Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Secure E-Voting with ETHVote

Master's Thesis

Tobias Ballat

ballatt@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Darya Melnyk, Tejaswi Nadahalli
Prof. Dr. Roger Wattenhofer

September 27, 2020

Acknowledgements

I would like to thank Tejaswi Nadahlli and Darya Melnyk for supervising my work and their weekly feedback. Prof. Dr. Roger Wattenhofer has my thanks, for making this thesis possible and his input. Also, Flavio Goldener, who implemented the basic version of ETHVote, and whose code I could build upon.

I also express gratitude to smoca AG to let me work in their office and for their advice about implementing details, and Mohammed Chelouti for his grammar advice. Many thanks go also to my girlfriend Meret Aschwanden and our families, who took care of our daughter much more often than usual. This allowed me to spend so much time on this thesis.

Abstract

This thesis extends the ETHVote system with another participant, called voting assistant. The protocol to vote is adapted and the actual voting process takes place on two devices. This improves the security of ETHVote significantly, as the voting device no longer must be trusted. Additionally, a Benaloh Challenge was added to increase the probability of manipulation detection.

Contents

Acknowledgements **i**

Abstract **ii**

1 Introduction **1**

2 Related Work **2**

2.1 Apollo & PrivApollo 2

2.2 Other E-Voting Systems 2

3 Protocol **3**

3.1 Participants 3

3.2 Model 3

3.3 Key Exchange 4

3.4 Voting Phase 5

3.4.1 Initialize 5

3.4.2 Vote 6

3.4.3 Cast 7

3.4.4 Audit 7

3.4.5 Initialize Audit 7

3.4.6 Reset 7

3.5 Non-Interactive Zero-Knowledge Proof 7

4 Implementation **10**

4.1 Voting Assistant 10

4.1.1 Main Activity 10

4.1.2 Voting Activity 11

4.2 Authentication & Authorization 11

4.3 State Updates 13

4.4 User Experience 13
4.4.1 Colors 14
4.4.2 Instructions 14
4.5 Deployment 14
4.5.1 Initial Administrator 16
4.6 Implementation Challenges 16

5 Security 18
5.1 Ballot-Privacy 18
5.2 Receipt-Freeness 18
5.3 Election-Fairness 18
5.4 Authenticity 19
5.5 Correctness 19
5.5.1 Malicious Data by the Terminal 19
5.5.2 Malicious Data by the Voting Assistant 19
5.5.3 Collaboration Between Terminal and Voting Assistant . . 20
5.6 Individual Verifiability 20
5.7 Universal Verifiability 20

6 Conclusion 21
6.1 Future Work 21

Bibliography 22

A Acronyms A-1

Chapter 1

Introduction

While e-voting has been established in Estonia since 2005, it has a difficult stand in Switzerland. There were two approaches in recent years. CHVote [1] was developed by the canton of Geneva and has been in use in several cantons since 2003. The development of version 2.0 which should have been available on a national scale, was stopped after two years in 2018 due to its costliness.

The second system which was a candidate for national e-voting was the Post E-Voting [2]. This system was developed by the Spanish company Scytl. While CHVote is completely open-source, the source code of the Post E-Voting is only available after registration. The project was stopped in 2019 because of serious flaws in the source code found in an intrusion test [3] in 2019.

Because both projects have been stopped, the Swiss government started a redesign of the trial phase[1] in June 2020. According to the government [4], the benefits of e-voting in Switzerland are the impossibility of invalid votes, the faster counting, and accessibility. Their basic principle is security before speed. A key element, that a system is assumed secure, is verifiability. While previous systems only had to be individually verifiable, future systems must be universally verifiable.

This thesis is a continuation of ETHVote [5]. The previous version of ETHVote was already universally and individually verifiable. But the voter could only verify that some encryption was saved, but not if it corresponds to their actual choice. We added a Benaloh Challenge [6] to increase the chance that vote manipulations are detected.

Even more important than verifiability is the correctness of a voting system. ETHVote was already correct under the assumption, that the voting device is honest. To ensure correctness with an untrusted device, a second device, which must actively participate, is added to ETHVote. With this thesis, correctness is ensured, as long as at least one of the devices is honest. Even if both devices are malicious, they must collude to manipulate the vote.

[1]https://www.bk.admin.ch/bk/en/home/dokumentation/medienmitteilungen.msg-id-79556.html

Chapter 2

Related Work

2.1 Apollo & PrivApollo

The Apollo voting protocol [7] adds a Benaloh Challenge [6] to Helios [8]. This allows the voter to audit the vote on a second device. While this helps to detect manipulations from the voting device, the voting device still must be trusted that system is correct.

PrivApollo [9], an extension to Apollo, delegates the actual vote to a second device. Through this separation, the terminal which encrypts the vote does not know what the voter votes, since this happens on the second device. With this, the criterium of correctness is fulfilled even if one of the devices is malicious.

2.2 Other E-Voting Systems

There are several other e-voting systems besides Helios, Apollo, and PrivApollo. Some of them work similarly in some parts, and some of them work completely differently.

Electionguard[1] from Microsoft is a software development kit to verify that votes were correctly encrypted. It is based on Josh Benaloh's PhD thesis [10] and uses a similar key generation mechanism as ETHVote, but a different encryption scheme. It was never used productively yet.

