Karlsruhe Institute of Technology

Seminar Kryptographie in der Praxis
Thema 7: Helios

Sommersemester 2013

Institut für Kryptographie und Sicherheit
Europäisches Institut für Systemsicherheit
Fakultät für Informatik
Karlsruher Institut für Technologie

Betreuer: Prof. Dr. Jörn Müller-Quade
Lic. Antonio Almeida
Dipl.-Math. Brandon Broadnax
Dipl.-Inform. Bernhard Löwe

IKS Institut für Kryptographie und Sicherheit

EISS

Europäisches Institut für Systemsicherheit
Institut für Kryptographie und Sicherheit
Fakultät für Informatik
Karlsruher Institut für Technologie
Am Fasanengarten 5
76128 Karlsruhe

Inhaltsverzeichnis

Kapitel 1

Topic 7: Helios

Ferdinand Gantert

1.1 Introduction

Voting is the foundation of a democratic society. More general voting is the basic decision mechanism in a society that is based on collaborative working processes. Because of this there needs to be a really high level of trust in the vote and therefore the voting system with which it was conducted. Trust is especially an issue since it is nearly impossible for an individual to verify even relatively small ballots conducted with traditional paper based voting protocols. Verifying an election becomes even harder the further apart the individual voters are located from each other. This is because to verify a paper based election one needs to observe the decentralized tallying at each location. This high level of trust and the wish to easily verify the result are two basic topics often seen in cryptography. So it seems to be quite obvious to try to design a cryptographic protocol for ballots. Another advantage of cryptographic methods is that in many cases such systems can be used in combination with computer networks like the internet. This reduces the preparations necessary for an individual ballot and cuts the necessity to be at a specific place at a specific time in order to vote. A protocol that uses cryptographic means and the internet to perform verifiable ballots is the Helios protocol, as presented by Ben Adida in his 2008 article "Helios: Web-based Open-Audit Voting"[Adi08] . Helios follows the so called open-audit-voting principles, these two principles are: 1. Voter secrecy, only the voter himself knows her vote and 2. universal audit, everyone, even someone how isn't entitled to vote, can verify the integrity of the vote. [ADMPQ09, page 1]
This article first gives a short introduction to the Helios protocol and its example implementation. Next we have some thought on potential security problems concerning Helios. We will also discuss some scenarios were Helios could be applied and finally have a section on the acceptance of electronic ballot systems and how basic voting principals are implemented in Helios.

1.2 Helios - Fundamentals

When talking about the Helios protocol one also needs to add about which version one is talking. At the moment there are two major versions of Helios: Helios 1.0[Adi08] and Helios 2.0 [ADMPQ09]. There also is Helios 3.0[Hvo10], however despite the fact that it was published in 2010 there is up to the date no article describing the changes made from Helios 2.0 to Helios 3.0[1]. Following the description on the website it seems as Helios 3.0 was a merely cosmetic upgrade to the reference implementation. There seem to be no real changes to the protocol since Helios 2.0, however in Helios 3.0 the Java VM is optional for the voter. The Current minor release seems to be Helios 3.1[2]

[1] There also is no official change-log to be found on the GitHub instance for Helios. Although to be fair a change-log could be created having a deep look into the commits on GitHub or using tools doing that.
[2] https://github.com/benadida/helios-server, last viewed May 21st 2013

The protocol described here is Helios 1.0[3], where necessary we will also discuss the enhancements and changes made in Helios 2.0.

1.2.1 The basic Helios protocol

The Helios protocol itself is quite general and only outlines the basic functionality of an open-audit voting protocol. Much of the security relies on the implementation and there probably needs to be at least small adjustments for most ballots[ADMPQ09, page 13 Conclusion]

1.2.1.1 Ballot preparations and casting

This part of the protocol is the part where the interaction with the voter takes place. Because of this it is the most visible part of the voting system. It is accessible in the world wide web without the need to authenticate. Because of these last two points it probably is also the most vulnerable part.

Ballot preparation This part of the protocol alone has no influence on the ballot whatsoever. The first interaction with the system managing the ballot takes place in the next step when casting the vote. However each vote needs to be prepared before it can be casted. The preparation of the vote ensures with cryptographic means the secrecy of the vote[4] therefore the user is not authenticated until the vote is completely prepared. The system handling this part of the protocol is called Ballot Preparation System (BPS), its process unfolds as follows[Adi08, page 336, section 2.1]:
The voter uses her browser to open the URL of the voting web site. There she starts the voting process by selecting the ballot she wants to vote in. Next the voter is presented all questions in the ballot and the BPS registers all choices made. It is possible to skip single questions and Helios 2 introduced a blank vote which needs to be explicitly recorded in some ballots because the abstentions add to the total vote count which is needed to calculate majorities[ADMPQ09, page 5, Additional Tweaks]. After answering all questions in the ballot the voter is presented a summary of her choices. If the voter confirms this summary the BPS encrypts the ballot and displays a hash of the ciphertext as commitment to the encryption. Helios uses the El Gamal encryption, because it enables shuffling methods[Adi08, page 337, section 2.3] about which we will take later on.
At this point the voter has the chance to audit the BPS. If she wishes so she can use an audit button. In this case the BPN shows the voter the cipher text and the randomness used to create it. So with the already known plaintext and the now given randomness the voter herself can perform the encryption and compare the resulting ciphertext with the ciphertext presented by the BPS[5]. When selecting the audit option the BPS not only shows the ciphrtext and the randomness used to create it but also prompts the voter to create a new encryption. After the re-encryption the voter is back in the state before she selected the audit option.

