Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Artemis: Solving the secure platform problem for the Helios e-voting system

Bachelor's Thesis

Mike Boss

mboss@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

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

October 4, 2022

Acknowledgements

I would like to thank Darya Melnyk and Tejaswi Nadahalli for their guidance, interest, constructive advice, and the lengthy discussions held during the course of this thesis.

Abstract

Many remote e-voting systems have been proposed providing verifiability and security of the tallying phase. Helios [1] is one of the most well-known implementations of such a system. Its simplicity derived from the ideas around homomorphic tallying and voter-initiated auditing have made it a much-used system in academia. However, most of these systems including Helios fail to provide security guarantees against malicious voting devices.

Two newly proposed systems, BeleniosVS [2] and PrivApollo [3], solve the issues of verifiability and privacy in the context of malicious voting devices. This thesis describes the background, creation, and implementation of Artemis on top of Helios. Artemis is a simpler combination of the ideas in PrivApollo and BeleniosRF [4], the predecessor to BeleniosVS, providing security and privacy against malicious voting devices.

Contents

Chapter 1

Introduction

E-Voting promises to deliver great benefits over its analog counterpart in terms of speed, cost, and voter participation. Trust in the integrity of such systems is of key importance as e-voting opens up a whole new area of possible attacks against elections. Modern end-to-end verifiable systems provide proofs of correct operation openly verifiable by all participants. Once a vote reaches the tallying authority, a voter can be certain that their vote was correctly counted towards the total. The greatest weakness remaining in these kinds of systems is correctly casting a ballot. In an electronic setting, each device a voter uses to cast their vote could be compromised. This allows the attacker to alter the vote, not to vote, re-vote or steal the voter's credentials depending on the protocol in question.

This thesis explores solutions to the secure platform problem for the Helios [1] e-voting system in a remote setting, finally implementing a protocol adapted from multiple different solutions to address the issue.

Artemis, the protocol described in this thesis, achieves the correctness and privacy of a vote as well as receipt-freeness using multiple devices under the assumption that at least one of the devices is not corrupted. Artemis can extend any generic e-voting system that uses homomorphic encryption and the Benaloh Challenge [5], a simple cast or audit choice for the voter. The implementation is based on a mix-net-tallying version of Helios although the underlying protocol can be adapted to be used with homomorphic tallying as well.

Artemis is easier to implement than the two protocols it is based on, BeleniosRF [4] and PrivApollo [3], as it requires no changes in the tallying phase as well as minimal changes in the vote submission step. Artemis uses no complex cryptographic primitives, alternatively relying on the Benaloh Challenge to provide security as well as providing privacy using an adapted indirection of the one described in PrivApollo. It provides receipt-freeness by having the bulletin board use the re-encryption property of the underlying homomorphic encryption to obfuscate the relationship between the submitted vote and the re-encrypted vote as described in BeleniosRF. In contrast to Helios, the audit in Artemis is not done locally on one device but several devices and involves the bulletin board.

Chapter 2

Background

Artemis is a combination of the PrivApollo and BeleniosRF protocols. PrivApollo is a privacy-preserving version of its predecessor Apollo [6]. PrivApollo uses two active devices to provide security and privacy as long as both devices do not act maliciously. Both devices interactively create the vote denying either the ability to alter the vote or learn its contents. BeleniosRF is a variant of the Belenios [7] protocol providing receipt-freeness. This is achieved by having the bulletin board re-encrypt the vote before making it public. No device controlled by the voter is able to provide a receipt as they have no access to the randomness used by the bulletin board. Apollo and Belenios are extensions of the ideas implemented by Helios which in itself is a digital implementation of the protocol described in Simple Verifiable Elections [8].

2.1 Simple Verifiable Elections

Adequate security has long been theoretically achieved for e-voting systems. The usability and complexity of these systems were still outstanding issues. Simple Verifiable Elections [8] addressed the latter by describing a much simpler e-voting system that derives its simplicity from the way it achieves ballot casting assurance [9] using the Benaloh Challenge.

The Benaloh Challenge, sometimes called voter-initiated poll station auditing, lets the voter challenge the correctness of the encrypted vote by offering the option of auditing it. The voter may either choose to cast their vote or initiate an audit. The audit forces the device that encrypted the vote to provide the randomness used for the encryption. As a malicious device cannot know how many times a voter chooses to audit their vote before casting it, it is highly unlikely for such a device to succeed in corrupting the votes of multiple voters without being discovered.

Of key importance to the audit is the physicality of the vote itself as the device first prints the encrypted vote. Afterward it either signs the printed vote if the voter wants to cast it or prints the randomness used for its encryption.

The device therefore physically commits to the encrypted vote before the voter reveals how they want to proceed.

The device is unable to know if a vote will be used in an election or audited as the voter's identity is not revealed to the device in the vote creation step which precedes the cast or audit choice.

After all the votes have been cast the set of ballots is shuffled multiple times by different parties using mix-nets while for each shuffle providing zero-knowledge proofs of correctness. As all the votes were initially encrypted using the election authorities key which was generated in a shared fashion using threshold encryption it is not possible to decrypt any votes before the shuffles have been completed. At least the threshold number of trustees are needed to decrypt the votes.

2.2 Helios

Helios [1] is a web-based implementation of the concepts introduced in Simple Verifiable Elections. The original protocol described in the Helios paper uses mix-nets different from the actual implementation that uses homomorphic tallying [10]. As the Helios system is simpler compared to other systems of its time it is often used as a base system for expanding research in academic contexts. Helios has been used in various elections including the IACR's and ACM's [3]. This simplicity stems directly from its base concepts inherited from Simple Verifiable Elections.

2.2.1 Protocol outline

A voter uses the Ballot Preparation System (BPS) to homomorphically encrypt their choices using the election authority's key. The BPS offers the voter the choice to either seal their ballot or audit it by revealing the randomness used for encryption.

Only when a voter chooses to cast their ballot, instead of auditing, are they asked to authenticate themselves to the tallying authority providing proof of eligibility for the given election. The delayed authentication allows anybody to audit as many ballots as they want even if they are not eligible for voting in the election. This keeps the identity of the voter secret during the vote creation step.

After receiving all the votes, the tallying authority shuffles the encrypted ballots using a re-encryption mix-net proving the correctness of the shuffle by providing zero-knowledge proofs. A re-encryption mix-net is the simpler version of mix-nets compared to a decryption mix-net [11, 12]. The encrypted ballots can be re-encrypted using the homomorphic property of the encryption. If the shuffler is trusted, one shuffle would be enough to guarantee anonymity. Even a once re-encrypted ballot cannot be traced back to the original encrypted ballot in contrast to a decryption mix-net requiring at least as many shuffles as initial encryption layers. In a decryption mix-net every ballot is encrypted with all the trustees public keys in order. Every time a shuffle is performed by a trustee in a decryption mix-net they use their share to provide a partial decryption of the encrypted ballots. Only once all partial decryptions in order have been performed the ballots are completely decrypted.

2.2.2 Implementation

The Helios protocol was not completely implemented as described in the original paper. The vote creation step remained mostly the same in contrast to the tallying that changed to homomorphic tallying with zero-knowledge proofs instead of the originally described re-encryption mix-net.

Homomorphic tallying uses the additive homomorphic nature of the encrypted ballots to tally the result while encrypted. The encrypted choices are added together resulting in a single ciphertext containing the summation of all encrypted choices. After summation, the single ciphertext can be decrypted in a shared fashion revealing the result of the election. No single votes are ever decrypted therefore guaranteeing the privacy of the vote during the tallying phase. All steps are proved to be correct by providing zero-knowledge proofs. A voter can still be certain that their vote is correctly tallied even though their vote is never revealed by checking the proofs.