Belenios [11] modifies the Helios protocol and does the key generation fully distributed, which means that no trustees are needed. Belenios is implemented as an online platform, like ETHVote. It was used already for academic, educational, and association elections.

While all mentioned systems are open-source, there exist also closed source solutions like Voatz[2] which has already been in use for local government elections in the USA. Voatz uses a blockchain to audit the votes.

[1] `https://github.com/microsoft/electionguard`
[2] `https://voatz.com/`

Chapter 3

Protocol

The main ideas of the protocol are still the same as in the previous version of ETHVote [5]. The only changed phase is the voting one, where an additional participant, the voting assistant was added and the protocol was adapted. As a consequence, the non-interactive zero knowledge proof (NIZKP) had to be adapted as well.

3.1 Participants

While the participants are the same as in the previous version, we took the terminology from PrivApollo [9] to clearly distinguish the two devices. This results in the following participants.

- **Voter:** The human voter can read and compare strings and colors. The voter decides which option they choose.
- **Terminal:** The web application, on which the voter votes. The terminal initializes and casts the vote. The voter can audit the voting on it. In the previous version, it was called "front end".
- **Voting Assistant:** The voting assistant is used to audit the voting. In ETHVote only active voting assistants exist, meaning a voting assistant sends the choice of the voter to the server.
- **Server:** The server stores the voting data and the voters. The terminal and the voting assistant communicate through the server. The server does no computations except checking zero-knowledge proofs.

3.2 Model

To create a new vote, the following parameters must be chosen:

- $\mathbb{Z}_{\mathbf{p}}^*$: Cyclic group in which all computations will be performed.
- **Trustees:** The trustees generate the keys for encryption and decryption.
- **Security Threshold:** k, the number of trustees that have to participate in key generation.
- **Option Primes:** To every option, a unique prime number is assigned.

For key generation, trustee i generates a secret polynomial s_i of degree $k-1$, and the corresponding public polynomial y_i computed as in Equation 3.1. To encrypt votes s and y as defined in Equations 3.2 and 3.3, are used as the private and public key.

$$y_i(x) := g^{s_i(x)} \tag{3.1}$$

$$s := s(0) := \sum_i s_i(0) \tag{3.2}$$

$$y := y(0) := \prod_i y_i(0) \tag{3.3}$$

In order to vote, a voter encrypts the prime number of their chosen option with ElGamal encryption [12] in $\mathbb{Z}_p^*$ with y as the public key. The resulting ciphertext is a pair (α, β), where α does not depend on the vote, but β does.

For decryption, trustee i sends point $s_i(j)$ to trustee j. Trustee j fetches $\prod \alpha_i$ from the server and can then compute their decryption factor D_j as in Equation 3.4. $d(x)$ is defined as a polynomial of degree k where $d(x) = D_x$ for all known D_x.

$$D_j := (\prod_i \alpha_i)^{s(j)} = (\prod_i \alpha_i)^{\sum_i s_i(j)} \tag{3.4}$$

When at least k trustees computed D_j, $d(0)$ can be computed by doing Lagrange interpolation. With $d(0)$ the product of the selected primes can be decrypted, however single votes can not be decrypted. Since the decrypted result is a product of primes, votes can easily be counted by doing prime factorization.

3.3 Key Exchange

The voting assistant needs some information about the vote to participate in the voting process. To exchange this data the terminal displays a *QR* code, which is scanned with the voting assistant and contains the following data:

- **Server Address:** The voting assistant can connect with the server listening at the server address.
- **Vote ID:** The ID of the vote, the voter currently participates in.
- **Token:** The authentication token of the terminal. The implemented authentication mechanisms are described in Section 4.2.
- **Secret Key:** The secret key sk is generated by the terminal and used to encrypt private data. sk is never sent to the server, meaning only the terminal and the voting assistant know this key.
- **Web-Socket Address:** The voting assistant opens a web-socket with the server through this address. Web-sockets are used to get fast state updates as described in Section 4.3.

Scanning the QR code is possible as long the vote is open. This allows the voter to reconnect the voting assistant with the server when the authentication expires or the connection is closed. The voter can also use another voting assistant and proceed there.

3.4 Voting Phase

The voter chooses their option on the voting assistant. This adds several actions to our protocol. Figure 3.1 shows the states a vote can assume during the voting phase and the actions needed to transition into those states. Blue actions must be executed by the terminal, red actions by the voting assistant, and violet actions can be executed by both devices. These states are voter dependent, meaning each vote can be in different states at the same time for different voters. In this section, the actions needed to transition between the states are described.

3.4.1 Initialize

To initialize the voting process, the terminal chooses n colors $[c_1, \ldots, c_n]$ where n is the number of options. It then generates a random permutation π used to assign each option a color. Additionally, a random number r_i for every option i, used as randomness for ElGamal encryption, is generated.

Since the terminal does not know which option will be chosen, it has to encrypt all options and send the ciphertexts together with the assigned color to the server. For every option, the terminal computes α_i and β_i as in Equations 3.5 and 3.6. g is a generator of $\mathbb{Z}_p^*$ and was defined when the vote was created, m_i is the prime corresponding to option i, and y is the public key as defined in Equation 3.3.

