting server to the verifier device [65]. Hence the STRIDE categorization is *Spoofing* for man-in-the-middle attacks and *Tampering* due to the possibility of inserting malicious service workers. This threat is summarized as *T2* in table 6.1 and can be mitigated with the strategy *M2*.

However, even with *HTTPS* enabled, there are attack vectors with which attackers can install malicious service-workers [65]. In the first one, cross-site-scripting and the upload of a use case-specific file are needed, and in the second one, a victim gets forced to install malicious service-workers on *localhost* with the help of a social engineering attack [65]. To prevent damage, service workers only have access to the cache storage but not to the local or session storage [65]. Since the user cannot upload a file within the PWA, the first attack vector cannot be applied by an attacker. The second one does affect only the *localhost* since *localhost* is excluded from the *HTTPS* service-worker installation policy due to simpler development of PWAs. Consequently, not running the verifier in production over *localhost* will resolve this threat. Furthermore, PWAs always run in the browser-sandbox even if installed on a device [65].

A native app would be verified by the respective App Store reducing the possibility of a malware-infected application [65]. This is not the case with PWAs since they are hosted like normal web sites. However, PWAs have the advantage that the updating is more straightforward, since the only thing needed is to host a newer version of the application.

Denial-of-Service (DoS) attacks against the entity providing the verification application such that the voters cannot access the verifier and thus cannot verify their votes can occur (threat *T6*). However, if the voting device is trustworthy and not malfunctioning, the result will not be influenced when a voter cannot verify the ballot. Here, common DoS mitigation techniques need to be applied by the provider, or the application must be provided by various entities. Distributing the provisioning of the verifier would make it more difficult for the verifier and voting device to work together since there are more possibilities of combinations (further details in section 6.1.3).

The coercion-resistance property (as defined in section 3.2.1) and, as a consequence, also receipt-freeness only holds when an honest voter is assumed (threat *T7*). It is classified as *Information Disclosure* because this threat would harm the vote secrecy property. Due to the nature of Provotum, a technically adept dishonest user can readout the encryption values at runtime without much effort. This would allow generating a receipt in the way of re-encrypting all possible voting options with the same randomness and then compare it against the re-encrypted ballot received from the randomizer. If this comparison reveals that the re-encrypted ballot and the newly created one belong together, a dishonest voter has created a receipt. This can not be mitigated with the current setup of Provotum.

For the implementation, described in chapter 5, open-source libraries used for the generation of the QR-codes and the scanning of those are applied. Furthermore, a library

containing all the cryptographic functions tailored for Provotum, the popular Redux library, and public frameworks and templates are applied in the code (details in chapter 5). Since about 7 out of 10 applications face security risks emerging from open source libraries, selecting these libraries must be conducted carefully [66]. The most crucial point here is that the cryptographic library gets adapted in the voting front end and in the verifier once it gets updated. Otherwise, the encryption may result in different results. Furthermore, the hashing with SHA-256 to create the commitment is currently considered secure enough, but it may must be replaced in the future. However, it should be kept in mind that the hash must be easily comparable by eye what is a limiting factor in regards to its length. Also, an already encrypted ballot is hashed. Regarding the hash icons, it could become problematic if the hash icons representing the commitment are not different enough. But as they are only additional to the alphanumeric hash values presented, comparing the character strings instead of solely comparing the hash icons is the user's responsibility as soon as the comparison is intricate.

Overall the PWA is technically implemented in a way that it does not use any sensitive data for verification (threat *T9* in table 6.1). The threat of revealing sensitive data would be classified as *Information Disclosure* since it reveals data to somebody not allowed to see it and *Tampering* since the source code or data storage needs to be manipulated beforehand. Technically, malfunctioning PWAs need an untrustworthy companion voter front end such that they can successfully create and verify malicious ballots. This gets hampered tremendously due to the offline usage possibility and due to the fact that the voting device has no knowledge of which device will be used as a verifier. Due to the fact that the verification is optional, a dishonest voter will simply not verify the vote. This is the case because the verifier is not needed to cast a ballot.

6.1.2 Process-related

In this subsection, The process of the challenge-or-cast verification is analyzed and discussed with a process-related focus. As a challenged ballot can never be cast, the verification will not reveal a meaningful receipt since the receipt an honest voter can use for vote selling or coercion is from a ballot that is not cast. Hence, a coercer cannot receive a ballot from an actual cast ballot. This is even strengthened because of the randomizer, which will blind the votes. Therefore, the process does not leak any unwanted information revealing the vote selection if honest participants are assumed. Furthermore, the randomizer also gets verified with the *Provotum-Vote-Verifier-App* which will increase security. Thus, the process becomes more secure, bringing less possibility for malicious devices or entities to alter ballots.