Ballot casting So besides the audit option there is the option to seal the vote. When this option is selected the BPS deletes the plaintext and the randomness which was used to create the ciphertext from the memory of the client. So after this all secret information on the client is gone and the vote is ready for casting. It is quite important to again emphasize that up until now the voter did not need to authenticate herself and therefore everyone with a web browser can test and verify the BPS as often as one wishes to. After

[3] following the description in [Adi08].
[4] For details on this see section 1.4.2 .
[5] For why this method of verification still is far from perfect see section 1.4.1 .

sealing the vote the BPS prompts the voter to authenticate. In Helios 1 the authentication relies on a user name and a pre shared secret which the voter got when registering for the ballot. Helios 2 has an open authentication model and generally accepts all means of authentication including already existing authentication services[ADMPQ09, page 4, section 2.3]. Helios 2 enables this feature by empowering a trusted server to cast votes as any person for whom the server is trusted.

1.2.1.2 Posting of the vote

Posting the vote is a big part of the open audit concept. Helios posts every casted vote on an online bulletin board[Adi08, page 337, section 2.2]. The vote is displayed next to the voter name or some other unique identifier. As only the Helios Server[6] should be able to decrypt the vote this does't compromise the confidentiality of the vote. Besides all individual votes all subsequent data processing is also posted to the bulletin board. The bulletin board is again accessible as a web site without any restrictions, especially no need to authenticate. So everyone can download and verify the data.

1.2.1.3 Shuffeling of the vote

Helios 1 Since we preserved the individual votes there is the risk that one could establish a connection between the votes and the voters. We need to preserve the individual vote in order to potentially support write-in candidates and revoting. So we need another way to ensure voter secrecy. Helios uses an approach based on a Sako-Killian mixnet[SK95].
This protocol was chosen although there are protocols which have a significant higher performance while delivering the same level of integrity, but the Sako-Killian protocol is relatively simple and easy to explain which Adida thought of as important features for a voting protocol[Adi08, page 337, section 2.3]. The Sako-Killian protocol needs El-Gamal ciphertexts and uses the re-encryption feature of El-Gamal. The re-encryption in El-Gamal can create a ciphertext $c' = (g^S\alpha, y^S\beta)$ which decrypts to the same plaintext as the original ciphertext $c = (\alpha, \beta)$ when choosing and s from the same group as the generator for the original encryption, the randomness used is r + s respectively r. The mix-server takes N inputs and re-encrypts them with re-encryption factors $\{s_i\}_{i\in[1,N]}$. After the re-encryption the cipher texts are permuted according to a random permutation π_N, so that $d_i = Reenc(c_(\pi(i)), s_i)$. As the encryption of the vote itself during the ballot preparation the mixing needs to be verifiable. Therefore the mix-server creates a second so called shadow mix . When someone wants to verify the mix she challenges the mix-server to reveal the permutation and either the re-encryption factors for the shadow mix or the difference between the two mixes. An honest mix server will surly be able to answer both challenges while a cheating mix server will at most be able to answer one of both. So a cheating mix server will be caught with at least 50% probability. But 50% is a much to low percentage for such a sensitive task as voting. To achieve a higher (worst case) probability of catching a cheating mix server Helios creates 80 shadow mixes which results in a overwhelming [7] (worst case) probably of catching a cheating mix server. This kind of proof is interactive which means it needs to be computed for each one who wishes to verify the mixing, but with 80 shadow mixes it uses quite an amount of computing resources so it may be impractical. Therefore in Helios this Honest-Verifier-Zero-Knowledge protocol is transformed to a non-interactive proof using the Fiat-Shamir heuristic[FS86]. The non-interactive proof computes the challenge bits as hash of all shadow mixes. This method

[6] In this context the Helios server is the whole system, handling the ballot, depending on the election there are different entities which are entitled and able to dencrypt a vote. There can be a single trustee able to decrypt the vote, there can be multiple trustee able to decrypt the vote and there can be a situation were only multiple trustees together are able to decrypt the vote.

[7] 0.99999999999999999999999917.

only works when, like in our case, the probability that a cheating mix server is not caught is insignificant. Otherwise the cheating mix server could create shadow mixes until it finds a set which produces the the right challenge bits.

Helios 2 In Helios 2 there is a switch from the mix net approach to a homomorphic tallying approach. Reasons for that are that homomorphic tallying is easier to implement, which is especially important for third-party verification tools, and more suitable for the also new distributed decryption[ADMPQ09, page 3, section 2.1]. Homomorphic tallying also relies on El-Gamal but uses Exponential El-Gamal where one encrypts g^m rather than m in order to achieve an additive homomorphism. For decrypting such a scheme we need a discrete-logarithm computation to simplify that each option of each question is encrypted as a single ciphertext. So a ballot is composed of three elements. First a ciphertext for each option of each question in the ballot. Secondly a disjunctive zero-knowledge proof that each such ciphertext encodes either a 0 or a 1. And lastly a disjunctive zero-knowledge proof that all ciphertexts for a given question summed up are the encryption of 0 to a pre set maximum of answers. The last one is necessary to ensure that a voter complied with the pre-defined maximum of chooses for each question.

