Institute of Information Security

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Individual Verifiability in the Polyas E-Voting System

Oliver Koppenhöfer

Course of Study:	Informatik
Examiner:	Prof. Dr. Ralf Küsters
Supervisor:	Nicolas Huber, M.Sc.
Commenced:	1. March 2023
Completed:	1. September 2023

Abstract

The demand for secure and accessible voting mechanisms has led to the exploration of electronic voting (e-voting) systems. Among these, end-to-end verifiable e-voting systems have emerged as a promising solution, ensuring transparency, integrity, and privacy throughout the voting process. This thesis delves into the realm of end-to-end verifiable e-voting, with a particular emphasis on the Polyas e-voting system and its method for individual verifiability.

Polyas is a German company developing a secure and verifiable remote e-voting system of the same name. They have already held elections for churches, political parties and other organizations like the GI, a German organization concerned with computer science. The GI holds an election soon which is of particular interest as the tool for individual verifiability in the Polyas e-voting system, which was developed in the scope of this thesis, will first be used by them.

Successful use of the tool assures voters that, as long as either their voting device or audit device is honest, their ballot will be accurately included in the final tally as they intended. The tool and the protocol it implements are discussed in-depth.

Contents

List of Figures

1 Introduction

1.1 Motivation

An election for a new student council president is probably not at risk of large-scale voter manipulation, and even if it were, not many people outside the school would care. Therefore, an election where a server is fully trusted to provide secrecy and integrity might be acceptable. On large political elections, on the other hand, ride billions. In the US "Political spending in the 2020 election totaled $14.4 billion" making "the 2020 election the most expensive of all time by a large margin" [9]. This is a drastic example but it should be kept in mind that even for smaller political elections there may be a huge incentive for some to invest in its manipulation, in larger ones, like the US presidential election, entire countries may try to sabotage it. In classical elections, every voter sees their ballot enter the ballot box and trusts that when the votes are counted afterwards no single person has the opportunity to hide or otherwise manipulate votes as multiple people of different sides will always look over each other's shoulders. To provide a similar level of integrity to e-voting there has been a lot of research in cryptographic verifiability and modern methods do a great job at securing elections. This lends credibility to the election results and makes it harder for mistrust to spread in the public. This thesis is especially concerned with individual verifiability, which answers an important question that a voter might have: Has my vote actually been counted?

1.2 Outline

Exploring the world of e-voting systems is the focus of chapter 2. It starts by explaining some of the desired properties of e-voting systems, including privacy, verifiability, and coercion resistance. Furthermore, the chapter explores two e-voting systems. Helios and Polyas, which is based on Helios, are introduced.

Verifiability takes center stage in chapter 3. The chapter begins by establishing definitions related to verifiability in e-voting systems. It then delves into the concept of individual verifiability, which is the main part of this thesis, explaining its definition and how it

can be achieved by implementing cast-as-intended verifiability and tallied-as-recorded verifiability. The chapter also covers the concepts of universal verifiability and eligibility verifiability.

The practical part of this thesis along with the theory behind it is detailed in chapter 4. It is a website implementing cast-as-intended verifiability for the Polyas E-Voting system.

In chapter 5 the previously introduced properties of E-Voting systems are investigated inside the Polyas e-voting system. The chapter explores how Polyas addresses these properties. The claim of end-to-end verifiability is briefly discussed

Finally, chapter 6 summarizes the main findings and provides a conclusion to the entire body of work while chapter 7 gives an outlook that brings the thesis to a close.

2 E-Voting Systems

2.1 Desired Properties

For a decent programmer, at first glance, constructing an E-Voting system does not seem too hard. A client sends votes to a server which then counts them and spits out a result. It only becomes hard once the system is held to a higher standard and thus, must provide some properties. Ideally these properties hold in a system, even if some of its components are not trustworthy. This can be partially achieved by separation of concerns.

2.1.1 Privacy

This is the first property that comes to mind. It seems to be trivially solved by using https like any decent Webserver, however, we would also like the E-Voting server to stay clueless about the voters decisions.

2.1.2 Accountability

Intuitively accountability means that it should be possible to detect misbehavior when the election has been corrupted and trace it back to the culprits. While this is nice to have few ,if any, e-voting systems can guarantee it.

2.1.3 Verifiability

Verifiability is the idea that individuals or independent auditors are capable of verifying that the election result corresponds to the will of the voters.

2.1.4 Coercion Resistance

For an election to be fair, it must be impossible for bad actors to threaten or bribe someone to influence their vote. In physical voting this is done by having private voting booths.

2.1.5 Receipt-freeness

Coercion resistance is hard to achieve, receipt-freeness relaxes the requirements. The voting system simply must not provide information that would allow a voter to proof to anyone how they voted. If someone wants to pay for a vote they must rely on an honer system without any hard proofs.

2.1.6 Current State

Modern cryptographic methods can be used to implement systems that provide these properties to a varying degree, as some of the formal definitions are not even agreed upon or can be measured instead of being an all or nothing deal. It is also plain to see that designing such a system is a challenging task and that they should be analyzed thoroughly, both in theory and in there implementation. Such a discussion is included in chapter 4 for Polyas' individual verification mechanism and its implementation which constitutes the practical part of this thesis.

2.2 Helios

Helios is an open-source, web-based electronic voting (e-voting) system that uses modern cryptography to ensure auditability of the election process, and thus, integrity. The following description is based on the alpha version of Helios and the paper which was released with it at the time[1].

The first major component of Helios is the server including the ballot preparation system (BPS), a public bulletin board containing a list of voter identifiers and their encrypted ballots and an authentication mechanism. The second major component is the web-based voting client.

Voters first navigate to the client URL and indicate their choice, then the BPS encrypts it and displays the hash of the encryption. Next, the voter can audit the ballot as often as they please and then, finally, authenticate and publish the ballot on the public

bulletin board. The late authentication allows auditing to be done by everyone, not only individuals that are eligible to vote. Every time a ballot is audited, the BPS displays the ciphertext and the randomness used so that the voter can verify that encryption and hashing were done correctly. As the voting client does not know whether the ballot will be audited, it would risk detection by tinkering with the vote. After the audit, voting has to be repeated, possibly with a different choice so that no receipt is generated. This auditing step will be mentioned again later in the context of individual verification.

Once the election is over, the election server shuffles, re-randomizes and only then decrypts the votes. Once ballots are decrypted, tallying is trivial and auditors can compute their own result and compare it to that of Helios. Importantly, auditors also need to check the proofs of correctness that Helios must produce for the shuffling and decryption to make sure the decrypted votes accurately represent the encrypted votes stored in the public bulletin board. These checks fit in the scope of universal verification.

Helios provides a high level of verifiability through its auditing capabilities. It is however not coercion resistant and, to make this absolutely clear, even has a "coerce me" button on its website. It is also only private if Helios is assumed to be trustworthy, as the voting server has the private election key and could use it at any time to decrypt votes.

The coerce me button allows a voter to send a receipt of a ballot (randomness, plaintext and ciphertext) to a coercer and then cast that same ballot so that the coercer can find the ballot on the public bulletin board. This apparently 'highlights the difference between a coercion-free auditing process that could potentially be used with in-person voting, and the inherent coercibility of online-only voting which is made more explicit with the "Coerce Me!" button' [1].

2.3 Polyas

As said before, Polyas is a German company that has carried out electronic elections for multiple big organizations. One example is the CDU, a German political party, which elected their leader via Polyas in 2021. Interestingly, due to the "Political Parties Act" the electronic election alone was not binding. All three candidates had agreed, however, that the electronic vote would count and that only the winner would then be electable in the legally binding postal vote which was held afterwards [14]. A nifty legal trick was required, nevertheless, Polyas has already been used in a big political election.

As this thesis is about the Polyas E-Voting System, its setup, voting and verification flow will be explained in more detail as it was the case for Helios. Still, this is just meant

to give an overview of the system. Some details for the implementation of individual verifiability are presented in chapter 4. This section is entirely based on [12].

When an organization wants to hold an election with the Polyas remote e-voting system, they should set up an Election Council. This council is basically the point of contact for Polyas and gets to make decisions about the election. It has three main tasks:

1. Choose trusted Election Administrators, Registrar and Distribution Facility. Together with the voters, Polyas and potential external auditors these are the main participants.
2. Create a list of eligible voters. This should include some kind of address for every voter.
3. Audit data provided by Polyas. The implementation of the tools necessary or even the whole process can be outsourced to trusted external auditors, e.g. to universities.

Task 3 is all about verification and will be handled in the following chapters. Task 2 should be clear. As for the entities introduced in task 1 their responsibilities will become clear throughout the following explanation of a Polyas remote e-voting election process.

An election is conducted in three phases: registration, voting and tallying phase. Auditing mostly happens after the election.