However, if dishonest voters are assumed and if additional tools such as screen recorders are applied, the voter can record the complete voting process and use this material for vote selling. Here tools against screen recording can be integrated into the voting front end to try fighting against it. Nevertheless, as soon as a voter is recording the screen with an additional camera or voting with a coercer/vote buyer next to him/her, tools are helpless. This can only be fought if voters are not allowed to vote remotely or if a voter can hand in multiple ballots while only the last one counts. A potential coercer can never be sure if the voter does not cast a ballot after providing the receipt.

The process can become safer when voters challenge the voting device with multiple verifier devices. This is the case because the possibility that numerous devices are malware-infected and work together is smaller than with only two devices. Using multiple verifier devices to verify the same ballot would voters allow to use verifiers from an untrustworthy domain. However, this also raises the extra effort a user must take to establish trust.

The extra effort a user must invest in generating trust is also criticised by *Kulyk et al. (2019)*. They state that the verification is used to remove voters' concerns about the voting system's integrity [48]. At the time of writing, it is not apparent how voters will behave once their initial concerns have vanished. This could result in the problem that experienced users will not challenge the votes since they already trusted the system the last time they used it [48]. Stating that this behaviour must be studied further in the future, it theoretically would result in less verification and, therefore, higher chances of malfunctioning devices succeeding in casting malicious ballots [48]. Preventing this would probably only be possible in making the verification mandatory, which is not possible with the challenge-or-cast mechanism since the voting device is not allowed to know how often the verification is done.

In Estonia, where people have some experience with Internet voting, in 2013 only 3.43% and in 2014, about 4.04% of the electronically cast ballots were verified with the provided verification system (described in section 6.3) [33]. Another study found that approximately between 26% and 31% of all verified ballots belonged to women, while slightly more than half of the electronically cast ballots were handed in by women [32]. Unfortunately it is not stated why these numbers are achieved, but they could, with care, be used as an indicator for the willingness or assumed verification necessity by voters. Also, it illustrates the need to motivate all genders of voters to verify their ballots.

Technically and processual, it is significant that there is enough easily accessible guidance during the process, especially since the challenge-or-cast approach is rather counter intuitive. The instructions on the next step must be clearly visible because they will be overlooked otherwise [48]. One common problem occurring is, that the voters do not scan the challenge since the commitment on the voting device and the verifier looks similar after scanning it [48, 56]. Voters mistakenly believe that they already verified the ballot correctly at this point [48,56]. Based on the process, this is not the case since the encryption has not proceeded on the verifier. Another common problem is the counter-intuitive handling since the voters are surprised that a challenged vote cannot be cast and a cast one is not challenged [56]. Here detailed descriptions or an approach where the challenged ballot can be cast would be beneficial [56].

In the challenge-or-cast approach, the verification device gets to know the selected option. This because the verifier needs the information about the selected option to encrypt the ballot locally again. However, due to the process design, the voting device does not know the actual cast ballot's included chosen voting option. The maximum the verifier knows is the intention a voter probably has, but it cannot be sure that a voter indeed voted for the same option with which the encryption is challenged.

The code-based approach has an advantage compared to the challenge-or-cast approach since it also verifies the connection to the voting server. This would raise the security level even more since, despite the Cast-as-Intended property, the Recorded-as-Cast feature is

verified as well. On the other side, the challenge-or-cast approach has advantages since no codes created by a single entity are needed.

6.1.3 Environment-related

In this subsection, the challenge-or-cast method is examined regarding the environment, including Provotum specific topics and involving the stakeholders defined in section 4.2. The verification possibility has a significant influence on voters and their behavior. Furthermore, there are points that should be considered for providing and hosting the Progressive Web App.

The voters can verify their ballots such that they can challenge the voting setup to prevent maliciously encrypted ballots. This increases the security for a voter since he/she must not rely on unsubstantiated trust. Hence the voting becomes more secure from a voter's perspective.

Making the PWA accessible for all voters is not as trivially as assumed, since if it is only provided by one entity, this entity must be a trusted one. Hence, the voters must, similar to the case with the voting authority, trust a Trusted Third Party. This trust can be avoided if various different entities host the verification application (threat *T8* and mitigation strategy *M8*). This could be achieved if the sealers, which are already distributed in a location-independent manner, furnish the application in addition to the validation of the blocks. Therefore, the need for a single TTP providing the verifier application is negligible since a user can select the sealer hosting the verifier independently. Giving the voters the opportunity to choose which provider they want to use reduces the possibility of the verifier and voting front end working together. Furthermore, it may be helpful if the voting authority provides a test environment where a voter can test the verifier before interacting with the real system. This would allow the verification of the verifier application if not provided by a TTP.

Not relying on one single randomizer entity would also enhance security. However, since the verifier application can verify the randomizer as well, the added value of multiple randomizers is not that big as the one given through multiple verifier applications.

Social engineering attacks with the objective to discourage voters from using the verifier and at the same time providing malfunctioning voting front ends can not be fought with the current setup. This is especially true because the verification is entirely optional, and there is no need to verify the voting front end at least once before voting. Hence, it is essential to mobilize voters to use the verifier application. This is classified as *Tampering* in the STRIDE scheme because it is a modification of the designed voting process.