1.2.1.4 Decrypting of the vote and tallying

The decription is again verified with a Honest-Verifier Zero-Knowledge proof. This time the proof is build on the Chaum-Pedersen[CP92] protocol and as it has an overwhelming probability of catching a cheating proofer we can again build a non interactive proof with the help of the Fiat-Shamir heuristic. Again we use the non-interactive version in order to save on resources and therefore enable the possibility that nearly a infinite number of observers can verify the decryption at once.
In Helios 1 the description is handled centrally on a single computer.
In Helios 2 the description is distributed among a number of trustees. All trustees are needed to decrypt the whole ballot. This also ensures that only the homomorphic tally of the ballot is decrypted and not an individual vote. For the distributed decryption method there are two options available. The first method is that every trustee generates her own public El-Gamal key with the same parameters (p, q, g) and combining all single public keys via multiplication. The second method is again that all trustees generate their own public El-Gamal key with the same parameters but then all trustees need to generate and publish a Lagrange coefficient to enable threshold decryption.

1.2.1.5 Verifying of the tally and overall ballot

Every individual encrypted vote was posted next to voter's name or some other unique identifier on the online bulletin board. Also all proofs for re-encrypting and mixing have been posted there. So in short all individual votes and all data concerning subsequent processing are posted there. The individual votes are there so the individual voter can check her own vote and the general public can check the voter turnout and the computation of the vote. All subsequent processing is posted in order to be verifiable too, as it is quite an effective way to manipulate a ballot is while tallying. The audit phase is a central security mechanism to ensure integrity, it is the pendant to a public tallying in a paper-and-pen election. Manipulations can not be prevented or made be undone this way, but there is a pretty good chance that they can be detected.

1.2.2 Some details on the implementation of Helios

Regarding the details on the implementation outlined by Adida[Adi08, page 338, section 3 and 4] one must keep in mind that since 2008 web technologies have made huge advance-

ments. Especially the incompatibilities associated with the Internet Explorer[Adi08, page 339 section 3.2 and 3.3] have been mostly fixed in newer versions. Also there are some new Technologies, for example HTML 5, which probably could prove useful. Regarding that we won't go into detail and only present the basic ideas how to implement the protocol. It would surly be interesting how the implementation would look when one would make a complete reimplementation with todays web and web browser technology, but considering the limited space we won't discuss that here.

1.2.2.1 Client side

The client side of the voting process includes the ballot preparation and casting. A specific concept ensuring privacy and secrecy of the vote is the design of the client side application. The client side application is a single-page web application written in java script. Single side meaning in this context that the whole application is loaded when the website is invoked. At this point the client also loads all election parameter including the public key used for encryption. So the whole application can run without interaction with the server[Adi08, page 341 section 4.3 the voting booth]. This is important because the client application stores the plaintext information of the vote in order to compute it and deletes it after encrypting and before posting. So the internet connection can physically be terminated until casting time. So there is no doubt that any plaintext information is submitted to the election server, or for that matter any other server.
To implement the client application in Java Script Adida uses the jQuery JavaScript library[Adi08, section 3.1]. Since the clarity and comprehensibleness of the code are important design goals, in order to facilitate easy audibility, there is a strict separation of presentation and computation. To achieve this Adida uses the templating portion of jQuery.
Java Script is quite a powerful scripting language but like almost all scripting languages it has some performances issues for computational heavy tasks. Therefore Helios uses the LiveConnect technology, with which it is possible to access a clients Java Virtual Maschine directly from java script, in order to perform the cryptographic tasks. However as it greatly benefits the performance this decision also has big negative implications. First Java has had major vulnerabilities in the near past which potentially than can be used to attack helios. Also Java is another pice of software that needs to be installed therefore it not only adds another piece of software that can be attacked but also adds to the complexity of the whole system needed to vote. Another thing that's critical about LiveConnect is that it has since been remove from major web browsers such as firefox [8]. Because of said problems since Helios 3 it is possible to vote without java script installed[Hvo10].
In Helios 2 there were no major changes to the client side application the focus seems to have been on improving the user interface on the client side. The same seems to be true for Helios V3, with the already mentioned exception that Helios 3 made LiveConnects optional.

1.2.2.2 Sever side

The server side application of Helios is written in Python, it runs in a CherryPy application server and stores all its data in a PostgreSQL database.
In Helios V1 the Google App Engine[9] was utilized to build the server application. But since this proprietary software raised some concerns everything based on it was rewritten in Python, using the Django web toolkit for Python for some portions, for Helios 2[ADMPQ09, page 4 section 2.4]. So now the client and the server side application of

[8] https://wiki.mozilla.org/Mozilla_2/Kill_LiveConnect, last viewed May 10th 2013.
[9] http://code.google.com/appengine/, last viewed May 10th 2013.

Helios are completely open source.