After the Election Council has finished tasks 1 and 2 the registration phase, and thus the election, can begin. First, the list of eligible voters and their addresses is handed to the Registrar. The Registrar then sets up a signing key pair for each Election Administrator and the voter credentials. The credentials basically just consist of a random password which is stored along with the voter's identifier in a distribution file. From these passwords, however, an asymmetric key pair can be derived deterministically. The Registrar does this for each voter. The encrypted distribution file is handed to the distribution facility, which decrypts it again and distributes the voter's login data along with a link to the Voting Client according to the addresses given. Another file containing identifiers, public key's and a hash derived from the password should be sent to the Election Council, checked by them, and then uploaded to Polyas Voting Server. The server will then upload the voter's public keys to a public board called registry. The hash is used for the login functionality of the Voting Client. If Polyas is chosen as the Registrar the server does these things automatically along with other tasks like creating the election key pair. If not, Polyas has written a credential generation tool that can be used and is also working to have one independently implemented.

Once the basic setup is done, the election can enter the voting phase. There are three pieces of software in use in this phase. The Election Server and the Voting Client are both

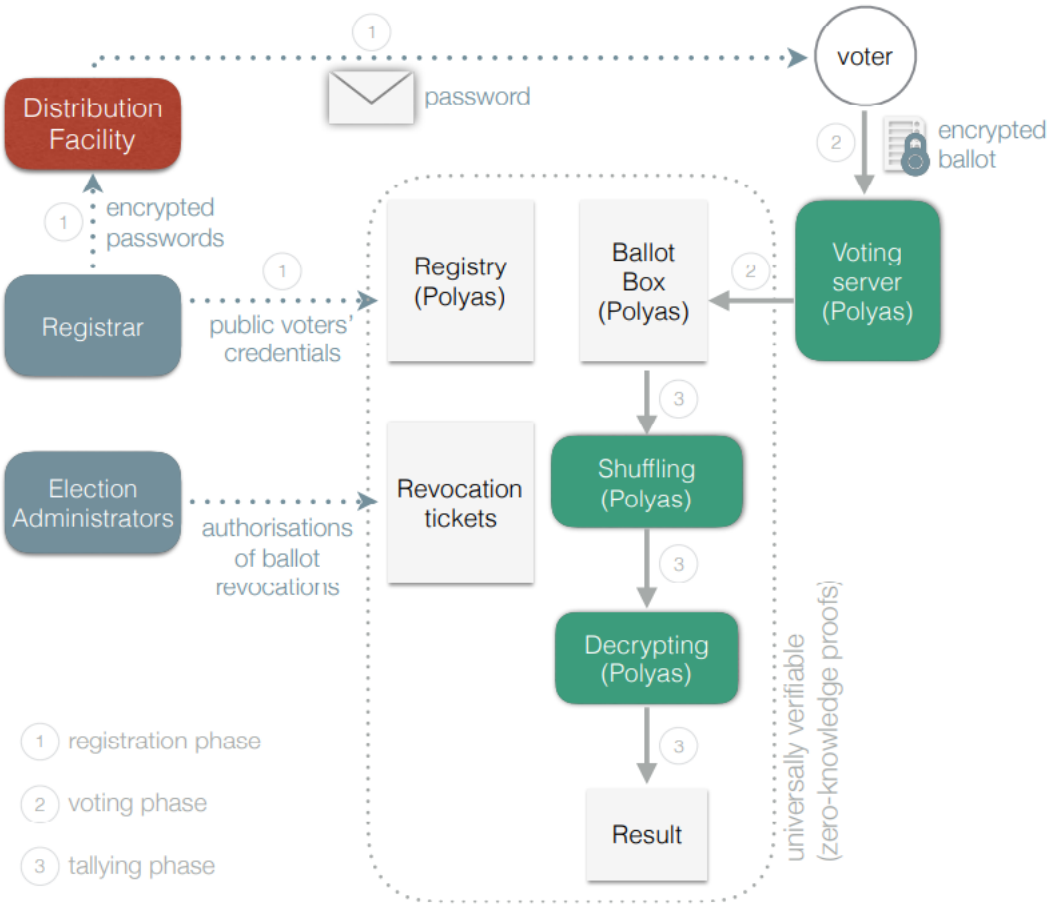

Figure 2.1: The structure of the system. Gray boxes represent bulletin boards. Ballot box is a bulletin board, where submitted ballots are published. Although not indicated on the picture, the intermediate results of shuffling (mixing) are also posted on bulletin boards and hence are subject to audit. In particular, verifiable shuffling and verifiable decryption produce zero-knowledge proofs of correctness of these operations. Source: [12]

implemented and hosted by Polyas. The third is the Second Device App, an individual verification tool that can be used by voters to verify that the Voting Client and Election Server did not drop or manipulate their ballot. Neither the setup of the Second Device App in the election nor the use by voters is mandatory. The app has its own chapter 4 and thus will not be further described here.

From the point of view of a voter, the voting process is easy. Once they received a letter from the distribution facility containing a link to the Voting Client as well as a password, they follow the link and log in with the Election Server. Then they fill out their ballot as they wish and cast it. Afterwards they are given the option to perform individual verification by scanning a QR-code and following the process from there.

Behind the scenes, the Voting Client uses the password the voter entered to deterministically derive both the voter's private key and the value necessary for logging in, just like the Registrar did in the last phase. It then logs in and, once the form has been submitted, encrypts the ballot using the public election key and signs it with the computed private key before sending it to the server. In the server's response should be the ballot cast confirmation, this is basically just a signature from the server on the ballot made with an extra signing key pair. This confirmation is important for tallied-as-recorded verifiability and should be handed to the election council.

The server should check the validity of the ballot and mark it invalid in case it is not well-formed, for instance. Additionally, the server checks the signature, this will also be done by the auditors after the election to prevent ballot stuffing. It also creates the ballot cast confirmation and sends it to the client. All ballots, including invalid ones, are put on the public bulletin board. Invalid ballots are included for transparency reasons. Ballot stuffing occurs when votes are added to the public bulletin board that were not cast by any eligible voter, or if there are multiple valid non-revoked votes by a single voter.

During this phase, it might happen that ballots are revoked. One reason for this could be that some voters should not have been on the Election Council's list of eligible voters. In such a case, a revocation ticket is uploaded on the Election Server's public board 'revocations'. For a revocation ticket to become valid, a certain number of Election Administrators have to publish their signatures on the ticket.

The last phase is the tallying phase, and it can start as soon as the voting phase's time has run out. Intermediate results produced during the tally are stored on public read-only boards so that auditing can take place afterwards. Valid non-revoked ballots are counted. To guarantee both voter privacy and verifiability ballots are first anonymized and decrypted afterwards.

The system is a lot more difficult to understand than a trivial voting system would be, so what is gained from the additional complexity? "The proposed system has the potential

to provide a whole range of state-of-the-art security features, including ballot privacy and end-to-end verifiability" [12]. The claim regarding end-to-end verifiability is crucial here. For ballot privacy, one might think that more conventional security approaches, such as encrypting the contents of your database—utilizing features like transparent encryption within the database system [3]—and employing robust encryption protocols like HTTPS, would yield a similar level of privacy. But Polyas supports threshold decryption meaning that multiple parties share a secret key and that no single party can decrypt ciphertexts. Even when there are no other parties, trusting Polyas for privacy is very reasonable and integrity and trustworthiness of elections, as provided by a well set up end-to-end verifiable election, are crucial for any democratic entity.

The system should be receipt-free unless the randomness used to encrypt ballots is known to the voter. It should be said, however, that encryption does happen on the client side so a good hacker could theoretically always watch the RAM carefully and try to extract the value. Even in a computer science community, such as the GI, this seems incredibly unlikely though and would hardly warrant much concern.

Unlike Helios, Polyas does not offer a "Coerce me!" button, nevertheless the remote e-voting system can not be called coercion resistant. Once a ballot is cast it cannot be altered by anyone so voters could always be forced to vote one way or the other by gunpoint. While revocations are possible, they are also public and so revoking a ballot in such a case might not be a comfortable option.

As for accountability, this property has not been mentioned as a goal for the Polyas E-Voting System in any of the used references. Still it can be discussed a bit. Practically speaking, proofs have to be provided for the tally by the server. If it cannot there is only really one component to blame. If, on the other hand, an auditor accuses the server even though the proofs are valid, this auditor would be to blame. The individual verification protocol also allows blame to be put on the Election Server in case ballots are silently dropped.

3 Verifiability

3.1 Definition

To make absolutely clear what the goal here is, at least one formal definition of verifiability should be provided. In [7] verifiability is proven to be a special case of accountability after a symbolic and a computational definition are established. The symbolic definition is introduced in the following.

For the complete formal definition of all components of the symbolic verifiability definition the original paper should be read. Here only the verifiability definition along with some explanations is given. To understand the definition some other terms should be introduced first:

A protocol P has agents Σ who run programs. These programs make up an instance of a protocol and the instances have multiple runs due to randomness and possibly different starting conditions. An agent $A \in \Sigma$ is honest, denoted by $hon(A)$, in an instance π and therefore in all its runs depending on the program A executes. If the program strictly follows its part of P then A is honest using it. Protocols have properties. These do not have to be met in every run, instead a property is defined as the set of runs in which it is met. Lastly, F_{hon} is the set of all "positive boolean formula over propositions of the form hon(a)" [7]. For a formula $\psi \in waq1F_{hon}$, $\pi \vDash \psi$ holds if the necessary agents, indicated by ψ, are honest in the instance π.