Homomorphic tallying is simpler as no complicated shuffles using mix-nets have to be performed. Instead, simply relying on the homomorphic nature of the encryption to tally all votes in a single step. As the single votes are never decrypted any malformed ballot could result in a malformed result either corrupting the result or allowing malicious voters to vote multiple times without being discoverable during the tallying step. Therefore voters have to provide zero-knowledge proofs to guarantee the correct forming of their ballots such as the validity of the selected choices and that the correct number of choices were selected.

2.2.3 Mix-net

Homomorphic tallying relies on zero-knowledge proofs to prove the correct forming of encrypted ballots. Not all types of elections can easily be encoded in a provable format allowing the use of corresponding ballots in a homomorphic tally. Approval voting is a natural fit for homomorphic tallying as the choice(s) with the most votes wins the election. A vote for a choice is encoded in an additive homomorphic system by either a 0 or 1. One needs to prove that the right number of choices have been encoded with either a 0 or 1 allowing the total to be a summation of these values.

In contrast, more complicated election modes such as ranked voting or party-list proportional voting are not easy to encode in a way usable by a homomorphic tallying system. On the other hand, a mix-net tallying system can produce results for any kind of election as there are no proofs required in the vote creation step. As a mix-net system completely decrypts all single decryptable votes a malformed vote is detected in plain-text form allowing for it to be simply discarded. The shuffle and following decryption of the votes are openly verifiable meaning that any malformed vote has been malformed before tallying.

The complete decryption of all votes allows for new attacks. In the "Italian Coercion Attack" [13] a coercer forces the victim to encode a specific order of all choices unimportant to the coercer in a ranked voting election. Therefore a coercer can circumvent some of the receipt-freeness guarantees provided by the e-voting system as they can check if their victim correctly voted the way they intended after the decryption.

Mix-nets are more complicated to implement, slower to compute a tally as multiple shuffles have to be performed, and allow for new attacks against the integrity of the election. Such tallying systems allow for any kind of election mode to be used in contrast to homomorphic tallying only working with simpler election types encodable in the correct format.

Mix-net support

Mix-net support was mentioned as a future goal of version 3.1 of the Helios e-voting system [10]. As of version 4.0 no mix-net support has been implemented. The basic data structures used in Helios support the eventual addition of mix-nets to the system. Zeus [14] and helios-server-mixnet [15] are two systems implementing mix-nets based on Helios created by two separate entities not affiliated with the original Helios designers.

1. **Zeus**

   Created for use in the governing councils of higher education institutions in Greece, Zeus was born out of the need to support single transferable vote (STV) elections. This election mode is not supported by homomorphic tallying. Therefore, the creators of Zeus re-implemented the main concepts of the original Helios paper using mix-nets on top of the Helios version 2 code base. Most of the cryptographic workflow remains the same, some organizational structures were changed.

2. **helios-server-mixnet**

   The helios-server-mixnet project created by Lee Yingtong Li implementing mix-net based approval and single transferable vote elections on top of the Helios code base. The mix-net implementation called Phoebus is taken from the Zeus code base. In contrast to Zeus, the Helios code base was not changed further than what was necessary to implement mix-net voting. Additionally, more authentication options were added. Support for homomorphic tallying has been disabled.

2.2.4 Attacks

Multiple attacks against the Helios system have been found and mitigated [10]. In its current form, the Benaloh Challenge performed during the vote creation step is a significant attack vector as it does not provide any strict security guarantees. This stems from the fact that the challenge is performed locally on the device allowing any corrupted device to deceive the voter by showing the correct information in place of the actual vote submitted by the device. This issue only appears in the remote e-voting setting as the device has full control over what the voter sees. In the booth setting the vote can be physically printed as described in Simple Verifiable Elections. No privacy guarantees can be given because a corrupted device will be able to directly observe all interactions of the voter, including their voting choices. Several solutions have been proposed to solve the security issue including the Apollo and Belenios systems as well as their extensions solving the privacy issue: PrivApollo and BeleniosVS.

2.3 PrivApollo

Apollo extends Helios by mitigating the corrupted device scenario by publicly committing to the vote before a possible audit. The audit is performed using multiple devices. The PrivApollo [3] protocol extends Apollo by adding privacy using a generated indirection between two interacting devices.

2.3.1 Apollo

Apollo [6] changes the audit to be performed on multiple devices called voting assistants unaware to the voting terminal, the device encrypting the vote. Therefore the audit is not performed locally on a single device, guaranteeing that if at least one device is not corrupted any attempt of alteration will be discovered. In each vote creation step performed by the voter, a session is created which all devices observe. The voting terminal commits its encrypted ballot to this session allowing all voting assistants to save it before the option of auditing is possible for the voter. If audit is chosen the voting terminal reveals the randomness used for encryption on the session. All voting assistants check if any possible encryptions of the choices using said randomness match the given encrypted ballot informing the voter about the result.

To hinder the voting terminal from casting a vote before the voter is convinced it is behaving honestly, cast codes and a lock-in code are provided to the voter over a separate channel not available to the voting terminal.

Cast codes and the lock-in code are generated by the registrar. The cast codes are entered when the voter decides to cast their vote. Once the voter is convinced their vote is correctly recorded on the bulletin board they can enter their lock-in code to make it final. Therefore, a malicious device is not able to deceive a voter by switching the ballot when a cast code is provided. If detected the voter can simply switch devices to use another cast code. As cast codes can only be used once the voter can be certain that the correct ballot is on the bulletin board before they enter their lock-in code.

The cast codes and lock-in code are sent to the registrar with the encrypted ballot. The registrar signs the encrypted ballot with the correct key if the provided cast codes and lock-in code are valid for the election. It then publishes the signed ballot on the bulletin board.

2.3.2 Privacy

Under Apollo, a corrupted voting terminal is unable to submit an altered ballot as long as one of the voting assistants behaves honestly. The voting terminal still creates the vote by itself, therefore, being able to observe all interactions by the voter including their choices. PrivApollo extends the voting assistant concept introduced by Apollo with an active voting assistant. The active voting assistant is directly involved in the vote creation step. The interaction between the active voting assistant and voting terminal guarantee privacy as long as they are not colluding.

This is achieved by the voting terminal generating an indirection in the form of a mapping between a permutation of the possible choices and a set of colors. It encrypts the choices with the tallying authorities public key. The complete mapping is encrypted with a secret symmetric key. The voting terminal commits to the encrypted mapping on the session. The symmetric key is only known to all voting assistants in addition to the session id.

The active voting assistant shows the voter the set of colors allowing the voter to select their choice corresponding to a certain color by observing the mapping on the voting terminal. The encrypted color choice is published on the session allowing all the voting assistants to observe it.

If the voter chooses to audit the voting terminal and active voting assistant reveal their randomness used for encrypting the permuted mapping as well as the color choice. The voting terminal and all voting assistants check if the choices were correctly encrypted.

Once all votes have been cast the tallying authority first has to solve the indirection of each vote before being able to compute the result. A mix-net based system has to be used as it is unclear how such an indirection is encodable for use in homomorphic tallying. The problem lies in the need to generate zero-knowledge proofs of correct forming of the ballots. As two parts by two devices are needed for a valid ballot, this is non-trivial. The mix-net first has to shuffle and re-encrypt the ballot pairs after which the color choice can be decrypted. This allows the indirection to be solved by matching the color choice with the colors in the mapping. In the second step, the mix-net decrypts the actual choices after first shuffling and re-encrypting them. The entity performing the shuffles provides zero-knowledge proofs of correctness for the shuffling and re-encryption.

