HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Attacks against Network Voting Systems

Masterarbeit

von

Alexander Schneider

aus

Krasnyj Jar, Kasachstan

vorgelegt am

Lehrstuhl für Rechnernetze und Kommunikationssysteme
Prof. Dr. Martin Mauve
Heinrich-Heine-Universität Düsseldorf

August 2015

Betreuer:
Philipp Hagemeister, M. Sc.

Abstract

The scientific community concentrates on formal or cryptographic safety regarding network voting systems. While this is necessary and valuable work there are also real-world threats to those systems that are not covered by this research. The results are several security issues with released voting systems discovered either during their pilot trials or first runs [HT15, WWIH12, SFD^{+}14].
Electronic network voting systems are potentially very vulnerable to mass manipulation of votes and / or voters. Therefore the goal is to develop systems that are against said mass manipulation while not imposing overly complicated cryptography on the voter.
We concentrate on historic attacks and universal goals for network voting systems to learn a lesson and develop according guidelines that can be universally used to design and implement network voting systems.
The guidelines are furthermore evaluated and applied to existing voting systems.

Acknowledgments

A lot of people supported me during my work on this thesis to whom I wish to express my gratitude. Thanks to all my friends, that had to endure my monologues about the thesis. Also special thanks to Christian Meter for the discussions during the work-time, Martin Mauve for providing the opportunity to write this thesis and of course Philipp Hagemeister for a lot of advise and an open ear at all times.
The various rap play-lists on Spotify also helped a great deal in keeping me entertained during the conception of the thesis. *Can't stop Hip-Hop running through these veins.*

Contents

List of Figures

Chapter 1

Introduction

Voting systems are the foundation of every legitimate democracy. They serve to assert the periodic delegation of power by the people. Network voting systems, especially those using the Internet, have become the focus of a few pilot projects regarding political elections around the world. The hope is to heighten participation among the technically literate young generations. Furthermore, voting over the Internet enables citizens that live out of state, for example soldiers and diplomats, to cast their vote during elections. With Internet voting Disabled, sick and elderly people can vote without having to leave their homes. Estonia is the prime example using their network voting system for municipal elections since 2005 and parliamentary elections since 2007.

Besides their advantages, Internet voting systems may have flaws that enable manipulation of an election. It is hard to manipulate the votes of a lot of people at once during a paper ballot election, but it is potentially easy to do during an online election as soon as an attack vector is found. Because of that an Internet voting system needs to have precautionary security measures to prevent a wide range of attacks.

This thesis concentrates on an attacker, that uses the networking and software components as an attack vector as opposed to systematic and cryptographic flaws. Usually voting systems are evaluated regarding their cryptographic components and not the final product and its implementation in a real world scenario.

Furthermore, we state that the attacker considered by the usual work in the field is not powerful enough. We argue that for instance a nationwide political election over the Internet would likely attract the attention of malicious countries or political parties, which have far more resources than the usual hacker trying to breach systems for profit and sport. Political coups orchestrated by foreign countries happened frequently during the last decade, e.g. in Venezuela in 2002, and therefore coups using Internet voting should be considered at all times.

Thus we have to consider an attacker that has large monetary resources that can be spent in order to facilitate an attack. The skill of the attacker should be considered very high as well. Many countries employ some of the bests security experts in the field through one agency or another. Many countries furthermore maintain countless zero day exploits which they buy from freelance hackers and security experts, see [Mil07]. Motivation should not be a limiting factor, as the attacker should have a very big interest in manipulating the political landscape of a country.

Lastly this thesis also keeps an insider attacker in mind. An insider of the election can be any election authority, an system administrator or a developer of the system, short: everybody with potentially more power over the voting system than any common voter.

1.1 Related Work

To our knowledge ours is the first work to develop specific guidelines for network voting systems regarding design and implementation. There is a handbook published by the Council of Europe [Eur11] which gives legal operational and technical advice regarding the introduction of electronic voting.

Our work differentiates by applying specifically to network voting systems, which are a more strictly defined category of voting systems than electronic voting systems, which also include e.g. DRE[1] voting machines. Furthermore the Council of Europe recommendations are a lot more general while our guidelines specifically target implementation and technical design of the system.