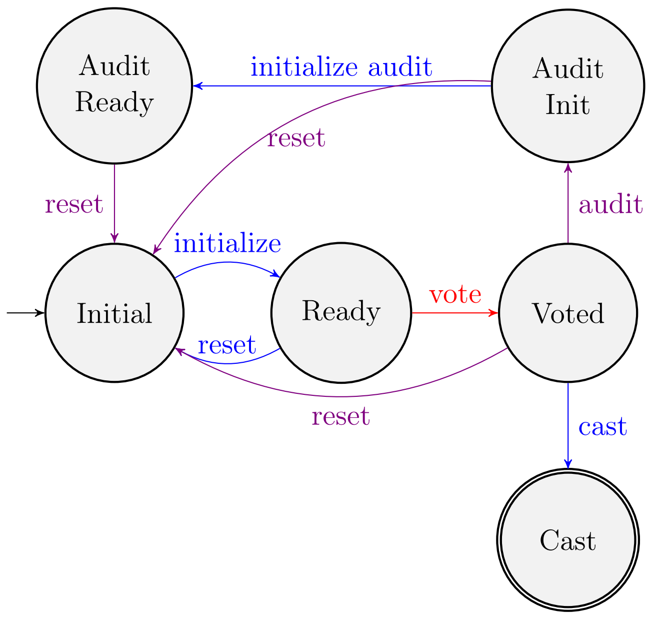


Figure 3.1: State machine for the voting phase

$$\alpha_i := g^{r_i} \tag{3.5}$$
$$\beta_i := m_i y^{r_i} \tag{3.6}$$

Next, the terminal sends then the triplets $\langle \alpha_i, \beta_i, c_{\pi(i)} \rangle$ to the server for every option i. At the same time, it displays the mapping from options to colors to the voter.

Additionally, it encrypts the list of colors with sk and sends the ciphertext to the server. When the server receives the ciphertext, initialization is finished and the vote transitions into the state “Ready”.

3.4.2 Vote

When the vote is ready, the voting assistant fetches the colors from the server, decrypts them, and shows a button for each color to the voter. When the voter chooses the color of their wanted option, the voting assistant sends the chosen color to the server.

When the chosen color is saved on the server, the vote is in the state “Voted” and the user has to choose whether they want to audit or cast the vote.

3.4.3 Cast

When the voter decides to cast the vote, the terminal computes a NIZKP to prove that all option primes were encrypted exactly once. Section 3.5 explains the mathematics behind the proof. When the server receives the NIZKP, the server checks if the proof is valid. If it is valid, the vote transitions into state "Cast" and can not be changed any longer.

3.4.4 Audit

To audit the vote, the voter presses the "audit" button on the terminal or the voting assistant. The targeted device then notifies the server that the voter wants to audit the vote and the vote transitions into the "Audit Init" state.

3.4.5 Initialize Audit

When the terminal receives the new state "Audit Init", it fetches the selected color from the server. With the color, it checks which option s the voter voted for and display this to the voter. Then the terminal looks up r_s and sends $\{|r_s|\}_{sk}$ to the server, where $\{|x|\}_k$ notates symmetric encryption of x with key k.

The voting transitions into "Audit Ready" state when the server receives $\{|r_s|\}_{sk}$. The voting assistant then fetches $\{|r_s|\}_{sk}$ and the selected encrypted option $\langle \alpha_s, \beta_s \rangle$ from the server and decrypts $\{|r_s|\}_{sk}$. With r_i all option primes can be encrypted until $\langle \alpha_i, \beta_i \rangle$ is found. When the encryption from the voting assistant and the server matches the voting assistant found the selected option and shows it to the voter.

After auditing the vote must be reset, such that the voter has no receipt for what they voted.

3.4.6 Reset

When something failed or after auditing, a vote can always be reset. As soon as the server receives the command to reset a vote, it will delete the already saved data and update the state to "Initial".

3.5 Non-Interactive Zero-Knowledge Proof

To create a NIZKP, the fact that ElGamal encryption is homomorphic is used. "Proof Systems for General Statements about Discrete Logarithms" [13] describes how proofs for linear relations among discrete logarithms can be created.

To prove that the terminal encrypted the option primes, it computes r as in Equation 3.7, where r_i is the randomness used to encrypt option i (see Section 3.4.1).

$$r := \sum_i r_i \tag{3.7}$$

Because the prime number m_i is known for every option i, the server can compute β and β' as in Equations 3.8 and 3.9 when it receives r.

$$\beta := \prod_i \beta_i \tag{3.8}$$

$$\beta' := y^r \prod_i m_i \tag{3.9}$$

When it computed β and β', the server can check that the following assumptions hold.

1. $\beta' = \beta$
2. Exactly one ciphertext per option was sent.

When both assumptions hold, the server knows that the terminal encrypted all option primes exactly once.

Proof. Equation 3.10 shows that β is equal to β' when every β_i is valid ciphertext of option i.

$$\beta = \prod_i \beta_i = \prod_i m_i y^{r_i} = y^{\sum_i r_i} \prod_i m_i = y^r \prod_i m_i = \beta' \tag{3.10}$$

If the terminal encrypts a number, which does not belong to the set of option primes, the product of encrypted numbers is different from the product of all option numbers because prime factorization is unique. It follows that $\prod \beta_i \neq \prod m_i y^{r_i}$ and further $\beta \neq \beta'$.