Another version using two indirection steps exists which uses codes instead of colors. The active voting assistant receives the set of codes called inner codes from the voting terminal. It generates a second mapping between the inner codes and a new set of generated codes called vote-codes. The correct vote-code corresponding to the voter's choice is then entered on the voting terminal instead of in the color version on the active voting assistant. The tallying has to run the first step twice to solve both indirections. This mitigates a possible randomization attack by the active voting assistant for which it randomly submits a different color choice.

2.4 BeleniosRF

Helios-C [16], standing for Helios with credentials, extends Helios by a second authority called the registrar. The registrar provides the voter with a private key used for signing while publicly releasing the set of all corresponding public keys. Belenios is a more formal description combining the concepts introduced in Helios as well as its extension Helios-C. The system has been implemented using OCaml allowing for better formal verification. BeleniosRF extends Belenios adding the receipt-freeness property by having the bulletin board re-encrypt the encrypted ballot once received. BeleniosVS is an extension of BeleniosRF using a device to scan a pre-encrypted vote for submission by the device adding privacy as well as verifiability against a dishonest voting device. Both of these systems rely on a cryptographic primitive allowing a signed ciphertext to be re-encrypted while still being able to extract the signature from the re-encrypted ciphertext. As both systems use homomorphic tallying the zero-knowledge proofs have to be re-randomizable as well. This is achieved by the usage of Groth-Sahai proofs [17].

2.4.1 Helios-C

In Helios, the tallying authority is single-handedly in control of all submitted ballots. This allows the tallying authority to stuff the ballots with generated votes not submitted by voters as no one can check the eligibility of the voters except the tallying authority. Helios-C [16] introduces the concept of a second authority called registrar mitigating the ballot stuffing attack. The registrar provides all registered voters with private keys used for signing their encrypted ballots as well as making the set of the corresponding public keys publicly available. The tallying authority is unable to generate new votes as it does not have access to a valid private key needed for signing the encrypted ballot.

2.4.2 Belenios

Belenios [18] describes a complete e-voting system based on the concepts introduced in Helios as well as Helios-C. It is a homomorphic tallying system with the tallying authority generating the public key in a shared fashion using threshold encryption. The registrar provides the voters with signing keys. The correct forming of encrypted ballots is proven by the voters providing zero-knowledge proofs. The encrypted ballots are signed using the provided private signing key. Only ballots for a given voter with one distinct signature are accepted by the tallying authority. The tallying authority does not know the link between the verification key used to check the signatures and voters beforehand. Therefore, the tallying authority memorizes the voters submitting an encrypted ballot and the corresponding verification keys. Encrypted ballots are formed by encoding the choices as a 0 or 1 vector encrypting each entry. The voter proves that all entries are either 0 or 1 and that the sum over all entries fulfills the given conditions.

2.4.3 BeleniosRF

In Belenios a voter can use their encrypted ballot available on the bulletin board as a receipt by revealing the randomness used for encryption. Therefore, allowing another party to check that the encrypted choices using these randomness match with the available encrypted ballot. BeleniosRF [4], standing for Belenios with receipt-freeness, adds the property of receipt-freeness to Belenios by having the tallying authority re-encrypt the encrypted ballot submitted by the voter before publishing it on the bulletin board. As the voter has no knowledge of the randomness used by the tallying authority they are unable to prove how they voted. In Belenios each encrypted ballot is signed with a private key provided by the registrar to each eligible voter. Using homomorphic re-encryption would lead to the invalidation of the signature as the ciphertext changed. BeleniosRF uses a cryptographic primitive introduced in Signatures on Randomizable Ciphertexts [19] using Waters signatures [20] to still be able to extract the signature even after re-encryption. Therefore the voter, as well as auditors, can verify that the re-encrypted ballot has been correctly signed with a valid signature by extracting the signature. The re-encryption leads to the invalidation of all zero-knowledge proofs that would be used in the vote creation step by Belenios. The primitive uses Groth-Sahai proofs which in contrast to the normal zero-knowledge proofs used in Belenios are re-randomizable.

2.4.4 BeleniosVS

A malicious device can easily deceive a voter in Belenios by simply showing the voter's intended ballot while submitting a different one to the bulletin board. BeleniosVS [2] extends BeleniosRF by adding privacy and verifiability against a dishonest voting device. A voter is provided with a voting sheet generated by the registrar containing signed encrypted ballots, the public key of the tallying authority, the signature verification key, and the randomness used for the encryption. All of information is encoded as QR-codes. The ballots are encrypted using the tallying authorities public key and signed using the private signing key of the registrar.

To vote, the voter simply scans the QR-code corresponding to the intended choice allowing the device to cast the signed encrypted ballot. It is of utmost importance that the voter only scans the intended QR-code. Scanning multiple QR-codes provides the device with multiple valid signed encrypted ballots. This allows the device to perform a randomization attack. Voters can audit the provided vote sheet by scanning the whole sheet with a different device. The device obtains the encoded private signing key and the randomness used for encryption. The device checks the correctness of the signed encrypted ballots. The audit of the vote sheet can be delegated to a third-party as the voter is required to login to the voting platform.

2.5 Related work

There are many different categories of approaches to electronic voting. In addition to the described voter-initiated auditing of note are code voting and secure devices as well as a diverse cast of other approaches. Only a small subset of the solutions provide privacy in case of malicious voting devices and servers.

Different variations of code voting exist although the underlying principles are mostly the same [21, 22, 23, 24, 25, 26, 27, 28]. Voters receive code sheets before the start of the election over a secure pre-channel. The code sheets may contain two kinds of codes. The first kind is entered into the voting application instead of the voter's choice. After submitting their vote the voter receives a code over a secure post-channel. The second kind of code is used to check if the received code matches the submitted choice. If both types of codes are used the voter can be certain that the voting device has not learned their vote nor changed it. The systems differ in the types of codes used, how codes are computed, and how they are transferred. Most code voting systems assume an honest voting server. A malicious server could generate codes incorrectly, reply with the wrong code, and even change the vote during tallying depending on the protocol.

The most well-known usage of secure devices in e-voting is the system operated in Estonia [29]. A prevalent issue of secure devices is the distribution and access of all the voters to such a device in a trusted manner. In the case of Estonia the device is a mandatory id-card issued to each citizen. The card has signing and encryption capabilities used during voting. More advanced devices exist ranging from similar capabilities to touch enabled devices to select the voter's choice [30, 31, 32, 33, 34].

Besides these three well-known categories of e-voting systems more approaches exist. Some combine different ideas of the other categories whereas others use more advanced cryptographic techniques to achieve security [35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45].

Chapter 3

Artemis

The Artemis protocol is a combination of the concepts introduced in PrivApollo by Hua Wu, Poorvi L. Vora, and Filip Zagórski as well as the ideas in BeleniosRF created by Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. Artemis uses the Benaloh Challenge for verifiability against corrupt voting devices as well as the tallying authority. As the PrivApollo protocol changes the tallying phase Artemis instead uses the re-encryption property of the homomorphic ciphertexts to achieve privacy through a simpler indirection without changing the tallying step. It provides receipt-freeness through the same re-encryption by the tallying authority as described in the BeleniosRF system. The Artemis protocol can be adapted to be either used with mix-net tallying or homomorphic tallying. Artemis does not rely on the same visual cryptography introduced in BeleniosVS alternatively providing security through the device interactions described in PrivApollo.