1.3 Potential weaknesses and known attacks on Helios

When regarding the security of Helios there is quite some difference between Helios 1 and Helios 2. Helios 2 fixed most of the potential weaknesses in Helios 1. For example the quite problematic authentication via e-mail in Helios 1 is only one option in Helios 2, where more secure methods were added.
A quite extensive analysis of the security of Helios 3 was done in 2012 by the Ruhr-University Bochums Chair for Network and Data Security[HFNS12]. Much of the text in this section will build on their findings, but we won't cover everything they found and will also cover other aspects and cover some aspects they found from an other angle. Saghar Estehghari and Yvo Desmedt[ED10] also did an extensive security-analysis of the then up-to-date Helios 2, however Heinrich et al. cover nearly the same subjects but look at Helios 3 and also have some advanced findings. Another in deep look at the vulnerabilities and how they can be fixed can be found in the article Attacking and fixing Helios[CS12] by Veronique Cortier and Ben Smyth.
Some findings are a combination of implementation issues and security problems concerning the used web browser such issues will mostly be cover as implementation issues.

1.3.1 Potential weaknesses regarding the implementation and used technologies

Strictly speaking Helios is a protocol and therefore outlines the basic principles how a voting system following this protocol could be implemented. But from the start Adida not only specified the protocol but did also a complete implementation for demonstration purpose and published the code of this implementation in 2009 on GitHub[10]. Regarding that this implementation therefore gains the character of a reference implementation an needs to undergo excessive security analysis.
This section is further divided into problems originating from the implementation itself, such as implementation faults, bugs and similar things, and security risks and general problems originating from the technologies used in example implementation.

1.3.1.1 Weaknesses regarding the example implementation

The description of the Helios protocol itself only takes up a relatively small part of the space in the original article Adida wrote on Helios[Adi08] an at least as big a part is about how Helios could be implemented [11]. As the implementation often is more critical in terms of security than the protocol itself, as even a perfectly secure protocol can still be implemented faulty and therefore be insecure, we should have a deep look into the proposed implementation.

Attacks on the voters' interface The *voter interface* is the most visible part of the voting system and it is accessible to everyone since authentication doesn't takes place until just before casting the vote. So regarding this the voters' interface is a quite obvious

[10] https://github.com/benadida/helios-server/commits/master?page=16, last viewed 13th May 2013.
[11] Adida did a full implementation of Helios it is up and running on an web server and everybody who wishes to can create an election or participate in a election, subject to eligibility. This demo system can be found following the URL http://heliosvoting.org, last viewed May 12th 2013.
The code of this implementation (despite the name of the project on GitHub it seems to contain the code of the client side application as well) can be found in a GitHub instance with the URL https://github.com/benadida/helios-server, last viewed May 12th 2013. Development in this GitHub instance is ongoing.

place to start an attack. As Heiderich et al. found out[HFNS12, page 91 section 2] there is indeed an possible exploit regarding problems with the Document Object Model(DOM) used in the in the implementation of the Helios voter interface on http://heliosvoting.org. This problems exist because the Helios client neither uses HTTP headers nor Java Script frame busting code to verify wether a browser is allowed to render a specific frame of a webpage or not. One way of making use of this inadvertence is sending out SPAM-Mails with invitations to the election[HFNS12, page 91 section 2.1]. The e-mails contain links to the actual ballot-web-site however we make use of the insufficient DOM control and use the "target" attribute to create a tab or window with which we can interact. As we have control over the window displaying the actual voting site we can do a cross-scripting attack and therefore record sensitive data. Such Cross Site Scripting (XSS) attacks will be recognized in modern web browsers and the user will be warned or the link will even be blocked. But even when modern browsers block the cross site scripting attempts there is still the possibility of timing attacks[HFNS12, page 92]. With the timing attack the attacker can find out in which state of the voting process the voter is right now and since he has control over the DOM change the URL to a web site controlled by her. There is a good chance the user won't notice that since thanks to the timing attack the change of the URL takes place in exactly the moment the voter proceeds from one step to another and therefore expects a new page to load. To perform the timing attack the attacker needs an additional window or tab to execute Java script the whole time, but since this window can be even a pop-under this shouldn't be too suspicious. Other attacks based on this problem in the implementation are possible. Manipulation of the voter interface can possibly go as far as rearranging the frames of the voting interface, using this ability one could possibly trick a voter into voting an candidate other than the one she intended to.
Such attacks need an active internet connection, the ballot preparation process in contrast is designed not to need one. Regarding that one could argue that this eliminates the danger. However, as it is possible for a suspicious voter to terminate the internet connection, while the browser knows about the plaintext of the vote, it seems a reasonable assumption that the vast majority would not do so. Ergo the risk isn't eliminated by this design feature.
Another problem regarding the client side implementation is that most of its logic works with global variables[HFNS12, page 93]. If one of this variables is compromised it will alter the election result. Depending what variable is altered and when, the manipulation potentially could be huge.