However, the terminal can combine multiple options. If there are for example three options, "yes", "no", and "maybe" with the primes 2, 3, and 5 assigned respectively, it can combine "yes" and "maybe". It then encrypts 10 and 3 instead of 2, 3, and 5. Because the product is 30 in both cases, $\beta = \beta'$ still holds. But since the server also checks that exactly one ciphertext for every option was sent, it detects the attack.

To create a valid NIZKP with invalid data, the terminal has to combine options as described before. Additionally, it has to send some encryption of 1 for every combined option, such that the total number of ciphertexts equals the number of options. However, the voter will recognize this attack, when they audit the vote because valid option numbers are always prime and neither 1 nor a product of multiple primes, is prime.

To learn anything about single options, the server has to decrypt ElGamal ciphertexts. For this, the private key or the randomness is needed. Because the private key is distributed under the trustees, nobody has the whole private key. A single randomness r_i can also not be computed, because there are infinite possibilities to sum up r. $\square$

CHAPTER 4

Implementation

The code of the previous version of ETHVote [5] was used as a base. While the terminal and the server only needed adaptions to work with the extended protocol, the voting assistant was built from scratch.

4.1 Voting Assistant

The voting assistant is implemented as an Android app. Kotlin is used to get a clear code with small overhead. For network requests and other background tasks as decryption, coroutines were used. Coroutines make it easy to define the thread on which a function should run.

To keep the app small and trustable, only a few libraries, besides standard Android libraries, are included. Namely, ZXing[1] to scan and decode *QR* codes, Retrofit[2] for network requests, and Moshi[3] to parse and write JSON.

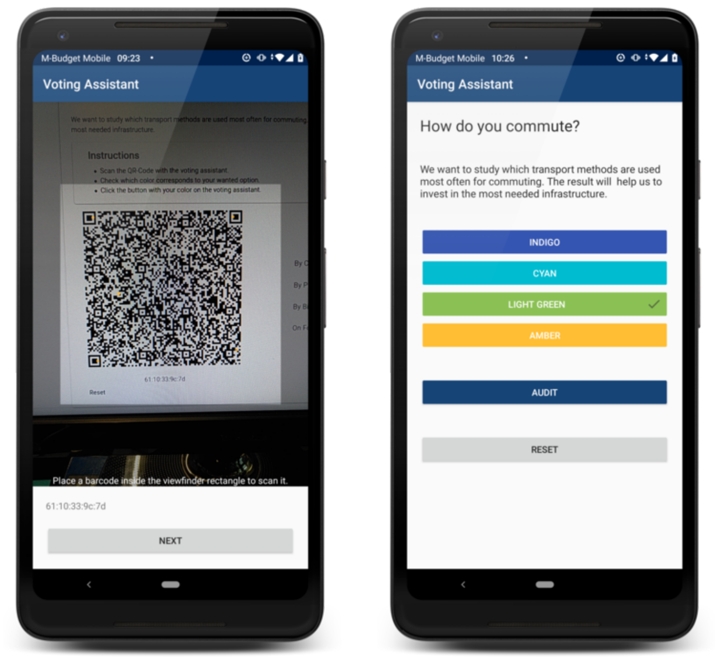
The app contains two activities. The main activity to scan and decode the *QR* code and the voting activity to vote and audit. Screenshots of both activities can be seen in Figure 4.1.

4.1.1 Main Activity

The main activity is responsible to scan the *QR* code displayed on the terminal. The *QR* code contains the five values described in Section 3.3, separated by ";". If the *QR* code contains valid data, the voting assistant saves the data and vibrates. This vibration tells the voter that the *QR* code was successfully scanned and decoded, and that they can proceed with the voting process.

To convince the voter that the data was read correctly, a fingerprint of the data is shown on the terminal and the voting assistant.

[1] `https://github.com/journeyapps/zxing-android-embedded`
[2] `https://square.github.io/retrofit/`
[3] `https://github.com/square/moshi`



How do you commute?

We want to study which transport methods are used most often for commuting. The result will help us to invest in the most needed infrastructure.

(a) Main Activity (b) Voting Activity

Figure 4.1: Screenshots of the voting assistant

4.1.2 Voting Activity

When the voting assistant enters the voting activity, it will first fetch all required data for the selected voting. Additionally, it subscribes to state updates. On every state update, it fetches the required data, decrypts it if necessary, and updates the user interface. In Section 3.4 is described what data is fetched in which state.

If the voter presses the audit or reset button, the voting assistant sends the command to audit or reset respectively the voting to the server. If the voter chooses a color, the voting assistant marks the chosen color and sends it to the server.

4.2 Authentication & Authorization

Shibboleth is used to authenticate voters. With this, a secure authentication mechanism is guaranteed, without much effort. It is also convenient for the voter as they can use their well-known username and password combination and do not have to register to yet another platform.

While Shibboleth already provides a name, an email address, and a unique

```
{
        "subId": "3",
        "sub": "Tobias Ballat",
        "registered": true,
        "exp": 1599570354,
        "device": "TERMINAL",
        "numberOfVotesWithTrusteeRole": 7,
        "subRoles": [
                "ADMINISTRATOR",
                "VOTING_ADMINISTRATOR"
        ]
}
```

Listing 4.1: Example payload of a JWT

ID, this is not enough to authorize the voter. ETHVote has a role system, where each voter can have one or multiple roles. Additionally, the voting administrator can only authorize registered voters to vote. Voters can either register themselves or be registered by a registrar or an administrator.