Definition 3.1.1
Symbolic verifiability
Let P be a protocol with the set of agents Σ. Let $J \in \Sigma$, $\psi \in F_{hon}$, and γ be a property of P. Then, we say that the goal γ is guaranteed in P by ψ and verifiable by J if the following conditions are satisfied:
(i) For every run r of an instance π of P such that $\pi \vDash \psi$, the agent J accepts r.
(ii) For every run r of an instance of P in which J accepts r, it holds that $r \in \gamma$.

In this definition J could be called judge as its about J making a decision in a run, will they accept or not? J could be any of the participant's, in E-Voting J could be just a regular voter. If the system is verifiable it likely provides mechanisms so that a competent

judge can realize when the property they care about holds. This is one reason why the definition does not just say: The judge accepts if and only if the property is achieved. As an example, in E-Voting, the server might perform the tally perfectly but not provide any proof, then the goal would be reached but the auditors would not accept.

It should also be noticed that if $\pi \vDash \psi$, so a 'correct' group of participants is honest in protocol instance π, then J accepts in all runs of π because of i. So due to ii, the property γ is always achieved in this instance. So some groups of participants can guarantee that the goal is met if they are honest. With everyone running honest programs, i.e.

$$\psi = \bigwedge_{a \in \Sigma} hon(a)$$

, the judge should probably accept, otherwise the Protocol itself would likely not be sensible. This is not applicable in E-Voting as, at the very least, the voters should be assumed to be dishonest. It is important to understand that the formula ψ is not an indicator for the judge J of whom to trust, and therefore not check their work. The formula indicates which components can guarantee the property, that does not mean they actually will.

The symbolic verifiability definition is a broad one which could be applied to many scenarios such as contract signing and auctions. Even when considering only E-Voting, the goal γ and formula ψ can be chosen to capture different scopes and parts of the election framework.

In E-Voting, there are different kinds of verifiability. Global or end-to-end verifiability is what Helios and Polyas strive for. To capture it with the introduced definition, the goal proposed in [7] is the set that contains all runs in which the election result correctly corresponds to the votes cast by honest eligible voters only, where each eligible voter casts at most one ballot. The reason why the goal only cares about honest voters votes being counted is that dishonest voters might not execute their part of the verification protocol, or they might have made a mistake in the voting process which causes their ballot to be disqualified. In classical elections, ballots with markings are often times disqualified as they might be used to identify a voter.

In systems that have only one server like Polyas and Helios, this server can ruin the election easily if they behave dishonestly, e.g. just shut down. Fooling the judge and secretly manipulating the election, however, should not be possible. Sadly, the symbolic definition is impractical, making tinkering impossible is too much to ask for. The authors extend the definition to a computational one which only sets probabilistic requirements, making attacks extremely computationally expensive, as usual in modern cryptography.

Often times an e-voting system is called end-to-end verifiable if it offers individual, universal and, sometimes, eligibility verifiability. The latter ensures that at most a

single vote per eligible voter is tallied. Individual verifiability lets individual voters check that their encrypted ballot was actually put on the public bulletin board and correctly represents their intended vote, which they indicated on the voting client. Lastly, universal verifiability asserts that the votes in the ballot box are counted correctly.

These 'smaller' verifiability notions can be expressed as goals and honesty formulas as well. This is, among other things, done in the following subsections.

3.2 Individual Verifiability

3.2.1 Definition

The goal, akin to the symbolic verifiability definition with a voter as judge J could be stated as "all runs in which J's voting intentions are correctly represented by a ballot in the ballot box". In most remote e-voting system's without special mechanisms in place, this goal could likely be guaranteed by the honesty of J, who must not make a formal mistake so that the ballot is not declared invalid, the Voting Client, who must encrypt the vote indicated by J, and the Election Server, who must put the ballot into the ballot box as received. Two approaches to loosen these trust requirements are explored in this thesis.

3.2.2 Overview

Just like end-to-end verifiability is often separated into individual and universal verifiability, individual verifiability can be separated into cast-as-intended and tallied-as-recorded verifiability. Putting the two together, individual Verifiability could also be called tallied-as-cast which sums up the intention pretty well. Tallying is done on the ballots that are in the ballot box, so the ballot has to be there the way it was handed in, tallied-as-recorded. And what was handed in, through an official or a web client in the remote e-voting case, should be as the voter intended.

As explained in the introduction to Helios, the voting process can be separated into ballot preparation and ballot casting. This distinction is not as relevant in Polyas, even so, it can help to think about them as two distinct actions. There are two approaches to cast-as-intended verifiability that will be considered in this thesis, cast-or-audit, used by Helios, and cast-and-audit, used by Polyas. The difference is that with the former approach, a prepared ballot, basically the voter's choice encrypted with the public election key, can be either audited or cast but never both. In cast-and-audit, this restriction does not hold. Polyas only offers auditing for ballots that were both prepared and cast.

3.2.3 Requirements

In [10] the authors, among them one of the developers of the Polyas E-Voting system, explain their protocol for individual verification and add some context. Individual verification protocols have a trade-off between security, usability and deployability. Therefore, the choice of which individual verifiability protocol to implement should depend on the election system, as some protocols might be hard to integrate, and the type of election and its requirements, as e.g. a student council election is an unlikely target for large-scale voter manipulation. Also, E-Voting is becoming more regulated, there are different standards that should be followed in e.g. Estonia and Germany concerning privacy but also (individual) verifiability.

Some properties by which to judge an individual verification protocol are

1. usability: The risk that Election Server and Voting Client take when dropping or manipulating ballots rises with the number of voters who execute the IV protocol successfully. To get voters to use the protocol the process must be quick, easy and have only reasonable additional requirements. The capability to scan a QR-Code with a second device is judged to be a reasonable requirement in this thesis, for instance.
2. receipt-freeness: The verification process must not provide the voter with additional proof of how they voted, otherwise this proof could be used by the voter to sell their vote to a coercer. This must also hold if the audit app misbehaves and does not follow the protocol.
3. monetary cost: Obviously no one likes spending money. Protocols could make the election process expensive if special hardware or additional infrastructure is required. This could be additional printing facilities or the maintenance of physical audit locations in non-remote E-Voting.
4. computational cost: The protocol should run on any smartphone within a reasonable time frame, otherwise voters might just quit halfway through. Back on the server side, the load should not be exploding dramatically when including IV though this might rather be said in the context of monetary cost, if the system is scalable.
5. privacy: The Election Server should not learn anything about the voters ballot that they did not know before. In the case of e.g. Polyas 3.0 this is not relevant as the ES is the sole owner of the private election key and could decrypt ballots at any time.

6. flexibility: There are several election types, e.g. Proportional Representation and Ranked Voting, that require different ballots. And even in one election type ballots may have complicated designs. Furthermore, it should be easy to integrate into many different e-voting protocols. Basically, the same protocol should be applicable to many scenarios.
7. minimal trust: Generally, in E-Voting the number of entities that have to be trusted in order for the election result to reflect the voters will is desired to be as low as possible. Some trust assumptions can be relaxed by having multiple entities implement a component. How good of an option that is depends on the concrete system and the component. In case of Polyas IV implementation this works really well.

3.2.4 Cast-Or-Audit approach

In [2] a method, the Benaloh challenge, is suggested that only requires a secure asymmetric encryption scheme E which offers verifiable decryption. This means, the system preparing a ballot must be able to prove that a ballot it encrypted actually decrypts to the voter's intended choice. As explained before, this prepared ballot that was audited may not be cast because the auditing leaves a voter with a receipt. Therefore, another ballot has to be prepared and then cast. Due to the random nature of any secure asymmetric encryption scheme, the cast ballot will differ from the audited one.

The preparation of a voter's choice for casting, basically encryption, is done by a ballot preparation system. Any such system "need not know the identities of voters using the device; it need not authenticate that users even have the right to vote; it need not limit voters to a single use; it need not record the encrypted ballots it creates; it need not have any remote communication abilities; and it need not be involved in the casting of ballots"[2]. Thus, to prepare their ballot, anyone, eligible or not, can be allowed to create and audit as many encrypted votes as they like. So long as the system has no accurate way to predict whether the next prepared ballot will be cast or audited, it is at risk of being discovered as dishonest each time it manipulates a ballot. Whether this happens due to e.g. corruption, malfunctioning or malicious intent is irrelevant.

While this thesis is primarily concerned with remote electronic voting, as Polyas fits into this category, it is also interesting to consider the cast-or-audit approach using two physically separated machines, one for ballot preparation and one for ballot casting, both located in public voting locations. In [2] it is suggested that the ballot preparation system could print the ciphertext on a "paper card with a magnetic stripe". To guarantee receipt-freeness the machine would add to the card either a proof of correct encryption or a digital signature, depending on whether the voter chooses to audit or cast their

ballot. Importantly, the vote casting system would have to reject ballots without a valid signature.