Besides the voters' interface there is another part of the client application where attacks could be potentially even more sever, namely the *administrator's interface*. A possible attack scenario found by Heiderich et al.[HFNS12, page 93 section 2.2] can create an open-redirect and therefore luring the administrator on a web site that is controlled by the attacker. This attack uses the continue_url parameter to place malicious code into a link. However for this to work the administrator has to be tricked into clicking a malicious link. The created link does not contain any secret data like the unique election ID which is also an URL token.
But this attack can be taken further by including a Data URI in the malicious link the attacker can execute script on the actual election domain and still don't have to know any secret data. However such URLs containing Data URIs will be blocked by modern web browsers.
Another vulnerability can be found regarding the use of JavaScript Object Notation (JSON), which is used to supply parameters such as candidate names for the voting application[HFNS12, page 94]. The URI used for requesting JSON literals supplying data to voting application contains the election ID. Therefore the otherwise secret election ID can be relatively easyly retrieved. With that information a full CrossSide Scripting (XSS) attack is possible obtaining read and write privileges for the election and even voter and

administrator accounts. This XSS attack even worked in modern browsers[12].
Another starting point for an XSS attack are the special characters used in Cascading Style Sheets(CSS) to mark the delimiters for a CSS property value assignment block preceded by a selector (namely U+007B and U+007D)[HFNS12, page 95, section 2.3]. An attacker could use this by using this characters to inject the code sequence "{}*{font-family:{" it will turn the Helios web site into a valid style sheet, this style sheet can then be used for an malicious website. When such a malicious website is visited by a voter or an administrator of an election the attacker can apply any text behind the Last U+007B symbol as the font family. Further the attacker can now use Java Script to extract any data from the CSS font family and send it to any domain she wishes. This relatively small problem with encoding of special characters facilitates large scale data theft. For example this method enables an attacker to extract the candidates list, the vote casted by a voter, the election ID and most notably the CSRF token. Using the stolen CSRF token an attacker can submit forms on behalf of the administrator.
So a general problem common to many of the attacks mentioned in this section were web browsers handling input in a way unintended by the developer. This is a more general problem and therefore discussed in detail in the next section. Besides that it became obvious that the server side seems to be secure while the client side needs more attention regarding the security[HFNS12, page 100, section 5]. This seems to be quite dangerous as the client side generates the input that the server sides computes. So regarding the result it is insignificant wether the data was manipulated on the client or server side.

1.3.1.2 General problems regarding the used technologies

The use of a standard web page as means of voting is somehow ambivalently:
First it is quite comfortable for most voters since they already uses webpages on a daly basis and the voting website isn't much different from other website. So the usability should be quite good for a overwhelming majority. Also voting on a website is much more time efficient and comfortable than physically moving to a voting office.
But secondly using a website for sensitive transaction, especially without a hardware token securing the transaction, has some risks since you heavily relay on the security of the used web browser. Since the chosen web browser is beyond our control this problem heavily relies on current user preferences and the focus the web browser vendors put on security.

Browser security and compatibility Browser security is particularly, but not only, a problem with old browsers since they often have known security problems or still have some limitations or errors in implementing new standards[HFNS12, page 94]. Regarding the different browser usage statistics found online[13] one could assume that there is a significant number of users using old versions of web browsers. This number probably will decline over time since most browsers nowadays have autoupdate functions[RBP09, page 6 figure 2]. But any significant number of old or up-to-date but still insecure browsers is a problem to us. And since any eligible voter needs to be able to vote a relatively small number is already significant in this context. So we need quite overwhelming security concerns to exclude a specific web browser from voting[14]. And even when we choose to exclude specific

[12] Modern browser in this context means the up-to date versions of the most common web browsers at the time of the publishing of Heiderich et al. article[HFNS12].

[13] for example see http://www.browser-statistik.de/statistiken/versionen/, last viewed May 8th 2013. Whether or not those statistics based on the user strings recorded by a set of website are really accurate is up to the debate, but since different statistics with different data sets show the same trends they can surly be used as an indicator.

[14] One could make the argument that it is really easy to install an alternative and more secure web browser. However since there are people who have difficulties with that, or actively choose not to do so, we can not make them do so in such a sensitive context as voting.

web browser we need to find a way to securely do that. The most obvious way would be blocking specific user agent strings however this is not reliable since some web browsers already use fake user agent strings to ensure compatibility with most web sites.
Browser security is also influenced by plugins and the browser preference. Some user deactivate java script in their browsers because of security concerns, however web applications such as the Helios client web application, needed to cast a vote, rely entirely on java script. But since the ballot casting web application is open-source and every concerned user could review it herself there should be no particular problem in that. Concerning plugins Helios does not need any plugins so plugins can only indirectly endanger Helios when an insecure plugin used at the same time as Helios corrupts the security of the browser, however this is a general problem in using a web browser so we can't fight that with Helios. Regarding plugins one could criticize the Live-Connect technology used by Helios to access the clients Java virtual machine (JVM). Since the JVM is a component outside the browser which needs to be installed it can be seen as plugin. But since Helios 3 doesn't need LiveConnect and the JVM anymore we won't discuss the security concerns associated with them. Lastly an increasingly problem with websites is the risk of Denial of Service (DoS) and distributed Denial of Service (dDoS) attacks. It seems quite probable that a website containing a election is a likely target for Denial of Service attacks.