Therefore, JSON Web Tokens (JWTs) [14] are used to authorize voters. Listing 4.1 shows an example payload of an ETHVote JWT. It contains the following data.

- **subId:** The ID of the voter, used to identify them by the system.
- **sub:** The name of the voter, used to identify them by other voters.
- **registered:** Whether the voter is already registered.
- **exp:** Timestamp when the token expires. To not bypass Shibboleth a new Token is valid only 10 minutes. This forces to authenticate the voter by Shibboleth every 10 minutes. Because the Shibboleth session is valid longer, this can be done by the terminal, without any interaction from the voter. When the terminal receives an unauthorized error from the server, it sends an authentication request with the Shibboleth session cookie, and if this is successful it sends the failed request again.
- **device:** The device type on which the voter is authorized. This is used by the server to check, that actions are only performed from a legitimate device.
- **numberOfVotesWithTrusteeRole:** A number indicating in how many votes the voter is a trustee. When the voter is a trustee in at least one voting, the terminal displays the trustee-interface to them.

- **subRoles:** A list of roles the voter has. It can contain `"ADMINISTRATOR"`, `"VOTING_ADMINISTATOR"` or `"REGISTRAR"`. The terminal reads the roles to show the corresponding interfaces to the voter and the server to authorize the performed actions.

To prevent a voter to manually log in again on the voting assistant, the token of the terminal is sent to the voting assistant via QR code as described in Section 3.3. The voting assistant uses the terminal token to authenticate the voter on the server and receives a valid voting-assistant token.

4.3 State Updates

With the voting assistant, multiple devices can send requests to the server, which will then update the voting state. Therefore, it is not enough sending the new state as a response to the request, because only the device that sent the request would receive the new state.

As a countermeasure, web-sockets were introduced. When a device wants to get state updates, it opens a socket to the server. The device sends the authorization token as the first message. If the token is valid, the server saves the connection and assigns the voter's ID to it.

The server sends the new state, together with the ID of the changed vote to all sockets belonging to the originating voter. This allows all devices to update their user interface right after a state change, even when another device initialized the change.

HTTP requests are parsed by Apache httpd[4], serving the terminal, and forwarded to Apache Tomcat[5], serving the server via AJP[6]. Because AJP cannot handle web-sockets the socket request must be sent directly to Tomcat. This is the reason that the QR code contains the server address and a separate web-socket address as described in Section 3.3.

4.4 User Experience

The user experience is very similar to the one in the previous version. Material design is still used on the terminal and now also on the voting assistant. The changes made for this new version are described here.

[4]`https://httpd.apache.org/`
[5]`https://tomcat.apache.org/`
[6]`https://tomcat.apache.org/connectors-doc/ajp/ajpv13a.html`

```
sudo docker-compose up -d --build
```

Listing 4.2: Command to build and start ETHVote

4.4.1 Colors

Colors for the options are taken from the Material design guidelines[7], which the components follow too. If there is a feasibly small number of options, colors with big distances are chosen, such that they are easily distinguishable. To ensure color-blind people can vote as well, the name of the color is also displayed.

When the voter resets their vote, the colors for the options are reshuffled as described in Section 3.4.6. As you can see in Figure 4.2, completely new colors are taken after a reset. This ensures that the voter notices that the colors have changed and does not press the same color again without checking to which option it now belongs.

4.4.2 Instructions

Since the complexity of the voting process has increased, instructions were added to the terminal. The instructions tell the voter what they have to do, and on which device they have to do it, to continue the voting process.