Accountability, as described in section 3.2.2, is only partly met, since if the verification fails, the user does not know directly whether the voting device or the verifier is untrustworthy. To determine which device is untrustworthy, a second verifier device needs to be used. This would allow cross-checking of the commitment generated and clearly indicate whether the voting device is malicious or not. If both verifiers result in the same commitment, it is likely that the voting device is not trustworthy. On the other side, it is an

indicator of an untrustworthy verifier if the voting device and one verifier result in the same commitment.

Furthermore, voters must have a trustworthy way to report and announce malicious devices. This is currently neither implemented in the verifier nor in the voting front end. Doing so would be trivial and straightforward since it is only displaying contact information. This cannot be automated because the voting device does not know if the verification succeeded, and it is the user's decision to report malfunctioning devices. The possibility to report malfunctioning devices would increase the overall security and trustworthiness.

6.2 Comparison with Provotum Before - After

Vote verification adds the possibility to verify the encryption solely generated on the voting device and thus allows the determining whether the voting device is honest or not. This possibility is the primary added value for a voter since it will enable voters to use a voting device they do not trust. This is definitely an advantage since trust in the voting setup can be established even in a remote environment. It allows users to verify that their vote was Cast-as-Intended which is one of the three pillars a voting system should serve.

With Provotum 2.0, as described in *Hofmann (2020)*, a voter did not have the verification possibility, and there was also no Cast-as-Intended feature implemented. So basically, a voter must trust the voting client but had no evidence of whether their trust is appropriate or not. With the addition of vote verification in Provotum, this trust can now be founded on experience since the voter can determine if a voting device is trustworthy. This results in a higher overall trust level what theoretically will reduce insecurity about electronic voting systems. In *Hofmann (2020)* this threat is listed with ID *T1* in the analysis of attack vectors and limitations. It is remedied with this work.

Besides the newly added Cast-as-Intended feature, Provotum already features Recorded-as-Cast and Counted-as-Recorded. Furthermore, an honest voter will still not receive a valid receipt of an indeed cast ballot and cannot transfer the re-encryption proof to a third party due to the usage of designated verifier proofs. Therefore, the receipt-freeness property holds for honest voters.

6.3 Comparison with other Implementations

In *Adida (2008)*, the Helios system is targeted for elections with a low coercion risk [9]. In their approach, a voter can challenge as many ballots as wished, and once the voter is convinced, he/she must authenticate himself/herself and cast the ballot then [9]. They state that all vote selections are only recorded in the browser without any network calls [9]. Only if the ballot is encrypted and the plaintext is discarded, network calls occur [9]. In Provotum, this is different since there are network calls to the randomizer entity after the ballot is encrypted but before the plaintext selection is discarded. This is partly caused by the nature of Provotum's encryption process and partly by storing the selection for

a possible challenge of the ballot. Furthermore, in the Helios version created by *Adida (2008)*, the auditing of the encrypted ballot is done in a ballot encryption verification program which takes the used randomness as input [9]. Exchanging the randomness is done via downloading the randomness and importing it into the verifier software [9]. Additionally, the voter gets redirected to the selection screen once he/she finished challenging the encryption [9].

In Provotum or the work described by *Kulyk et al. (2019)*, the verification is air-gapped, which may not be the case with the above-introduced Helios version since there the verification program can run on the same device. Furthermore, in Helios the voter is bound to a computer since the downloading and importing of the used randomness into a specific Python program is needed. Probably voters will use the same computer they voted with, which can become problematic if this device is in the same trust domain, which may be untrustworthy.

Other systems, such as the one described in *Bell (2013)* or *Ben-Nun (2012)*, which are not for remote voting, includes paper as a physical medium to transfer specific data [11, 12]. The latter makes use of dual voting, including electronic and paper-based voting for the same election [12]. Since these mechanisms are used onsite, they will not be discussed further here.

In the Estonian electronic voting system, a smartphone app is used for vote verification [33, 75]. The fundamental difference in the Estonian voting system is that voters are allowed to cast multiple ballots, and only the last one is counted [70]. This brings the possibility to verify a ballot after it is cast [33]. In their approach, an active Internet connection is needed since the verifier application needs to communicate with the vote collection [33,75]. For the verification, the cast ballot is downloaded to a voter's smartphone, which compares it to the encryption of all possible voting options [33,75]. If there is a match between the received ballot and a newly encrypted one, the voter has an indication that the ballot was Cast-as-Intended if the match occurs for the indeed selected voting option [33,75]. If the ballot matches with an undesired voting option, a cheating malfunctioning voting device is detected. While with this approach, a voter gets a receipt, the nature of the voting process prevents coercion since a coercer cannot be convinced that the presented ballot was the last one cast.