1.3.2 Potential weaknesses in the protocol

The Helios protocol itself doesn't gets too much into detail, most details are shown in the example implementation. It sure makes sense only to specify the really essential parts in the protocol to preserve flexibility while ensuring compatibility and basic security. So most potential weakness lie within the implementation and not the protocol itself.
A possible weakness within the protocol is the authentication: In Helios 1 every voter gets her credentials via e-mail. As e-mail is inherently insecure it could be possible that e-mails containing credentials are intercepted and the credentials are used for impersonation. The identification of the voter or to be more precise the impersonation of a voter possibly could be one of the biggest targets for attackers. So the more advanced identification methods in Helios 2 are a step towards possibly more security . However as this method is an interface to other identification methods and not an identification method itself a major security aspect of the voting process now isn't part of the protocol anymore.
Another protocol within the protocol is the encryption algorithm. The past has shown that almost no encryption protocol is save beyond any reasonable doubt, since there are so far no useable encryption algorithm which are mathematical proof-able secure and ready for application. So the used encryption algorithm may be save at the time off setting up the election but an attacker can find a weakness in the encryption at any time. In a worst case scenario the attacker would find such a weakness while a ballot is ongoing. Since the attacker can't necessarily temper with the authentication she still can't manipulate the election with that knowledge, but she can breach the confidentiality of the election. So the confidentiality of the election in a cryptographic voting protocol will be an issue as long as we don't have verifiable safe encryption algorithms. To be fair this sure is an issue with the protocol but mostly it is an issue for cryptography as a whole.

1.4 Acceptance of electronic ballots and consequences

1.4.1 General thoughts on acceptance of electronic ballot systems

The acceptance of electronic voting systems varies by region. An example with which this can be illustrated is the distribution of voting machine throughout the industrialized world. While they are rather common in north america they are quite rare in most regions

of Europe.
Part of the problem sure is that a paper-and-pen based voting system is as simple as it can be. Therefore such a system can be understood by every voter without any special knowledge whatsoever. Open open-audit-voting systems, which build on cryptographical methods, on the other hand need profound knowledge in cryptographics and also programming to asses the protocol and the implementation. Such deep knowledge is only possessed by a small number of people. So here again is the classical problem when designing something: there need to be a tradeoff between complexity and power of the features.
But one can argue that not everybody needs to fully understand the technology used for voting when there are people how do and are trusted by all voters. This principle today is common in our increasingly complex world. For example most of us do not fully understand the security technology in cars or trains but we do trust the engineers that designed it, or some sort of certification authority that certified its security for that matter. But even if we use this principle we need to keep the system as simple as possible as most trustees probably still don't have a background in computer science[ADMPQ09, page 6 section 3.1]. So in a real world scenario it seems to be quite probable that we need to use transitive trust, as many trustees probably still need an expert who checks the system for them and explains it to them. For this reason Helios is kept as simple as possible without compromising security.
So even if we assume that electronic voting in general is accepted, the used systems still has to fulfill some criteria to be actually used. Christian Schaupp and Lemuria Carter did an interesting study in 2005 on the acceptance of e-voting in the age group of 18-24 year olds in the USA[SC05]. They described seven constructs which seem to be important for the acceptance and use of electronic voting. These constructs are most likely age independent while the importance of each is most likely linked to the age. This seven constructs are: usefulness, ease of use, image, relative advantage, compatibility, trust of the internet and trust of the government. The usefulness more precisely is the perceived usefulness for the voter, it is higher when the voter believes that the system is a effective way of participating. The Perceived ease of use describes how much effort an voter has to invest to use the system, so in our case it is manly a question of the user interface. The image is a quite general category following the assumption that a positive reputation will enhance the chances that something will be used. The relative advantage describes the advantage as compared to the way it used to be up unit now, so in most cases paper-and-pen based voting. The compatibility category defines to which degree the system complies with own values, experiences, needs and believes. And lastly the trust in the internet respectively the government measures the surrounding conditions. Meaning that when someone has no trust whatsoever in the internet is it unlikely that she will use e-voting even when the protocol is secure beyond any doubt. Also it is unlikely that someone will use e-voting when she does not believe in government at all. In the examined age range only the structures of perceived usefulness, compatibility and both trust variables were found to be significant[SC05, page 593 results]. There is a chance that the significance of single structures changes with age, to be sure more research concerning that needs to be conducted.

1.4.2 Basic Voting system principles and their implementation in Helios

The principles of an ballot can slightly differ depending on the kind of ballot and where it takes place. But the most basic principles are the same in every democratic election. Since we need a reference we used the principles specified in the Bundeswahlgestzt[15]used for electing the german Bundestag. The principles specified there are: 1. The vote is general

[15]¤ 1 Abs.1 Satz 2 BWahlG: Sie werden in allgemeiner, unmittelbarer, freier, gleicher und geheimer Wahl von den wahlberechtigten Deutschen nach den Grundsäitzen einer mit der Personenwahl verbundenen Verhäiltniswahl gewäihlt.

(no discrimination), 2. the vote is direct (you directly vote and don't vote someone how votes for you), 3. the vote is free (no coercion), 4. the vote is equal (every vote counts the same) and 5. the vote is secret.

1.4.2.1 The vote is general

This principle does not relate directly to the protocol used for the ballot. The ballot protocol only needs to check if someone how wishes to vote is eligible to vote by checking a voter register. So the principle of generalness needs to be considered when creating the voter register. For checking the eligibility Helios 2 and above can use a already existing user database which simplifies the use of Helios since there no longer is a need for building a new voter register for each ballot.

