Artemis: E-Voting with Indirection and Re-encryption

Abstract. Secure and private e-voting on user-owned devices is a challenging task when building a large-scale e-voting system. In this paper, we propose a new voting scheme called *Artemis*, that combines features from two existing voting schemes - PrivApollo and BeleniosRF. Artemis achieves verifiability and privacy against a malicious voting server as well as against the possibly malicious voting device. The indirection created between two active devices introduced in PrivApollo for ballot secrecy is simplified by the usage of re-encryption steps described in BeleniosRF. These steps additionally allow for the adaption of PrivApollo to support homomorphic tallying and provide receipt-freeness. Artemis moreover features ranked election capabilities. This novel combination of indirection and re-encryption as well as ranked voting has been implemented on a fork of the well-known Helios E-Voting system as a proof of concept.

Keywords: E-Voting · Malicious Device · Verifiability · Privacy · Helios.

1 Introduction

Voting is the central part of democracy. It allows individuals to contribute to a political change in their country. Unfortunately, the current physical system is far from perfect: in only a few countries it is possible to verify that a vote has been counted by overseeing the counting process. There are also concerns about security and lack of convenience. As an example, roughly 43 percent of eligible voters in the US did not vote in the 2016 presidential election [18]. E-voting aims to solve many drawbacks of physical voting: it uses a remote e-voting system, where voters have the convenience to vote over the internet; it provides verifiability of the note, such that every voter can make sure that the vote has been counted; and it allows to verify the correctness of the election by letting the authority provide proofs for every election step.

A secure remote e-voting scheme presents itself as a harder challenge to create than its non-remote counterpart. Since the voters participate from home, they must use their own devices to cast their vote over the internet. Such voting schemes need to resemble physical voting and therefore require secure authentication and the possibility to submit a ballot anonymously. Special hardware that prints physical information or helps to generate keys for cryptographic schemes is costly and may not be widely available. Physical interactions with the voting authorities to prove a voters' identity would have to be completed prior to the start of the election. In such a case, a voter may as well vote physically. With growing digitalization and a growing interest in expressing opinions online, it seems inevitable that online voting will replace physical elections in the future.

Modern end-to-end verifiable remote e-voting systems provide proofs of correct operation. Such proof can be verified by every participant in the system. Once a vote reaches the bulletin board, a voter can be certain that her vote was correctly counted towards the total. Just like with electronic voting machines at polling stations, the greatest weakness remaining in fully online systems is that a malicious voting device will not cast a ballot correctly. Each device that is used to cast a vote could be compromised. Many voting schemes have been proposed in the literature that provide verifiability even if a corrupt device is present. However, only a few of them provide privacy. If they do, they usually do this at an additional cost of increasing complexity of an election or constricting possible voting schemes. For remote e-voting schemes to be plausible alternatives to their non-remote counterparts both properties have to be fulfilled without excessive trade offs.

This paper presents *Artemis* – a protocol that helps to extend the electronic voting platform Helios by verifiability and privacy against malicious devices. The protocol is based on two known protocol schemes: PrivApollo and BeleniosRF. PrivApollo uses simple cryptographic schemes and therefore does not satisfy verifiability. BeleniosRF, on the other hand, manages to satisfy verifiability and privacy, but at a cost of complex cryptographic primitives and a physical interaction with the election authorities. We therefore use a combination of two devices to guarantee privacy from PrivApollo and the re-encryption provided by BeliniosRF on the tallying side in order to implement a system that

- guarantees verifiability and privacy of a vote, even if a malicious device has been used for vote casting, and
- is a simple protocol that does not require an interaction between the voter and the voting authority ahead of the election by using multiple private devices, and
- has ranked election capabilities with multiple ranking methods in order to support a broader set of elections.