Furthermore, *Yilmaz et al. (2020)* bring up the possibility of visual comparison of the matching vote selection with the help of a color representation consisting of 32 colored cells [75]. Whether this or the in Provotum implemented hash icons are more straightforward for comparison is depending on the personal preference. The possibility to compare the values in an alphanumeric form is missing in the Estonian verification application in the form presented by *Yilmaz et al. (2020)*. This may become problematic for color blind people.

Chapter 7

Summary, Conclusions & Future Work

This chapter summarizes the most essential points of this thesis. Furthermore, a short conclusion and some future work suggestions are given.

7.1 Summary & Conclusion

This thesis aims to introduce vote verification such that the Cast-as-Intended feature is present in Provotum. This is achieved with the implementation of the challenge-or-cast method, including a second device running the verifier application. The challenge-or-cast property of either challenging or casting a ballot makes use of the circumstance that the voting device does not know whether it will be challenged or not while creating the commitment. The commitment must contain the selected voting option, and it must be ensured that it is impossible to challenge and cast the same ballot. Since the commitment is immutable, the voting device cannot alter the selected voting option once it learns the voter's challenge-or-cast decision. As this commitment is scanned with the voter's verifier device independent of the planned decision (challenge-or-cast) and the selected option is included in encrypted form, the voting device cannot manipulate the commitment later. Since the implementation of the verifier application is done with a Progressive Web App, which can be used offline, the potential agreement on manipulating a ballot between the voting device and the verifier becomes more complex than it would be when only an air-gap exists.

Having a second air-gapped device that is solely used for verification allows a voter to verify whether the voting device is trustworthy or not. This allows voting in an unfamiliar, potentially insecure environment where the voting device could be in an untrustworthy trust domain. This thesis shows that trust can be established with the challenge-or-cast approach and that counterfeit ballots can be detected with the help of a separate verification device. The additional value generated allows voters to remotely and electronically cast their votes while still being sure that the selected option is encrypted and thus included in the ballot. The air-gapped verifier device functioning in offline mode combined with a communication flow only from the voting device to the verifier minimizes the

possibility of the two devices communicating with each other. This enhances the complete voting and tallying process's security and verifiability tremendously since it detects malicious input and malfunctioning devices at the first stage of the voting process.

7.2 Future Work

Future work on this topic, the Cast-as-Intended property, is possible in multiple directions. Overall, future work on Provotum could focus on getting away from the *honest voter* assumption. This assumption should be attenuated since it sadly does not perfectly coincide with reality. This would have a significant influence to topics like *coercion-resistance* and *receipt-freeness* since, at the time of writing, a dishonest voter can generate receipts.

Usability & Verification Willingness

While this thesis focuses more on the technical aspects of Cast-as-Intended and the implementation of a method to achieve it, further studies about usability are needed. Especially since the current process is somewhat counter-intuitive. Usability studies could be realized in an observed setting such that the bystander directly sees where voters get stuck or struggle. In a second phase, studies about the number of verifications performed in a vote would be fascinating. This could only be achieved once proper identity management is available since it must result in a quantitative result which is only achieved through observing many participants.

In terms of user studies, it is also important to detect and define how much information and help needs to be given to the voters. The current implementation's handling instructions need to be actively opened. The reason for this is that a voter familiar with the verification process does not need to be informed all time. However, this is only based on assumptions, and it is probably better to display visual aids every time a voter verifies a ballot.

It needs future work in the direction of verification willingness, especially once voters are used to voting with a remote electrical voting system. At the time of writing, it is unknown what will happen once users gained trust in the voting system. Because the challenge-or-cast approach brings extra effort, users who have gained trust through verification during a previous election may not challenge the voting setup again, even though it can be malfunctioning this time. This is also mentioned by *Kulyk et al. (2019)* [48].

Other Verification Approaches

Once studies about usability and willingness to verify the vote are conducted, it would be interesting to see if another method (*e.g.* code-based or challenge-and-cast) would achieve other scores and results. This would allow to identify the most suitable verification technique for the broad population.

Identity Management

Identity management matching the intention of a remote electrical voting system is crucial for the long-term success of Provotum. Without this, it will probably not get further than a research project. This, because it is not possible to manage access privileges for a large number of voters by hand. Probably it would be a good idea to think of a decentralized identity management system that matches the intention of Provotum. This minimizes the need for a trusted third party as an identity management entity which would increase the application area's scope.

Bibliography

[1] Anteil der Besitzer von Smartphones in der Schweiz von 2017 bis 2020. https://de.statista.com/statistik/daten/studie/537944/umfrage/besitz-von-smartphone-bzw-tablet-in-der-schweiz/. Accessed: 11.03.2021.

[2] e-governance. https://e-estonia.com/solutions/e-governance/i-voting/. Accessed: 4.11.2020.

[3] E-voting. https://www.evoting.ch/en. Accessed: 24.02.2021.

[4] How, Where, and When to Vote. https://www.usa.gov/how-to-vote. Accessed: 31.01.2021.

[5] Manipulation von Wahlen und Abstimmungen. https://www.ch.ch/de/demokratie/abstimmungen/falsificazione-dei-risultati-elettorali-e-frode-elettorale/. Accessed: 17.02.2021.