Helios uses a pretty simple version, first of all the voting client handles both ballot preparation and casting, thus no signature is needed. Once a voter has indicated their choice v, the voting client (VC) chooses randomness r and computes, using the public election key pk, either $c = E(pk, v; r)$ or $c = E(pk, v'; r)$ for some manipulated vote v'. VC then displays the Hash $h = H(c)$, with H being a secure collision resistant hash function. This hash serves as a commitment to the ciphertext. Now, if the voter chooses to audit their ballot, the VC should display r, but must display some r' to at least give the appearance of it following the protocol. Then the voter should compute $h' = H(E(pk, v; r'))$ and complain if not $h' = h$. Due to the collision resistance of H and $E(pk, v; r) \neq E(pk, x; y)$ for all $x \neq v$ and all y, the voter can be sure that the ciphertext to which VC committed itself was, in fact, a correct encryption of their vote. Of course, the VC can manipulate ballots and hope to get lucky, meaning that the voter chooses to cast instead of auditing. If the system allowed voters to cast and audit it would not be receipt-free because a voter could reveal (v, r, c) to a coercer. This coercer can check that $c = E(pk, v; r)$ and that c appears on the public bulletin board next to the voter's identifier. Thus, voters could sell their votes and provide receipts as proof that they kept their word.

As it is unreasonable to assume that a significant number of voters will perform such a check themselves, a Ballot Verifier System (BVS) is needed. The BVS should be independently implemented by a trusted entity and could be just another website, allowing a voter to open it in another browser tab and copying information via the clipboard. Because the BVS is given the plaintext vote used in encryption, a voter should not be likely to cast the same vote that they audit. Therefore, after auditing voters should be given a clean ballot instead of the VC filling it in automatically or even just re-encrypting the ballot immediately which would, doubtlessly, be much more convenient [11]. This depends, however, on the majority of voters knowing what they are doing as for an uninformed voter, filling in the ballot with choices that they do not like does not make sense.

3.2.5 Cast-And-Audit approach

In cast-and-audit methods, the voter does not have to decide between casting and auditing their ballot. One might think that this means the Voting Client can not opt to take a risk and manipulate a ballot like it can in cast-or-audit methods, but this is not true. Cast-and-audit methods may not force the voter to execute the cast-as-intended protocol and Polyas does not, so the percentage of voters that use the verification option is critical and might be very low.

Polyas protocol basically uses a second device to take a look at the ciphertext that is stored in the ballot box and makes sure it reflects the voter's choice. It was implemented in the scope of this thesis, so this section is rather short. Instead the implemented protocol is explained in chapter 4.

To briefly compare this method to the previous one. In both, a second application is required and ideally run on a different computer to prevent the risk of viruses. The Benaloh challenge does not trust the second device with ballot privacy, as the same ballot that was audited is not cast. This is also a weak point, because protection is only based on a high risk of detection for the Voting Client. Both protocols can be independently implemented and run on different devices so that the trust requirements become less relevant.

3.2.6 Tallied-As-Recorded Verifiability

While cast-as-intended verifiability should check that the ballot was cast, i.e. prepared and handed over, correctly, tallied-as-recorded verifiability checks that the same prepared ballot is actually located in the ballot box, which the final tally will use, and was not manipulated or just discarded by the Election Server.

In [10] the authors write that "the standard mechanism to achieve tallied-as-recorded verifiability" is "a public bulletin board and signed receipts". As this approach is rather easy to understand, widely used and the method of choice in the Polyas E-Voting System, it is described here. After the Voting Client VC sends the encrypted ballot c to the Election Server ES the following steps are taken.

1. ES signs c and sends the ballot cast confirmation to VC. The confirmation contains c and the signature s
2. VC checks that s is a valid signature on c and hands the confirmation to V
3. V hands the confirmation to a trusted auditor who checks that c was counted in the tally

Step three should be performed on the final public bulletin board as part of a universal verifiability process.

The signature on the ballot basically serves as a promise by ES, stating that it did put the encrypted ballot in the ballot box. In this analogy, an invalid signature would correspond to the ES crossing its fingers behind its back and a missing ballot in the ballot box to a broken promise. This misbehavior could be easily proven with the valid signature.

For this method to be effective, the voters have to know what to do with the ballot cast confirmation, otherwise many would not check the bulletin board or would even complain to the ES itself, which could then just fix the error before more knowledgeable parties learn of the event. It would do this by simply adding the ballot to the public bulletin board, thus destroying the proof of misbehavior. Ideally, the ballot cast confirmations would be sent to all universal verification applications automatically. Those would verify that all ballots are included in the final tally once the voting phase is over and the ES can not correct its potential manipulation attempts anymore.

3.3 Universal Verifiability

The goal of Universal Verifiability in the context of the Symbolic Verifiability definition introduced earlier would be the set that contains all runs in which the tally result exactly corresponds to the contents of the ballot box. There could be many ways to achieve some level of Universal Verifiability, in [5] two general approaches are stated.

The first approach is based on homomorphic encryption schemes. These allow computations to be performed on ciphertexts directly. Consider an encryption scheme that is homomorphic with respect to addition. With (pk, sk) as key pair and c_1, c_2 encrypted with pk this would mean that $D(sk, c_1 + c_2) = D(sk, c_1) + D(sk, c_2)$. A vote for the i-th candidate could then be a vector containing a one at position i and zeros everywhere else. The encrypted versions of these vectors can then be added up, and only the sum will be decrypted. Besides exceptionally small elections, there is then no way for auditors to know how individual voters voted from the encrypted votes and the decrypted result alone. Additionally, the component decrypting the result also has to provide a proof of correct decryption. Otherwise, it could just lie about the computed encrypted results contents.

In the second approach, encrypted votes are only decrypted after they have been anonymized. The anonymization is done by dropping any identifiers that were attached to the encrypted ballot in the ballot box and then shuffling the votes. Shuffling does not just randomize the order of the votes, it also re-encrypts them. Basically, if vote v was represented by its ciphertext c at position i of the public bulletin board, after shuffling, it is represented by c^* at position j where most likely $c^* \neq c$ and $i \neq j$. Proofs have to be provided here as well. These proofs ensure that each voter's unencrypted ballot, originally present within the ballot box prior to the shuffling process, continues to be represented by an encrypted ballot within the ballot box following the shuffling.

3.4 Eligibility Verifiability

For eligibility verifiability the goal is the set of runs in which all votes contributing to the election result originated from eligible voters as decided by the entity holding the election. Mostly the goal is achieved by the Election Server only accepting votes cast by authenticated voters, basically a login. ES now also needs to prove this fact to external judges. If there is a public key infrastructure in place for the eligible voters, then a simple solution is to require all encrypted ballots in the ballot box to be signed by the voters who cast them. The judge can then check whether all ballot signatures are valid. This is also the approach Polyas took [12]. This also prevents the server from tampering with the ballot, as it would lead to an invalid signature.

Note that the signatures proposed here and in the individual verifiability section are different. To verify eligibility a voter's signature is needed, for tallied-as-recorded the election server's signature can later serve as evidence of misconduct at the server side.

In [6] private eligibility verifiability is introduced. Here, the goal includes hiding which eligible voters did or did not vote. An approach for this is also given: A voter can have one real vote and multiple "null votes" on the public bulletin board. The ElGamal encryption scheme is used so that multiplying two ciphertexts also multiplies the plaintext. Null votes are all encryptions of the group element 1 so that then the final vote of a voter can be computed by a product over all their votes, as multiplying by 1 does not change the final outcome. If the outcome itself is just 1, then no vote was cast.

4 Second Device App

4.1 Overview

The second device app was developed as the practical part of this thesis. It implements individual verification for the Polyas Core 3 Verifiable e-voting system and is based on their specification of the protocol [10]. The app can, of course, be used for many different elections but was developed with the upcoming election of the Gesellschaft für Informatik e.V (GI) in mind. The GI is a German computer science organization which represents the interests of computer science in e.g. science and politics. If all goes well, the app will be hosted either by the University of Stuttgart or the GI itself, in addition to the app developed by Polyas for the same purpose.

As the main voting device may be corrupted, the app should be run on a second device, one with the ability to scan a QR-Code for usability reasons. The QR-Code contains just a URL to the app with query parameters, so an alternative would be the use of tools to extract this URL and visit the website that way. As the voter can view their vote via the app, the device used and the app itself have to be trusted with ballot privacy in addition to Polyas Voting Client and Server.

4.2 Cast-As-Intended Protocol

This section is based on the paper [10] co-written by one of Polyas developers and the protocol specification provided to developers by Polyas [13]. The protocol is proven to be zero knowledge, here only an intuition for its correctness is given.