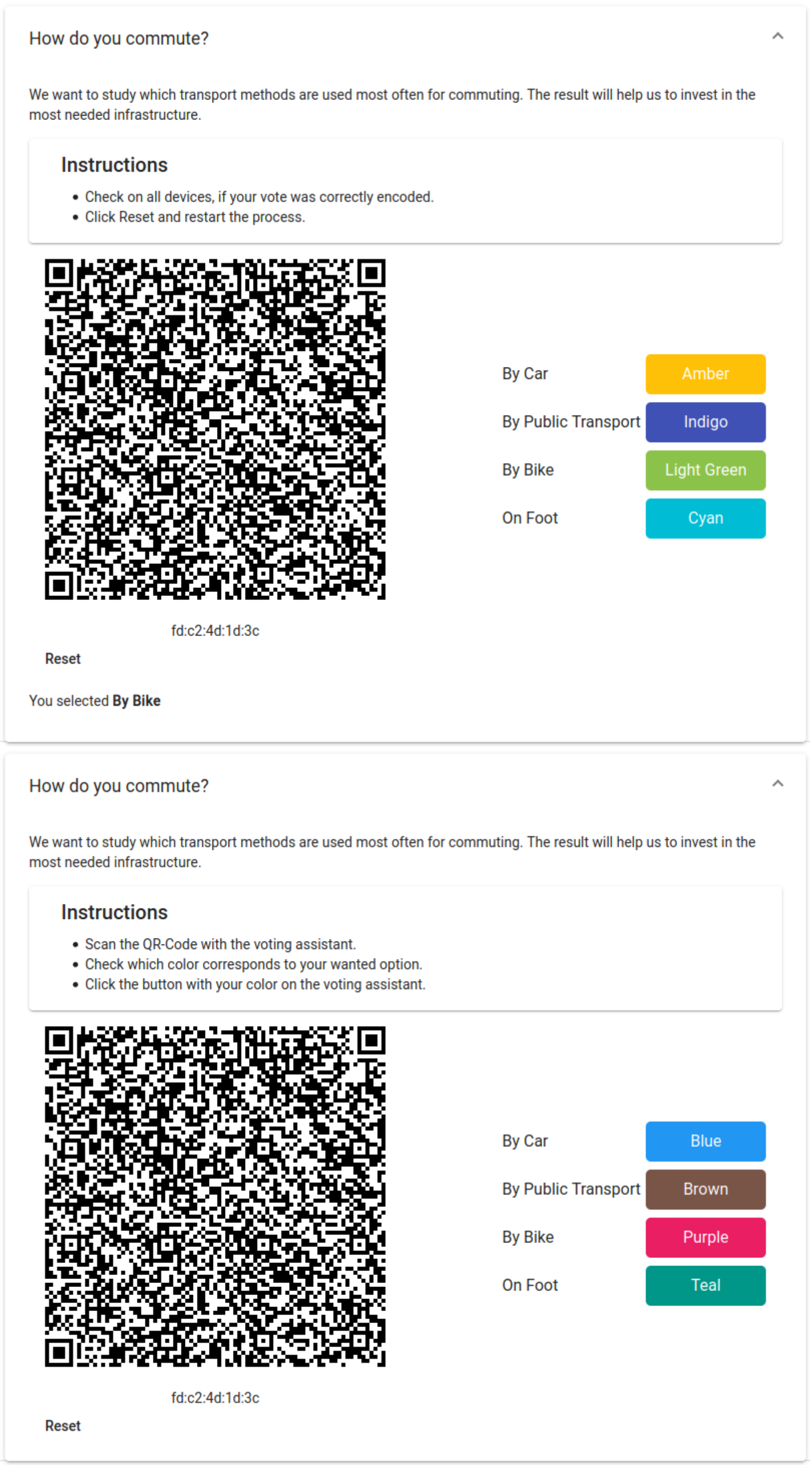
4.5 Deployment

To deploy ETHVote, Docker containers[8] are created for the database, the server, and the terminal. Docker Compose[9] is used to start all containers with one command (see Listing 4.2). Docker Compose also creates a local network between the docker containers and backs up the database to the host operating system.

For configuration, environment variables stored in a file `.env`, are used. This makes it easy to configure and deploy ETHVote on different environments, without having to adapt the code. It also ensures that all passwords and secrets used are only stored locally, and not for example in a git repository.

The voting assistant is an Android application, which can be built via script. The passwords needed to sign the application, are also set in `.env`. When the script is executed, it will copy the built application into the terminal container. This ensures that a voter always can download a voting assistant which works

[7] https://www.materialui.co/colors
[8] https://www.docker.com/resources/what-container
[9] https://docs.docker.com/compose/


How do you commute?

We want to study which transport methods are used most often for commuting. The result will help us to invest in the most needed infrastructure.

Instructions

- Check on all devices, if your vote was correctly encoded.
- Click Reset and restart the process.

By Car	Amber
By Public Transport	Indigo
By Bike	Light Green
On Foot	Cyan

fd:c2:4d:1d:3c

Reset

You selected **By Bike**

How do you commute?

We want to study which transport methods are used most often for commuting. The result will help us to invest in the most needed infrastructure.

Instructions

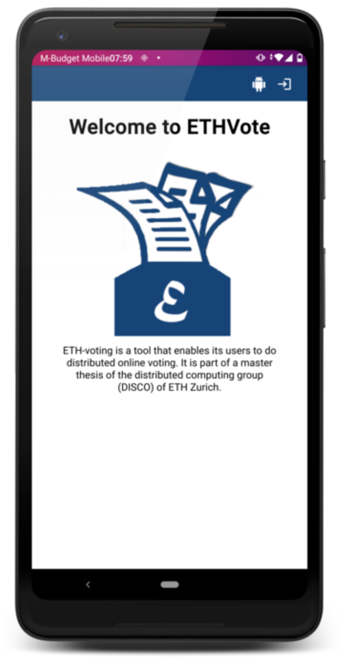
- Scan the QR-Code with the voting assistant.
- Check which color corresponds to your wanted option.
- Click the button with your color on the voting assistant.

By Car	Blue
By Public Transport	Brown
By Bike	Purple
On Foot	Teal

fd:c2:4d:1d:3c

Reset

Figure 4.2: Terminal before and after reset



Welcome to ETHVote

ETH-voting is a tool that enables its users to do distributed online voting. It is part of a master thesis of the distributed computing group (DISCO) of ETH Zurich.

Figure 4.3: Terminal on a mobile phone

with the current version of ETHVote. In Figure 4.3 the button to download the app can be spotted in the top bar. The button is already visible before the voter is authenticated, to prevent that the voter has to log in on the smartphone.