[6] Total of Estonia. https://rk2019.valimised.ee/en/voting-result/voting-result-main.html. Accessed: 4.11.2020.

[7] What is a QR code? https://www.keyence.com/ss/products/auto_id/barcode_lecture/basic_2d/qr/. Accessed: 16.03.2021.

[8] Ben Adida. *Advances in cryptographic voting systems*. PhD thesis, California Institute of Technology and Massachusetts Institute of Technology, 2006.

[9] Ben Adida. Helios: Web-based open-audit voting. In *17th USENIX Security Symposium (USENIX Security 08)*, San Jose, CA, July 2008. USENIX Association.

[10] Giuseppe Ateniese and Breno de Medeiros. On the key exposure problem in chameleon hashes. In *Security in Communication Networks*, pages 165–179, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-30598-9_12.

[11] Susan Bell, Josh Benaloh, Michael D. Byrne, Dana Debeauvoir, Bryce Eakin, Philip Kortum, Neal McBurnett, Olivier Pereira, Philip B. Stark, Dan S. Wallach, Gail Fisher, Julian Montoya, Michelle Parker, and Michael Winn. Star-vote: A secure, transparent, auditable, and reliable voting system. In *2013 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 13)*, Washington, D.C., August 2013. USENIX Association.

[12] Jonathan Ben-Nun, Niko Fahri, Morgan Llewellyn, Ben Riva, Alon Rosen, Amnon Ta-Shma, and Douglas Wikström. A new implementation of a dual (paper and cryptographic) voting system. In Manuel J. Kripp, Melanie Volkamer, and Rüdiger Grimm, editors, *5th International Conference on Electronic Voting 2012 (EVOTE2012)*, pages 315–329, Bonn, 2012. Gesellschaft für Informatik e.V.

[13] Josh Benaloh. Simple verifiable elections. In *2006 USENIX/ACCURATE Electronic Voting Technology Workshop*, volume 6, 2006.

[14] Josh Benaloh. Ballot casting assurance via voter-initiated poll station auditing. In *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, EVT'07, page 14, USA, 2007. USENIX Association.

[15] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '94, pages 544–553, New York, NY, USA, 1994. Association for Computing Machinery. https://doi.org/10.1145/195058.195407.

[16] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 626–643. Springer, 2012.

[17] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, 1988. https://doi.org/10.1016/0022-0000(88)90005-0.

[18] Bundeskanzlei. Chronik. https://www.bk.admin.ch/bk/de/home/politische-rechte/e-voting/chronik.html. Accessed: 28.10.2020.

[19] David Burkett. Digitisation and Digitalisation: What Means What? https://workingmouse.com.au/innovation/digitisation-digitalisation-digital-transformation, December 19. 2017. Accessed: 28.10.2020.

[20] Véronique Cortier, Jannik Dreier, Pierrick Gaudry, and Mathieu Turuani. A simple alternative to benaloh challenge for the cast-as-intended property in helios/belenios, November 2019. https://hal.inria.fr/hal-02346420.

[21] Véronique Cortier, Fabienne Eigner, Steve Kremer, Matteo Maffei, and Cyrille Wiedling. Type-based verification of electronic voting protocols. In *International Conference on Principles of Security and Trust*, pages 303–323. Springer, 2015. https://doi.org/10.1007/978-3-662-46666-7_16.

[22] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge proof systems. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 52–72. Springer, 1987.

[23] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In George Robert Blakley and David Chaum, editors, *Advances in Cryptology*, pages 10–18, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg. 978-3-540-39568-3.

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

[25] Adrienne Fichter. Passwort: "Wahlen". `https://www.republik.ch/2020/09/25/passwort-wahlen`, September 25. 2012. Accessed: 28.10.2020.

[26] Rojan Gharadaghy and Melanie Volkamer. Verifiability in electronic voting-explanations for non security experts. In *4th International Conference on Electronic Voting 2010*. Gesellschaft für Informatik eV, 2010.

[27] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 291–304, New York, NY, USA, 1985. Association for Computing Machinery. `https://doi.org/10.1145/22145.22178`.

[28] Sandra Guasch and Paz Morillo. How to challenge and cast your e-vote. In *Financial Cryptography and Data Security*, pages 130–145, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg. `https://doi.org/10.1007/978-3-662-54970-4_8`.

[29] Sandra Guasch Castelló. *Individual verifiability in electronic voting*. PhD thesis, Universitat Politècnica de Catalunya, Feb. 2016.

[30] Feng Hao. Schnorr Non-interactive Zero-Knowledge Proof. `https://tools.ietf.org/html/rfc8235`. Accessed: 27.02.2021.

[31] Patrick Hayes. benaloh challenge. `https://crates.io/crates/benaloh-challenge`. Accessed: 23.02.2021.