For the protocol, an assymetric encryption scheme $S = (X, Gen(1^n), E, D)$ which allows for special decryption and re-randomization, including zero-knowledge-proof for the latter, is required. Both algorithms must be in polynomial time and not require the private key sk. Special decryption allows decrypting a ciphertext $c = E(pk, m; r)$, encrypted with randomness r, not only as usual, by knowing the private key sk and computing $m = D(sk, c)$, but also by knowing the randomness used during the encryption. This method is denoted here as $m = D'(pk, c; r)$. Re-randomization takes public key pk,

ciphertext $c = E(pk, m; r)$ and randomness x. With those it spits out $c* = E(pk, m; r*)$, Polyas assumes this operation to be homomorphic with respect to randomness meaning that $E(pk, m; x + r) = ReRand(pk, E(pk, m; r); x)$ should hold true.

As an example, here it is shown how ElGamal, which is used by Polyas, fulfills these requirements. The definition is an extension of the ElGamal definition from the modern cryptography course at Uni Stuttgart.

Definition 4.2.1
Let $\eta \in \mathbb{N}$. The ElGamal (asymmetric) encryption scheme with ppt algorithm ReRand for re-randomization and pt algorithm D' for special decryption based on GroupGen is the tuple $S_{ElGamal} = (X, Gen(1^\eta), E, D)$, where

- $Gen(1^\eta)$ *executes* $GroupGen(1^\eta)$ *and obtains (G,n,g). Then,* $Gen(1^\eta)$ *chooses* $b \in \{0, ..., n-1\}$ *uniformly at random and outputs* $((G, n, g, g^b), (G, n, g, b))$.
- $X = \{G\}_{(G,n,g,h) \in K_{pub}}$. *That is, the plaintexts are interpreted as elements of the group G.*
- $E((G, n, g, h) : K_{pub}, m : G)$:
  $a \xleftarrow{\$} \{0, ..., n-1\}$
  return $(g^a, m * h^a)$
- $D((G, n, g, b) : K_{priv}, (y_0, y_1) : G \times G)$:
  return $y_1 * ((y_0)^b)^{-1}$
- $D'((G, n, g, h) : K_{pub}, (y_0, y_1) : G \times G, a \in \{0, ..., n-1\})$:
  if $y_0 = g^a$ *then return* $y_1 * (h^a)^{-1}$*, otherwise indicate failure*
- $ReRand((G, n, g, h) : K_{pub}, (y_0, y_1) : G \times G)$:
  $x \xleftarrow{\$} \{0, ..., n-1\}$
  return $(y_0 * g^x, y_1 * h^x)$

The correctness of both the special decryption and re-randomization algorithms is pretty easy to see. In ElGamal a ciphertext has the form $(g^r, m * h^r) = E((G, n, g, h), m; r)$. Therefore special decryption works: $D'((G, n, g, h), (g^r, m * h^r); r) = m * (h^r) * (h^r)^{-1} = m$ and re-randomization is homomorphic: $E((G, n, g, h), m; r + x) = (g^{r+x}, m * h^{r+x}) = (g^r * g^x, m * h^r * h^x) = ReRand((G, n, g, h), (g^r, m * h^r); x) = ReRand((G, n, g, h), E((G, n, g, h), m; r); x)$.

As for the zero-knowledge proof of re-randomization in the ElGamal encryption scheme, the authors of [10] explain that they adapted the sigma-protocol for equality of discrete logarithms to become an interactive zero-knowledge proof. The sigma protocol can proof the fact that the prover knows a value x for which holds $X = g^x$ and $Y = h^x$. With

$c = (y_0, y_1) = E((G, n, g, h), m; r)$ and $c^* = (y_0^*, y_1^*) = E((G, n, g, h), m; r^*)$ the protocol can be used by setting

$$X = \frac{y_0^*}{y_0} = \frac{y_0^*}{g^r} = g^x$$

and

$$Y = \frac{y_1^*}{y_1} = \frac{y_1^*}{m * h^r} = h^x$$

. Then, if the proof is accepted, it follows that there is some x such that $y_0^* = g^r * g^x = y_0 * g^x$ and $y_1^* = m * h^r * h^x = y_1 * h^x$ i.e. $c^* = (y_0 * g^x, y_1 * h^x)$. This is exactly the definition of the ReRand algorithm. In the practical implementation the server only sends the second device app c and (G, n, g, h), instead of c^* X and Y are sent with them. With c, X and Y $c^* = (y_0 * X, y_1 * Y)$ can be computed anyway.

The components partaking in the protocol are Voter V, Audit Device AD, Voting Client VC and Election Server ES with public key pk. An entire vote casting and auditing procedure where steps 1-4 belong to casting and steps 5-9 belong to auditing then works as follows:

1. V indicates their vote v to VC
2. VC chooses random r and sends $c = E(pk, v; r)$ to ES
3. VS chooses and sends to VC a random blinding factor x
4. VC displays blinded randomness $r^* = x + r$ for V
5. V opens the audit app on AD and enters r^*
6. AD notifies ES and is sent (c, c^*) in response, where $c^* = ReRand(pk, c; x)$
7. AD and ES run the zero-knowledge-proof of correct re-encryption with AD as verifier, ES as prover, (pk, c, c^*) as input to both and x as secret input to ES. AD displays an error and aborts if the ZKP is invalid
8. AD uses special decryption to obtain the plaintext vote $v^* = D'(pk, c^*, r^*)$ and displays it to the voter
9. V accepts if $v = v^*$

If the voter does not audit the ballot, the only difference to the usual ballot casting process is ES choosing x and VD computing r^*. It is noteworthy that ES knows neither r nor r^* meaning that ES can still only learn the voter's choice by decrypting the ballot with the secret election key. This is relevant in case the key is distributed among parties, e.g. when threshold decryption is used.

With all parties behaving honestly, correctness can be explained as follows. The two ciphertexts are encryptions of the same plaintext as c^* is a re-randomization of c using randomness x. Furthermore, due to ReRand being homomorphic with respect to the randomness it even holds that $c^* = E(pk, v; x + r) = E(pk, v; r^*)$. Since AD knows r^* it can decrypt c^* without the secret election key using special decryption: $v^* = D'(pk, c^*; r^*)$. The voter, ideally remembering his prior choices, will recognize $v^* = v$ and accept this as proof that the ballot stored on the server is indeed the one they cast.

The protocol could, however, be much easier if VD just handed r to AD directly. No re-randomization would be required at all, AD could simply compute $v = D'(pk, c; r)$. This is actually the approach taken in the Estonian election system. There, the plaintext vote is simply bruteforced [8] by encrypting every possible voting choice with randomness r and comparing it to the ciphertext fetched from the server. There is no problem with this, but it thus make the system look a bit like the unsophisticated little brother of Polyas. The real issue with this approach, however, is that it leaves the voter with a receipt, made up of v, r and c. Thinking back about the Helios E-Voting System, the same was also the case there when auditing a ballot. In Helios, a new ballot then had to be prepared, in this approach, though, the ballot is already cast. There are solutions to this problem, Estonia, for instance, allows voters to change their votes, which makes it possible for voters to sell a vote for candidate A, and then secretly change it to candidate B. Still, this shows that the simpler version should not be used in conjunction with some E-Voting Systems, e.g. Polyas as votes can not be altered after casting them.

Seemingly, a similar receipt is given by ES in Polyas' version in the form of v, $r*$, $c*$ and c. Crucially though, while ES has proven to AD that $c*$ is a re-randomization of c, so $D(sk, c*) = D(sk, c)$, AD can not proof this fact to potential vote buyers. This is because the protocol used was a zero-knowledge proof with AD as the verifier. Without the knowledge of the secret input that ES had, namely, the randomness used in the re-randomization x, AD can not proof the fact to a coercer. Even if AD were to give the coercer the transcript gained from the ZKP, the coercer could not know whether AD faked a transcript for the choice that the coercer would like to see. Intuitively, the difference lies in this fact: in the ZKP AD got to choose at least one random value, but AD can not give the coercer the same opportunity.

4.3 Implementation

Voters first visit the voting client with the link handed to them, likely with their credentials. After logging in, they can enter and confirm their choices to submit their ballot. If the submission is successful, the Voting Client shows its 'thank-you' screen, as in Figure 4.1, containing a QR-code and a time-based one-time password. The QR-code is