4.5.1 Initial Administrator

Newly registered voters have no special roles by defaul, but only administrators can add roles to voters. To ensure that ETHVote always has an administrator, the environment variable `ADMIN_SHIBBOLETH_IDS` can be set. As the name suggests a list of Shibboleth IDs is taken, and if a voter is registering and the Shibboleth ID is in this list, the administrator role is assigned to this voter.

4.6 Implementation Challenges

While some elements could be easily implemented because a codebase already existed, there were also more challenging elements. For example, making the whole system configurable with one single configuration file was quite tricky because the server, terminal, database, and voting assistant must listen to this file. While in the previous version each component had a different way of handling the configuration, there is now only one single place to configure the whole system.

Another tricky part was the QR code scanning. There exist a lot of libraries for this, but this made it difficult to choose the right one. And camera management is difficult in Android apps because there exist a lot of different devices.

While adding new endpoints to the server was easy, modifying the terminal was more difficult. To make the terminal more adaptable, a lot of code had to be refactored first. Also implementing a secure authentication mechanism, that is easy to use for the voter, was challenging. Checking the NIZKP on the server was another challenge because the server did not any mathematical computations at all before.

Chapter 5

Security

In this thesis, the security of the previous version of ETHVote [5] was improved. All security requirements holding in the previous version still hold. Therefore, the focus in this chapter is set to the improved parts.

5.1 Ballot-Privacy

As long the trustees follow the protocol, ballot-privacy is achieved, because all options are encrypted before they are sent to the server and the decryption factors can only be used to decrypt the product of votes, but not to decrypt single votes.

Ballot-privacy can be broken, when k trustees collude. For this k trustees must exchange their parts of the secret polynomial. With k points of the polynomial $s(0)$ can be computed, which is the private key of the encrypted votes.

5.2 Receipt-Freeness

While the new protocol makes getting a receipt of the vote more difficult, it is still possible. For this, a malicious terminal saves all r_i, used to encrypt the options, as a receipt. When the vote is published it encrypts all option primes once with every r_i. When the encryption matches the published encryption, it found the voted option.

To get the voted option of a vote with n options, an attacker needs at most n^2 tries. However, because n is very small, this does not matter.

5.3 Election-Fairness

It is not possible to receive any information about the voting outcome before the vote is closed, because the voting phases are strictly separated.

5.4 Authenticity

Authenticity is achieved because the terminal and the voting assistant must authenticate the voter. The server accepts only authenticated vote requests, and only as long as the vote is not cast.

5.5 Correctness

The attacks described in “ETHVote - A Distributed E-Voting Application" [5] during key derivation and decryption phase are still mitigated by the same countermeasures because those phases did not change.

Attacks during the voting phase are even more difficult to perform. While in the previous version the terminal must be trusted, the new protocol ensures correctness also when at least one of the terminal and the voting assistant is honest. The following two attacks are mitigated by the new protocol.

5.5.1 Malicious Data by the Terminal

While in the previous version sending an invalid option was already detected, the terminal could just send a wrong but valid option. To encounter this, in this version the voting assistant has to send the vote. And the new NIZKP as described in Section 3.5 forces the terminal to send valid ciphertexts for every option available, such that sending invalid options is still not possible.

The manipulation, the terminal can do is showing a wrong mapping to the voter. For example, it can send the mapping (``yes'' $\mapsto$ ``red'',``no'' $\mapsto$ ``blue'') to the server but display (``yes'' $\mapsto$ ``blue'',``no'' $\mapsto$ ``red'') to the voter. This falsifies the outcome, but an attacker has no benefit from it because the selection of the options is made on the voting assistant. Therefore, the probability to get voted is 50% for both options. Additionally, the voter notices the manipulation when they audit the vote.

5.5.2 Malicious Data by the Voting Assistant

The color chosen by the voter is the only data the voting assistant sends to the server. Therefore, the only manipulation is to send a different color than the chosen one. This manipulation has the same limitations as described in Section 5.5.1 where the terminal displays a wrong mapping. Namely, an attacker has no benefit from it and the manipulation is noticed when the voter audits their vote.

5.5.3 Collaboration Between Terminal and Voting Assistant

While neither a malicious terminal nor a malicious voting assistant can falsify the voting on their own, there exists an attack if both collude. For an attack, the terminal must be able to send the mapping of options to the voting assistant. With this information the voting assistant sends the color, corresponding to the preferred option, no matter which color the voter has chosen.

Since the terminal and the voting assistant are malicious, they also collude during the audit phase. This is necessary so the voter does not recognize the attack. But when both devices display the option wanted by the voter instead of the transmitted one, the voter will not notice the attack.

To mitigate this attack, the voting assistant and the terminal runs on physically different devices. This makes it difficult to communicate directly. The direct communication channel of the QR code cannot be used, because the voter could reset the vote after they scanned the QR code. But it is still possible for the terminal and the voting assistant to communicate with each other. For example, they could encrypt data with their secret key s_k.

To convince the user that a malicious terminal is the official one, a man-in-the-middle-attack must be performed. Therefore, ETHVote is only available through HTTPS. Imitating a mobile app is even more difficult. Each mobile application is signed. When the attacker presents the malicious app as an update, the operating system will prevent the user from installing the update, because it is signed with a different key.

If ETHVote is used for a security-relevant vote, the voting assistant app should also be published to the official app stores. When the voting assistant can be downloaded from the terminal, it is easy for a malicious terminal to serve a colluding voting assistant.