2 Background

The Helios e-voting system [4] is one of the first widely used systems achieving verifiability. As a digital remote implementation of the concepts introduced in Simple Verifiable Elections [5] it stood out amongst its complex contemporary counterparts as a more accessible system. Most of the simplicity is derived from the cast-audit choice presented to the voter after creating their ballot called Benaloh Challenge [6]. Using a physical voting booth as described in Simple Verifiable Elections, the challenge is secure. However, in a remote context, it is not. A malicious voting device is able to submit an altered ballot while presenting the voter with the correct auditing information. Additionally, vote privacy is not guaranteed as the malicious device learns the voter's choice and credentials. To solve these shortcomings Helios has been extended multiple times. Two branches of extensions are the Belenios [15] and Apollo [13] systems culminating in the BeleniosVS [11] and PrivApollo [29] schemes. Both BeleniosVS and PrivApollo provide verifiability and privacy against malicious voting devices. In addition, BeleniosVS provides receipt-freeness first introduced in BeleniosRF [8].

BeleniosVS uses a complex cryptographic primitive called Signatures on Randomizable Ciphertexts [7] to achieve both verifiability and privacy. This primitive allows a signed ciphertext to be re-encrypted preserving both the encryption and signature while randomizing both. A voting sheet with the signed encrypted ballots for each possible choice encoded as QR codes is sent to each voter. A voter simply scans the desired choice. Re-encryption of the ballot by the voting device guarantees privacy. As signature checking is preserved, verifiability is guaranteed. The ballot is re-encrypted again before being published on the bulletin board providing receipt-freeness. Signatures on Randomizable Ciphertexts allows the voting procedure to be straightforward from a voter's perspective.

In contrast, PrivApollo uses simple cryptographic primitives. It relies on indirection created between two active voting devices. The first device creates a mapping between encrypted choices and a set of generated colors. The second device shows the set of colors to the voter. The voter selects the color corresponding to their choice allowing the second device to submit the encrypted choice. The indirection guarantees privacy given non-conspiring devices as neither sees the full information encoding the voter's intention. Optional passive voting devices may be used to verify the behavior of the two active devices. The indirection is resolved in a private and verifiable manner in an altered mix-net based tallying phase.

The goal of Artemis is to extend Helios with verifiability and privacy against malicious voting devices given the requirements of simplicity, generality, and compatibility with Helios. We do this by adding to Helios the following:

- PrivApollo's idea of indirection to thwart malicious devices.
- BeleniosRF's tallying side re-encryption to provide homomorphic verifiable receipt-free tallying.

As the current Helios implementation is based on homomorphic tallying we altered the indirection created during PrivApollo to be used homomorphically using the cryptographic primitive described in BeleniosRF. The primitive called Signatures on Randomizable Ciphertexts proved to be complex and computationally expensive. However, the idea of using ballot re-encryption was used to allow the tallying phase of PrivApollo to consist of a standard mix-net based approach instead of having to be altered to resolve the indirection. In addition, re-encryption on the bulletin board provides receipt-freeness as described in BeleniosRF. We implemented Artemis on top of a mix-net based extension of Helios. The corresponding code has been published here: `https://bit.ly/2FPJ52h` .

3 Related Work

The effort of conducting votes with polling stations or postal voting has encouraged many countries, including Estonia, Norway, and Switzerland, to evaluate the introduction of electronic voting. Electronic voting does however imply that people would have to use their personal computer to vote, which is a major source of vulnerabilities in a system. There are several ways to approach this so-called *secure platform problem* [25], some of which we present next.

A simple way to design a distributed system that satisfies authenticity is to use blind signatures [9]. In the corresponding *anonymous broadcast systems* [12,22], the voters use blind signatures in order to authenticate their vote by a voting authority, and then send the vote in plaintext together with a signature to a public bulletin board. While the system is fully distributed due to the separation of the voting authority and the bulletin board, the authentication step only works if the broadcast channel is anonymous, which is very difficult to implement in practice.

A system that guarantees the privacy of a voter against malicious voting clients is called *code voting* [10,23,19,26]. In this system, a central authority issues personalized code sheets that are sent to the voters via a private channel. This enables the voter to enter codes instead of candidates in the client. While the voting client does not know the content of the vote entered, it can still corrupt the whole ballot and thus violate the integrity of the corresponding vote. In order to prevent this step, a system needs to satisfy verifiability properties.