The US Election Assistance Commission also published guidelines [Com15]. Those

[1]Direct-recording electronic

guidelines also concentrate on the setup, documentation and testing of physical electronic voting machines, rather than on general network voting devices like our guidelines do.

1.2 Structure

Firstly, assumptions and definitions are presented in chapter 2. As an example chapter 3 showcases historical cases of vulnerable systems which were breached or shown to be dangerous for use during real elections. In combination chapter 4 takes a look at systems which are trivially flawed to demonstrate how easy a voting system can be unsuitable for political elections. Both previously mentioned chapters are relevant for developing an intuition for the kinds of security flaws we are researching. Following is chapter 5 in which the security goals an Internet voting system should aspire to are defined together with rough sketches of solutions. Following up on the goals guidelines how to achieve those goals are developed in detail in chapter 6. Since the guidelines are of a theoretical nature they need to be legitimized, which is done trough evaluation and application. The evaluation is performed in chapter 7 followed by the application to existing systems in chapter 8. With the application specifically we show that the guidelines can be used predictively as well. Tying up the thesis, the guidelines are used to develop a blueprint for an attack on the Estonian voting system in chapter 9 to demonstrate the practical relevance. Lastly, chapter 10 is concluding the thesis with a summary and future work suggestions.

Chapter 2

Preliminaries

This chapter helps to understand the upcoming work, which uses definitions and assumptions made in the following sections. The definitions are equipped with accompanying examples if needed to deepen the understanding.

2.1 Assumptions

To restrict the thesis to a limited scope some assumptions are necessary. The guidelines introduced in chapter 6 need those assumptions and definitions to function safely.

Assumption 1. The election organizer is not malicious
The election organizer is the political instance that organizes the election before its start, e.g. accepting candidates, creating lists of eligible voters, determining election dates, overseeing the correct implementation of the election, etc. Note that the organizer is not a technical instance like an administrator or any of the instances during the election.
The need for this assumption is best explained by counter-assumption. Assume the election organizer is corrupt. Now the election organizer can manipulate the election in several ways without the technology we are discussing in following chapters having any impact. The organizer can exclude rightful candidates or eligible voters or design the

election procedures in a way that excludes certain parties from voting.
Thus the election organizer must be assumed non-malicious.

Assumption 2. There exists at least one trustworthy off-line registrar
To prevent potential leaking of election credentials every voter must have the opportunity to obtain the credentials off-line through a registrar before the election begins. If that is not possible, a multitude of things can be done to prevent the user from obtaining credentials, e.g. the inhibiting of a voter connectivity during the registration period.

Assumption 3. No two different types of election authority described in chapter 6 are malicious at the same time and cooperate with one another.
Some of the safety properties of several voting systems are based on the fact that no two authorities are malicious and cooperate. This assumption may sound like it is very unlikely but consider that there are potentially only three different instances in total, with some being distributed (and therefore needed to be compromised together).

Assumption 4. At least one person uses public audits
If there is an option to publicly audit (parts of) the election we assume that there is at least one person doing so correctly. If not one single person audits the election if possible, the public auditing measures show no effect. When the system is publicly auditable, even people that are not participating in the election can audit the system and make sure all functionality is as intended.

2.2 Definitions

This section introduces and defines voting system properties and cryptographic procedures needed to understand the following chapters. Definitions are accompanied by examples for better comprehension if required.

Definition 1. Unlinkable Channel
A channel ensuring that there is no possible way of detecting who is communicating with whom. Onion-proxies like Tor do not provide unlinkable channels, since timing attacks are possible, see [LRWW04], which provide a link between sender and receiver.

Definition 2. End-to-End verification
An end-to-end verifiable system provides stringent (cryptographic) proof for every step it takes. Everybody getting a hold of this proof can mathematically ascertain that the system behaved correctly during the steps.

Definition 3. Zero-Knowledge-Proof
A zero-knowledge-proof (ZKP) is a protocol that can be used by one entity, Alice, to cryptographically reliably prove to another entity, Bob, that a certain information is true. Alice does not reveal anything more to Bob with the ZKP than the information being true.
Some ZKPs can be remade to be non-interactive, meaning that the protocol does not require input from Alice and Bob together, but only Bob during the verification.

