Panos Louridas (GRNET)
George Tsoukalas (GRNET)
Dimitris Mitropoulos (GRNET)

Requirements and User Interface Design

Deliverable D 5.1

June 8, 2016
Panoramix Project, # 653497, Horizon 2020
http://www.panoramix-project.eu

Contents

Chapter 1

Overview and Goals of the e-Voting Use-case

The e-voting use case concerns Zeus [6], [1] [2] GRNET's e-voting system. Zeus is in production since late 2012. So far it has been used in 419 real-world elections involving more than 51,000 voters. Zeus mixes votes with a Sako-Kilian re-encryption mix-net [4], based on the ElGamal cryptosystem [5]. By employing the technology and the tools developed in the context of the PANORAMIX project, GRNET seeks to enhance the security, the performance, and the usability of Zeus.

The following sections identify issues with the current system and motivate the use-case in the context of the project.

[1] http://zeus.grnet.gr
[2] http://github.com/grnet/zeus

Chapter 2

Overview of Zeus voting

Zeus is derived from Helios, a web-based open-audit e-voting system [1]. The original Helios system proposed cryptographic mixing via Sako-Kilian [4] / Benaloh [2] re-encryption mixnets of ballots for their anonymization. Specifically, this is achieved in the following manner: The mix-net itself consists of at least two mix-servers and one decryption layer. A mix-server first re-encrypts every ciphertext with specific public keys. Then it randomly permutes the new ciphertexts and outputs them. The same procedure is repeated by each mix-server included in the mixnet. Finally, in the decryption layer, a decrypting entity that knows the secret keys decrypts its inputs and outputs the resulting plaintexts. Figure 2.1 presents the architecture of Zeus.

Zeus uses essentially the same cryptographic workflow as Helios. In particular, the workflow has been implemented around a logical document, which we call the *poll document*. This document contains every cryptographically significant piece of data for a poll and can be stored in a portable binary *canonical form*.

This canonical form always maps two identical documents to a unique byte array. Inversely, the same byte array always decodes into the same document. This is necessary since the unique serialized representation of the document is used to obtain a unique identifier based on cryptographically secure hashing. Unfortunately, JSON does not offer a unique representation and therefore a simple binary format was developed and implemented for Zeus.

The handling of the poll document is done by a generic and standalone software module. This module can be extended by a web application and a corresponding user interface to create a usable voting system. This core module implements all the required cryptographic functions and validations. In turn, the web application extends the workflow module and builds a usable voting system upon a database that handles authentication, election formulation, vote submission, and key setup and ballot decryption for the members of the committee of trustees responsible for overseeing the voting. There is also a user interface through which voters may access the web application by using their web browser, and perform vote preparation and encryption locally on their computing devices. Finally, an interface that can be used for trustee key generation and (partial) ballot decryption is also included in Zeus.

The core module handling the poll document is completely independent of the other aspects of the system. Therefore it can provide accessible and reliable verification of the tabulated results by third parties, as they are not obliged to dig through irrelevant software layers to validate results. Ideally, Zeus users should have an API through which they could query the poll document at will. This enables auditors to more efficiently inspect documents and provides a substrate for verifiability services which will receive vote submission receipts and run a validity checks for voters. However, Zeus does not provide such an API right now, but it is clearly something worth implementing.

2.1 Workflow Stages

For every poll, Zeus passes through five consecutive stages: 1) initialization, 2) voting, 3) mixing, 4) decryption, 5) finalization. We describe each stage in the upcoming subsections. Figure 2.2 presents the