3.1 Requirements

The main requirement for Artemis is to introduce security as well as privacy against malicious voting devices for the Helios e-voting system. Of key importance to the protocol is simplicity and compatibility with the Helios code base as one of the main goals of this thesis is the implementation of the protocol on top of the Helios system.

3.2 Participants

1. Tallying authority (TA): Manages the voter's login credentials. Tallies the cast ballots from the bulletin board in a publicly verifiable manner.
2. Registrar (R): Generates cast and lock-in codes for each user.
3. Bulletin board (BB): A secure append-only-authenticated-write and public-read access bulletin board available to all participants.

4. Voter (V): A voter who can read, write as well as generate and remember short strings.
5. Voting terminal (VT): The main voting device used to generate initial ballot encryptions, submit the generated indirection and for entering the cast and lock-in codes.
6. Active voting assistant (AVA): A secondary device on which the voter makes their choice. The device is used for privacy.
7. Voting assistant(s) (VA): Additional devices used for checking the correctness of VT, AVA, and BB.

3.3 Description

3.3.1 Setup

The tallying authority generates an asymmetric key pair in a shared fashion using threshold encryption in collaboration with multiple trustees. The public key is published on the bulletin board. The registrar generates two asymmetric key pairs as well as a set of cast codes and one lock-in code for each voter. One of the key pairs is the cast signing pair. The other one is used for the locking signatures. The registrar publishes the public keys on the bulletin board and distributes the codes over a secure channel to all voters before the election.

The voting terminal generates a symmetric key and creates a new session in conjunction with the tallying authority. The session id is appended by a short string entered by the voter. The voting terminal encodes this information in a QR-code. The voter scans the QR-code with at least one voting assistant that will be actively involved in the protocol. Optionally more passive voting assistants acting as auditors of the two active devices may be used. The voting assistants observe the session during the protocol by using the provided session id.

3.3.2 Vote creation

The voting terminal generates a mapping between choices and colors by encrypting a permuted set of all possible choices using the tallying authority's public key and a generated set of colors as seen in table 3.1. It publicly commits to a symmetrically encrypted version of the mapping on the bulletin board. Afterward, it displays the non-encrypted mapping to the voter.

The voting assistants decrypt the mapping containing the encrypted ballots and corresponding colors displaying the set of colors to the voter. The voter chooses the color matching their choice as visible on the voting terminal. The

Answer	**Color**
$\{Tony\}_{TA}$	🟦
$\{Alice\}_{TA}$	🟩
$\{Bob\}_{TA}$	🟥

Table 3.1: Mapping generated by the voting terminal of the choices [Alice, Bob, Tony] and colors [blue, green, red].

active voting assistant re-encrypts the encrypted ballot afterward committing to it publicly on the bulletin board.

The bulletin board re-encrypts the already re-encrypted ballot before publishing it. This provides receipt-freeness as neither voter controlled device has knowledge of the randomness used.

The voter may either choose to cast the encrypted ballot on the bulletin board or audit it. If audit is chosen, the voting terminal, active voting assistant, and bulletin board reveal the randomness they used. All devices including the voting assistants check the correctness of the encrypted ballot by following the encryption steps using the provided randomness. A voter may audit their devices any number of times. Each time a new session is started the vote creation step starts anew.

3.3.3 Vote submission

Once the voter decides to cast their encrypted ballot they enter a valid cast code previously received by the registrar on the voting terminal. The voting terminal sends the encrypted ballot with the cast code to the registrar which if the code is valid signs the encrypted ballot using the private cast signing key. The registrar sends the signed encrypted ballot back to the voting terminal that publishes the signed encrypted ballot on the bulletin board.

All devices check if the signed encrypted ballot is the same as the encrypted ballot published by the bulletin board before signing. If this is not the case the voter may switch devices using a different cast code on the other device. Once the correctly signed encrypted ballot has been published the voter enters their lock-in code on the voting terminal. The voting terminal sends the signed encrypted ballot to the registrar that once again signs the ballot this time using the private locking signing key. The registrar sends the twice signed version back to the voting terminal.

The voter enters their login credentials previously created with the bulletin board on the voting terminal. Finally the twice signed encrypted ballot is submitted using the login credentials. The bulletin board only accepts the encrypted ballot if it is signed with the locking key and is the same as the last signed encrypted ballot with the cast key. Each cast code may only be used once.

3.3.4 Tallying

Mix-net tallying

As a mix-net based tallying system does not require any proofs created in the vote creation step any votes cast as described can be shuffled and afterward decrypted. The shuffled votes can only be decrypted if at least trustees in the number of the threshold use their partial keys in combination for decryption.

Homomorphic tallying

For a homomorphic tallying system to be viable, the voter must produce proofs of the correct forming of their ballot. The described protocol significantly hardens this problem as the encrypted ballots are twice re-encrypted making any normal zero-knowledge proof invalid. Fortunately, Groth-Sahai proofs can be used as they are re-randomizable. Groth-Sahai proofs are computationally expensive. If generation on the voting terminal is deemed non-viable the encrypted ballots with proofs can be sent in a pre-generated form. This is done in BeleniosVS as the protocol uses the primitive described in Signatures on Randomizable Ciphertexts. The signatures are not needed in Artemis as any malformed ballot is detected during the audit step. On the bulletin board the re-randomized proofs can be checked to be valid. The tallying is done as in any other homomorphic tallying system.

3.4 Protocol

Artemis supports homomorphic and mix-net based tallying. For both types, the protocol is mostly the same except for the tallying phase. Homomorphic tallying requires proofs to be generated in the creation of the ballot-color-mappings.

3.4.1 Notation

The tallying authority's public key is denoted by TA, the registrar's public key by R.

Given an asymmetric key pair k, encryption of ciphertext c using the public key is denoted by $\{c\}_k$. Signing using the private key is denoted by $\{c\}_{k^{-1}}$. Homomorphic re-encryption is denoted by $\{c\}_k^n$ for a ciphertext encrypted n

times. Initial encryption is denoted by $\{c\}_k^1$. Any re-encryption increases the upper index by one.

3.4.2 Assumptions

All ballot encryptions are homomorphic and re-encryptable. The zero-knowledge proof belonging to the encrypted ballot used in the homomorphic tallying version can be adjusted to still be valid after re-randomization. This can be achieved using Groth-Sahai proofs. Any mix-net used is a re-encryption mix-net.

3.4.3 Protocol description

1. **Election Setup.**
   (a) TA generates an asymmetric key pair in shared fashion using threshold encryption with multiple trustees and publishes the public key $(_{TA})$ on BB.
   (b) R generates an asymmetric key pair, publishes the public key on BB and sends each voter multiple casting codes and one lock-in code.
   (c) Voters create login credentials with the bulletin board.
2. **Session Setup.**
   (a) V enters a short string on VT which is appended to the *sessionID*.
   (b) VT starts a session using a new *sessionID* appended with the short string.
   (c) VT displays the *sessionID* as well as a QR code encoding a newly generated secret symmetric key k_{secret} and the *sessionID*.
   (d) V checks that the short string is appended to the *sessionID*.
   (e) V scans the QR code with AVA and optionally with additional VAs.
   (f) VAs display the received *sessionID*. VAs find the session corresponding to the *sessonID* on BB.
   (g) V checks that all the *sessionID*'s match with the original.
3. **Ballot-color-mapping setup.**
   (a) VT obtains the list of choices from BB.
   (b) VT generates the ballots based on the list of choices encrypting them with TA's public key $(\{B_i\}_{TA}^1)$. If the tally is done homomorphically, Groth-Sahai proofs are generated to prove the correct forming of the encrypted ballot.