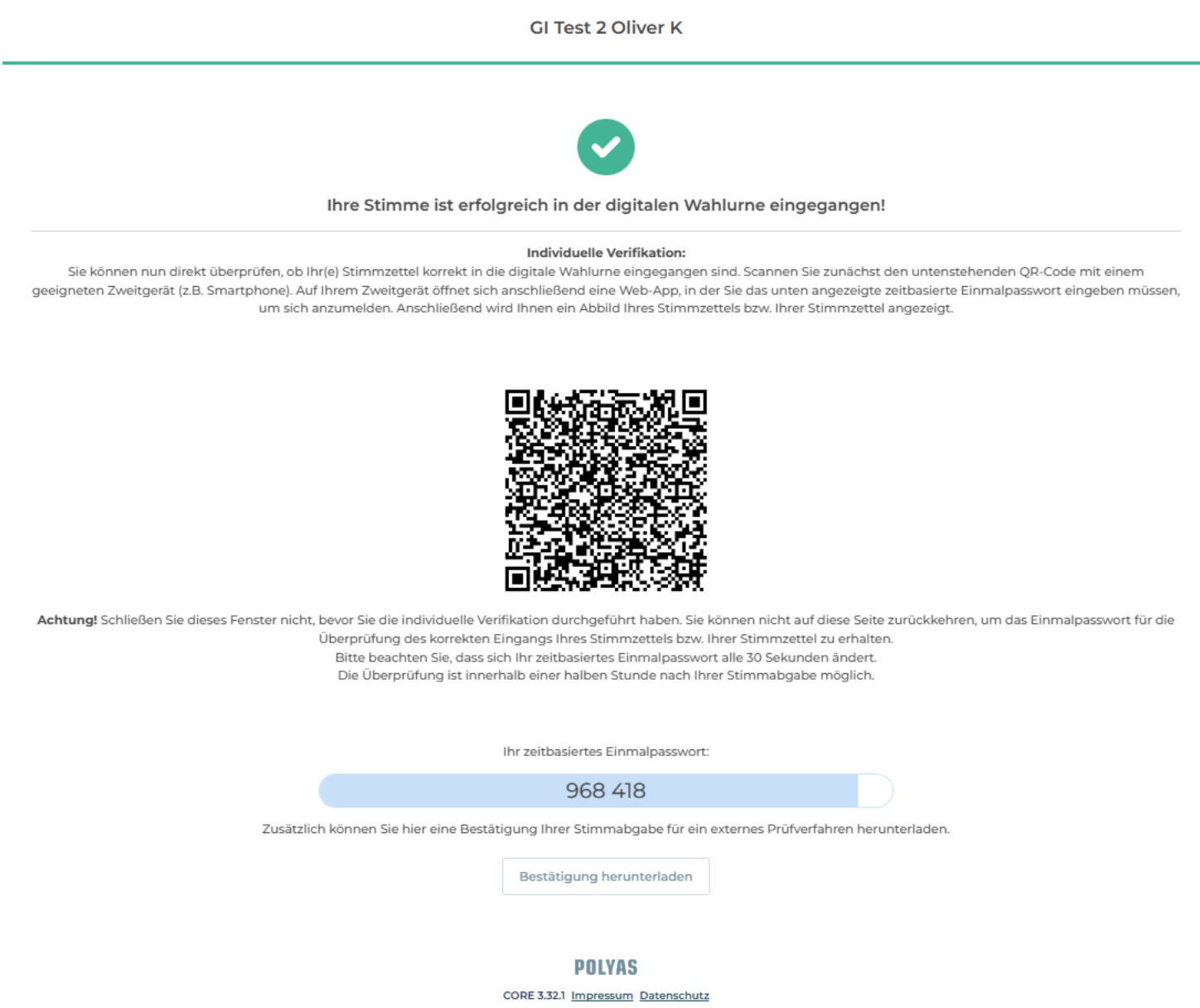

Figure 4.1: The Polyas Voting Client's thank-you page showing the QR-Code and TOTP for the Second Device App

a link of the form 'http://localhost:4300/?c=y6[...]iO&vid=voter6&nonce=04[...]b5'. In production, the app should have a proper domain and use https. The link can be configured on the server side and by the time of the GI election users can hopefully choose in the Voting Client which individual verification app they want to use as there will be multiple.

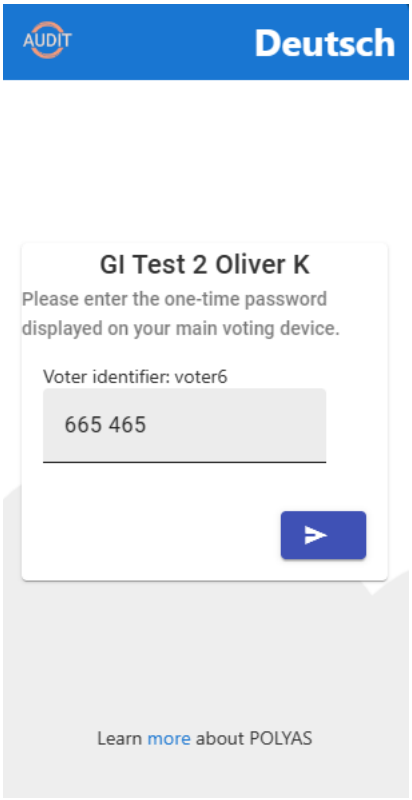

Figure 4.2: The Second Device Apps login. Basic election infos have already been fetched from the backend.

The link carries three URL-query parameters, the vid and nonce are sent along with the login request. The c parameter contains r^*, using the notation from the previous section. Meaning, the randomness that can be used for special decryption.

Since the ballot could be large, it might have to be stored over many ciphertexts. Thus, the r^*'s itself could require too many bits to be carried in a QR-code. To solve this a value called 'randomCoinSeed' is transmitted instead, it is AES encrypted and Base64 encoded in such a way that it can be put into a valid URL. From the randomCoinSeed the random coins, meaning the r^* values can be derived deterministically with an Algorithm given in [12] that makes use of 'HMAC-SHA512' as a pseudo-random function.

The login endpoint used is specially made for the cast-as-intended protocol and already requires (besides vid, nonce, and time-based one-time password) the first value of the zero-knowledge proof of correct re-encryption. The server responds with a token that can be used in all further requests, the ballot specification, and some values for the ZK proof.

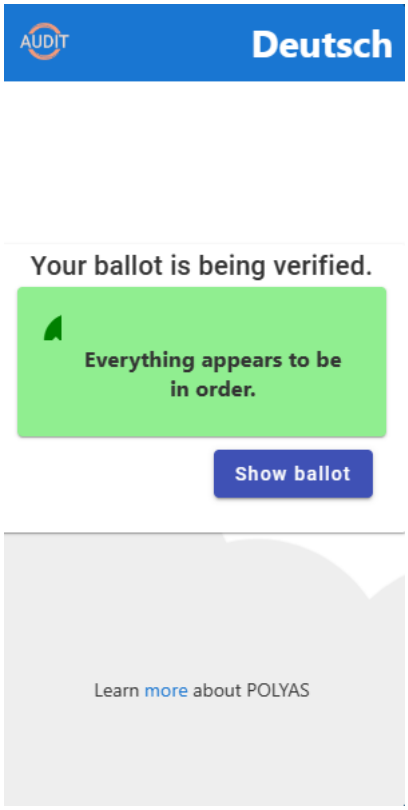

Figure 4.3: The Second Device Apps success screen

After successful login, the protocol continues immediately. It is very fast, so a loading spinner is displayed only briefly, then a success message is (hopefully) shown as in Figure 4.3. If the ZK proof were to fail, the message and coloring would be adjusted accordingly, and the ballot could not be viewed.

Pressing the 'Show ballot' button triggers the special decryption algorithm. As said before, a ballot is encrypted in multiple ciphertexts and the ballot sent to the app is actually the original encryption as well as X and Y, as explained in the previous section. Thus, with ciphertexts $c_i = (x_i, y_i) = (x_i, m * h^r)$ and $Y = h^x$ for $i \in \mathbb{N}/\{0\}$ the relevant second component of the re-encryption is $y_i^* = y_i * Y = m * h^{r+x}$. With it, the public election key and the random coins from the QR-code the encoded plaintext ballot can then be obtained:

$$m_i = \frac{m * h^{r+x}}{h^{r^*}}$$

.

The implementation uses the elliptic curve secp256k1 as an instantiation of ElGamal, so the plaintext consists of elliptic curve points and still has to be decoded. First, each point is mapped to a number in $\mathbb{Z}_q$ where q is the order of secp256k1. Interestingly,

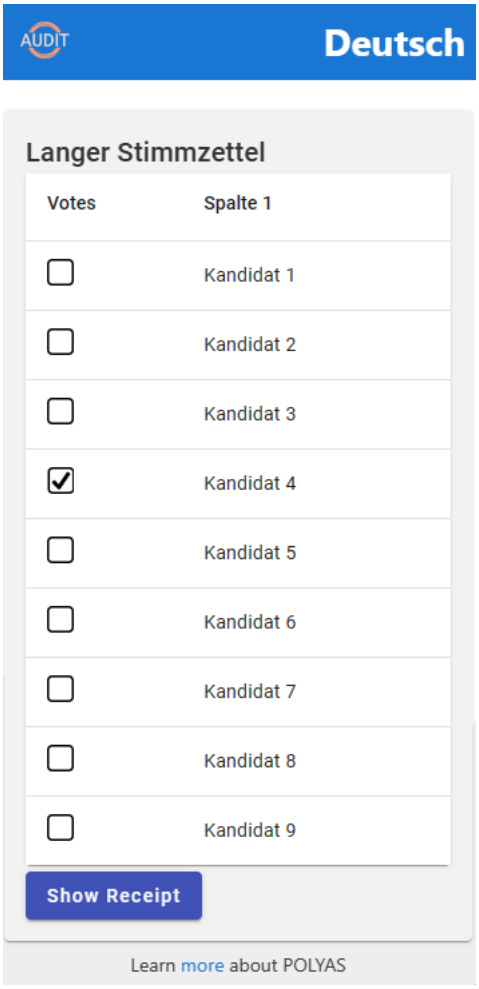

Figure 4.4: View of the decrypted and decoded ballot

even though both groups have size q "in general there exists no efficiently computable homomorphism between the integers modulo q and an elliptic curve of order q"[12]. Instead, a method is used that only utilizes numbers smaller than some upperbound $u < q$, so many group elements are not used. The sequence of numbers in $\mathbb{Z}_q$ is then converted to a byte array with Algorithm 6 of [12]. By using the ballot specification received in the login response, the byte array can be rendered just like in Figure 4.4.