Definition 4. Homomorphic Encryption
Homomorphic encryption allows certain operations, typically addition and few multiplications, to be performed on the data while encrypted. As an example: Let ε be an encryption, α, β be arbitrary data and $*$ some mathematical operation. ε is called homomorphic regarding $*$ when $\varepsilon(\alpha) * \varepsilon(\beta) = \varepsilon(\alpha * \beta)$.

Definition 5. Threshold Encryption
A threshold encryption scheme splits the private key needed for decryption between n participants. The message encrypted with the corresponding public key can only be decrypted if at least k of the n participants cooperate and contribute their share. Such a scheme is also often called a (k,n)-threshold encryption.

Definition 6. Re-Encryption Mix-Net
A re-encryption mix-network typically has n parties cooperating. Every party re-encrypts the entries of an input-list, permutes its order and hands in the output as input for the next party. The output of the final party is the result of the mix-network.

Chapter 3

Historical Attacks on e-Voting Systems

This chapter highlights weaknesses that were found and possibly exploited in existing electronic voting systems. It is important to have a thorough understanding of those attacks to effectively implement guidelines to prevent them in future applications.

Washington Digital Vote-by-Mail, U.S.A

One popular example of how an Internet voting system can be vulnerable to network attacks, is the Washington D.C. project "Digital Vote-by-Mail (DVBM)".
Public trials were held prior to live deployment of the system, during which a team of scientists used a row of vulnerabilities to gain access to all sensitive information regarding the test-election - see [WWIH12].
The scientists used a simple injection attack which allowed them to execute file-extension names as shell code on the elections web-server. Furthermore, the Intrusion Detection System (IDS), which was monitoring the traffic to and from the web-server, was not configured to monitor TLS-encrypted traffic which the scientists used to get data out of the server. In the end, the scientists were able to retrieve several cryptographic secrets, insert forged ballots and change the outcome of the whole election unnoticed.
This case alone showcases the severe implications that a simple bug in the election web-server code can have.

i-Vote, Estonia

Another case of insecure Internet-voting is presented in [SFD+14]. A team of scientists is invited to inspect the Estonian Internet-voting system during the municipal elections in 2013. During their observations, they find several potential flaws during the execution of the system such as insecure devices being used to prepare the election software or transfer votes, passwords input in front of publicly accessible cameras and errors that could indicate tampering with the systems that are completely ignored.
The scientists also inspected the publicly available code of the server-side part of the application. They reconstructed the client-side and were able to propose attacks which would lead to vote manipulation without detection.
The main issue of the Estonian voting system is that an air-gapped machine is used to count the votes that does not provide any audit trails whatsoever. In theory, if an attacker would somehow compromise the machine, i.e. though corruption of basic soft- or hardware, the outcome of the election could be changed without a hint of foul-play.
This particular case shows that the soft- and hardware of Internet-voting systems has to be designed to be fault-tolerant against administration errors. Even if a voting system is administrated by trained individuals, errors can not be ruled out entirely. Additionally, all systems should possess some form of audit-ability since manipulation is theoretically always possible and should be detectable.

Gujarat Internet-Voting, India

Khare [Kha14] inspects the voting system of the Indian state Gujarat as used in municipal elections in 2011. The Indian system undertook several steps to guard against malicious attacks, such as DoS Protection, IDS, SSL Encryption, etc. However, manipulation of the voting client was not mitigated at all. Furthermore, Man-in-the-Middle attacks through preinstalled, fake certificates are in theory still possible and would be undetected. Issues like vote-selling or coercion mitigation were not heeded to at all. This case shows that it is not sufficient to concentrate on certain areas of security.