   (c) VT generates a mapping ($\{\{B_i\}^1_{TA}, c_i\}$) between a permutation of the list of encrypted ballots and a generated set of colors (c_i).
   (d) VT publishes the mapping ($\{\{B_i\}^1_{TA}, c_i\}_{k_{secret}}$) encrypted by the pre-shared secret symmetric key (k_{secret}).

4. **Ballot Choice.**
   (a) VAs obtain the ballot-color-mapping from BB decrypting it with the pre-shared symmetric key. AVA displays the colors to V.
   (b) V chooses the color on AVA corresponding to the choice they want to make as displayed on VT.
   (c) AVA re-encrypts the encrypted chosen ballot ($\{B_i\}^2_{TA}$) and submits it to BB.
   (d) BB re-encrypts the received ballot ($\{B_i\}^3_{TA}$) before making it publicly visible.

5. **Audit. (Optional)**
   (a) V initiates auditing from either VT or AVA. The respective device sends a message to BB that the ballot is used for auditing. BB informs the other device of the audit.
   (b) VT, AVA, and BB reveal their randomness used for encryption and re-encryption respectively on BB.
   (c) VT and VAs check if all encryption steps were performed correctly informing V about the result. BB is unable to learn the choice as the initial mapping was encrypted with the pre-shared symmetric key.
   (d) V checks if their choice was correctly recorded. If one of the devices is at fault they may change devices else they may report BB. If the ballot is correctly formed, they start again as the audited ballot is invalid.

6. **Cast.**
   (a) V enters a cast-code on VT. VT submits the encrypted ballot with the cast code to R.
   (b) If the cast code is valid R signs the encrypted ballot with the cast signing key sending the signed encrypted ballot to VT.
   (c) V enters their login credentials on VT. Using the login credentials VT publishes the signed encrypted ballot on BB.
   (d) VT and VAs display a message that a new ballot using a cast-code was published on BB. If the ballot is not the same as the previously encrypted ballot they warn V.
   (e) V enters their lock-in code on VT. VT submits the already signed encrypted ballot with the lock-in code to R.

(f) If the submitted encrypted ballot is signed with the cast signing key and contains the last used cast code R signs it again with the locking signature key sending the signed encrypted ballot to VT.

(g) VT publishes the received ballot on BB.

(h) VT and VA's display a message that the final ballot was published on BB.

7. **Tally.**

   Mix-net.

   (a) TA uses a mix-net to obfuscate any relation between Vs and their published ballots. TA generates proofs of correctness for the mixing and publishes them on BB. Mixing may be done multiple times by different entities.

   (b) The trustees each provide their partial decryption of the mixed encrypted ballots allowing the ballots to be decrypted if at least as many as the threshold partial decryptions are provided for each ballot. They provide proofs of correct decryption for all partial decryptions.

   (c) TA invalidates any ballot that does not correctly encode a valid voting choice.

   (d) TA tallies all available valid votes and reveals the final tally on BB.

   Homomorphic.

   (a) TA verifies all the zero-knowledge proofs provided with the encrypted ballots. All invalid ballots are publicly discarded.

   (b) TA reduces all encrypted ballots to a single encrypted result using the homomorphic property of the encryption.

   (c) The trustees each provide their partial decryption of the encrypted result allowing the result to be decrypted if at least as many as the threshold partial decryptions are provided. They provide proofs of correct decryption for all partial decryptions.

   (d) TA reveals the final decrypted tally publicly on BB including all partial decryptions and corresponding proofs.

Chapter 4

Implementation

Artemis has been implemented on top of the helios-mixnet-server [15] fork, a mix-net extension of Helios. It is not a complete implementation of the described theoretical protocol because of of time constraints and technical deviations. The foundation given by the helios-mixnet-server has been adapted to accommodate the interactive vote creation step between the two main devices, the voting terminal and active voting assistant, as described in Artemis.

4.1 Foundation

4.1.1 Considerations

Of key importance to the choosing of the foundation on which Artemis is implemented is simplicity as well as compatibility with the Helios code base for ease of implementation. As such the main two contenders are the Zeus [14, 46] e-voting system and the helios-mixnet-server implementation. The latter fits both properties better as it is the same mix-net implementation without the other structural adaptions made by the Zeus system. Another possibility would have been the extension of the current Helios version to support mix-nets as the path of mix-net support has changed since the helios-mixnet-server forked the project. As of time constraints, such an option has been disregarded even though such a contribution would have been of more value to the e-voting system.

4.1.2 Helios

The Helios code base consists of the back end, the voting booth front end and a verifier used for checking audited ballots. The back end is a Django server using a PostgreSQL database to store the election data. The voting booth front end consists of a single booth JavaScript class. The state of the class is updated allowing the application to be run as a single page application (SPA). As such the voting booth may be run completely offline only requiring contact with the server for the initial data transfer and vote submission. Communication between the back end and front end is handled over a REST API based on JSON formatted data. One may refer to the original Helios paper [1] as well as the technical documentation [10] for a more detailed description.

4.1.3 helios-mixnet-server

As an extension and fork of the Helios code base, the main workflow remains the same. The back end tallying workflow has been extended with a mix-net workflow as well as the Phoebus mix-net implementation included in the code. Several cryptographic changes have been made to the base data structures when used in the mix-net workflow to accommodate the mix-net implementation. The booth has been altered to allow for single transferable vote (STV) elections. A ballot is encoded as the vector of the encrypted choices either as an encrypted 0 or 1. Proofs are generated for each choice being either a 0 or 1 as well as the sum confirming to the constraints.

4.2 Adaptation

The Artemis protocol changes only the vote creation step and submission of the vote. The tallying step is not changed as the final vote is in the same form as a vote cast in a non-Artemis mix-net based system. In this implementation, the vote submission step has not been altered by reasons of time constraints allowing a malicious tallying authority to carry out ballot stuffing attacks. The encoding of the ballots is not adapted to Artemis and no alterations have been done to do so. This leads to a slow initial encryption phase as the helios-server-mixnet way of encoding the choices as 0-1-vectors is inefficient if all possible choices have to be encoded. Better suited for Artemis would have been to encrypt all choices individually as the current way of encoding scales non-linearly with the number of choices.

The implementation uses the same code base, data structures, and overall logic as the base system. It is written in a similar style although no considerations were made for backward compatibility for older browsers. The original code base goes back to 2009. The implementation uses modern JavaScript instead of the older primitives and language features used by the base system. For easier implementation the data structures originally implemented were used such as their BigInt implementation even though BigInt officially exists as a language feature for JavaScript as of two years now.

4.2.1 Election creation

A settable flag has been added to indicate the usage of the Artemis vote creation step. The flag is sent to all voting booths allowing them to show the voter the correct booth. There is no check if a voter completes the Artemis vote creation steps. If the vote submission had been correctly altered they would be forced to at least enter cast and lock-in codes though this still may be done without the device interactions. A malicious voting booth or server could trick a voter into performing a simpler protocol. Therefore, voters need to be informed about the steps they have to take beforehand to make sure they recognize any malformed protocol implementation. Technical enforcement is deemed implausible as a malicious voting terminal can show the voter a simpler protocol while simulating the device interactions to the tallying authority.