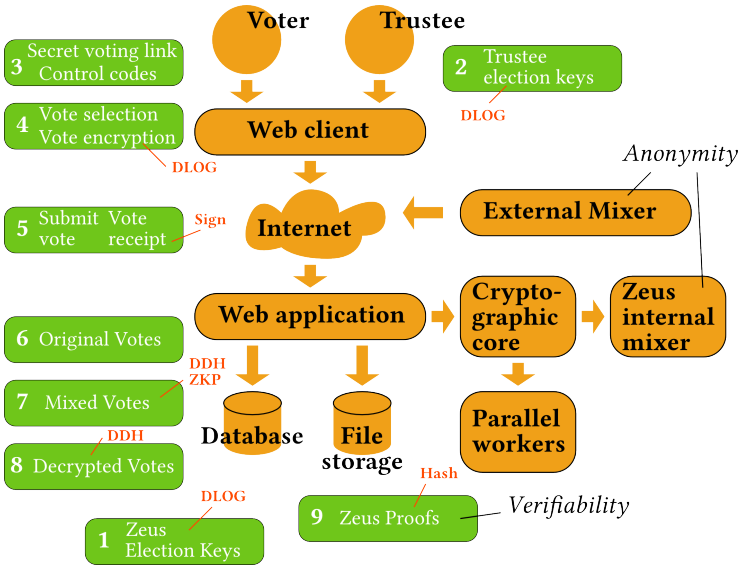

Figure 2.1: The Architecture of the Zeus e-voting System.

general workflow of the Zeus e-voting system.

Note that all data necessary for election results and election audit are recorded into the poll at the various stages. Once recorded in the poll document, data cannot be modified, leaving a reliable trace. To safeguard against data corruption from accidental errors or other unforeseen circumstances, each stage is validated upon transition into the next one. During the final stage, the document represents a full and verified account of a completed poll.

2.1.1 Initialization

At start, candidates, voters, and trustees are registered into the poll document. The candidate list is just an ordered list containing unique candidate names. Different types of elections put special structure within the candidate list, which is used at tallying time to verify and count the ballot.

Voters are likewise represented by uninterpreted but unique names that must be provided by authorities. For each voter, Zeus additionally generates several random numbers which we call *voter codes*. These codes can be used both for *audit votes* (see below) and voter authentication.

Each trustee provides their public key for each poll. All keys are combined into a single election public key that is used to encrypt votes. At decryption time, each trustee must provide partial decryptions which are all combined to yield the final plaintext ballots. Zeus automatically creates a key and operates as a trustee for every election. As per the Helios workflow, this allows the service operator to join the appointed committee of trustees in providing anonymity guarantees.

2.1.2 Voting

When all the trustees of a poll have registered with the election and their public keys have been recorded, the poll advances to the *voting* stage. In the beginning of this stage Zeus combines (by a multiplication) the trustees' public keys into a single *poll public key*. Then, this key is used by the voters to encrypt their votes. After the formulation of the public key of the poll, no trustees, voters, or candidates can be modified.

The voter submits to the server the encrypted ballot a (discrete log knowledge) proof that they possess the randomness the ballot was encrypted with and optionally a voter-provided *audit code* to verify that their local system really does submit the vote they choose and that it does not alter it in any way as per a variation of the Benaloh challenge [2]. The process is detailed in Section 2.1.2.

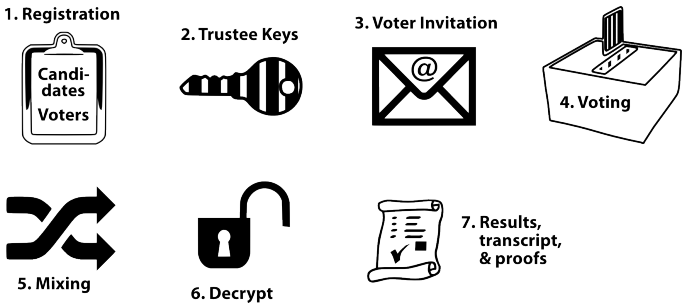

Figure 2.2: Zeus's General Workflow.

For each submitted vote, Zeus generates a cryptographically signed receipt for each voter. With this receipt, the voter can later prove that his vote submission was acknowledged as successful, (e.g. during an investigation for a complaint). Receipts are detailed in Section 2.2.

Each voter may submit a vote multiple times. Each time, the new vote replaces the old one and the new receipt explicitly states which vote was replaced. Vote submissions are strictly serialized and therefore this does not create a race where a voter may end up with multiple valid votes in their name.

Audit Codes

Audit codes are random numbers generated by Zeus and sent to each voter. The purpose of these codes is dual. First, they can be used to submit audit votes to test that the local system, mainly the web browser, faithfully submits the votes it is being instructed to. Secondly, and optionally, they can be used to authenticate voters in close integration with audit votes.