If the voter is satisfied with the ballot, they can view and download a ballot cast confirmation containing their ballot (or its fingerprint) and a signature on it. This confirmation should then be handed to auditors so that they can make sure the ballot is included in the tally. This process is as described in the tallied-as-recorded section of chapter 3.

4.4 Additional facilitation of vote-selling

The authors are correct and have even proven that the protocol is zero-knowledge. This means that the transcript is useless as proof of a voter's choice and thus the protocol is receipt-free. Receipt-freeness is important because a receipt enables voters to sell their votes to coercers. About the design goal "No facilitation of vote-selling" the authors write "we require that the cast-as-intended mechanism should not additionally provide voters with receipts that they can use to trivially prove towards a vote-buyer how they voted" and later "we achieve this by providing cryptographic denialability" [10]. With cryptographic denialability they roughly mean that the coercer can not be sure which plaintext is contained in the encrypted ballot. Even after the voters attempted proofs, there are still plausible alternatives.

The way it is implemented in Polyas, vote selling is, however, aided by this protocol in another way. Whether this is relevant in Polyas depends on the rest of the system being receipt-free or not. Credentials can be sold directly of course, other than that the voter could try to learn r as it is chosen on the client side. Either way, as I certainly would have made the mistake of not authenticating the audit device properly, this section is included anyways. Proper authentication could just mean the same method used in the voting client, which could be tied to something that can not easily be given to a coercer, as the negative consequences would overshadow the financial gain. In Estonia, for instance, the election is tied to a multipurpose identity document/smart card [4].

A coercer Eve, willing to spend enough money to sway Alice to vote for Eve's candidate C could tell Alice that they should both stick to the following protocol, which should be executed in an agreed upon time window:

1. In this window range Alice votes for C
2. Alice extracts the QR-Code payload and sends the link to Eve
3. Eve opens the cast-as-intended app by following the link
4. Alice sends the time-based one-time password to Eve
5. Eve uses the TOTP to login
6. Eve aborts if the ballot contains no valid vote for C or is revoked
7. Eve pays Alice

There are two reasons that a time window should be agreed upon. Firstly, both participants have to be online at the same time, as the time-based password works for only 10 seconds or so. Secondly, the individual verification protocol must be executed before 30 minutes or so after voting. This is just the way Polyas implemented its server. In

this protocol, Eve and Alice only need their communication to have no multi-second delay. If Alice were to try and tamper with r^* then the zero-knowledge proof and special decryption algorithm would fail.

Potentially, Alice could try to fool Eve and vote for another candidate or not at all in step 1 and still get paid. In steps 1, 2 and 4, Alice actively does something but step 4 can be ruled out relatively safely, any tampering with the TOTP leads to a failed login and her not getting paid. In step 1 she could try and choose the initial randomness and in step 2 the blinded randomness sent to Eve. Only choosing the initial randomness does nothing, the protocol should work for any such value. If she also changes the blinded randomness r^*, then the ZK proof of correct re-randomization will still use the blinding factor x which the server chose, so it will fail. It appears that to get paid, Alice must vote for C.

The problem that the protocol enables vote selling does not exist if ballot revocations or revotes are possible in the election system, at least if they are not noticeable for Eve. As said before, proper authentication would also do the job.

4.5 Technologies used

As the individual verification process requires a voter to scan a QR-code with a second device, the app is ideally run on a smartphone. Thus, the initial idea was to program a smartphone app in a multi-platform framework such as Flutter or React-Native. There are two big practical issues with this, however.

1. People, in general, do not like to download strange apps by (potentially) unknown developers on their smartphone. Especially not if they need only use it once every year. And the situation is even worse in this case since there is no need for voters to use the app at all.
2. The app would have to be published on both the App Store and the Play Store so that (almost) everyone could have access to the individual verification process. Publishing apps can take time as well as cost money and whose responsibility publishing would be was unknown at the time of choosing a technology. It should not be Polyas because they could theoretically alter the app, that only leaves the GI and the developer.

Therefore, while it would have been possible to write a smartphone app, a web app seemed like the better decision as it can be visited without a download and the deployment process should be manageable by any of the parties involved. At the time of

writing this, it is still not clear if the app will be used in the upcoming GI election and, if so, who will host it.

There certainly is no shortage of web frameworks to choose from, Angular simply won due to familiarity. Type Script was used as a step towards clean code and the Karma test framework to write unit tests. Unfortunately, some bugs still found their way in.

Different libraries had to be used for the cryptographic functionalities, as there was no library for which everything was provided. The libraries are

- CryptoJS provides the sha256 and sha512 hash functions
- rsasign to verify the ballot cast acknowledgment's signature
- SubtleCrypto for decrypting a part of the QR-code payload using GCM
- noble-secp256k1 as an Elliptic Curve based implementation of the ElGamal asymmetric encryption scheme

Even though the individual verification protocol should explicitly be independently implemented and hosted, Cross-Origin-Resource-Sharing (CORS) is not enabled on the Polyas E-Voting Server. As this CORS policy denied resource access to websites hosted on a different domain than the server, Polyas suggested using a proxy server. In their own implementation, NGINX was used, the second device app uses a very short Node.js script and the http-proxy-middleware library instead. Therefore, the git repository consists of two projects called frontend and backend, both of which are written in JavaScript. Since there is already a Node.js server, it also hosts the app using the Express library and counts the number of users, just out of interest.

For deployment there are two options, the first is to build the angular app and run the server natively and put the built webapp in the '/public' directory, the second is to install docker and run the image associated with the second device app. This option should be preferred as there are fewer pitfalls. Node.js does not have to be installed, for instance, as building happens inside the container. The docker image is built by a two-stage Dockerfile, in the first stage, the angular app is built, in the second stage the static files resulting from the first stage are copied and put into the '/public' directory automatically and the server is started.

4.6 Struggles

The project moved along quite smoothly at the start, Polyas handed out a protocol specification that explained the steps that had to be taken in easy terms. No deep theoretical knowledge of e-voting was required, and so a basic cryptography class was

enough. The necessary algorithms, e.g. to derive keys and randomness, that had to be implemented were given in pseudocode. While not all of them could be copied line by line, they were relatively straightforward. Then came the cryptographic functionalities, these were more of an issue in general as they required the use of libraries, and they did not always work as expected. The spec also sometimes was not detailed enough, e.g. not providing the precise AES mode or the key format.

Since there was no server instance set up, the only way to test were Unit Tests using data that was captured by Polyas in a single run. This was very annoying as sometimes it would have saved a ton of time had there been a way to check intermediate results. When hashing, for instance, it would have been helpful to know both the correct input and output of the hash, so that errors can be pinpointed more easily.

Once all Unit Tests were green, the functionality was put together into an application, but there still was no server instance and thus no way to test the app. After some time, an instance was put up, though, and nothing worked at first, which was kind of expected. E.g. communication with the REST API did not immediately work but was fixed quickly. Then, however, the cryptography did not work either, which was odd as this functionality had pretty good test coverage. The creation of the byte representation of the ballot, used e.g. to verify the ballot cast confirmation, seemed a very likely candidate but was ruled out due to all tests, still based on the test data by Polyas, being successful. It took far too long to find this bug and was probably the most time-consuming single task in this entire thesis. It did end up being the byte representation of the ballot, in particular the byte representation of Bigints, which only worked in rare cases such as the example data. This was incredibly frustrating and even a bit nerve racking as there were only five weeks or so left to hand in the thesis and the application did not work.

5 Verifiability in Polyas

Polyas implements the three smaller verifiability notions and can thus call itself end-to-end verifiable. Here another look is taken at how things are done in the system and whether there are risks.

Starting with eligibility verifiability. Its main purpose is to prevent ballot stuffing. This is implemented by allowing only encrypted ballots with an eligible voter's signature in the ballot box. Seemingly only the voter and the Registrar know a voter's signing key, meaning, so long as the Registrar is honest, all ballots in the ballot box should have been cast by an eligible voter. There is however one more component to consider, the Voting Client. The Voting Client learns the credentials of voters when they log in. Should they decide to login and then not vote, the Voting Client could still add a ballot under the name of the voter. Honest voters, however, do not abort protocol after log in and if they cast their own ballot then they also verify it using a second device, thus they would notice. In chapter 3 it was decided that the election result only honest voters have to be represented accurately. Here also practical issues will be considered. If the Voting Client were to do something like this, however, it would be a gamble. Voters can check whether a ballot is tallied under their name. Hypothetically, the Registrar could set up trap credentials, voters that login and then not vote from the VCs perspective. This 'weak point', if it can even be called that, does seem like a stretch, though.