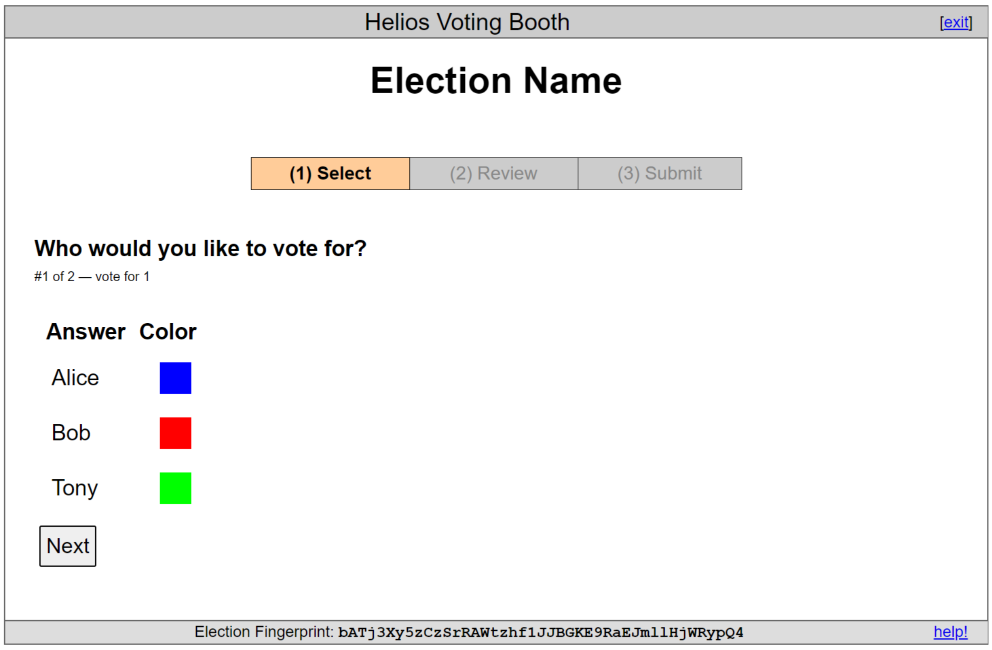
4.2.2 Vote creation step

Sessions have been introduced to the model as well as given REST APIs to interact with them. A voter is given the choice of their device being a voting terminal, active voting assistant, or voting assistant. Once voting terminal has been selected the voter is asked to enter a short string used against clash attacks. The terminal sends the string to the server initiating a session with the short string appended to its id. The voting terminal displays a QR code encoding the session id as well as a newly generated symmetric key. The voter opens the booth on a second device this time selecting active voting assistant. A QR scanner allows the voter to easily transfer the session id and symmetric key to the active voting assistant. Optionally, the voter may choose to add more devices as voting assistants.

The voting terminal creates a mapping of a permuted set of encrypted choices to a set of generated colors. The encryptions are created without proofs for better performance as the proofs are invalid as soon as the first re-encryption is applied. The voting terminal submits a symmetrically encrypted version of the mapping with the session id to the server. It displays the mapping between plain text choices and colors to the voter as seen in figure 4.1.

The active voting assistant is actively polling for the encrypted mapping. Once received, it decrypts it with the symmetric key displaying the set of colors to the voter as seen in figure 4.2. The voter can select their choice for each question by observing the mapping on the voting terminal and selecting the color corresponding to their choice on the active voting assistant. The active voting assistants re-encrypts all encrypted ballots mapping to the chosen colors submitting them with the session id to the server afterward.

Upon reception of the choices, the server re-encrypts all of them by transforming them to EGCiphertext objects, re-encrypting them and then encoding


Helios Voting Booth [exit]

Election Name

(1) Select | (2) Review | (3) Submit

Who would you like to vote for?

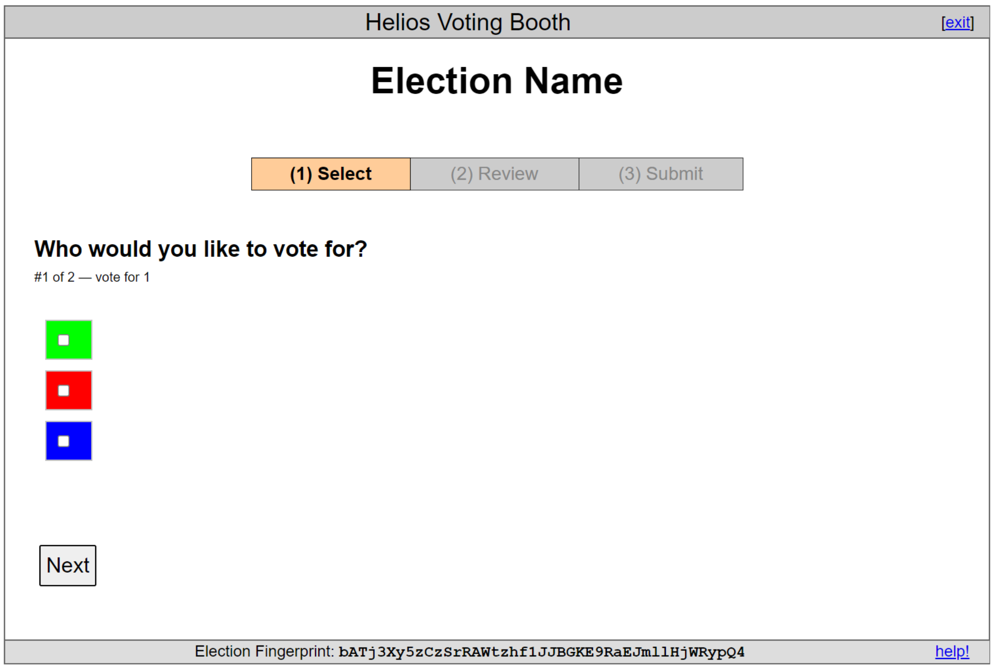
#1 of 2 — vote for 1

Answer	Color
Alice	🟦
Bob	🟥
Tony	🟩

Next

Election Fingerprint: `bATj3Xy5zCzSrRAWtzhf1JJBGKE9RaEJml1HjWRypQ4` help!

Figure 4.1: Mapping of plain text choices to colors displayed on the voting terminal.


Helios Voting Booth [exit]

Election Name

(1) Select | (2) Review | (3) Submit

Who would you like to vote for?

#1 of 2 — vote for 1

Next

Election Fingerprint: `bATj3Xy5zCzSrRAWtzhf1JJBGKE9RaEJml1HjWRypQ4` help!

Figure 4.2: Set of colors displayed on the active voting assistant.

the result back to JSON. The server stores the original encrypted choices for a possible audit. The active voting assistant is informed that the submission was successful. It displays the possibility of auditing to the voter.

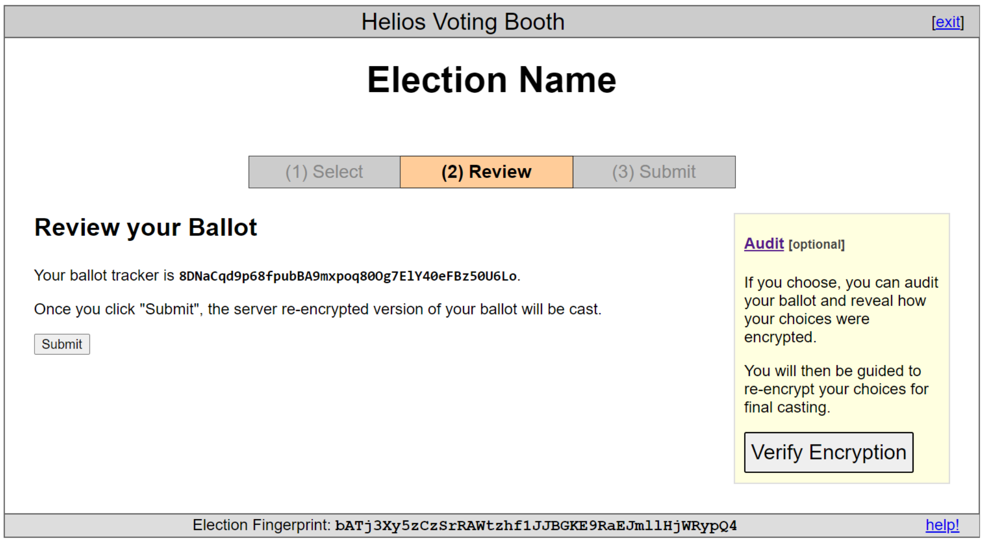
The voting terminal actively polls for the submitted choices. After receiving the server re-encrypted choices it displays the possibilities of either casting or auditing them as seen in figure 4.3. If either on the voting terminal or active voting assistant audit is chosen the respective device sends a start audit request to the server. If such a request is received, the server sets the audit flag on the session to true invalidating the ballot from future use. As the vote submission has not been implemented it is still technically possible to submit vote without a session. The usage of cast codes and a lock-in code mitigates this issue. All devices actively poll for the setting of the audit flag.

The passive voting assistants display to the voter that they have received the encrypted mapping and re-encrypted choices. Both are saved for a possible audit.

Once the audit has been initiated the active devices publish their randomness used for encryption and re-encryption respectively. The server allows for the original encrypted choices by the active voting assistant to be requested as well as the randomness used by the server for re-encryption. The devices publish the actual choices made as well to allow for a more efficient audit process. After receiving all needed data the devices perform the same audit process consisting of checking all steps in reverse. The server's re-encryption is first performed again on each device, then the re-encryption by the active voting assistant and finally the initial encryption by the voting terminal. If any of the steps fail the voter is informed that their audit did not succeed correctly. Only if all steps are valid the voter is shown the actual plain text choices encrypted by the ballot. The voter is offered to start a new session as the previous session is no longer usable after the audit flag has been set. The voter has to select restart on all devices. The result page can be seen in figure 4.4.

4.2.3 Vote submission

Upon the selection of the cast option by the voter, the voting terminal submits them to the server using the standard vote submission process provided by the base system. It first transforms the encrypted choices to the correct format adding additional needed data. Afterward, the voting terminal uses the provided cast REST API to submit the choices.



Review your Ballot

Your ballot tracker is `8DNaCqd9p68fpubBA9mxpoq80Og7ElY40eFBz50U6Lo`.

Once you click "Submit", the server re-encrypted version of your ballot will be cast.

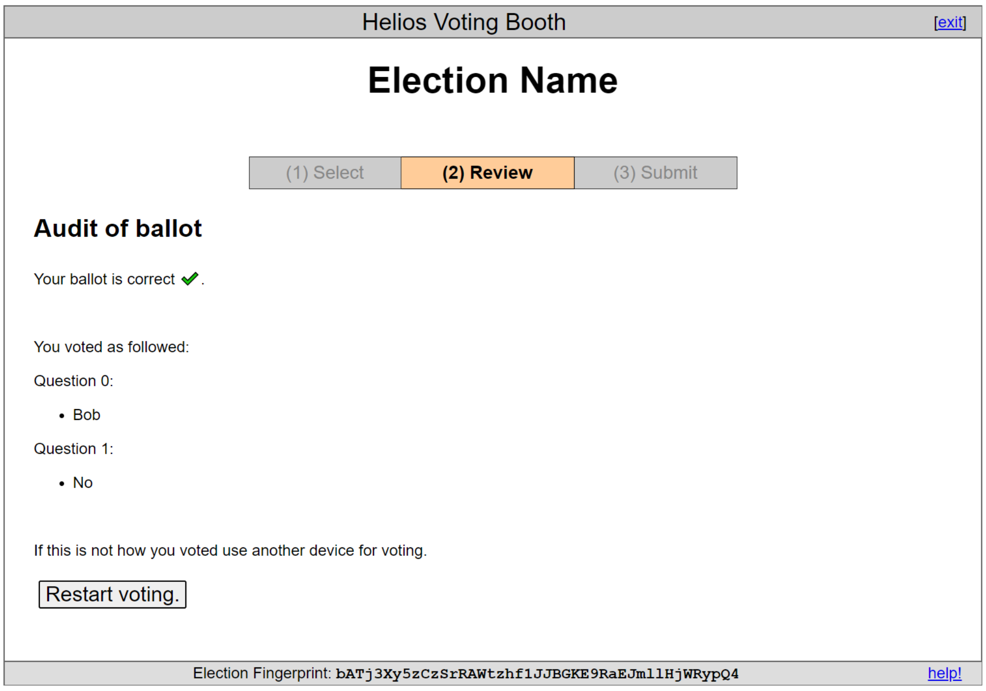
Audit [optional]

If you choose, you can audit your ballot and reveal how your choices were encrypted.

You will then be guided to re-encrypt your choices for final casting.

Election Fingerprint: `bATj3Xy5zCzSrRAWtzhf1JJBGKE9RaEJmllHjWRypQ4`

Figure 4.3: Choice between casting and auditing the server re-encrypted ballot on the voting terminal.



Audit of ballot

Your ballot is correct ✓.

You voted as followed:

Question 0:

- Bob

Question 1:

- No

If this is not how you voted use another device for voting.

Election Fingerprint: `bATj3Xy5zCzSrRAWtzhf1JJBGKE9RaEJmllHjWRypQ4`

Figure 4.4: Audit result page visible on all devices.

Chapter 5

Conclusions

Many practical remote e-voting implementations have been considered in the last few years in different settings. Most lacked secure foundations resulting in a loss of trust by the populace once severe issues were revealed. It is plausible to implement a secure e-voting system given that strong assurances are provided in the form of cryptographic and symbolic proofs of the underlying protocol as well as of its implementation. The level of carefulness one has to follow implementing such a system differentiates such a task strongly from standard software engineering practices.
Helios was one of the first systems of this type. The fundamental principle of simplicity followed throughout the design of Helios allowed for various new ideas and systems to be created improving and extending Helios. These improvements and extensions are fundamental to e-voting as Helios itself should not be used in real elections of importance. It does not provide security against malicious voting devices nor any privacy guarantees rather focusing on the validity of the tallying phase. Only once e-voting is secure during the vote creation, submission and tallying steps real-world elections should be carried out electronically as only then comparable properties to paper-based elections are achievable.

In recent times multiple such systems have been proposed leading e-voting in a more secure and private direction. This thesis presented Artemis a simpler protocol based on two such systems, PrivApollo and BeleniosRF / BeleniosVS, both providing security and privacy properties against malicious voting devices. Both systems are extensions of previously implemented systems with proofs of correctness though neither has been implemented as of now. This thesis describes the implementation of Artemis based on Helios. The implementation should not be used in real elections as no security analysis has been conducted of the implementation and it remains unfinished missing key security features. This holds as well for the underlying protocol which provides no cryptographic nor symbolic proofs.
All of these systems provide a great insight into how a secure e-voting system may be designed. One of the ideas or extensions of the concepts introduced by them may be thoroughly analyzed and implemented into a real remote e-voting system.

Bibliography

[1] B. Adida, “Helios: Web-based open-audit voting.” in *USENIX security symposium*, vol. 17, 2008, pp. 335–348.

[2] V. Cortier, A. Filipiak, and J. Lallemand, “Beleniosvs: Secrecy and verifiability against a corrupted voting device,” 2019.

[3] H. Wu, P. L. Vora, and F. Zagórski, “Privapollo–secret ballot e2e-v internet voting.”

[4] P. Chaidos, V. Cortier, G. Fuchsbauer, and D. Galindo, “Beleniosrf: A non-interactive receipt-free electronic voting scheme,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1614–1625.

[5] J. Benaloh, “Ballot casting assurance via voter-initiated poll station auditing.” *EVT*, vol. 7, pp. 14–14, 2007.

[6] D. Gaweł, M. Kosarzecki, P. L. Vora, H. Wu, and F. Zagórski, “Apollo–end-to-end verifiable internet voting with recovery from vote manipulation,” in *International Joint Conference on Electronic Voting*. Springer, 2016, pp. 125–143.

[7] V. Cortier, P. Gaudry, and S. Glondu, “Belenios: a simple private and verifiable electronic voting system,” in *Foundations of Security, Protocols, and Equational Reasoning*. Springer, 2019, pp. 214–238.

[8] J. Benaloh, “Simple verifiable elections.” *EVT*, vol. 6, pp. 5–5, 2006.

[9] B. Adida and C. A. Neff, “Ballot casting assurance.” *EVT*, vol. 6, p. 7, 2006.

[10] “Helios documentation,” accessed: 2019-12-18. [Online]. Available: https://documentation.heliosvoting.org/

[11] D. Chaum, “Untraceable electronic mail, return addresses and digital pseudonyms,” in *Secure electronic voting*. Springer, 2003, pp. 211–219.

[12] C. A. Neff, “A verifiable secret shuffle and its application to e-voting,” in *Proceedings of the 8th ACM conference on Computer and Communications Security*. ACM, 2001, pp. 116–125.

[13] J. Otten, “Fuller disclosure than intended,” *Voting matters*, vol. 17, p. 8, 2003.

[14] G. Tsoukalas, K. Papadimitriou, and P. Louridas, “From helios to zeus,” *USENIX Journal of Election Technology and Systems (JETS)*, vol. 1, no. 1, pp. 1–17, 2013.

[15] L. Y. Li, “helios-server-mixnet,” accessed: 2019-12-27. [Online]. Available: https://yingtongli.me/git/helios-server-mixnet/

[16] V. Cortier, D. Galindo, S. Glondu, M. Izabachene *et al.*, “A generic construction for voting correctness at minimum cost-application to helios.” *IACR Cryptology EPrint Archive*, vol. 2013, p. 177, 2013.

[17] J. Groth and A. Sahai, “Efficient non-interactive proof systems for bilinear groups,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 2008, pp. 415–432.

[18] S. Glondu, V. Cortier, and P. Gaudry, “Belenios–verifiable online voting system,” 2015.

[19] O. Blazy, G. Fuchsbauer, D. Pointcheval, and D. Vergnaud, “Signatures on randomizable ciphertexts,” in *International Workshop on Public Key Cryptography.* Springer, 2011, pp. 403–422.

[20] B. Waters, “Efficient identity-based encryption without random oracles,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 2005, pp. 114–127.

[21] J. Helbach and J. Schwenk, “Secure internet voting with code sheets,” in *International Conference on E-Voting and Identity.* Springer, 2007, pp. 166–177.

[22] Y. Desmedt and S. Erotokritou, “Towards usable and secure internet voting.”

[23] D. Galindo, S. Guasch, and J. Puiggali, “2015 neuchâtel’s cast-as-intended verification mechanism,” in *International Conference on E-Voting and Identity.* Springer, 2015, pp. 3–18.

[24] A. Brelle and T. Truderung, “Cast-as-intended mechanism with return codes based on pets,” in *International Joint Conference on Electronic Voting.* Springer, 2017, pp. 264–279.

[25] R. Oppliger, “How to address the secure platform problem for remote internet voting,” *Sis*, vol. 2, pp. 153–173, 2002.

[26] R. Oppliger, “Addressing the secure platform problem for remote internet voting in geneva,” *Chancellery of the State of Geneva*, 2002.

[27] M. Backes, M. Gagné, and M. Skoruppa, “Using mobile device communication to strengthen e-voting protocols,” in *Proceedings of the 12th ACM*

workshop on Workshop on privacy in the electronic society. ACM, 2013, pp. 237–242.

[28] S. Heiberg, H. Lipmaa, and F. Van Laenen, "On e-vote integrity in the case of malicious voter computers," in *European Symposium on Research in Computer Security.* Springer, 2010, pp. 373–388.

[29] E. Maaten, "Towards remote e-voting: Estonian case," *Electronic Voting in Europe-Technology, Law, Politics and Society*, vol. 47, pp. 83–100, 2004.

[30] H. Lipmaa, "A simple cast-as-intended e-voting protocol by using secure smart cards." *IACR Cryptology ePrint Archive*, vol. 2014, p. 348, 2014.

[31] A. Brioso, "Virus resistant e-voting."

[32] R. Haenni and R. E. Koenig, "Voting over the internet on an insecure platform," in *Design, Development, and Use of Secure Electronic Voting Systems.* IGI Global, 2014, pp. 62–75.

[33] R. Joaquim and C. Ribeiro, "Codevoting: Protecting against malicious vote manipulation at the voter's pc," in *Dagstuhl Seminar Proceedings.* Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2008.

[34] C. Gallegos and D. Shin, "A novel device for secure home e-voting," in *Proceedings of the 2015 Conference on research in adaptive and convergent systems.* ACM, 2015, pp. 321–323.

[35] R. Haenni, R. E. Koenig, and E. Dubuis, "Cast-as-intended verification in electronic elections based on oblivious transfer," in *International Joint Conference on Electronic Voting.* Springer, 2016, pp. 73–91.

[36] Y. Desmedt and S. Erotokritou, "Making code voting secure against insider threats using unconditionally secure mix schemes and human psmt protocols," in *International Conference on E-Voting and Identity.* Springer, 2015, pp. 110–126.

[37] J. Clark and U. Hengartner, "Selections: Internet voting with over-the-shoulder coercion-resistance," in *International Conference on Financial Cryptography and Data Security.* Springer, 2011, pp. 47–61.

[38] R. Joaquim, P. Ferreira, and C. Ribeiro, "Eviv: An end-to-end verifiable internet voting system," *Computers & Security*, vol. 32, pp. 170–191, 2013.

[39] M. Kutyłowski and F. Zagórski, "Verifiable internet voting solving secure platform problem," in *International Workshop on Security.* Springer, 2007, pp. 199–213.

[40] M. Volkamer, A. Alkassar, A.-R. Sadeghi, and S. Schulz, “Enabling the application of open systems like pcs for online voting,” in *Proc. of Workshop on Frontiers in Electronic Elections*, 2006.

[41] D. Chaum, A. Florescu, M. Nandi, S. Popoveniuc, J. Rubio, P. L. Vora, and F. Zagórski, “Paperless independently-verifiable voting,” in *International Conference on E-Voting and Identity.* Springer, 2011, pp. 140–157.

[42] G. S. Grewal, M. D. Ryan, L. Chen, and M. R. Clarkson, “Du-vote: Remote electronic voting with untrusted computers,” in *2015 IEEE 28th Computer Security Foundations Symposium.* IEEE, 2015, pp. 155–169.

[43] A. Juels, D. Catalano, and M. Jakobsson, “Coercion-resistant electronic elections,” in *Towards Trustworthy Elections.* Springer, 2010, pp. 37–63.

[44] B. L. K. Kim, “Receipt-free electronic voting through collaboration of voter and honest verifier,” 2000.

[45] M. Clarkson, S. Chong, and A. Myers, “Civitas: A secure remote voting system,” in *Dagstuhl Seminar Proceedings.* Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2008.

[46] “Zeus,” accessed: 2019-12-27. [Online]. Available: https://zeus.grnet.gr/zeus/