Audit votes work by ruining the plans of an attacker who might otherwise compromise the voter's computer and alter the voter's selections. To ruin those plans one introduces uncertainty; either the vote will be submitted normally and be counted, or the vote will be revealed and published for all to see what it was. If the attacker does not know what will be the fate of the vote, they cannot decide whether to cheat or not. If they cheat and the vote is published, they will be exposed. If they don't and the vote is counted, they will fail.

Audit votes in Helios create this uncertainty by having the voter decide on the fate of the vote, only *after* it has been uploaded to the server and therefore is made immune to the local attacker. After submission of each vote, the voter is prompted to choose between a normal vote and an audit one.

This is explicit and safe, but for Zeus it was deemed too dangerous to *require* all voters to make such a decision, because it would require of all voters to understand what an audit vote does, while a wrong button press, even accidentally, would cause the true intended vote to be published.

In Zeus, a vote can be submitted either with an accompanying code, or with no code at all. Submissions without an accompanying code are accepted as normal votes to be counted and no auditing is performed. If a voter distrusts their local system and wishes to verify it and submit their vote safely, then they should never submit a vote without a code.

If a vote is submitted with an accompanying code, two things may happen. If the code is *not* among the ones the voter has received, then the server considers it to be an audit. If the code *is* among those supplied by the server, then it is cast as a normal vote. Two different votes must never be submitted with the same code, because the second time the attacker will know what to do. The process in detail is

as follows:

1. The voter enters one of the audit codes they have received via e-mail.
2. The voter proceeds to submit the vote to the server.
3. The server receives the encrypted vote. If the encrypted vote is accompanied by a correct audit code or is not accompanied by any audit code, the vote is treated as a normal vote. Otherwise, that is, if the vote is accompanied by an audit code but the audit code is not correct, we continue with the following steps.
4. The system stores the vote ciphertext as an audit vote request.
5. The system asks the voter to confirm that they have cast an audit vote.
6. If the voter responds negatively, the audit process is canceled and the voter is redirected to the initial ballot casting view. The voting request remains in the system but is not displayed to the users.
7. If the voter responds in the affirmative, the browser reposts the vote ciphertext along with the ElGamal randomness with which the vote has been encrypted.
8. The system tries to match the fingerprint of the re-posted vote with that of an existing vote audit request. Failure to do so implies that the ciphertext initially posted is mangled and the system informs the user that the audit request cannot be accepted. This guards against malware that would submit an altered vote and then after realising that the vote is an audit vote would send the actual ciphertext. If the match succeeds the system stores the audit vote and returns to the voter a unique identifier of the audit vote.
9. The decrypted contents of each audit vote, based on the randomness sent previously by the client, are posted on an audit vote bulletin board. The voter can go to the audit bulletin board and look for the vote using the identifying number they have received from the server.

The user interface of Zeus makes this process intentionally somewhat obscure to the layman, so that its use without being certain what to do is discouraged, but it is intended to be easy enough for those who do understand: vote with a random "wrong" code many times and check the audits, then vote with a "correct" and be done.

Therefore, Zeus makes auditing optional, by creating the required uncertainty for auditing using the audit codes. However, Zeus can be configured to *require* a code with each vote submission in which case the audit codes provide a secondary function as an additional authentication factor. Making the audit code compulsory would also strengthen the system against an attack in which a compromised browser notices when a ballot is submitted without an audit code and so will cheat only then.

This function helps combine a very convenient but potentially insecure means, for example the e-mail, with a quite secure and personalized one, for example the mobile phone. Invitations, notifications, and descriptions relating to voting, which can be voluminous texts with graphics, can be sent by e-mail over the internet, while the audit codes, which are conveniently small as a text, can be sent to a mobile phone. The voter would need both the e-mail and the mobile phone message in order to vote successfully.

2.1.3 Mixing

After the voting stage ends, the poll progresses to the *mixing* stage, where all encrypted ballots are anonymized. Only the ballots eligible for counting are selected. Note that all audit votes, replaced votes, and the votes by excluded voters are not included.