[32] Sven Heiberg, Arnis Parsovs, and Jan Willemson. Log analysis of estonian internet voting 2013-2014. In *E-Voting and Identity*, pages 19–34, Berlin, Heidelberg, 2015. Springer-Verlag. doi: 10.1007/978-3-319-22270-7_2.

[33] Sven Heiberg and Jan Willemson. Verifiable internet voting in estonia. *2014 6th International Conference on Electronic Voting: Verifying the Vote, EVOTE 2014 - IEEE Proceedings*, 01 2015. doi: 10.1109/EVOTE.2014.7001135.

[34] Alexander Hofmann. Security analysis and improvements of a blockchain-based remote electronic voting system, November 2020. University of Zurich.

[35] Arcitura Education Inc. Trust boundary. `https://patterns.arcitura.com/cloud-computing-patterns/basics/roles-and-boundaries/trust_boundary`. Accessed: 13.04.2021.

[36] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, 1996. `https://doi.org/10.1007/3-540-68339-9_13`.

[37] Hugo Jonker, Sjouke Mauw, and Jun Pang. Privacy and verifiability in voting systems: Methods, developments and trends. *Computer Science Review*, 10:1–30, 09 2018.

[38] Fatih Karayumak, Maina M Olembo, Michaela Kauer, and Melanie Volkamer. Usability analysis of helios-an open source verifiable remote electronic voting system. *EVT/WOTE*, 11(5), 2011.

[39] Mojtaba Khalili, Mohammad Dakhilalian, and Willy Susilo. Efficient chameleon hash functions in the enhanced collision resistant model. *Information Sciences*, 510:155 – 164, 2020. https://doi.org/10.1016/j.ins.2019.09.001.

[40] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. End-to-end verifiable elections in the standard model. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 468–498. Springer, 2015. https://doi.org/10.1007/978-3-662-46803-6_16.

[41] Christian Killer, Bruno Rodrigues, Raphael Matile, Eder Scheid, and Burkhard Stiller. Design and implementation of cast-as-intended verifiability for a blockchain-based voting system. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, SAC '20, pages 286–293, New York, NY, USA, 2020. Association for Computing Machinery. https://doi.org/10.1145/3341105.3373884.

[42] Christian Killer, Bruno Rodrigues, Eder Scheid, Muriel Franco, Moriz Eck, Nik Zaugg, Alex Scheitlin, and Burkhard Stiller. Provotum: A blockchain-based and end-to-end verifiable remote electronic voting system. *IEEE 45th Conference on Local Computer Networks (LCN)*, pages 1–12, November 2020. Sydney, Australia.

[43] Loren Kohnfelder and Praerit Garg. The threats to our products. *Microsoft Interface, Microsoft Corporation*, 33, 1999.

[44] Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *Computer Security – ESORICS 2010*, pages 389–404, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[45] Robert Krimmer, Stefan Triessnig, and Melanie Volkamer. The development of remote e-voting around the world: A review of roads and directions. In Ammar Alkassar and Melanie Volkamer, editors, *E-Voting and Identity*, pages 1–15, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-77493-8_1.

[46] Caroline Kudla and Kenneth G Paterson. Non-interactive designated verifier proofs and undeniable signatures. In *IMA International Conference on Cryptography and Coding*, pages 136–154. Springer, 2005.

[47] Mirko Kuhr. 10 Gründe für Progressive-Web-Apps. https://www.new-communication.de/neues/detail/10-gruende-fuer-progressive-web-apps, June 27. 2019. Accessed: 13.03.2021.

[48] Oksana Kulyk, Jan Henzel, Karen Renaud, and Melanie Volkamer. Comparing "challenge-based" and "code-based" internet voting verification implementations. In *Human-Computer Interaction – INTERACT 2019*, pages 519–538, Cham, 2019. Springer International Publishing. https://doi.org/10.1007/978-3-030-29381-9_32.

[49] R. Küsters, J. Müller, E. Scapin, and T. Truderung. sElect: A Lightweight Verifiable Remote Voting System. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 341–354, 2016. doi: 10.1109/CSF.2016.31.

[50] R. Küsters, T. Truderung, and A. Vogt. A game-based definition of coercion-resistance and its applications. In *2010 23rd IEEE Computer Security Foundations Symposium*, pages 122–136, 2010. doi: 10.1109/CSF.2010.16.

[51] R. Küsters, T. Truderung, and A. Vogt. Verifiability, privacy, and coercion-resistance: New insights from a case study. In *2011 IEEE Symposium on Security and Privacy*, pages 538–553, 2011. doi: 10.1109/SP.2011.21.

[52] R. Küsters, T. Truderung, and A. Vogt. Clash attacks on the verifiability of e-voting systems. In *2012 IEEE Symposium on Security and Privacy*, pages 395–409, 2012. doi: 10.1109/SP.2012.32.

[53] Ralf Küsters and Johannes Müller. Cryptographic security analysis of e-voting systems: Achievements, misconceptions, and limitations. In *International Joint Conference on Electronic Voting*, pages 21–41. Springer, 2017. `https://doi.org/10.1007/978-3-319-68687-5_2`.