Norwegian I-Voting

Norwegians Electoral Management Body (EMB) conducted remote electronic voting trials in 2011, which are described by Stenerud and Bull [SB12]. The Norwegian system is an end-to-end verifiable system which uses return codes delivered out of band (via SMS) for the cast as intended verification on the voters end. Return codes are unique numbers representing candidates, which can be used by the voter to verify if her vote is stored by the voting system as it was cast. The most obvious hurdle such a system has to take is the printing of the return code cards without allowing any one entity to deduce the link between the unique return code sheet and the corresponding voter. The EMB solves this problem by partitioning the printing and linking process into physically and logically separated steps operated by different personnel.
The trials had a largely positive outcome and the return codes seem to improve the voters trust in the voting system. The System nonetheless has open issues, which could theoretically be abused to manipulate the election. There is for example no way to prove that the received return code SMS is sent by the election authorities. This could be used to send out large numbers of fake return codes, triggering an investigation and termination of the possibility to vote electronically during the election as intended per protocol. Another team of researchers was able to replicate the voting homepage to trick people into entering their return code sheets, thus being able to steal their votes. When the credentials are stolen as well the attacker is even able to re-vote on the voters behalf while possibly tricking them into ignoring the fact that they received multiple return code SMS. This indicates that the voting protocol is not clear enough for all voters to follow safely through, even though such attacks were not encountered during the official trials.

New South Wales iVote, Australia

Recently Halderman and Teague [HT15] published an article describing a Man-in-the-Middle attack on the Internet voting system of New South Wales, Australia. This shows that, while adding protocols, like TLS, may improve security to a degree, it does by no means imply that the connection is completely secure as shown by several discovered attacks on TLS implementations. One instance being the FREAK attack on TLS which was also used in this case[1]. Before the flaw was fixed, it was possible to intercept communication between the voter and the voting server through the use of external Javascript code. The flaw enabled an attacker to steal the voter PIN, which is used for certification purposes and her return code, which is used to confirm that the cast vote was stored correctly. An attacker furthermore could insert arbitrary code into the voter website. This naturally gave the attacker the possibility to manipulate votes at will. Researchers furthermore point out flaws in the verification system. The exploit was possible for almost a week after the election went live. This case illustrates again that it is not trivial to secure an election against network-attacks. A big problem is that we currently do not have a set scientific way to deal with partially compromised voting systems in live use. The administration of the New South Wales voting system chose to keep the system running after fixing the exploit, although potentially about 66000 votes could have been compromised. This is not acceptable and should not set a precedent.

[1]Another TLS-flaw being Heartbleed [hea15], which was possible for several years before discovery and would have been a good potential attack point on election-services if available.

Chapter 4

Bad Examples

A lot of proposed systems or ideas have conspicuous fundamental flaws when they would be implemented. The goal of this chapter is to have a categorization of flaws. This helps to see with a quick glance if a system even needs further investigation or is flat out flawed.

4.1 Theory Not Meeting the Real World

To elaborate how cryptographically safe systems are not meeting safety standards once implemented, we analyze a voting system based on the Discrete Logarithm Problem by Chen et. al. [CCJC14].
Firstly, the voter contacts a Certificate Authority (CA) to authenticate and acquire a certificate, which gets embedded in the voters browser. Next the voter contacts the Authentication Center (AC), which is only possible with a valid certificate from the CA. The AC issues a voter-pseudonym signature to the voter. This signature has to be used to validate a vote. As the last step the voter sends her vote including the voter-pseudonym signature to the tallying center. The voter uses a public proxy center for anonymity. A Supervision Authority consisting of all political parties supervises the tallying center.
As shown in previous work [SMHM15], a channel usually considered anonymous is often not sufficient for electronic voting. From now on we will use the phrase unlinkable

channel when we want to describe a channel, which not only provides the anonymity of e.g. a mix-net, but also has the property that timing attacks and other kinds of attacks can not reveal a link between packets used to cast a ballot and a voter.
The above voting system utilizes a public proxy center to anonymize the IP address inside the packets used to cast a vote by sending them through a series of proxy servers. This only obfuscates the IP of the true sender and does not provide an unlinkable channel, since this system is a weaker version of a onion mix-network like Tor, which also does not provide an unlinkable channel for voting purposes.
The whole system is furthermore fragile against Distributed Denial of Service (DDoS) attacks. Three out of the five components, namely the Certificate Authority, the Authentication Center and the Tally Center, are Single Points of Failure. If one of those three authorities is attacked by a DDoS attack, no voting can take place while the attack is ongoing.
An even more severe possible attack is a permanent Denial of Service (DoS) made possible by the Authentication Center. The system allows every voter only one single chance to ask the AC for its voter-pseudonym signature during the so-called authentication phase. One can easily imagine malware which is activated during the authentication phase and causes the voter-machine to not obey the authentication protocol correctly. This would lead to the voter not receiving a correct voter-pseudonym signature. Since every voter only has one chance at receiving one, the voter is now barred from participating in the election. Alternatively, malware can just steal the signature and use it to vote for a certain candidate. Since re-voting is not allowed, the vote can not be changed[1]. This could even be done unnoticed by pretending to the user that the vote was sent according to her wishes. Since the system does not provide an end-to-end verification for the user, this is entirely probable.
As with DDoS, insider-attacks where someone like an administrator has (legal) access to vital components of a system, can be quite devastating. Insider-attacks on the Certificate Authority could entail the handing out of certificates which authorize fake voting-websites which are used for fishing purposes. Since there is no end-to-end verification of the vote, the voter can not know if she voted correctly. An insider in the Authentication Center could collude with an insider of the Tally Center to provide a link between voter-pseudonym signature and ballot, which undoubtedly links a voter to her cast ballot,