5.6 Individual Verifiability

Individual verifiability is fulfilled because all encrypted votes are published. Therefore, a voter can check if some encrypted vote was saved for their voter ID. Additionally, the voter can audit their vote before casting. With this, they challenge the system to prove that it encrypts correctly.

5.7 Universal Verifiability

Universal verifiability is fulfilled because all encrypted votes and decryption factors are published. To check that everything is correct, the decryption of the encrypted votes can be recomputed.

Chapter 6

Conclusion

ETHVote was secured by adding a new participant, the voting assistant. This made ETHVote correct even with an untrusted Terminal. To break correctness an attacker must now manipulate two applications, namely the terminal and the voting assistant, and convince the voters to use the manipulated devices.

6.1 Future Work

There still exist several options to improve ETHVote. While it is theoretically already possible to add up to 45 options to a vote, it is not very usable. Because the voter must be able to distinguish from 45 colors and scroll on a mobile phone until they find the wanted color. There is the possibility to extend ETHVote similar to the system used for Swiss national elections. Meaning that the voter first has to choose a color for the party of the candidate and after this a color for the candidate.

Another extension would be to allow multiple votes. Right now it is only possible to choose one option. For an election with n candidates for m seats, a voter should probably have m votes. Or it should even be possible to give multiple votes to one option. In Swiss national elections, a voter has several votes and can give either zero, one, or two votes to a candidate.

While the separation idea of PrivApollo [9] is already implemented, PrivApollo also shows a message for every data posted on the server on all devices. This helps the voter to ensure that the correct data was saved on the server. This addition would also be possible in ETHVote.

The voting assistant is now only an Android application. To allow more voters to participate in a vote an iOS application should also be available.

To improve confidence about ETHVote it would also be possible to verify the used protocol. For this, a tool like for example Tamarin[1] could be used.

[1] `https://tamarin-prover.github.io/`

Bibliography

[1] R. Haenni, R. E. Koenig, P. Locher, and E. Dubuis, "Chvote system specification," Cryptology ePrint Archive, Report 2017/325, 2017. [Online]. Available: https://eprint.iacr.org/2017/325

[2] "Post e-voting: system documentation," 2018. [Online]. Available: https://www.post.ch/-/media/post/evoting/dokumente/evoting-system-dokumentation.pdf?la=en&vs=3

[3] "Final report, public intrusion test (pit)," jun 2019. [Online]. Available: https://www.post.ch/-/media/post/evoting/dokumente/abschlussbericht-oeffentlicher-intrusionstest-post.pdf?la=en&vs=1

[4] S. Sommaruga, G. Parmelin, U. Maurer, A. Berset, I. Cassis, V. Amherd, and K. Keller-Suter, "Faktenblatt - vote électronique," jun 2020. [Online]. Available: https://www.bk.admin.ch/dam/bk/de/dokumente/pore/Faktenblatt_DE.pdf.download.pdf/Faktenblatt_DE.pdf

[5] F. Goldener, "Ethvote - a distributed e-voting application," Apr. 2019.

[6] J. Benaloh, "Ballot casting assurance via voter-initiated poll station auditing," in *Proceedings of the 2007 Electronic Voting Technology Workshop*, June 2007. [Online]. Available: https://www.microsoft.com/en-us/research/publication/ballot-casting-assurance-via-voter-initiated-poll-station-auditing/

[7] D. Gawel, M. Kosarzecki, P. L. Vora, H. Wu, and F. Zagorski, "Apollo - end-to-end verifiable internet voting with recovery from vote manipulation," Cryptology ePrint Archive, Report 2016/1037, 2016. [Online]. Available: https://eprint.iacr.org/2016/1037

[8] B. Adida, "Helios: Web-based open-audit voting," in *USENIX Security Symposium*, 2008.

[9] H. Wu, P. L. Vora, and F. Zagórski, "Privapollo - secret ballot e2e-v internet voting," in *Financial Cryptography and Data Security - FC 2019 International Workshops, VOTING and WTSC, St. Kitts, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, ser. Lecture Notes in Computer Science, A. Bracciali, J. Clark, F. Pintore, P. B. Rønne, and M. Sala, Eds., vol. 11599. Springer, 2019, pp. 299–313. [Online]. Available: https://doi.org/10.1007/978-3-030-43725-1_21

[10] J. Benaloh, “Verifiable secret-ballot elections,” Yale University, September 1987. [Online]. Available: https://www.microsoft.com/en-us/research/publication/verifiable-secret-ballot-elections/

[11] V. Cortier, P. Gaudry, and S. Glondu, *Belenios: A Simple Private and Verifiable Electronic Voting System*, 04 2019, pp. 214–238. [Online]. Available: https://hal.inria.fr/hal-02066930

[12] T. Elgamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[13] J. Camenisch and M. Stadler, “Proof systems for general statements about discrete logarithms,” ETHZ, Tech. Rep., 1997.

[14] M. Jones, J. Bradley, and N. Sakimura, “JSON Web Token (JWT),” Internet Requests for Comments, RFC Editor, RFC 7519, May 2015. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7519.txt

Appendix A

Acronyms

NIZKP	non-interactive zero knowledge proof
JWT	JSON Web Token