[54] Soonhak Kwon and Aaram Yun. *Information Security and Cryptology-ICISC 2015: 18th International Conference, Seoul, South Korea, November 25-27, 2015, Revised Selected Papers*, volume 9558. Springer, 2016.

[55] Stefan Luber and Peter Schmitz. Definition Air Gap: Was ist Air Gap? `https://www.security-insider.de/was-ist-air-gap-a-933764/`, April 15. 2020. Accessed: 21.03.2021.

[56] Karola Marky, Oksana Kulyk, Karen Renaud, and Melanie Volkamer. *What Did I Really Vote For? On the Usability of Verifiable E-Voting Schemes*, pages 1–13. Association for Computing Machinery, New York, NY, USA, 2018.

[57] Karola Marky, Oksana Kulyk, and Melanie Volkamer. Comparative usability evaluation of cast-as-intended verification approaches in internet voting. *SICHERHEIT 2018*, 2018.

[58] R. Matile, B. Rodrigues, E. Scheid, and B. Stiller. Caiv: Cast-as-intended verifiability in blockchain-based voting. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 24–28, 2019. doi: 10.1109/BLOC.2019.8751413.

[59] Raphael Matile and Christian Killer. Privacy, verifiability, and auditability in blockchain-based e-voting. `https://files.ifi.uzh.ch/CSG/staff/rodrigues/extern/theses/mp-raphael-christian.pdf`, April 4. 2018. Accessed: 28.10.2020.

[60] Andrew C. Neff. Practical high certainty intent verification for encrypted votes, 2004.

[61] Domhnall O'Sullivan. Die Schweiz wird zum Briefwahl-Paradies. `https://www.swissinfo.ch/ger/direkte-demokratie_die-schweiz-wird-zum-briefwahl-paradies/46073018`. Accessed: 31.01.2021.

[62] ACE Project. Country experiences with electronic voting. http://aceproject.org/electoral-advice/archive/questions/replies/410523171, August 1. 2013. Accessed: 4.11.2020.

[63] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 393–403. Springer, 1995. https://doi.org/10.1007/3-540-49264-X_32.

[64] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989. https://doi.org/10.1007/0-387-34805-0_22.

[65] Vincent De Schutter. Deep dive into the security of Progressive Web Apps. https://blog.nviso.eu/2020/01/16/deep-dive-into-the-security-of-progressive-web-apps/, January 16. 2020. Accessed: 17.03.2021.

[66] Help Net Security. How secure are open source libraries? https://www.helpnetsecurity.com/2020/05/21/secure-open-source-libraries/, May 21. 2020. Accessed: 25.03.2021.

[67] Adam Shostack. *Threat modeling: Designing for security*. John Wiley & Sons, 2014.

[68] Nigel P Smart. *Cryptography made simple*. Springer, Cham, 2016. doi: 10.1007/978-3-319-21936-3.

[69] Ben Smyth, Steven Frink, and Michael R. Clarkson. Election verifiability: Cryptographic definitions and an analysis of helios, helios-c, and jcj. Cryptology ePrint Archive, Report 2015/233, 2015. https://eprint.iacr.org/2015/233.

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

[71] Timothy. Making a Progressive Web App. https://create-react-app.dev/docs/making-a-progressive-web-app/. Accessed: 14.03.2021.

[72] Melanie Volkamer and Robert Krimmer. Die online-wahl auf dem weg zum durchbruch. *Informatik-Spektrum*, 29(2):98–113, 2006. doi: 10.1007/s00287-006-0064-1.

[73] Scytl Secure Electronic Voting. Swiss Online Voting Protocol. https://www.post.ch/-/media/post/evoting/dokumente/swiss-post-online-voting-protocol.pdf?la=en, November 2017. Accessed: 17.02.2021.

[74] Jordan Wilkie. America's new voting machines bring new fears of election tampering. https://www.theguardian.com/us-news/2019/apr/22/us-voting-machines-paper-ballots-2020-hacking, April 22. 2019. Accessed: 28.10.2020.

[75] S. Yilmaz and I. Sertkaya. Improving the individual verification of estonian internet voting scheme. In *2020 International Conference on Information Security and Cryptology (ISCTURKEY)*, pages 38–47, 2020. doi: 10.1109/ISCTURKEY51113.2020.9308016.

Abbreviations

CSG	Communication Systems Research Group
DKG	Distributed Key Generation
DoS	Denial-of-Service
E2E	End-to-End
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IfI	Institut für Informatik, Departement of Informatics
NFC	Near Field Communication
NIZKP	Non-interactive zero knowledge proof
NIZKPK	Non-interactive zero-knowledge proof of knowledge
PBB	Public Bulletin Board
PWA	Progressive Web App
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial-of-Service, and Elevation of Privilege
TTP	Trusted Third Party
UZH	University of Zurich
QR-Code	Quick-Response-Code
ZKP	Zero-Knowledge Proof

Glossary

Air-gap An air-gap occurs when two devices are physically and logically separated [55]. It satisfies the highest safety and security requirements [55].