Modern large scale votes make use of so-called *Mix-Net systems*. Next to authenticity and privacy, they are also specifically designed to satisfy individual and universal verifiability. The Estonian [27], Norwegian [14] and Swiss [17] systems are examples for such e-voting systems. In Mix-Net systems, there are several authorities. Voters encrypt their ballot with each of the authority's public keys, sign and submit the vote to the bulletin board. Authentication can be done using the voter's signature. After the voting phase, the authorities shuffle and decrypt all the votes one after another. Most such systems use three authorities for decryption and argue that the shuffling step between the authorities is sufficient to provide anonymity of the voters.

Mix-net support was also mentioned as a future goal of the Helios e-voting system [2]. As of version 4.0, no mix-net support has been implemented. The basic data structures used in Helios support the possibility to extend the system to mix-nets. Two systems, Zeus [28] and helios-server-mixnet [21], were later implemented based on Helios created by two separate entities not affiliated with the original Helios designers. Zeus was born out of the need to support single transferable vote (STV) elections. 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. The helios-server-mixnet project created by Li implements 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.

Besides Helios, there are also other open source implementations of voting systems. One example is ElectionGuard [3], an election system developed by Microsoft that uses homomorphic threshold cryptography to tally the votes. ElectionGuard has been successfully tested during an election in a small Wisconsin town in the USA [24]. Another system is the tally-hiding system named Ordinos [20]. Tally-hiding is a method, where an e-voting system does not publish the final tally after the election, but only the necessary results, such as the winner or the k best candidates. This is done to improve the anonymity and reduce the bias of various elections.

4 Protocol Description

In this section, we provide a detailed description of the Artemis protocol. We first give a high-level overview and introduce the participants as well as the notation used in the protocol. In order to explain the full protocol, we will split the description into four phases: the setup, vote creation, vote submission, and tallying phase. The description of each phase contains an explanatory overview and more formal descriptions of individual protocol steps.

4.1 Participants

The participants can be split into two separate groups: authorities and voters. There are two authorities, the registrar and tallying authority. Using two separate authorities mitigates against the problem of ballot stuffing. The public bulletin board is an independent functional unit but is considered to be controlled by the tallying authority. The voters are in control of at least two active devices called the voting terminal and active voting assistant. Optionally, more passive voting assistants may be used.

The names of the participants match the participants described in PrivApollo, except for the renaming of the election authority to the tallying authority and the voting assistant(s) to the passive voting assistant(s). This is done to better reflect their respective roles in the protocol. The name voting assistant (VA) is used in the protocol to describe the union of both, the active and passive voting assistants.

1. *Registrar* (R): Distributes cast codes and lock-in code to each voter. Signs an encrypted ballot if a correct cast or lock-in code is provided by a voter.
2. *Tallying authority* (TA): Manages the voter's login credentials. Tallies the cast ballots from the bulletin board in a publicly verifiable manner.
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 the 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. This device is used for privacy.
7. *Passive voting assistant(s)* (PVA): Additional devices used for checking the correctness of VT, AVA, and BB.

4.2 Overview

Before the election, all eligible voters must be registered to vote with the registrar and have created login credentials with the tallying authority. The two authorities publish their public keys on the bulletin board. The registrar sends a list of multiple codes, so-called cast codes, and one single lock-in code to each voter.

A voter uses two active devices, the voting terminal and the active voting assistant, to cast a ballot in an election. Optionally, a voter may choose to use more passive voting assistants to verify the correct behavior of the two active devices. The voting terminal and the voting assistants are linked by the voter scanning a QR code on the terminal with the assistant. For security, this QR code also contains a symmetric key that can be used to encrypt data between these two devices.

The voting terminal submits a symmetrically encrypted mapping between a permuted set of the possible choices and a set of arbitrarily generated colors. The choices are encrypted with the tallying authorities' public key. The voting terminal shows the non-encrypted mapping to the voter. The colors are then displayed on the active voting assistant. The voter selects the color corresponding to their choice by observing the mapping on the voting terminal. The active voting assistant re-encrypts the chosen ballot before submitting it to the bulletin board. The bulletin board re-encrypts the ballot before publishing it for receipt-freeness.