1.4.2.2 The vote is direct

This principle is not used in every democratic election. For example in the US presidential election the voter votes the electoral collage and the members of the electoral collage vote the president. However usually whenever the vote is not direct that is only for practical reasons. Regarding that aspect electronic voting has the advantage that no matter how complex or large the ballot is direct voting should be no problem. Direct voting is achieved in Helios as each voter casts her vote which is later part of the mixing procedure and therefore directly part of the tallying.

1.4.2.3 The vote is free

The vote wouldn't be free if there was some form of coercion. We already talked about that and again have to state that we can't guarantee coercion freeness. However Helios tries to counteract coercion by trying to educate about it and giving the possibility to re-vote. So in this point Helios can't guarantee coercion freeness but is at least as coercion proof as postal voting, the possibility of re-voting can possibly make it a bit more coercion resistant.

1.4.2.4 The vote is equal

This principle should be used for most democratic elections but there are some scenarios where weighting of votes makes sense. This insight also can be found in the evolution of Helios. In Helios 1 the vote is equal by design as the mix-net approach only works when each vote has the same weight[ADMPQ09, page 2 Helios]. This should be the best method to ensure that all votes are counted equal since it is mathematical impossible to weight a vote. The only way to destroy the equality of the vote would be if it would be possible for one person to cast more than one vote. But such behavior should be easily recognizable in the audit of the individual encrypted votes. In Helios 2 on the other hand different weights for votes depending on voter-categorys where an explicit design goal.
So wether vote equality is achieved is a question of which Helios version one is using. More specifically it can be guaranteed by using a mix-net based shuffling.

1.4.2.5 The vote is secret

Keeping secrets secret is a big part of cryptography. So a cryptographic voting protocol should be pretty good at this principle. However in contrast to most paper-pen based voting protocols Helios at least in theory has the possibility of matching voters and vote directly as it needs to keep track which vote belongs to whom in order to enable revoting and audibility of individual votes on the bulletin board. The connection between voter and vote should be scrambled during the mixing phase. So using a fair mix the secrecy is

reestablished. So in Helios the trust in the trustees must be really high because in principle they had the means to decrypt the votes before shuffling and therefore destroy secrecy. To lesser that fear in Helios 2 there are several trustee who are only able to decrypt the ballot together. So secrecy in Helios is a bit ambivalent: when everything or most things went according to protocol the secrecy is better than in voting protocols in use today. Because with a pseudonym voter identification on the bulletin board nobody besides the voting board can even tell that an individual voted. On the other hand when a big security breach occurs during which the ballot is decrypted before shuffling the secrecy is completely gone as there is a list of voter names and their choices. This risk can be a bit lessened by using pseudonyms on the bulletin board such as in Helios 2. In that scenario there also needs to be a security breach in the system handling pseudonymisation.

1.5 Potential Applications for Helios

Not every voting protocol is suitable for any kind of election. This has been true even for traditional pen-and-paper based voting protocols, which is part of the reason why we use different protocols for local, state and federal election and even within in this entities there are different voting protocols.
One type of election for which Helios isn't suitable was pointed out by Ben Adida himself, as he stated that Helios has the same problem as any voting protocol where the vote can be casted anywhere and anytime, namely coercion[Adi08, page 335] He argues that for an election to be truly coercion resistant there needs to be some truly private interaction. Bearing that in mind Helios normally utilizes mechanisms to lower the coercion risk, for example it offers the possibility to revote until the ballot closes. Also even for high stake elections like the Bundestagswahl there is the possibility of postal voting. So coercion surly is a risk for every election and this risk grows when there is no personal interaction, but it probably shouldn't rule out Helios as a voting protocol in most cases. However that said one always should have the risk of coercion in mind when utilizing online voting systems. As already mentioned all non-presence voting protocols inherently bear the risk of coercion. But as electronic protocols have the ability to implement methods of lowering that risk one should do everything one could to lower it.
The reason why different voting protocols are used for different ballots usually lies within the body which should be elected. So there can be things like voting lists as opposed to voting individual candidates, write-ins can be allowed or not, every vote can be weighted the same or there can be different weights. That are only some features which characterise a voting protocol and since some are contradicting it seems unrealistically that a voting protocol will ever be able to support all of them at once. However the basic idea behind Helios seems to be robust enough that nearly all possible features of an ballot could be implemented when needed. So in principal Helios should be suitable for almost any kind of election when one bares in mind that every election is an important project and probably needs some adjustments[ADMPQ09, page 13 conclusion]. For hight stake elections however Helios could be unsuitable for other reasons, most notable client security. Besides problems in the implementation of the user agent, in this context the web browser and the operating system it is running on, is always an inherent risk as shown in section 1.3.1.2.

1.6 Conclusion and future development

So after this brief overview about the Helios voting protocol, its security and how it is perceived by the voter and implements basic voting principles, it is time for a wrap-up.
The protocol itself has been peer-reviewed[16] and is abstract enough to be flexible enough for

[16] To be absolutely precise this is only true for Helios 1 and Helios 2.

a variety of different elections. In the protocol itself there were no major weaknesses or other problems in the design. However there were major weaknesses in the implementation of the client side application that could be used to seriously interfere with a election. It isn't really fair to blame the Helios protocol for weaknesses in it's example implementation since the problems aren't within the protocol. However this findings point out a general issue: The most secure protocol can still fail when its implementation is spotty. And since Ben Adida and Olivier Pereira are respected security experts it shows another problem: The almost exclusive focus on the server-side. However development of this example implementation seems to be ongoing and other implementation based on this code seem to be developed. This is reflected by the fact that the last commit to the GitHub repository was only two month ago and that there are several forks to the repository[17].