Cast-as-Intended Cast-as-Intended ensures that encrypted ballot truly contains intended voting selections, and thus, a ballot is cast as the voter intended [8, 28, 29]. If this property is met, a corrupt/malfunctioning voting device is unable to cast a vote containing a different voting option without being detected by the voter [29, 58].

Clash Attack A clash attack is an attack where malfunctioning voting devices or dishonest authorities can cast manipulated ballots since the same receipt is shown to different voters who selected the same voting options [53]. Thus, clash attacks are attacks where malfunctioning devices provide different voters with the same receipt [52].

Countes-as-Recorded Counted-as-Recorded ensures that previously received and recorded ballots are counted correctly [8]. Furthermore, it ensures that no accepted ballot is uncounted or unaccepted ballots are counted [53].

Counter-strategy A counter-strategy is a strategy that can be applied by a coerced voter to defend himself/herself against coercion [51]. The goal of such a strategy is that the coerced voter can vote for the intended choice instead of the one ordered by a coercer [51].

Designated Verifier Proof A Designated Verifier Proof is a proof where only the designated, selected verifier can verify the proof [36]. Thus, only the specified, designated verifier can be convinced of a proof [36].

Digitalisation Digitalisation is the process used to enhance business processes with the help of digitisation [19].

Digitisation Digitisation is the process of converting physical, analog data and information to digital one [19].

Localhost Localhost is a domain name used for the local development of web-based applications and websites. Usually, it is accessible on the IPv4-Address 127.0.0.1.

Public Bulletin Board A Public Bulletin Board (PBB) is an authenticated immutable append-only channel that provides transparency and verifiability in eVoting systems [34]. Furthermore, it is public and provides a consistent view to all entities looking at it [34].

Recorded-as-Cast Recorded-as-Cast allows voters to verify whether their vote is recorded in the ballot box correctly or not [8, 29].

Randomizer Entity which randomizes and thus blinds encrypted ballots [34]. This entity is used in Provotum 3.0 [34].

STRIDE The STRIDE threat modeling approach was brought up in 1999 by Loren Kohnfelder and Praerit Garg to model specific threats [43]. STRIDE stands for Spoofing, Tampering, Repudiation, Information Disclosure, Denial-of-Service, and Elevation of Privilege [43].

Trust Boundary A trust boundary is a boundary, where a user's level of trust in an application, a system or hardware changes [35].

Trust Domain A trust domain is an abstract area with the same level of trust or a collection of devices a voter has equal confidence.

List of Figures

List of Tables

List of Listings

Appendix A

Installation Guidelines

The easiest possibility to run and test the verification mechanism is to follow the instructions listed under **Infrastructure**. How the source code can be accessed is listed in the **Source Code** section.

Infrastructure

Installing the components in a working environment can be done with the help of the *Infrastructure* repository in GitHub. This contains all needed docker containers, settings, and instructions such that straightforward installation and usage are possible.
Infrastructure: `https://github.com/provotum/Provotum-Infrastructure-Fabio`

This repository contains two voter front ends. These run on *localhost:9000* and *localhost:9001* if set up with the *infrastructure* repository. The one running at port *9000* is trustworthy, meaning it encrypts the vote indeed selected by the voter, while the one running on port *9001* is malfunctioning. The second one is only for demonstration purposes and should never be used in a production environment since this front end alters the selected vote to the opposite.

Since the verifier application must be running on a second device, it is hosted on Firebase at the time of writing. Alternatively, the *Docker-Compose* fires up the verifier application at port *localhost:9002*, despite scanning the QR-codes is not possible due to the verifier and the voting front end running on the same device.
Verifier application: `https://provotum-vote-verifier-app.web.app/`

Source Code

The implementation was done with the usage of GitHub. Therefore, accessing the source code is most straightforward with the below-presented URLs. Also, there are instructions on how to install the specific part of the system.
Voting front end: `https://github.com/febe19/voter`
Verifier application: `https://github.com/febe19/provotum-vote-verifier-app`

Appendix B

Contents of the CD

- Thesis:
  Voting-Verification-Mechanism-for-a-Distributed-Ledger-based-Remote-Electronic-Voting-System_Fabio-Maddaloni.pdf
- Thesis Latex source code folder:
  Voting-Verification-Mechanism-for-a-Distributed-Ledger-based-Remote-Electronic-Voting-System_Fabio-Maddaloni.zip
- Mid-Term Presentation:
  MidTermPresentation_Fabio-Maddaloni.pptx
- Folder with figures used in thesis:
  Figures
- Source Code:
  - Provotum-Vote-Verifier-App:
    provotum-vote-verifier-app
  - Voter:
    voter
  - Infrastructure:
    Provotum-Infrastructure-Fabio
- Abstract and Zusammenfassung
  Abstract.txt & Zusfsg.txt