The voter either casts the ballot or audits it. All devices offer the voter the choice of auditing the published ballot. To audit, the two active devices and the bulletin board reveal the randomness used for encryption allowing the voter to check on all devices if the correct choice has been submitted. This check has to be done by the voter by encrypting (with the tallying authority's public key) their vote with the revealed randomness, and checking if it matches the vote published on the bulletin board.

Casting the ballot requires one of the cast codes the voter received from the registrar. The voting terminal submits the encrypted ballot and cast code to the registrar. If valid, the registrar signs the ballot and sends it back to the voting terminal. All devices check that the same ballot as before has been submitted. If the voter wants to finalize this vote, they enter their lock-in code on the voting terminal. The ballot is signed again by the registrar. The voter enters their login credentials allowing the voting terminal to publish the final twice signed encrypted ballot on the bulletin board.

After the voting ends, the tallying authority uses either a mix-net based approach or homomorphic tallying to publicly and verifiably tally the votes. To use homomorphic tallying, proofs of correct ballot creation have to be submitted during the vote creation.

4.3 Notation

All participants are referred to by their abbreviations (R, TA, BB, V, VT, AVA, PVA) in the protocol steps. The tallying authority's public/private key pair is denoted by TA/TA^{-1}, the registrar's key pair by R/R^{-1} respectively. Initial encryption of ciphertext c using the public key k is denoted by $\{c\}_k^1$. Homomorphic re-encryption is denoted by $\{c\}_k^n$ for a ciphertext re-encrypted $n-1$ times after the initial encryption. Signing with the private key k^{-1} is denoted by $\{c\}_{k^{-1}}$.

4.4 Setup

The setup phase starts prior to the election and ends before the voter creates his ballot. It includes key generation, code distribution by the registrar, session setup, and linking of the voting devices. It is assumed that voters have registered with the registrar and created login credentials used to publish the final ballot on the bulletin board beforehand.

The setup of the election entails both authorities publishing their public keys and the registrar distributing the necessary information to the voters as described in Protocol 1. 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. One of the key pairs is the cast signing pair. The other one is used for the lock-in signatures. Cast codes and lock-in codes are used by a voter to prevent a malicious device from switching their ballot. The registrar publishes the public keys on the bulletin board. The registrar then generates multiple cast codes and one lock-in code for each voter. These codes are distributed over a secure channel to all voters before the start of the election.

The voting terminal creates a new session in conjunction with the bulletin board as described in Protocol 2. This session is used to share data between the devices. The voter enters a short string on the voting terminal. The string is submitted by the voting terminal to the bulletin board to start the creation of a new session. The session ID is appended with this short string by the bulletin board. This is done to mitigate against possible clash-attacks. The bulletin board sends the new session ID to the voting terminal. The voting terminal generates a symmetric key. This key is used to encrypt the generated mapping in the vote creation phase. This information is encoded in a QR-code by the voting terminal. 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 then observe the session during the protocol by using the provided session ID. The voter checks that all the session IDs are the same as the original one and are appended by the short ID they entered.

It is to note, that the short ID entered by the voter does not offer additional security against a malicious voting terminal. The voting terminal can create two separate sessions showing the session ID of the correct one to the voting assistants. However, it is unable to submit an incorrect ballot as the entered cast code may only be used once. This is described in the vote submission phase.

Protocol 1 Election Setup.

1. TA generates an asymmetric key pair in shared fashion using threshold encryption with multiple trustees and publishes the public key (TA) on BB.
2. R generates two asymmetric key pairs, publishes the public keys on BB and sends each V multiple cast codes and one lock-in code.

Protocol 2 Session Setup.

1. V enters a short string on VT.
2. VT submits the short string to BB to start a new session. BB appends the short string to a new *sessionID*. BB sends the *sessionID* to VT.
3. VT displays the *sessionID* as well as a QR code encoding a newly generated secret symmetric key k_{secret} and the *sessionID*.
4. V checks that the short string is appended to the *sessionID*.
5. V scans the QR code with AVA and optionally with additional PVAs.
6. VAs display the received *sessionID*. VAs find the session corresponding to the *sessonID* on BB.
7. V checks that all the *sessionIDs* match with the original.

4.5 Vote creation

The vote creation phase starts with the generation of the mapping used for the indirection and ends once the voter is convinced their ballot has been correctly formed. The indirection created is split between the two active devices. Neither device independently gains enough information to learn the voter's intent. Vote creation may be done multiple times if the voter chooses to audit their vote.

The voting terminal generates a mapping between choices and colors as described in Protocol 3. A permuted set of all possible choices is encrypted using the tallying authority's public key and mapped to a newly generated set of colors. It publicly commits to a symmetrically encrypted version of the mapping on the bulletin board using the previously generated symmetric key. Encrypting the mapping prevents observers of the session, such as the bulletin board, from learning the contents of the ballot if audited.

The voting assistants obtain and decrypt the mapping containing the encrypted ballots and corresponding colors. The active voting assistant displays the set of colors to the voter. The voter chooses the color matching their choice as visible on the voting terminal. The active voting assistant re-encrypts the encrypted ballot before submitting it to the bulletin board as described in Protocol 4. The active voting assistant is unable to learn the mapping as the choices are encrypted. The voting terminal is unable to learn the vote as the chosen ballot is also re-encrypted. 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 by the bulletin board.

Protocol 3 Ballot-color-mapping setup.

1. VT obtains the list of choices from BB.
2. VT generates the ballots based on the list of choices encrypting them with TA's public key ($\{B_i\}^1_{TA}$). If the tally is done homomorphically, Groth-Sahai proofs [16] are generated to prove the correct forming of the encrypted ballot.
3. 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).
4. VT publishes the mapping ($\{\{B_i\}^1_{TA}, c_i\}_{k_{secret}}$) encrypted by the pre-shared secret symmetric key (k_{secret}).

Protocol 4 Ballot Choice.

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

Protocol 5 Audit. (Optional)

1. 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.
2. VT, AVA, and BB reveal their randomness used for encryption and re-encryption respectively on BB.
3. 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.
4. V checks if their choice was correctly recorded. If not V can change devices. If the ballot is correctly formed, they start again as the audited ballot is invalid.

The voter may either choose to cast the encrypted ballot on the bulletin board or audit it. In case of an audit, the voting terminal, the active voting assistant, and the 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 as described in Protocol 5. A voter may audit their devices any number of times. Each time a new session is started and the vote creation step starts anew. A malicious device is unlikely to predict the correct number of times a voter intends to audit if voters audit in a somewhat random fashion.

4.6 Vote submission

The vote submission phase entails entering the previously received cast and lock-in codes on the voting terminal. The codes are used to prevent malicious devices from submitting ballots before the voter is convinced that the correct ballot has been formed. As the codes are known only by the voter and registrar. Their usage mitigates ballot stuffing by the tallying authority as well.

Once the voter decides to cast their encrypted ballot, he enters a valid cast code on the voting terminal. The voting terminal sends the encrypted ballot with the cast code to the registrar. If the cast code is valid and has not been used before, the registrar signs the encrypted ballot and cast code using their private cast signing key. The registrar sends the signed encrypted ballot back to the voting terminal. The voting terminal submits the signed encrypted ballot to the session. These steps are described in Protocol 6 1-4.

All devices check if the signed encrypted ballot is the same as the previously submitted unsigned ballot. 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 submitted, the voter enters their lock-in code on the voting terminal. The voting terminal sends the signed encrypted ballot with the lock-in code to the registrar. The registrar signs the ballot and lock-in code this time using their private lock-in signing key. The registrar sends this twice signed ballot back to the voting terminal as described in Protocol 6 5-6.

The voter enters their login credentials previously created with the bulletin board to submit the final ballot as described in Protocol 6 7-8. 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 lock-in key and is the same as the last signed encrypted ballot with the cast key. The bulletin board then publishes the signed encrypted ballot.

Protocol 6 Cast.

1. V enters a cast-code on VT. VT submits the encrypted ballot with the cast code to R.
2. If the cast code is valid and has not been submitted before R signs the encrypted ballot and cast code with the cast signing key. R sends the signed encrypted ballot to VT.
3. VT submits the signed encrypted ballot to the session. BB checks the validity of the signature.
4. VT and VAs display a message that a new ballot using a cast code was published on BB. The message contains the used cast code. If the ballot is not the same as the previously encrypted ballot they warn V allowing V to switch devices.
5. V enters their lock-in code on VT. VT submits the already signed encrypted ballot with the lock-in code to R.
6. If the submitted encrypted ballot is signed with the cast signing key and contains the last used cast code R signs the ballot and the lock-in code with the lock-in signature key. R sends the twice signed encrypted ballot to VT.
7. V enters their login credentials on VT. Using the login credentials VT publishes the twice signed encrypted ballot on BB. BB checks the validity of both signatures.
8. VT and VA's display a message that the final ballot was published on BB containing both the cast code and lock-in code used.

4.7 Tallying

The tallying phase is the final phase of the protocol entailing counting the votes in a verifiable and private manner and then publicly releasing the result. Tallying may either be done with a mix-net based approach or homomorphically. Homomorphic tallying is simpler in concept as the votes are reduced to the result in encrypted form using the homomorphic property of the underlying encryption operation. However, it requires proofs of the correct forming of the ballots to be generated in the vote creation phase. As the ballots are re-encrypted during vote creation the proofs have to be re-encryptable as well requiring more complicated proof systems.

Mix-net tallying: As a mix-net based tallying system does not require any proofs to be created in the vote creation step, any votes cast as described can be shuffled and then decrypted. The shuffled votes can only be decrypted if at least the threshold number of trustees use their partial keys in combination. Proofs of correct mixing and decryption are published on the bulletin board.

Protocol 7 Mix-net Tally.

1. 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.
2. 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.
3. TA invalidates any ballot that does not correctly encode a valid voting choice.
4. TA tallies all available valid votes and reveals the final tally on BB.

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. Such proofs are used in BeleniosVS which uses re-encryption as well. The additional re-randomizable signatures described in BeleniosVS are not used in our protocol as the cast or audit challenge provides voters with the means to check the correct forming of their ballot. On the bulletin board, the re-randomized proofs are checked to be valid. The need for proofs restricts the possible voting modes as the ballots have to be in a format such that proofs can be generated. The tallying is done as in any other homomorphic tallying system.

Protocol 8 Homomorphic Tally.

1. TA verifies all the zero-knowledge proofs provided with the encrypted ballots. All invalid ballots are publicly discarded.
2. TA reduces all encrypted ballots to a single encrypted result using the homomorphic property of the encryption.
3. 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.
4. TA reveals the final decrypted tally publicly on BB including all partial decryptions and corresponding proofs.

5 Ranked Elections

Artemis adds additional support for ranked elections: The voters can submit a personal ranking of the candidates during the voting stage. It uses the same technique of indirection as described above. The election will result in either a single winner or a final ranking of candidates, depending on how it was set up. Multiple ranking methods are available for selection during election creation.

5.1 Problem Transform

During the election creation phase, Artemis creates every possible permutation of candidates, where each permutation corresponds to a ranking of candidates. Each of these permutations is then registered as a candidate in the Artemis voting system. Thus, a voter does not vote for an individual candidate, but for a permutation of candidates represented by an appropriate ranking.

For M candidates this yields $M!$ different options. If the number of rankings R is strictly smaller then the number of candidates M, then the space can be reduced to $M!/(M-R)!$ possible rankings.

5.2 Voting Phase

The vote creation phase remains similar: The voter requires an active voting assistant, a voting terminal, and, optionally, an arbitrary number of passive voting assistants to create the desired ballot. One difference is that there is no candidate-color mapping but rather a ranking-number mapping. This is because the number of rankings grows exponentially with the number of candidates, and the voter would be overwhelmed such an exponential number of colors. A simplified ranking selection process is implemented in the voting terminal, where the voter enters her desired ranking to get a code instead of reading all possible $M!/(M-R)!$ ranking-number mappings. In order to not reveal the candidate ranking to the voting terminal, the voter enters multiple rankings, where only one of these choices corresponds to the correct ranking. This way the voting terminal cannot determine the voters' ranking preference. The code corresponding to the ranking of choice is then entered into the active voting assistant.

The optional audit phase differs from the original one in that after finishing the audit step, the voter is shown the final ranking of candidates, instead of the final candidate.

5.3 Tally to Rank Mapping

The tally phase of ranked elections does not have to be adjusted in order to implement ranked voting correctly. After the rankings are tallied, they will have to be combined to a final result. A method to map the tally of individual rankings by voters to a final ranking of candidates is selected at the election creation, where the administrator can decide whether the election should yield a single winner or a ranking of candidates. Among the ranking options are Borda counting, plurality counting, pairwise counting. Each of these methods satisfies a different set of fairness criteria. The ranking method that the election administrator chooses is entirely dependent on the nature of the election. Note that it is simple to augment the code base to support additional ranking methods if one wishes to do so in the future.

6 Implementation

Artemis is based on the helios-mixnet-server [21] which is a fork of Helios. Note that the current Helios branch uses homomorphic tallying and we therefore selected to extend the fork that is using mix-net tallying. Artemis is a selectable option in the election creation page leading to a different client side path other than the classic helios-mixnet-server if selected.

The election dashboard, election creation, and tallying are implemented in Python using the Django framework. The election administrator creates and sets the parameters: such as who is eligible to vote and what sort of tallying procedure should be used. The Python server acts both as the TA and the BB in this implementation. Once the election is frozen on the admin dashboard, a PUT request is sent to the registrar which contains the election ID and a list of ID's and email addresses of all eligible voters. The registrar creates a new session using this information and acts as a REST server, with which the VT will communicate during the election. The registrar is implemented in Typescript, using a simple Express Server. The voter will interact with the VT, the PVA, and the AVA throughout the election. All three devices are implemented in JavaScript as Single Page Applications which communicate with the registrar and the Django server using REST calls. The registrar generates and sends five cast codes and one lock-in code to each voter per email address specified by the election admin.

During the casting step the encrypted vote is sent twice to the registrar for signing. In order to be able to verify the signature on the VT and on the TA during tallying, a common protocol has to be used. We decided to use the standardized JWS protocol using a 2048 bit RSA signature. During the tallying phase, the TA will only count votes which it has successfully verified in order to prevent fraud.

It is important to know that the existence of a registrar is not necessary for the security of the Artemis voting system: It is possible for each voter to possess a private key which she uses to sign the encrypted vote herself. The tallying authority would own the public key of every voter in this case and thus can verify the signed encrypted ballot and only count it towards the final tally if the signature is valid. The issue with this approach is that each voter would have to type the long private key in the terminal which is an inconvenience. Instead of this, we have chosen the usage of cast codes and lock in codes and a registrar which acts as a signing proxy. Thus, the voter only has to type a few letters as opposed to the long key. There are also only two sets of key-pairs needed for signing, of which the private keys are only known to the registrar while the public keys are known to the VT and the TA.

7 Conclusion

E-Voting and electronic participation, in general, has become more important during the Covid19 pandemic [1], as more and more people tend to vote online and not assemble in person. Since E-Voting is used for more serious elections, it is imperative to get verifiability, receipt-freeness, and privacy to the best possible extent from any E-Voting system that we design. Additionally, we also want to keep system design and implementation as simple as possible - so that they can be understood and reviewed by independent auditors and interested parties. To this end, we have extended the time-tested implementation of Helios with protection against malicious voting devices using indirection, and protection against corrupt tallying authorities using re-encryption based receipt-free and verifiable tallying.

References

1. The covid-19 crisis – a much needed new opportunity for online voting?, `https://www.idea.int/news-media/news/covid-19-crisis-%E2%80%93-much-needed-new-opportunity-online-voting`, accessed: 2020-09-24
2. Helios documentation, `https://documentation.heliosvoting.org/`, accessed: 2020-09-24
3. Electionguard (2016), `https://github.com/microsoft/electionguard`
4. Adida, B.: Helios: Web-based open-audit voting. In: USENIX security symposium. vol. 17, pp. 335–348 (2008)
5. Benaloh, J.: Simple verifiable elections. EVT **6**, 5–5 (2006)
6. Benaloh, J.: Ballot casting assurance via voter-initiated poll station auditing. EVT **7**, 14–14 (2007)
7. Blazy, O., Fuchsbauer, G., Pointcheval, D., Vergnaud, D.: Signatures on randomizable ciphertexts. In: International Workshop on Public Key Cryptography. pp. 403–422. Springer (2011)
8. Chaidos, P., Cortier, V., Fuchsbauer, G., Galindo, D.: Beleniosrf: A non-interactive receipt-free electronic voting scheme. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1614–1625. ACM (2016)
9. Chaum, D.: Blind Signatures for Untraceable Payments. In: Advances in Cryptology. pp. 199–203. Springer US (1983)
10. Chaum, D.: Surevote: Technical Overview. In: Proceedings of the workshop on trustworthy elections (WOTE'01) (2001)
11. Cortier, V., Filipiak, A., Lallemand, J.: Beleniosvs: Secrecy and verifiability against a corrupted voting device (2019)
12. Fujioka, A., Okamoto, T., Ohta, K.: A Practical Secret Voting Scheme for Large Scale Elections. In: International Workshop on the Theory and Application of Cryptographic Techniques. pp. 244–251. Springer Berlin Heidelberg (1993)
13. Gaweł, D., Kosarzecki, M., Vora, P.L., Wu, H., Zagórski, F.: Apollo–end-to-end verifiable internet voting with recovery from vote manipulation. In: International Joint Conference on Electronic Voting. pp. 125–143. Springer (2016)
14. Gjøsteen, K.: The Norwegian Internet Voting Protocol. Cryptology ePrint Archive, Report 2013/473 (2013)
15. Glondu, S., Cortier, V., Gaudry, P.: Belenios–verifiable online voting system (2015)
16. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 415–432. Springer (2008)

17. Haenni, R., Koenig, R.E., Locher, P., Dubuis, E.: CHVote System Specification. Cryptology ePrint Archive, Report 2017/325 (2017)
18. Ingraham, C.: About 100 million people couldn't be bothered to vote this year, `https://www.washingtonpost.com/news/wonk/wp/2016/11/12/about-100-million-people-couldnt-be-bothered-to-vote-this-year/`, accessed: 2020-09-24
19. Joaquim, R., Ribeiro, C., Ferreira, P.: Veryvote: A Voter Verifiable Code Voting System. In: International Conference on E-Voting and Identity. pp. 106–121. Springer (2009)
20. Küsters, R., Liedtke, J., Müller, J., Rausch, D., Vogt, A.: Ordinos: A verifiable tally-hiding remote e-voting system (full version) (2019)
21. Li, L.Y.: helios-server-mixnet, `https://yingtongli.me/git/helios-server-mixnet/`, accessed: 2020-09-24
22. Liu, Y., Wang, Q.: An E-voting Protocol Based on Blockchain. Cryptology ePrint Archive, Report 2017/1043 (2017)
23. Malkhi, D., Margo, O., Pavlov, E.: E-Voting Without 'Cryptography'. In: Financial Cryptography. Springer Berlin Heidelberg (2003)
24. Novet, J.: Microsoft's voting software is getting its first test in a small wisconsin town (2020), `https://www.cnbc.com/2020/02/18/microsoft-electionguard-software-gets-first-test-in-fulton-wisconsin.html`, accessed: 2020-09-24
25. Oppliger, R.: Addressing the Secure Platform Problem for Remote Internet Voting in Geneva. Chancellery of Geneva (2002)
26. Ryan, P.Y., Bismark, D., Heather, J., Schneider, S., Xia, Z.: Prêt à Voter: A Voter-Verifiable Voting System. IEEE Transactions on Information Forensics and Security **4**(4), 662–673 (2009)
27. State Electoral Office of Estonia: General Framework of Electronic Voting and Implementation thereof at National Elections in Estonia (2017), available at: `https://www.valimised.ee/sites/default/files/uploads/eng/IVXV-UK-1.0-eng.pdf` (Sep. 2018)
28. Tsoukalas, G., Papadimitriou, K., Louridas, P.: From helios to zeus. USENIX Journal of Election Technology and Systems (JETS) **1**(1), 1–17 (2013)
29. Wu, H., Vora, P.L., Zagórski, F.: Privapollo–secret ballot e2e-v internet voting