A problem that is inherent to any complex protocol is that it probably won't be understand by everybody who is using it. This seems to be especially true for cryptographic voting protocols since in many scenarios everybody should be able to vote and only very few people got a deep understanding of cryptography. Helios's way of enabling trustees to understand the protocol while the huge majority only needs to trust the trustees seems to be the most practical and pragmatic take on that problem. However this way there will probably always be at least some doubt wether the system really works fair under any circumstances. On the other hand open-audit voting offers an so far unknown level of transparency since any individual can verify the vote. And not only can any individual verify the vote, the verification can also be done any time at any place with a computer with an internet connection(if the data is accessible otherwise not even the internet connection is needed).

However using a standard web browser as interface posses gigantic security risks, as this system is administered by a user who in most cases has no spacial knowledge in computer security.

Since the setup of an election is quite simple and the voting takes place in something which is nearly ubiquitous today, namely a web browser, it drastically reduced transaction costs associated with elections. Therefore such a voting protocol could strengthen democracy since the reduced effort and cost associated with elections could lead to more ballots on individual subjects.

Since elections are complex processes for a voting protocol to become as good as possible it needs to be tested on a variety of real world scenarios. Why this is important can be observed regarding the changes made moving from Helios 1 to Helios 2, as the specific requirements of the president election of the Université catholique de Louvain lead to some small and big changes in the protocol and example implementation. This real world testing phase seems to be the phase in which Helios is at the moment. Since there is such a variety of requirements for different ballots this phase could last some time. With the president election of the Université catholique de Louvain[ADMPQ09], the Princeton Undergraduate Student Government election[18] and the deployment as voting protocol for the IACR[19] there already are a wide verity of different ballots conducted with Helios. Probably there need to be much more for Helios to gain general acceptance. A quite important step for Helios would be the use in an true public election since all elections conducted so far where elections within organizations which require to become a member in some form.

Despite all written up to that point there probably won't be one voting protocol that fits all sorts of ballots since there could be even contradicting requirements. Therefore probably there need to be adjustments to the protocol for every election or the protocol needs to define a broad toolbox from which components could be assembled to a specific protocol

[17] https://github.com/benadida/helios-server, last viewed May 21st 2013.
[18] http://heliosvoting.org/2009/10/13/helios-deployed-at-princeton/, last viewed May 21st 2013.
[19] http://www.iacr.org/elections/eVoting/heliosDemo.pdf, last viewed May 21st 2013.

custom made for the election. So regarding that one needs to keep in mind the conclusion from the work on Helios 2: ”each election is a significant project on its own”[ADMPQ09, page 13 section 5].

Literaturverzeichnis

[Adi08] Adida, Ben: Helios: web-based open-audit voting. In: *USENIX Security Symposium* Bd. 17 USENIX Association, 2008, S. 335–348

[ADMPQ09] Adida, Ben ; De Marneffe, Olivier ; Pereira, Olivier ; Quisquater, Jean-Jacques: 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*, 2009, S. 10–15

[CP92] Chaum, David ; Pedersen, Torben P.: Wallet databases with observers. In: *Lecture Notes in Computer Sience*. Brickell, Ernest F., 1992, S. 89–105

[CS12] Cortier, Véronique ; Smyth, Ben: Attacking and fixing helios: An analysis of ballot secrecy. In: *Journal of Computer Security* 21 (2012), Nr. 1, S. 89–148

[ED10] Estehghari, Saghar ; Desmedt, Yvo: Exploiting the client vulnerabilities in Internet e-voting systems: Hacking Helios 2.0 as an example. In: *Proc. 2010 Electronic Voting Technology Workship/Workshop on Trustworthy Elections (EVT/WOTE)(Aug. 2010)*, 2010

[FS86] Fiat, Amos ; Shamir, Adi: How to prove yourself: Practical solutions to identification and signature problems. In: *Lecture Notes in Computer Sience*. Odlyzko, Andrew M., 1986, S. 186–194

[HFNS12] Heiderich, Mario ; Frosch, Tilman ; Niemietz, Marcus ; Schwenk, Jörg: The bug that made me president a browser-and web-security case study on helios voting. In: *E-Voting and Identity*. Springer, 2012, S. 89–103

[Hvo10] http://heliosvoting.org/2010/08/09/helios-v3-released/

[RBP09] Reis, Charles ; Barth, Adam ; Pizano, Carlos: Browser security: lessons from Google Chrome. In: *ACM Queue* 7 (2009), Nr. 5, S. 1–8

[SC05] Schaupp, L C. ; Carter, Lemuria: E-voting: from apathy to adoption. In: *Journal of Enterprise Information Management* 18 (2005), Nr. 5, S. 586–601

[SK95] Sako, Kazue ; Kilian, Joe: Receipt-free mix-type voting scheme - a partical solution to the implementation of a voting booth. In: *EUROCRYPT '95*, 1995, S. 393–403