In contrast to traditional elections with a physical ballot box where anonymity is achieved when ballots are cast, in a mixnet-based voting protocol the encrypted ballots cast remain eponymous until mixing. This allows the exclusion of a voter's ballot after it has been cast but before mixing. The election officials may choose to disqualify voters if it is discovered that they were wrongly allowed to vote, or if their conduct during the election was found in breach of rules. Neither the disqualified voter nor their votes are deleted from the poll document and the audit trail. Instead, the voter is added into an exclusion list

along with a statement justifying the decision to exclude them so that the authorities cannot hide their action.

The eligible encrypted votes are first mixed by Zeus itself by a Sako-Kilian mixnet [4]. The bulk of the computational work in mixing is to produce enough rounds of the probabilistic proof. The essence of the proof is to commit to a sufficient number of additional mixes (i.e., re-encryptions and permutations) of the ballot set, and then for each one of the committed mixes, and according to a random challenge (made non-interactive via Fiat-Shamir), either reveal the mix or reveal a mix that can translate the committed mix into the original mix being proven. The work is distributed to a group of parallel workers, one round at a time. Our current deployment uses 128 rounds and 16 workers. Verification of mixes is likewise parallel.

After the first mix is complete, additional mixes by external agents can be verified and added in a mix list using Zeus's command line toolkit.

2.1.4 Decryption

After mixing is complete, the final mixed ballots are exported to the trustees to be partially decrypted. The resulting decryption factors are then verified and imported into the poll document. Zeus's own decryption factors are computed last.

2.1.5 Finalization

During the final stage of the election process all decryption factors are distributed in parallel workers to be verified and combined together to yield the final decrypted ballots. The decrypted ballots are recorded into the poll document, and then the document is cast into the binary canonical representation that was discussed in Section 2. This representation is hashed to obtain a cryptographically secure identifier for it, which can be published and recorded in the proceedings.

2.2 Vote Submission Receipts

Zeus incorporates the concept of *vote submission receipts*. For each vote cast a receipt is generated. This is returned to the voter and registered in the poll document. Typically, the receipt is a text file with simple structure, listing cryptographic information about the vote and the poll. The receipt text is signed with Zeus's own trustee key for the poll. This is possible because Zeus is a trustee for every poll, as described in Section 2.1.1.

The receipt contains:

1. a unique identification of the ballot cast,
2. the cryptosystem that was used,
3. the poll's public key along with the public keys of all trustees,
4. the list of candidates,
5. the cryptographic proof that the encryption was a valid,
6. the signature itself, and
7. in case of a replacement, the replaced vote.

The validity of the encryption is established by checking the discrete log proof submitted along with the encrypted ballot.

Private information such as user names, IPs, and how many times a user logs in the Zeus servers do not appear in the receipt. This is because we need the entire poll document to be publishable if the election authority decides so.

Note that the vote submission receipts are not enough to guarantee that a malicious Zeus mix-server cannot submit a vote on behalf of a user, and then argue that this user has simply lost their receipt; or that a user, taking care to delete their receipt, does not come forward and argues that their last vote on the server is a bogus one. The problem could be solved by posting encrypted ballots in a bulletin board after the polls close and before tallying starts.

2.3 Ranked Ballot Encoding

In the second version of Helios (which used homomorphic tallying instead of mixnets), ballots consisted of answers that corresponded to binary questions, framed in the appropriate way. For instance, a voter indicates k out of n candidates on a ballot by selecting them, by answering questions like "Would you like X as a Y?" (yes (1) or no (0)). If an election uses the Single Transferable Vote (STV) system though (as in any system in which we would need the whole ballot to be decrypted in the end), the ballot must be encoded in some way. We encode each ballot as a integer by assigning a unique number to each possible candidate selection and ranking. When enumerating, we first count the smaller selections (i.e. take *zero* candidates, then *one*, then *two*, ...) so that for small numbers of choices the encoded ballot will have a small value, thus saving valuable bit-space.[1] Intuitively, if voter uses fewer words to write down their choices then the encoding would choose a smaller number for the ballot.

[1] For example, selecting up to all of 300 candidates needs 2043 bits, while selecting 10 out of 1000 candidates needs only 100 bits.

Chapter 3

Review of Mixing in Zeus e-Voting

In this chapter we review the mixing process of Zeus and identify existing issues and possible enhancements in the context of an e-voting application, and a general mixing solution. Our review supports the mixing requirements and the design of the next generation e-Voting application presented in Chapters 4 and 6.

Availability Currently, there are no viable open-source implementations for mixing algorithms. In addition, efficient mixing algorithms are also not free to implement and use because of patents. Beyond the various basic algorithm implementations that can be found as independent software components, a complete system to support the whole mixing process while addressing practical challenges in engineering, security, and usability appears to be a necessity in the field.

Performance The Sako-Kilian mixnet, while linear in the number of ballots, has very large constant factors in its complexity, both in time and in storage space. An efficient mixnet significantly boosts applicability and usability. Currently, there are algorithms much more efficient ($\sim$50 times faster). Zeus should take advantage of such algorithms to improve its functionality.

Another issue concerns the scalability of the implementation. Currently, Zeus parallelizes the mixing in the same node but does not support multiple nodes.

Standardization One of the goals of the PANORAMIX project is to provide a common method to deploy mixnets and integrate them into applications. Standardizing a mixnet API offers multiple benefits. First, it allows a robust solution to be developed and maintained by experts who can then apply it in different contexts. Additionally, a product that integrates mixing can deploy multiple mixnets with different characteristics depending on the requirements of each specific application.

This has been evident in the use of Zeus, where different security requirements, trust environments, or different cryptographic parameters would ideally be served by a different mixnet system. For example, some mixnet proofs rely on more robust theoretical foundations but are slower than others, and some mixnets may operate with elliptic curve groups while others in integer groups.

Usability An important security parameter in the mixing process is the independence of mix contributors and their servers. This implies independent physical infrastructures. Our experience has shown that politically independent contributors are willing to participate in the mixing process but do not have the expertise or technical background to provide a truly independent deployment, free of the influence of a central technical consultant.

Therefore, the ability of non-experts to easily deploy a mix server in order to contribute to the process would improve significantly trust in the whole process. Similarly, it would be very desirable and useful if non-experts had the opportunity to verify a mixnet.

Community Trust is generally earned slowly with reliable performance within a community. It is much easier and safer to trust an established community of independent peers to mix for a very wide range of applications. Mixnets and contributors that are only participate once and for a specific application do not inspire trust and are suspect of having malicious intentions in that one case.

A platform that can offer some standardization can help create a trusted and reliable community.

Research Platform A mixnet platform should provide easy access to production systems and real-world scenarios where everyone can test their ideas and prototypes. Furthermore, it will accelerate evaluation and adoption of new technologies and help identify new areas of research.

Chapter 4

Requirements of PANORAMIX Mixing

4.1 Security

4.1.1 Verifiable Reliable Anonymous Private Messaging

Voting is essentially a procedure where voters send messages to the committee of trustees. These messages must be:

Verifiable meaning that senders must be convinced that their message is delivered.

Reliable meaning that exactly all sent messages will be received.

Anonymous meaning that senders cannot be linked to the messages they submitted.

Private meaning that only the recipient can read and publish the messages.

Making it easy for independent experts to write their own verification software helps strengthens the argument for using e-voting and can boost adoption the adoption of the service. The mixing component should facilitate this.

4.1.2 Distribution of Trust

An election is as trustworthy as the committee of trustees that oversees it. The committee should include an independent institution, the computer system administration, and all major competing interests in the election.

Therefore, a mix-net must be easily composable by the contributing servers of parties that do not share infrastructure, or even good faith among them. The distributed parties must also be able to practically verify the security of the mix-net.

4.1.3 Cryptography and Key Management

The e-voting application requires proofs of the mixing integrity. This typically means a re-encryption mix-net must be used but alternatives such as decryption mixnets may also be considered, especially if they are part of the same framework and API.

As part of the general verifiability requirement it is desirable that a vote can be proven to be included in the mix, and therefore in the counted results, without the need for full access to the proof data, for both security and efficiency reasons.

The mix-net must provide a utility for key generation, encryption, decryption, signing, and safe-keeping separate from the service runtime so that trustees can use them in their own controlled computers.

4.2 Performance

4.2.1 Mix-net Computational and Storage Complexity

The computation of a mix proof and its verification would be able to process over 10 ciphertexts per second per CPU core, producing a proof that is no more than 10 times the total volume of the mixed ciphertexts.

The computation and verification of proofs must be scalable to many CPUs and machines.

4.2.2 Available bit size for plaintexts

The bit size for messages should be unlimited, and configured at mix-net creation. 256-bits is an absolute minimum.

With a 256-bit plaintext a ballot can:

- Rank any subset of 57 candidates in order of preference.
- Provide a 53-character write-in vote in only 26 uppercase latin letters and space.
- Provide a 16-character write-in vote in UTF-16 format.

4.3 Engineering

The mix-net should offer a network-based API and client libraries for server and web browser environments. The clients should abstract away low-level crypto and network functions.

Since a mix-net's basic function is to anonymously deliver private messages to its recipients, it would be desirable to expose some of its low-level communications functions to API clients.

The client applications can then re-use those functions to implement protocols among their different actors without having to re-implement a separate communication stack.

An e-voting protocol involves more than mixing in implementing the election workflow and application-specific constraints. Different actors in the system need to communicate securely by using appropriate protocols with cryptographically signed messages. Binding signed receipts must be issued to voters. Therefore, building upon an established communication framework can reduce both development effort and security risks.

Chapter 5

Review of the e-Voting Application

This chapter reviews the e-voting application Zeus for its voting protocol and usability. See section 3 for the mix-net review.

5.1 Distribution of Trust

As presented in Section 2, currently there are too many functions under the control of a single system. Voting management, voter and trustee communications, vote submission, booth server, mix server.

An intrusion to this single system or a corruption of its administrator can compromise the entire procedure. Distribution of services would create a human or institutional network that would check each other and challenge misbehaviour.

5.2 Voter Verifiability

In the years of operating Zeus, it was observed that the committee of trustees usually decides not to publish the cryptographic proofs and audit logs of the election for fear that anonymity will be possible to breach some time in the future. The committee then requires voters to formally submit their request for access to proofs.

While there have been no such incidents yet, it would be much better for the perceived and effective reliability of the system if voters could interact with the system to receive some kind of verification.

5.3 Performance

Trustees interface with Zeus e-voting via a web browser. The most computationally demanding trustee operation is the (partial) decryption of ballots. Performed in a browser, a large ballot set (e.g. 10000 ballots) takes several hours and runs the risk of crashing due to memory exhaustion.

Furthermore, since trustees are already involved due to the election keys they hold, it is practical to also have some of them mix ballots. However, we expect trustees to not be experts and to only have access to a personal computer. Therefore mixing computation has to become efficient to enable easy trustee mixing.

5.4 Usability and Security

Experience has shown that usability is a critical issue. Trade-offs between usability and security have been resolved towards usability. For example, the audit vote feature was made inconspicuous for the

voter whereas originally in the Benaloh challenge [3] the voter has to make an explicit choice every time.

Another example is that in the voter workflow adopted, the e-voting system relies on the user to record cryptographic identifiers themselves. This is not reliable since users do not bother. However, requiring that the identifiers are copied and pasted into a form would force users to acknowledge and perhaps keep the identifier around enabling them to later verify the procedure.

The election control panel and the control panel for trustees is not intuitive enough for non-technical trustees to operate on their own. As a result trustees often rely too much on the technical operator of the user interface for controlling the voting process and even delegating trustee operations to them (e.g., put their key in portable storage).

Enhanced usability will make participants more independent promoting overall security and trustworthiness.

Chapter 6

Design of the E-Voting Application

The next generation of Zeus will serve as the e-voting application for Panoramix. It inherits from the current Zeus system in production and by extension from its system of origin, Helios [1]. It incorporates patterns proven in practice but also introduces new elements to address issues identified in Chapter 5.

Since mix-net details are not yet available, this design only concerns the general workflow of the application that is deemed to be independent of the underlying cryptography.