There is another practical issue concerning tallied-as-recorded verifiability. The issue is not with the system but the setup in the upcoming GI election.

Polyas individual verifiability method was already explained in detail. With x audit devices running x independent individual verification implementations, the voter can be sure that their ballot is being tallied as intended if only one of the x+1 application/device pairs is honest. One downside is that each device has to be trusted with ballot privacy. At the end of the protocol, a receipt / ballot cast confirmation is offered, the same one is also offered in the VC after casting a vote. [10]

These confirmations should be used to check the final ballot box content for the ballot and should be done by auditors. Ideally, handing over the confirmation would be automatic. The universal verification tools could, for instance, offer a REST endpoint for posting these confirmations. This would not be very hard to implement. Polyas

was asked what the voter's instructions were concerning these confirmations and how they would get them. They only say that the voter should hand the confirmation to the election council, who can then hand them to the auditors. It is, in this case, the GIs job to educate the voters about this.

But the means by which the confirmations should be handed over are not entirely clear yet. The suspicion at the moment is that no regular voter's confirmation will actually find its way to an auditor. Auditors themselves can not vote, so they can not create confirmations for themselves. Hopefully some at the GI involved in the e-voting process, who can also vote, validate that their ballots are included in the final tally, but this small group seems to be the only realistic option. While this sounds bad in theory, it would still be a considerable risk for the server in practice, and so it should be just fine for the upcoming election. There is also the option that this is wrong and it will be explained, e.g. in the instructions voters get with their credentials, and that many will follow protocol.

Finally, it is time for Polyas version of universal verifiability. In chapter 3, two approaches were introduced. Polyas follows the second approach which uses anonymization of the ballot box before decryption, its auditing tasks are explained in detail in [12], and an overview is given here using the document as a reference.

The foundational framework involves a server hosting bulletin boards. These bulletin boards function as append-only data structures, categorized into two types: external boards, sourced from another election component, and internal boards, computed directly by the server. It's essential to note that both types necessitate auditing, given that the election server isn't the sole potential source of malfunctions. Post-election, the boards are transmitted to auditors.

Perhaps most importantly, the auditors perform universal verification, i.e. they check the proofs of correct shuffling and decryption. This includes verifying that all ballots marked as invalid, for whatever reason, are dropped from further processing. This also applies to ballots that were revoked, i.e. there is a revocation ticket which has valid signatures on it from enough election administrators. And then, of course, the tally has to be performed to check the actual election result. But Auditors have other responsibilities in addition to those concerning universal verification. A selection of those is outlined here.

Firstly, the ballot cast confirmations introduced for individual verifiability should also be handled by the auditors as explained before. Secondly, the registry board needs to be checked so that no credentials are given to non-eligible voters and that the credentials given are well-formed. The Election Council's list of eligible voters should be the authority.

The takeaway here is: Polyas is end-to-end verifiable. This holds, so long as no two or more components collude. Otherwise, the Election Server and the auditor (if there is

only one) could make up a result or the Registrar could give passwords of voters, who did not participate, to the server for ballot stuffing.

6 Conclusion

In conclusion, with end-to-end verifiability the integrity of E-Voting can be secured. The implementation often happens in three parts: eligibility, individual and universal verifiability. In this thesis, multiple approaches of implementing verifiability have been introduced. As Polyas implements one of each type, it is judged to be end-to-end verifiable. Especially the method Polyas uses for individual verification was explained in detail as it was also implemented in the practical part of this thesis. Two issues were found, however.

Firstly, it can be argued that vote-selling is facilitated by the protocol as it is implemented. This is because the audit device is not properly authenticated. A voter can, after voting, send the QR-Code content as well as the time-based password to a coercer to prove their choice. Nevertheless, the protocol is an improvement over its simpler version, as no receipt is given. Secondly, as individual verification depends on voters actually following the protocol, the process should be as easy as possible for them. The ballot cast confirmations given to the voter by the app should therefore be handed to auditors automatically instead of leaving it up to the voter. Voters that fail to send the confirmation to an auditor run the risk that their ballot could be dropped from the ballot box.

7 Outlook

While working on this thesis, I learned that a well-thought-out remote e-voting system can be very secure if all data is publicly verifiable and there are many trusted auditors. Even privacy can not be broken by a single component. This is what Polyas does. They have implemented a version that uses a threshold decryption scheme, meaning that a certain number of independent parties must cooperate for decrypting and shuffling. This option should definitely be used in more important elections in the future.

There are further optimizations possible. For instance, in the paper about the cast-as-intended protocol [10] it is mentioned that e-voting systems aiming for everlasting privacy often commit to their votes instead of encrypting them and show that their protocol can also work in such a system. Everlasting privacy is an important property if a system, with public ballot box, is used in an important election. Voters in important elections do not want their choice to become public knowledge with the development of some next generation computer chip, even if there are 50 years in between.

Personally, I hope that end-to-end verifiable remote e-voting will see more adoption in the future as it is secure if auditing is set up properly and makes voting much more comfortable, thus increasing voter turnout. To achieve this goal, laws currently restricting entities to only postal or physical votes should be changed. Legal tricks, as the one used by the CDU to elect their leader electronically, should not be necessary. Important elections allowing remote e-voting should, however, consider the risk of vote-selling and coercion so that potential large scale operations to manipulate or coerce votes can be shut down before they do much harm.

Furthermore, public discussion on the topic of e-voting is often times not concerned with verifiability, it is a foreign concept to most and harder to understand at first than some in the field may think. As, in the public eye, both end-to-end verifiable e-voting and sending in ballots per email are just two sides of the same coin, trust in any electronic election can be eroded very quickly.

Bibliography

[1] B. Adida. “Helios: Web-based Open-Audit Voting.” In: *USENIX security symposium*. Vol. 17. 2008, pp. 335–348 (cit. on pp. 12, 13).

[2] J. Benaloh. “Simple Verifiable Elections.” In: *2006 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT 06)*. Vancouver, B.C.: USENIX Association, Aug. 2006. URL: https://www.usenix.org/conference/evt-06/simple-verifiable-elections (cit. on p. 23).

[3] *Data-at-Rest Encryption Overview*. https://mariadb.com/kb/en/data-at-rest-encryption-overview/. Accessed: 2023-08-26 (cit. on p. 17).

[4] *Electronic Voting in Estonia*. https://en.wikipedia.org/wiki/Electronic_voting_in_Estonia. Accessed: 2023-08-27 (cit. on p. 37).

[5] R. Gharadaghy, M. Volkamer. “Verifiability in electronic voting - explanations for non security experts.” In: *4th International Conference on Electronic Voting 2010*. Bonn: Gesellschaft für Informatik e.V., 2010, pp. 151–162. ISBN: 978-3-88579-261-1 (cit. on p. 26).

[6] O. Kulyk, V. Teague, M. Volkamer. “Extending helios towards private eligibility verifiability.” In: *E-Voting and Identity: 5th International Conference, VoteID 2015, Bern, Switzerland, September 2-4, 2015, Proceedings 5*. Springer. 2015, pp. 57–73 (cit. on p. 27).

[7] R. Küsters, T. Truderung, A. Vogt. “Accountability: definition and relationship to verifiability.” In: *Proceedings of the 17th ACM conference on Computer and communications security*. 2010, pp. 526–535 (cit. on pp. 19, 20).

[8] K. Marky, O. Kulyk, M. Volkamer. “Comparative usability evaluation of cast-as-intended verification approaches in internet voting.” In: *SICHERHEIT 2018* (2018) (cit. on p. 32).

[9] *Most expensive ever: 2020 election cost* $14.4billion$. https://www.opensecrets.org/news/2021/02/2020-cycle-cost-14p4-billion-doubling-16/. Accessed: 2023-08-15 (cit. on p. 9).

[10] J. Müller, T. Truderung. *A Protocol for Cast-as-Intended Verifiability with a Second Device*. 2023. arXiv: 2304.09456 [cs.CR] (cit. on pp. 22, 25, 29, 30, 37, 41, 47).

[11] S. Neumann, M. M. Olembo, K. Renaud, M. Volkamer. “Helios Verification: To Alleviate, or to Nominate: Is That the Question, or Shall we Have Both?” In: *Electronic Government and the Information Systems Perspective*. Ed. by A. Kő, E. Francesconi. Cham: Springer International Publishing, 2014, pp. 246–260. ISBN: 978-3-319-10178-1 (cit. on p. 24).

[12] Polyas. *POLYAS 3.0 Verifiable E-Voting System*. Unpublished specification. 2023 (cit. on pp. 14, 15, 17, 27, 34, 36, 42).

[13] Polyas. *Polyas-Core3 Second Device Protocol*. Unpublished specification. 2023 (cit. on p. 29).

[14] *POLYAS case study: Digital CDU party conference*. https://www.polyas.com/parties/party-board-elections/case-studies/cdu-party-conference. Accessed: 2023-08-27 (cit. on p. 13).

All links were last followed on August 24, 2023.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature