From Helios to Zeus

GEORGIOS TSOUKALAS, Greek Research and Education Network (GRNET)
KOSTAS PAPADIMITRIOU, Greek Research and Education Network (GRNET)
PANOS LOURIDAS*, Greek Research and Education Network (GRNET)
PANAYIOTIS TSANAKAS, National Technical University of Athens

We present Zeus, a verifiable internet ballot casting and counting system based on Helios, in which encrypted votes are posted eponymously to a server, then are anonymized via cryptographic mixing, and finally are decrypted using multiple trustee keys. Zeus refines the original Helios workflow to address a variety of practical issues, such as usability, parallelization, varying election types, and tallying through a separate computing system. In rough numbers, in the first nine months of deployment, Zeus has been used in 60 elections, tallying a total of more than 12000 votes.

1. INTRODUCTION

Helios [Adida 2008] (http://heliosvoting.org) is a well known system for internet voting, in which the entire process is carried out through digital means—there is no paper trace, nor ballots in physical form. Instead, there is a trail of mathematical proofs linked together that can verify the correctness of every step in the voting process. Helios has been used in several real world elections and its basic architectural design has proven robust.

In the summer of 2012 our organization was asked to provide a system for electronic voting to be used in elections in universities in Greece. Based on its good track record and the body of work that had been published on it, we decided to see whether Helios could fit our needs. Unfortunately, we found that it could not, because of the voting system that would be used in the elections we were called to run.

We therefore had the problem of creating a system that could be used for the required elections, scheduled to take place in approximately three months time. Having no time to start from scratch, and having no intention to re-invent the wheel or to embark on cryptography research, we set out to develop a system based on Helios, as much as possible, with the necessary modifications to suit our needs. The system came to be known as Zeus (thus remaining in the pantheon of Greek gods) and has been used with unequivocal success for many elections over several months now. That was especially satisfying since the use of Zeus was not without critics, for reasons that we will explain below.

The contribution of this paper is not any new electronic voting protocols or an entirely new voting workflow; we have adopted and used proven systems and ideas that have already been described in the literature. Our contribution is:

(1) An extension of an existing e-voting system, Helios, so as to be able to handle any kind of voting systems, such as Single Transferable Vote in a completely open source implementation.
(2) The organization of the Helios cryptographic workflow as a process of populating a single document with all relevant cryptographic content in a well-defined portable format, independent of any other details of the voting system.
(3) A new use of audit votes, both as a means of checking the integrity of voting from the user's side, and as a channel of authenticated communication with the user.
(4) A description of a working e-voting system that has been used in many real-world elections, by not especially technically-savvy users.
(5) An account of the experience of using an e-voting system in a particularly adversarial context, in over 60 real-world elections involving more than 12000 voters in total.
(6) A description of the usability, accessibility, and logistical issues we had to tackle.
(7) An outline of the internal design and implementation choices.

*Corresponding author.

Items 1–3 relate to technical differences with previous implementations of Helios and related systems. The other items relate to lessons from real world usage of the system in an especially challenging setting.

The rest of this paper proceeds as follows. In Section 2, we discuss related work. In Section 3 we discuss the birth of Zeus. In Section 4 we offer an overview of the voting process from the user's point of view. In Section 5 we go through the complete cryptographic workflow elaborating on technical details and the population of the *poll document*, which contains all data and proofs to verify the election. In Section 6 we relate our experience with the deployment and use of Zeus as a service for elections over the internet. We conclude with some general discussion points in Section 7.

2. RELATED WORK

Zeus is based on Helios [Adida 2008], an electronic voting system developed by Ben Adida. The original Helios publication [Adida 2008] proposed cryptographic mixing via Sako-Kilian [Sako and Kilian 1995] / Benaloh [Benaloh 2006] mixnets [Sako and Kilian 1995] of ballots for their anonymization. Mixnets were dropped in version 2 of Helios, which was used for elections at the Université Catholique de Louvain in 2008, because in the then-existing implementation they could not accommodate the need for votes having different weights according to the voter's category [Adida et al. 2009]. In place of mixnets homomorphic tallying [Cohen and Fischer 1985] was adopted, as it was argued that it was "easier to implement efficiently, and thus easier to verify" [Adida et al. 2009].

Homomorphic tallying operates directly on *encrypted* votes, tallying them all up in a single encrypted result, which is the only piece of information that is decrypted. This is much easier computationally but sacrifices generality because the tallying algorithm must be encoded in the cryptographic processing itself—one cannot get results in the form of individual ballots as they were submitted.

On the other hand, mixnets shuffle the set of votes in a provably correct but practically untraceable way, therefore anonymizing it. The output is the same set of unencrypted ballots as they were before encryption and submission, only in random, unknowable order. Then any method of tallying can be applied to compute results. Although Helios itself is not using mixnets any more, a variant of Helios with mixnets has been developed that does [Bulens et al. 2011]; however the code is not publicly available.

Due to the circumstances around the birth of Zeus (see Section 3) and the controversy around it (see Section 6) there was no way we could have used any implementation whose source was not completely open source and freely available to the community. Hence, even if after contacting the developers of Helios with mixnets we were given access to the code, we could still not have been able to use it unless it were completely open sourced. Judging this unlikely, we proceeded on our own, without contacting them. As a result, Zeus and Helios with mixnets share many conceptual similarities, although we did not try to avoid the submission of related ballots using the Cramer-Shoup cryptosystem, as such behaviour is not prohibited by law. Moreover, due to time constraints and ample hardware at our disposal, we had the luxury of not having to use more efficient mixnets than the Sako-Kilian model.

From the voter's point of view, there has been a published usability analysis of Helios [Karayumak et al. 2011]. From the elections we held we have gathered our own experience on usability, and in fact the user interface of Zeus is completely re-worked.

Over time researchers have explored possible vulnerabilities in different versions of Helios [Estehghari and Desmedt 2010; Heiderich et al. 2012]. To the best of our knowledge these attacks are not applicable to Zeus.

More recently, the question of endowing Helios with everlasting privacy has been explored [Demirel et al. 2012]. Helios offers computationally secure privacy, i.e., it is currently prohibitively expensive, from a computational point of view, to break voter anonymity. This does not preclude, however, that it will be practical in several years' time. Protocols for everlasting privacy can be used for both homomorphic and mixnet-based elections. Zeus, like Helios, relies on computational privacy and it would be possible to explore the adoption of an everlasting privacy approach.

Due to the switch from homomorphic tallying to mixnets and other requirements and refinements presented below, the Zeus software only retains 50% of the original Helios source code, including the entire homomorphic tallying support, which is unused.

3. THE BIRTH OF ZEUS

Zeus was initiated after the use of electronic voting was permitted by decree for the election of the Governing Councils of Higher Education Institutions in Greece. In each institution the Governing Council is directly elected by its faculty and is its main governing body. The election uses the Single Transferable Vote (STV) system, in which voters do not simply indicate the candidates of their preference, but also rank them in order of preference.

When we were charged with providing an implementation of a system implementing electronic voting we decided to investigate Helios's suitability, as we needed a mature system with a proven record in real world elections and published open source code. The current version of Helios (version 3) allows internet elections from end-to-end: from the moment the voter casts a ballot through a web browser to the publication of election results. It does that by never actually decrypting the ballots but performing a series of homomorphic calculations on them. In the end, the results of the calculations are decrypted and published.

Although using a single system for the whole process is appealing, the use of homomorphic tallying in Helios cannot accommodate voting systems in which not just the individual choices on the ballot matter, but the whole ballot itself. In STV, homomorphic tallying as currently implemented in Helios could pass to the STV algorithm the information that a certain candidate has been selected in rank r by n voters, but this is not enough, as the whole ballot and not just each rank separately is passed around during STV's counting rounds.

That does not mean that homomorphic tallying cannot be used with STV. It is indeed possible to do homomorphic STV using the Shuffle-Sum approach [Benaloh et al. 2009][1], although Helios does not provide this capability now.

At that point we realized that it is not necessary to use Helios's homomorphic tallying capabilities. We decided to use Helios for *counting the ballots*, not for producing the election results. Once we do have a verifiable ballots count, this can be fed to an STV calculator, or indeed to a calculator of any voting system. Since the ballots are published, and the algorithm is also published, a third party can always verify that the results are correct.

4. OVERVIEW OF THE VOTING PROCESS

Each poll in Zeus comprises the following phases: poll preparation, voting, processing, and tallying.

4.1. Poll Preparation

A Zeus account is created for the organizers of the poll. Logging into the system with this account enables creation of new polls through a submission form. The account holder, that is the *poll administrator*, sets the title, description, dates, and other information for the new poll, and chooses who the trustees are going to be. Next, they proceed to select the type of the election and define the content of the poll accordingly. Depending on the election type, the content can typically be either a single list of candidates to rank, or multiple party lists with candidates to select candidates from one, or multiple questions for each to be answered by choosing one or more of the provided answers; see Section 5.3 and Section 5.4 for details on the types of elections supported. Then, the administrator uploads the list of voters that may participate in the poll. A simple CSV format is used, easily exported from spreadsheets.

[1]We are grateful to one of the reviewers for pointing out this work to us. We have not worked out how to use Shuffle-Sum in the particular STV variant used in the elections we had to support, since in order to get elected, a candidate must pass the preferences quota (similar to the Droop quota) and not violate a quota of two elected candidates per school of the university. A candidate that would be elected but violates the per school quota is eliminated. At the end, if the STV iterations result in empty seats, eliminated candidates are brought back and are elected, provided the per school quota is not violated.

Before voting starts each of the trustees must visit Zeus's website and generate and register their encryption keys for the poll. The trustees do not need an account to enter the system. Instead, upon their designation as trustees, Zeus sends them a confidential invitation which they can use to log in to the system and perform their duties.

Thus far, trustees are designated by their e-mail address, and the invitation is sent via e-mail as a secret link to Zeus's website. However, it is easy for Zeus to be adapted to use any system for designation and notification of trustees. Indeed, an adaptation for notifications via mobile phones was actually implemented but has not been used or streamlined within the software yet.

4.1.1. Trustee Key Generation. Trustees must follow their invitation and log in to generate their key pair. The generation takes place entirely in the web browser, assisted with randomness downloaded from the server itself, due to poor randomness in browsers, a practice inherited from Helios. After key creation, the trustee is prompted to save their private key. Once this is done, the trustee is automatically logged out and the browser session ended. Zeus then sends to the trustee a new invitation to verify that they do indeed have their private key. The trustee, following the link, starts a new browser session, in which they are shown the hash value of their public key, and are asked to locate the private key that they have saved. The browser verifies the key locally by comparing its hash value to the one registered by the server. The whole process is roughly equivalent to having to type a password twice, since it is important that the trustees have indeed saved their keys, or decryption will not be possible.

4.1.2. Finalizing the Poll, and Inviting Voters. The administrators may edit all aspects of the poll, up until they decide to *finalize* (or *freeze*) the poll. After that, only limited editing is possible, and most importantly, all registered voters are sent their confidential invitations to vote.

As with trustees, voters do not need an account, but use the secret information in their invitation. Visiting Zeus through the invitation link will initially display information about the poll along with contact details and voting dates. Once the appointed voting time has come, the voting booth is automatically enabled and visiting voters are transferred there directly.

We decided to embed the login credentials in the confidentatial invitation URL for usability and logistical reasons: the users would not have to keep track of a personal account in Zeus, and we would not have to deal with users forgetting their passwords etc. This of course requires that users have access to their e-mails; if they do not have access to the link, they cannot vote. As we report in Section 6.1 this was used as an attack vector, but was dealt with by extending the duration of the attacked polls and adding the capability to send the links by SMS messages. A remaining worry is that a malicious Zeus server, knowing each voter when they come to vote before they cast their ballot, could target them with malicious code. This could be mitigated by requiring the user to submit with their vote audit codes received by a separate channel, like SMS, although this is not immune to attacks (see Section 5.5).

4.2. Voting

The voting booth, once opened, operates entirely locally without further interactions with the Zeus server or any other site. The voter may even disconnect from the internet while inside the booth. The booth appearance depends on the election type, and after vote selection users are asked to review and confirm their choices. After vote submission, a receipt (see Section 5.2) is generated which can immediately be downloaded, but which is also sent to them via e-mail.

During voting administrators have access to information about the ongoing election. The information includes the number of voters that have voted, and for each voter their e-mail, name, whether they have voted, when was the latest e-mail sent to them and the last time the visited the voting booth.

Section 5.1.2 elaborates on the technical issues.

4.3. Processing and Tallying

After the appointed end time, voting is automatically disabled. However, administrators may choose to extend the voting to a new end time and date, even after expiration. For the poll to be finally closed, the administrator has to explicitly select so. Then, the first mixing by Zeus proceeds automatically. Once the first mix is complete, additional mixes by external agents may be added one by one using generated confidential links. Zeus provides a command line toolkit that, among other things, can perform mixing given these confidential links.

Finally, the administrator ends the mixing process, and the trustees are notified again for the decryption process. Each one visits the Zeus website through the invitation link they have just received. The mixed encrypted ballots are downloaded to their client browser, and they are prompted to present their secret poll key. When this is provided, the ballots are partially decrypted and the output is sent back to Zeus.

When all partial decryptions are available, they are combined to produce the final anonymous and unencrypted ballots. Depending on the poll type, the ballots are either sent into another system for tallying and reporting the results (if the tallying and results presentation algorithm is implemented externally, as was the case in the majority of elections we have run), or the ballots are tallied and reported on by Zeus itself. After the results are available, the administrator may download an archive with all generated proofs.

5. CRYPTOGRAPHIC WORKFLOW

Zeus uses essentially the same cryptographic workflow as in Helios [Adida 2008]. In this section we contribute the organization of the Helios workflow around a logical document that contains every cryptographically significant piece of data for a poll, which can be stored in a portable textual canonical form.

The handling of this *poll document* has been implemented in a generic and standalone software module, which is then extended by the web application and the user interface to create a usable voting system.

This core module implements all the required cryptographic functions and validations. The web application extends the abstract workflow module and builds a usable voting system upon a database that handles authentication, election formulation, vote submission, and trustee interactions for key setup and decryption. The user interface offers access to the web application through a web browser, and locally performs vote preparation and encryption, as well as trustee key generation and (partial) ballot decryption.

The core module handling the poll document is completely independent of the other aspects of the system, and therefore can provide much more accessible and reliable verification by third parties, than if they had to dig through irrelevant software layers to validate results. Ideally, users of Zeus should have an API which they could use to query the poll document at will. Zeus does not provide such an API right now, but it is clearly something worth implementing.

From the Zeus point of view, it does not matter where or how the poll document is created, one can inspect and verify the actual process that took place.

5.1. Poll Stages

Zeus represents each poll as a structured document and defines five consecutive stages for it, from creation to completion. At each stage more data are recorded into the document, either coming from the outside, such as votes, or as results of processing existing data, such as vote mixing, or both, such as final ballots decryption.

Once recorded in the poll document, data are never modified, and each stage is validated upon transition into the next one.

At the final stage, the document represents a full account of a completed poll, containing everything needed to verify the process, from votes, to results, to proofs.

5.1.1. Creating. At this stage, candidates, voters, and trustees are recorded into the document. The candidate list is just an ordered list of unique candidate names, which are otherwise not important and are not interpreted. However, different types of elections put special structure within the candidate list, which is used at tallying time to verify and count the ballot, as discussed in Section 5.4.

Voters are likewise represented by uninterpreted but unique names. For each voter, Zeus generates several random numbers, the *voter codes*, that can independently be used both for audit votes and voter authentication, as discussed in Section 5.5.

For trustees, only their public key is recorded. For each poll, and additionally to any external trustees, Zeus always creates and registers a new trustee key for each poll. As per the Helios workflow, this allows the service operator to provide anonymity guarantees in addition to those provided by the appointed committee of trustees.

5.1.2. Voting. When all intended trustees have been recorded, the poll may transition to the *voting* stage by combining (by a multiplication) the trustees' public keys into a single *poll public key* that the voters must use to encrypt their votes. After the formulation of the poll public key, no trustees, voters, or candidates can be modified.

The voter submits to the server the encrypted ballot, (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. The process is detailed in Section 5.5.

For each submitted vote, Zeus generates a cryptographically signed receipt for the voter. With the receipt the voter can later prove that his vote submission was acknowledged as successful, perhaps in the context of an investigation for a complaint. Receipts are detailed in Section 5.2.

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 it replaces. No vote can be replaced more than once.

5.1.3. Mixing. After voting has ended, the poll is put into the *mixing* stage, where the encrypted ballots are anonymized. Only the ballots eligible for counting are selected, by excluding all audit votes, all replaced votes, and all votes by excluded voters.

Voter exclusion is a digital convenience owning to the voter's identity still being known after voting has ended. Anonymity is achieved at the end of the mixing stage, and not at cast time as in traditional ballot box polls. The poll's 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. Instead, the voter is added into an exclusion list along with a statement justifying the decision to exclude them.

The eligible encrypted votes are first mixed by Zeus itself by a Sako-Kilian mixnet. The bulk of the computational work in mixing is to produce enough rounds of the probabilistic proof. The work is distributed to a group of parallel workers, one round at a time. Our deployment used 128 rounds and 16 workers. Verification of mixes is likewise parallel.

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

5.1.4. Decrypting. 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.

5.1.5. Finished. In the transition to *finished*, 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 a canonical representation in text. This textual representation is hashed to obtain a cryptographically secure identifier for it, which can be published and recorded in the proceedings.

5.2. Vote Submission Receipts

For each vote cast a receipt is generated and returned to the voter, as well as registered in the poll document. The receipt is a text file with simple structure listing cryptographic information about the vote and poll. The receipt text is signed with the Zeus's own trustee key for the poll, which is possible because Zeus is a trustee for every poll as described in Section 5.1.1. The ElGamal signature is according to [Schneier 1995].

The receipt contains a unique identification of the cast vote, the vote (if any) which it replaces, the cryptosystem used, the poll's public key along with the public keys of all trustees, the list of candidates, the cryptographic proof that it was a valid encryption, and the signature itself. The validity of the encryption is established by checking the discrete log proof submitted along with the encrypted ballot.

Note that in the current version of Zeus we do not exclude the submission of related ciphertexts; however this was not a major problem for us, as in real elections it is not forbidden for candidates in a party list to give their voters prepared ballots which they can use to vote.

A related problem, indicated by one of the anonymous reviewers, is that "a voter who would like to compromise the privacy of another voter can simply copy the ciphertext of that voter, maybe modify it a bit in a controlled way using the homomorphic property of the encryption scheme, and then look for duplicate ballots (or for a pair of ballot with a difference matching the modification made on the ciphertext) after decryption". It was easy to guard against that, and it was an oversight that we did not do it from the outset. To calm our worries, we checked all past elections and verified that no such chicanery occurred. We subsequently addressed this for any submited ciphertext (g^r, c_2) by including c_2 in the hash that produces the challenge for the proof of knowledge of r. This way, the proof of knowlegde is bound to the specific ciphertext and therefore is not reusable.

Although private information such as user name, IPs, and times are logged in the Zeus servers, they do not appear in the receipt, because we need the entire poll document to be publishable if the election authorities decides so.

The vote submission receipts are not enough to guarantee that a malicious Zeus server cannot submit a vote on behalf of the user, and then argue that the 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. However this was not possible in the elections we had to support, since it would add a step to the users' sequence of actions, and a delay in the production of results; the speedy announcement of results was of major importance, especially as there were days where more than one election was taking place in different institutions.

5.3. Ranked Ballot Encoding

In Helios, ballots consist of answers to binary "yes" or "no" questions, framed in the appropriate way. For instance, a voter indicates k out of n candidates on a ballot by selecting them, in which case the answer "Would you like X as a Y?" gets a yes (1), otherwise a no (0). In STV, 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.[2]

5.4. Tallying Multi-Question Approval Polls

In Section 5.3 we have established the encoding used for ballots that select a subset of the candidates and ranks each one in it, in a strict order.

Multi-question approval ballots, however, are different. There is a number of questions separated into distinct groups, and each question has a number of possible answers. Each question can demand a minimum and a maximum number of answers to be selected for a ballot to be valid. The minimum can be zero, which means a ballot may select a question without selecting any of its answers. This is useful for party list elections, where you can vote for a party list without having to vote for any of the candidates in it. Only questions from the same group can be selected in the same ballot. It is not allowed to select an answer without selecting the question it belongs to. Selecting no answer at all creates an empty ballot and a blank vote.

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

In order to encode this into an ElGamal plaintext we had two options; either create a new encoding for multi-question approval ballots, or translate the approval ballots into ranked ballots and use the existing encoding. Due to time constraints we used approval-to-ranking translation for its simplicity and malleability, a solution that proved to be very practical.

The translation works mainly by structuring the candidate names. Each candidate name has two parts, the question and the answer. All answers to the same question are required to occur together on the candidate list. The first candidate of a question is special, and instead of an answer, the second part of its name contains configuration information such as the minimum and maximum answers, and the group it belongs to. Selecting a questions is signified by selecting this candidate entry. Question groups are identified by consecutive integers starting with 0.

Given this structure, each approval ballot can be represented by correct choices among those entries, i.e. those that respect grouping, minimum-maximum answers, etc. The selections are also required to appear in a strictly ascending order within the ranked ballots.

Note that this embedding of approval ballots within ranked ones is only relevant at ballot preparation and ballot counting. Between those two stages they are treated as ranked ballots. The user interface will never produce an invalid ballot, but Zeus will always validate them at counting time. An invalid ballot is an alert, because it means there was either an error in encoding, or someone has voted incorrectly without using the official web interface. Voting non-interactively via Zeus's programmatic web interface should not pose any problem in itself, but because the voter might get it wrong, or might be executing an attack, it is prudent to look into such incidents.

5.5. 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 is very useful because it 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.

Although the use of audit codes improves confidence in the system, it does not make it invulnerable to attacks. Such an attack would be by a malware that scans the voter’s email for emails coming from Zeus and getting the codes from there. Then the malware could know when the user is submitting a normal vs an audit code, and behave accordingly. The risk could be reduced by having the audit codes received from a second channel, e.g., SMS-messages, although the SMS channel could also be attacked, as shown in possible attacks described against the Norwegian system [Koenig et al. 2013].

5.6. Performance

Mixing the votes dominates the total processing time and exponentiation dominates the mixing calculations. Zeus is written in Python. Python as a programming language has native support for arbitrarily sized integers and exponentiation. It is quite fast but we have used a high performing implementation from `libgmp` via the `gmpy` python module. This sped our exponentiations by about 7 times.

Still, the computational cost of mixing is so high that it only became practical when we parallelized it, which was an algorithmically straightforward undertaking. Hardware was not a problem for us (we are a cloud services provider, among other things), so we could deal with the cost in the old-fashioned way of throwing hardware at the problem. Had we more time in our disposal, we could have explored more efficient mixnet implementations [Bayer and Groth 2012; Terelius and Wikström 2010; Wikström 2009].

Zeus is by default a 2048-bit cryptosystem with a 2047-bit order. Using 16 cores at 2.26 GHz it can mix 10000 votes for 128 rounds, a total of 12.8 million ciphers, in about 65 minutes, producing about 2 billion bytes of on-disk proof text data, compressible to about half of it, since numbers are in a hexadecimal canonical form and thus 2 bytes of text correspond to 1 byte of raw number data. Although not negligible, this load has allowed us not to allocate more hardware so far, our biggest election tallying about 1600 votes.

6. EXPERIENCE

At the point of this writing (June 2013), Zeus has been used in 60 elections around Greece. Figure 1 shows the number of registered voters and the turnout in each election. The registered voters in all elections totaled 15599, while the voters that actually took part in the elections totaled 12171; Elections were carried out successfully in all planned institutions. This was a significant success, as the stakes were particularly high and the voting process charged with emotions. To understand that it is necessary that we provide some surrounding context.

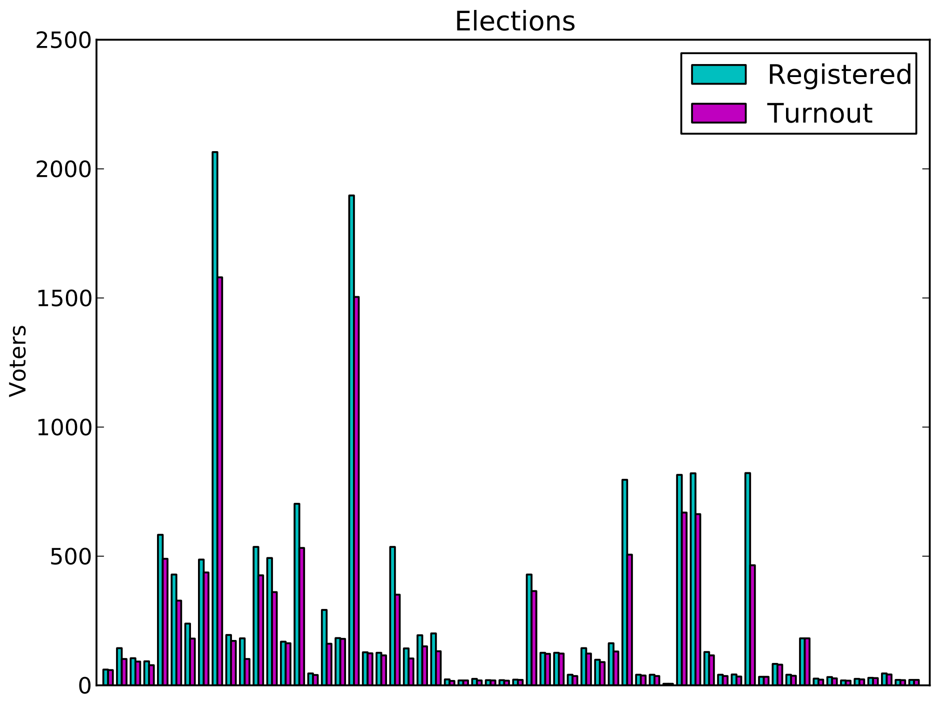

Fig. 1. Zeus held elections to June 2013.

The law that instituted the Governing Councils in Greek educational institutions met resistance from some political parties, student and academics unions. A widespread means of protest was to disrupt the election of Governing Councils, so that they could not be selected. Successive aborted elections pushed forward the adoption of electronic voting, which then in brought Zeus into the controversy.

In particular, the controversy meant that some political parties were against the use of Zeus, or any form of electronic voting. As a result, Zeus was the subject of a number of public denouncements and letters, detailing weaknesses of electronic voting systems in general and Helios in particular. These did not challenge Zeus, as the weaknesses applied to older versions of Helios. Although Zeus was open sourced from the start and we welcomed constructive criticism or the exposure of unknown vulnerabilities, we did not receive any. The atmosphere was at times vitriolic, as when a mainstream political party called the people behind Zeus "techno-fascists", thus introducing a new term in the greek lexicon and bemusing the people involved in the whole effort.

This led to attacks on Zeus, and the voting process supported by Zeus. None of them was eventually successful. We describe them briefly in Section 6.1. A discussion on other security-related aspects follows in Section 6.2. We round up with a presentation of voting logistics in Section 6.3.

6.1. Attacks

6.1.1. University of Thrace. The elections at the University of Thrace had been planned for October 24, 2012. Due to a coding bug, the polls opened the at the time the election was frozen, on October 22. This was discovered by some voters, who went on to vote before the official poll opening. The issue was publicized and the election was annulled and repeated at a later date. Technically, the election could have proceeded without a problem, since there was no way anybody could have tampered with the election results, but the issue was sensitive and politicized, so that the election committee took the most cautious route.

Elections were called again for October 29, 2012, with polls opening at 9:00. At dawn our Networks Operations Center noticed a large number of connections to Zeus. Upon investigation, it was found that Zeus was the subject of a slowloris attack [ha.ckers.org 2013]. The attach was swiftly dealt with, and apart from some initial inconvenience did not cause any real problems. Moreover, the attackers used static IP addresses and could easily be traced and blocked.

6.1.2. University of the Aegean. The University of the Aegean had planned its elections for November 2, 2012. After freezing the election, and when the notifications to the voters had been sent, voters started receiving additional notifications containing bogus URLs from a cracked university server. Although the bogus URLs were not functional, a number of users were confused. The electoral committee decided to cancel the elections and call new elections at a later date. To avoid similar situations in coming elections we recommended the use of Sender Policy Framework (SPF) [Wong and Schlitt 2006] to institutions that had not been using it.

The repeat elections were held on November 9, 2012. The authorities at the university had authorized the set up of a new mail server, to be used solely for the elections, in which all voters were opened accounts. The mail server was hosted outside the university's premises. This avoided a problem that would have risen from a sit-in at the building housing the main mail server, which had been switched off by protesters at election day.

6.1.3. Agricultural University of Athens. On election day, November 5, 2012, protesters staged a sit-in at the mail server building of the Agricultural University of Athens. The mail server was switched off, cutting off access to e-mail, and depriving voters that had not downloaded the access link from voting. The university authorities responded by asking voters to come physically to a location outside the campus, where they could be issued with a new URL.

6.1.4. University of Patras. The University of Patras held its elections on November 7, 2012. Just prior to the opening of the polls a slowloris attack was noticed and dealt with.

6.1.5. University of Athens. A little while after polls opened at the University of Athens, on November 12, 2012, protesters staged a sit-in at the university's Network Operations Server and cut off internet access. This took offline mail access to members of the university and, like in the Agricultural University of Athens, deprived voters that had not downloaded the access link from voting. More dramatically, the internet shutdown affected all sorts of university services, including internet connectivity to university hospitals.

The university responded by extending polling by three days. At the same time, an alternative voter's notification scheme was implemented, via which the voters would receive instructions for voting via SMS. The sit-in was lifted on the last election day, as it was realized that the elections would go on regardless.

6.1.6. National Technical University of Athens. A sit-in similar to the one at the University of Athens was planned at the National Technical University of Athens. However, it did not last long, as the SMS-based infrastructure was already in place, so that it would be irrelevant, and the university authorities made it clear that they would not tolerate the disruption. The election took place without problems on December 10, 2012.

A different, more insidious kind of attack was revealed later on. A few days before the election, one voter had contacted us complaining that the link he had received did not seem to be functioning at all (it should link to a page notifying the user about the coming elections; instead, the user was redirected to the main Zeus page). Two days before the election it was discovered that the user's e-mail stored in Zeus had been changed, and a new e-mail had been sent to the newly entered mail address. The user complained that he never used that address, so another URL was issued and sent to his institutional mail address. The voter voted without a problem on election day.

A few days afterwards a protesters' site announced they had circumvented Zeus security by taking hold of a voter's URL after contacting the election committee and asking them to update the voter's contact details. That explained the e-mail change, and it meant that it was the result of a social engineering attack. Luckily, it had been caught in time—something the protesters did not let on.

6.1.7. Recreate Greece. Encouraged by the success of Zeus in previous elections, a new political party, "Recreate Greece" ("dimiourgia, xana!" in Greek) approached us and enquired whether they could use Zeus for the elections leading to, and during, their party congress. Although not among our target constituency, we decided to allow the use of Zeus in a "best-effort" basis from our part, since that entailed the development of new, interesting functionality—specifically, party list elections, elections with multiple questions and answers, and elections where blank votes are not allowed.

It turned out that in one of their elections a voter wanted to show that e-voting cannot be trusted since anybody can vote with a voter's access credentials, if these are leaked. On April 6, 2013, the voter posted his voter URL on a Facebook page. The electoral committee noted the incident, and therefore moved to disqualify the ballots cast under that voter's name.

6.2. General Security Issues

As explained above, Zeus was born in controversy, and its use was accompanied by much nay-saying. An important argument against the system was that we were not to be trusted. After all, to the voters, as well as to each election committee, Zeus is a black box running in some remote virtual machine. It could be that we were really running a show: instead of Zeus, voters were interacting with something else, and we were putting on a pretense of anonymity and security.

As we were not aware of a practical method to verify the integrity of a running virtual machine, doubters were presented with two choices: either they would have to take us on our word that we were indeed honest, or they could download the code and run it themselves. Nobody took the second option.

Still, at several times throughout different elections we became aware that people did not really believe us in our assertions that we were honest and that the system behaved as we had argued it would. In two distinct occasions members of the election committee discovered that their election key could not be read by Zeus—a bad file. We had instructed them to keep two copies on two different USB sticks. Luckily, in both instances the back-ups were readable. People, though, were to various degrees sure that somehow we would be able to "do something" about it, disregarding our assurances that the only thing we could really do would be to re-run the elections from the beginning.

A similar veil of doubt appeared when people called us confidentially during elections, and shortly after the ballot box had been closed, to ask us "how things were going", and whether we had an indication of the results. We were at pains to explain that there is no way we could have such knowledge.

Doubt in whether Zeus does indeed what is purported to do manifested itself also in a request we received twice to provide to the election committee a log of the exact times each voter had cast a ballot. We explained to the election committee that if the issue at hand was to verify that all votes had been counted correctly, the correct procedure was other, and that the requested information could in fact be damaging in enabling fine-tuned attacks on privacy (an attacher would check that a voter who had been coerced to vote did not vote again afterwards).

Zeus gives to the election committee a full set of cryptographic proofs that can be used to verify the results. We had stressed that these proofs would be the sole arbitrer of any dispute, and the only material that could be really trusted. At some point in the elections, though, we decided to provide, just for convenience, a PDF summary output of the results. We found out to our chagrin that a plain, unsigned PDF document carries more weight than any of our pronouncements.

In particular, we used ReportLab's open source libraries for PDF creation (http://www.reportlab.com/software/opensource/). In one election (elections for president of the Piraeus Technical Educational Institute, held on March 26, 2013), a disgruntled candidate complained to the election committee that the PDF results' creation date was before the ballot closing time. It turned out that he was right, and we found a bug with the PDF libraries (which we reported to ReportLab). The fact that we never meant the PDF document to be the definitive results record of an election seemed to matter little. To resolve the issue we issued an official explanation, complete with details of the bug and possible fixes. We had thought the matter settled, but the candidate came back to us about 20 days later saying that when he had communicated directly with ReportLab he had been told that the problem we had identified could not explain the actual time difference he saw in the PDF. It turned out that he was right again, and the actual source of the error was that the creation timestamp was memoized; since Zeus uses long running processes, the timestamp was stored with its initial value and re-used by all documents that were created by the same process. Again we had to issue an official explanation with reference to the problematic lines of code. The authority of the PDF document is something we came up against repeatedly: people did not seem to care for any other kind of authority apart from that given by the printed matter. Perhaps we should have followed the example of [Adida et al. 2009] where *signed* PDF documents were an integral part of the process: since they somehow assumed that role anyhow, we had better invest them with more substantial authority.

Regarding usability, we did not receive any particular complains from the voters, except for one: that the system would not work with Internet Explorer. The users were informed of that in the notifications they received, but apparently not all of them noticed it, or knew exactly how to download and install one of the supported browsers (recent editions of Firefox, Chrome). The problem was pointed out to them when they contacted the respective election committee about their difficulties with and was thereby resolved; we did receive an angry e-mail, though, from a voter who felt snubbed that she had to go through such an ordeal. Be it as it may, we could extend the system to support recent versions of Internet Explorer, perhaps for the voters if not for the election committee. At the other end of the spectrum we saw in an interview a mention of a voter who had voted in an airplane and was very happy for it.

Voters were not expected to be knowledgeable about cryptography, and we took care to smooth the use of Zeus for them. A lack of understanding led to a few interesting interactions, with some voters trying to read and understand their voting receipts, being simple text files—although the e-mail attachments that contained them did not have a text suffix in order to discourage careless handling of them. They then inquired why they could not understand the receipt contents, and how they could be sure that it was a valid one. This is something that could be solved in time with better informational material and voter education on the basic premises of the system.

Taking into account the feedback we had from our users, it came as no surprise that nobody bothered to use the audit vote feature, whereby a voter tests that their browser is not compromised.

A few asked about it, and when they understood what it was about decided it was not worth the trouble. In truth, the voters that asked about it found it confusing, and we had to visually downplay the related user interface elements in order not to confuse the electorate. In more than one election the election committee asked us if it would be possible to remove any reference to audit votes from the user interface, as well as the audit codes from the notifications sent to voters. We declined; we should probably have put more effort into educating voters and recruiting champions among the voting institutions.

In the same vein, nobody bothered to use our support for external mixnets, although we did explain to those who asked that this was the only way to guarantee anonymity if they could not trust us, otherwise a malicious Zeus server could break the anonymity of all the users. We understood there were two reasons for their reluctance: they were wary of adding any complexity to the voting process, even though this complexity would not be reflected to the voters but only to the election administrator and the election committee. In addition to that, their typical view was that they trusted us that we would do our job properly and take every step possible to safeguard the elections, and that we would not want to shoot ourselves in the foot and compromise the elections ourselves. An important factor may have been that some people in the various election committees knew us personally. Since a flattering view does not really solve the underlying risks, we have been investigating whether it would be possible to have an independent third party that would run an additional mixnet: in this way users would not be inconvenienced and they would no longer need to put their trust in us. For example, Bulens et al. [Bulens et al. 2011] have suggested having a set of servers offering mixing services around the world that could be used by election organizers depending on availability.

Finally, Zeus, when used to run STV elections, suffers from the Italian attack: it is easy to "mark" ballots, by asking voters to put unlikely candidates low down in their preferences, and to check that such ballots appear in the results. An incisive academic that voted in an election in a relatively small institution (so the risks of identification were higher) noted the danger and asked us about it, even though he had had no previous experience with STV, that particular attack, or e-voting in general. He suggested that we could devise a mechanism by which only the part of each ballot that had actually been used in the STV rounds would be displayed in the final results.

6.3. Logistics

Zeus is offered as a hosted service; hosting here entails more than just a well-endowed virtual machine. Elections are a sensitive matter, since people exercise their basic democratic right. They must feel secure, and they must feel they are not slighted by their lack of technical knowledge.

In every Zeus election we have a helpdesk of three persons involved part time (and not all of them concurrently) before the election starts, during the election itself, and after the ballot box closes.

Before the election started the helpdesk is tasked with helping the election committee setting up the election. Although we have produced manuals, for both the election committee and voters, people frequently requested more engagement from our part. As an aside, we should point out that RTFM (Read the F* Manual) rules, as it usually does, and the more so the more technical a user is. We found that users with no particular computing experience were more careful in their use of the system and more conscientious in their use of the manuals provided, than users who, emboldened by their prowess in computing, went on without bothering to consult the documentation.

Our helpdesk colleagues help the election committee to generate their keys and to set up the voters list, ensuring that the input file, a simple CSV document, was well formed—not a trivial matter, as it turned out, as despite our detailed instructions college registrars often failed to produce an up to date list of eligible voters with their current e-mails, or to create a spreadsheet with the required columns. Once the e-mails containing the voters links are sent the helpdesk handles bounces. The bounces are forwarded to the election committee with the voter links removed, so that the election committee can contact the voters and correct their e-mail addresses but cannot abuse the system and start voting in their stead.

It could be possible, in this setting, for the administrators of Zeus not to forward the bounces and to therefore compromise the system by doing the voting themselves. This, though, would require of us to know in advance which voters would not bother to contact the election committee inquiring why they have not received their voter links, at which point the election committee would discover that somebody else has been voting for them. Even if we were crooks we would not run a risk as high as this one.

During the election the helpdesk guides the election committee when they are in doubt about how to correct a voter's details. This happens when a voter contacts the election committee asking them to change their e-mail address to another one. Then, at the end of each election process, the helpdesk is sometimes asked to show the steps for the mixing and decryption process, as well as the presentation of the actual results.

In general, even though the helpdesk workload is not particularly high, it has special time requirements, such as working out of normal business hours to ensure that the communication with the election committee and the voters is as prompt as possible. Both election committees and voters can get very anxious when things are not working in the way they may have anticipated.

7. DISCUSSION

Zeus was a success beyond our expectations. It carried out all the elections we were called to run by law. People must have been reasonably happy with it, since following our official mandate we were asked, and we continue to be asked, to help organize other elections around the country: appointment of representatives for university teachers' union congress, various elections for the selection of rectors, deans, and department presidents, congress election for a political party. We understand that the practical advantages of e-voting outweighs any reservations the corresponding constituencies may have. In a telling interchange, a member of an election committee expressed his gratitude in high spirits, mentioning that in previous elections they would have to spend the night counting, re-counting, and dealing with possible complaints—while with Zeus they packed up early in the evening.

We are continuing to maintain and improve Zeus in various ways, mainly in allowing more kinds of elections with different ballot configurations and improving usability based on the feedback we receive. Zeus mixing is embarrassingly parallel, so we can easily scale it up to bigger elections; decryption from the part of the election committee however takes place on a local computer and may be limited by the resources there. We are exploring way in which we could improve performance, for instance, by developing and distributing a native application.

As explained above, Zeus was controversial, its use was actively opposed, and in the initial set of elections that we were mandated by law to support there were calls to boycott the whole process by abstaining. We want to stress that we had no part in the political debate. We dealt with Zeus as a purely technical problem of implementing a system that would allow elections to take place even when voting in physical ballot boxes was unfeasible. In the end, it came out that people wanted to vote. In these first election batch, which comprised 23 elections, the average turnout percentage was 80.76%, widely accepted as being very high. It was no small relief to know that we did not seem to be doing something against the will of the people.

That points also to a wider application area of e-voting. E-voting is often presented as a adjunct to traditional voting, and no substitute of it. We saw that it can be used as an effective substitute when the conditions on the ground prohibit normal elections. It is our belief, though, that for this to happen any e-voting system must be based in compete openness, the opaqueness of many existing solutions being one of the main arguments against wider adoption of electronic elections [Simons and Jones 2012; Jones and Simons 2012]. If Zeus had not been open source, we could not have been able to defend its use against protesters and detractors. We tried to be as open as possible. The complete Zeus code was available from the beginning of the project, not just released versions, but the whole repository, complete with all our commits and activity. So, in the issue we described with the University of Thrace (recall Section 6.1.1) we were able to issue an explanation with an exact diagnosis to the point of highlighting the commit and the problematic lines of code. As we were not only the developers of the system, but also its administrators, we tried hard to ensure that nobody could doubt our professional integrity.

In hindsight, there are two things that we wished we had done at the appropriate time. First, better educate the voters about e-voting, the concepts of Zeus itself, and vote auditing. Secondly, engage third parties in the process to run mixnets and audit the whole election process independently from us. The use of Scantegrity II at Takoma Park provides a good example both of working together with the local community and employing two independent auditors [Carback et al. 2010].

At the time of this writing Zeus is actively used, and new elections are announced that are using it in the coming days.

8. AVAILABILITY

Zeus is open software, available at https://github.com/grnet/zeus.

9. ACKNOWLEDGMENTS

We would like to thank Professor Aggelos Kiayias for the interesting discussions we had in the start of the project.

REFERENCES

Ben Adida. 2008. Helios: web-based open-audit voting. In *Proceedings of the 17th conference on Security symposium (SS'08)*. USENIX Association, Berkeley, CA, USA, 335–348.

Ben Adida, Olivier De Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. 2009. Electing a university president using open-audit voting: analysis of real-world use of Helios. In *Proceedings of the 2009 conference on Electronic voting technology/workshop on trustworthy elections (EVT/WOTE'09)*. USENIX Association, Berkeley, CA, USA.

Stephanie Bayer and Jens Groth. 2012. Efficient zero-knowledge argument for correctness of a shuffle. In *Proceedings of the 31st Annual international conference on Theory and Applications of Cryptographic Techniques (EUROCRYPT'12)*. Springer-Verlag, Berlin, Heidelberg, 263–280. DOI:http://dx.doi.org/10.1007/978-3-642-29011-4_17

Josh Benaloh. 2006. Simple verifiable elections. In *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop (EVT'06)*. USENIX Association, Berkeley, CA, USA.

Josh Benaloh, Tal Moran, Lee Naish, Kim Ramchen, and Vanessa Teague. 2009. Shuffle-Sum: Coercion-Resistant Verifiable Tallying for STV Voting. *IEEE Transactions on Information Forensics and Security* 4, 4 (December 2009).

Philippe Bulens, Damien Giry, and Olivier Pereira. 2011. Running mixnet-based elections with Helios. In *Proceedings of the 2011 conference on Electronic voting technology/workshop on trustworthy elections (EVT/WOTE'11)*. USENIX Association, Berkeley, CA, USA.

Richard Carback, David Chaum, Jeremy Clark, John Conway, Aleksander Essex, Paul S. Herrnson, Travis Mayberry, Stefan Popoveniuc, Ronald L. Rivest, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. 2010. Scantegrity II municipal election at Takoma Park: the first E2E binding governmental election with ballot privacy. In *Proceedings of the 19th USENIX conference on Security (USENIX Security'10)*. USENIX Association, Berkeley, CA, USA.

Josh D. Cohen and Michael J. Fischer. 1985. A robust and verifiable cryptographically secure election scheme. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science (SFCS '85)*. IEEE Computer Society, Washington, DC, USA, 372–382. DOI:http://dx.doi.org/10.1109/SFCS.1985.2

Denise Demirel, Jeroen Van De Graaf, and Roberto Araújo. 2012. Improving Helios with everlasting privacy towards the public. In *Proceedings of the 2012 international conference on Electronic Voting Technology/Workshop on Trustworthy Elections (EVT/WOTE'12)*. USENIX Association, Berkeley, CA, USA.

Saghar Estehghari and Yvo Desmedt. 2010. Exploiting the client vulnerabilities in internet E-voting systems: hacking Helios 2.0 as an example. In *Proceedings of the 2010 international conference on Electronic voting technology/workshop on trustworthy elections (EVT/WOTE'10)*. USENIX Association, Berkeley, CA, USA, 1–9.

ha.ckers.org. Accessed in 2013. Slowloris HTTP DoS. http://ha.ckers.org/slowloris/. (Accessed in 2013).

Mario Heiderich, Tilman Frosch, Marcus Niemietz, and Jörg Schwenk. 2012. The bug that made me president a browser- and web-security case study on Helios voting. In *Proceedings of the Third international conference on E-Voting and Identity (VoteID'11)*. Springer-Verlag, Berlin, Heidelberg, 89–103. DOI:http://dx.doi.org/10.1007/978-3-642-32747-6_6

Douglas W. Jones and Barbara Simons. 2012. *Broken Ballots: Will Your Vote Count?* CSLI Publications, Stanford, California.

Fatih Karayumak, Maina M. Olembo, Michaela Kauer, and Melanie Volkamer. 2011. Usability Analysis of Helios—An Open Source Verifiable Remote Electronic Voting System. In *Proceedings of the Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE)*. USENIX Association, Berkeley, CA, USA.

Reto E Koenig, Philipp Locher, and Rolf Haenni. 2013. Attacking the Verification Code Mechanism in the Norwegian Internet Voting System. In *Proceedings of the 4th International Conference on e-Voting and Identity (VoteID 2013)*. Springer-Verlag, Berlin, Heidelberg.

Kazue Sako and Joe Kilian. 1995. 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)*. Springer-Verlag, Berlin, Heidelberg, 393–403.

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

Barbara Simons and Douglas W. Jones. 2012. Internet voting in the U.S. *Commun. ACM* 55, 10 (Oct. 2012), 68–77. DOI:http://dx.doi.org/10.1145/2347736.2347754

Björn Terelius and Douglas Wikström. 2010. Proofs of restricted shuffles. In *Proceedings of the Third international conference on Cryptology in Africa (AFRICACRYPT'10)*. Springer-Verlag, Berlin, Heidelberg, 100–113. DOI:http://dx.doi.org/10.1007/978-3-642-12678-9_7

Douglas Wikström. 2009. A Commitment-Consistent Proof of a Shuffle. In *Proceedings of the 14th Australasian Conference on Information Security and Privacy (ACISP '09)*. Springer-Verlag, Berlin, Heidelberg, 407–421. DOI:http://dx.doi.org/10.1007/978-3-642-02620-1_28

M. Wong and W. Schlitt. 2006. Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1. RFC 4408 (Experimental). (April 2006). http://www.ietf.org/rfc/rfc4408.txt