6.1 Components

Protocol Controller The protocol controller orchestrates voting by registering the state of the process and requesting actions from human or computer actors according to protocol.

The Protocol Controller is responsible for constructing and signing out the final results guaranteeing that the protocol was observed.

Voter Controller The voter accounting server authenticates submitted ballots. It hosts the voter registry and guards the privacy of the electorate.

Ballot Box The ballot box records authenticated encrypted ballots. It is responsible for cryptographically validating submissions.

Trustee Agent Each trustee has their own instance of the trustee agent as their interface to the entire platform. It needs to verify the procedure from the perspective of the trustee.

Voting Booth Each voter has their own instance of the voting booth as their interface to the entire platform. It needs to verify the procedure from the perspective of the voter.

Mix network The mix network mixes the encrypted output of the ballot box into a shuffled and decrypted ballot list at the voting controller for tabulating the results.

Within the mix network there have to be multiple administrative domains that contribute to the mix network, presumably with an appropriate agent. These entities may need to somehow be linked to and interacting with the Protocol Controller, Trustee Agent, and Voting Booth for the purpose of verification.

Details of this relationship will have to be developed after a technical mix-net solution is available in the course of the project.

6.2 Users

There are three kinds of users planned in the next iteration of GRNET's e-voting system, trustees, who oversee the whole process, mix-net node administrators who collectively maintain the computing infrastructure, and voters and auditors that need to participate and verify the process.

6.2.1 Trustees

The combined keys of the trustees constitute the election key that encrypts and decrypts all ballots. Trustees guard the anonymity of the decrypted ballots by refusing to decrypt anything other than what the protocol describes.

Additionally, to secure tight oversight of the procedure, each control action for an election (e.g., a change to the voter list) must be approved and signed by all trustees. Otherwise they would all have to delegate to a technical operator who could have used their access without instruction or knowledge of the trustees.

The critical operation for anonymity is the mixing itself. Trustees have the ability to monitor, audit, and participate in the mix-net.

6.2.2 Voters

Voters receive an invitation to participate in voting through an appropriate channel (e.g., email, SMS). Using the invitation they authenticate themselves to the e-voting system and are allowed to vote.

Voters are considered to be non-experts and non-technical in general, and therefore they need to have a simple and usable interface, and are unlikely to bother or be able to audit the system. However, it is expected that there will always be a small number of voters who are technical people or even experts who can effectively audit the system. The end-to-end verifiability of the system enables auditing at a later time after the fact, without requiring presence at the place or time of the event, and with the effort of only a few auditors.

Verifiability, both apparent and effective is very important for voters. Therefore, the software clients they interact with must offer intuitive and effective controls for verification, at least at the scope of the individual voter concerns.

Voter concerns are mainly to establish the authenticity of the authorities operating the voting procedure, that proper protocol was observed, and that their own votes are being counted towards the final results.

6.2.3 Administrators

Administrators deploy and maintain the various components of the system. Defending the systems from tampering and abuse they critically contribute in the overall system security. Different components may belong to different administrative domains for enhanced isolation and monitoring. Administrators do not need to know themselves details for each voting performed for the benefit of security. They need only secure their local server software, and the software itself automatically cross check its various components belonging to different administrators that cannot not all have incentive to collude with each other.

Bibliography

[1] Ben Adida. Helios: Web-based open-audit voting. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, pages 335–348, Berkeley, CA, USA, 2008. USENIX Association.

[2] Josh Benaloh. Simple verifiable elections. In *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, EVT'06, pages 5–5, Berkeley, CA, USA, 2006. USENIX Association.

[3] Josh Benaloh. Simple verifiable elections, 2006.

[4] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme: A practical solution to the implementation of a voting booth. In *Proceedings of the 14th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'95, pages 393–403, Berlin, Heidelberg, 1995. Springer-Verlag.

[5] Bruce Schneier. *Applied Cryptography (2Nd Ed.): Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[6] Georgios Tsoukalas, Kostas Papadimitriou, Panos Louridas, and Panayiotis Tsanakas. From helios to zeus. In *2013 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '13, Washington, D.C., USA, August 12-13, 2013*, 2013.