[1]As introduced in section 6.1.5 it is possible to let the voter cast her vote securely, despite malware being active on the device in other systems.

completely destroying anonymity. The public proxy center should not be trusted in either case, since there is no guarantee who controls the proxies and whether the route taken through the proxies and its contents are stored somewhere.

This section demonstrates how systems can be theoretically safe part for part, but are vulnerable as soon as network-attacks are possible. Theoretically safe design and cryptography are not sufficient for the implementation of a network voting system without testing it against real world attacks.

4.2 Insufficient Authentication

If a system uses a form of authentication which can be easily intercepted or forged, malicious parties can use the stolen or forged credentials to vote instead of the voter. Although in some systems this would be detectable, it is nonetheless a major design flaw.

One example of this category is the "E-Vote" Software [DiP15]. The authentication consists of a unique token sent via email to eligible voters. Since the email is not encrypted, Man-in-the-Middle attacks can be executed and the identifier can be stolen. Hacking of the E-Mail account is also a possible entry vector to gain the access token.

Another instance of this category is a voting scheme by Nikam et. al. [NRRK]. Their system uses Near Field Communication (NFC) tags for authentication. Every voter submits some required information and gets an NFC tag. This NFC tag, when brought near the voters Android-device, authenticates the user for casting a vote. The NFC tag only checks some characteristic of the device to authenticate. Every unique trait of a mobile phone, i.e. MAC adress, IMEI number, device ID, etc., can be spoofed and is therefore not safe. This obviously enables the sale and theft of any voters NFC tags to cast votes in their stead.

4.3 Insufficient Security

The Virginia Information Technologies Agency [Vir15] published an analysis of the Virginia voting machine WINVote. The technical report is a good example how easy it is to corrupt an election if security (probably) is not involved during the design process of the system. The WINVote system runs a wireless network with weak exploitable security (WEP). The password can be easily extracted from the network flow and thus access to the network is gained. Network-scans reveal a multitude of open ports and that the machines are running an outdated operating system (Windows XP Embedded 2002), which is unpatched. A multitude of publicly available exploits can again be used to gain administrative privileges on the machine. The database containing the votes has an access password but is not encrypted, which allows to e.g. manipulate the database in memory. Every system that is not using cutting edge security and the newest software[2], can per default not be considered secure.

The security of the deployed physical system, specifically its network ecosystem, is as important as the security of the software that is deployed. Unsafe networks allow attackers to compromise the elections structure, which is not acceptable. Flawed authentication is furthermore a high danger for any voting system, since it allows manipulation of votes, without regard to the security of the remaining system parts. Lastly, theoretically safe concepts are not necessarily safe under real-world conditions, as shown by section 4.1, and should therefore be extensively tested under those conditions before deployment.

[2]Regarding security updates.

Chapter 5

Goals of a securely implemented e-Voting system

To accurately construct guidelines for the secure implementation of an Internet voting system, we first must define clear goals and therefrom derive desirable properties the system needs to posses and precautions the system needs to undertake. This chapter does that by inspecting the typically desirable goals an electronic voting system should meet when implemented. We list and explain different approaches, which can be used to achieve every one of the listed goals. Note that not all methods listed are compatible to each other - the implementation of one can mean the exclusion of another.

5.1 Integrity

Integrity is, together with anonymity, the greatest goal of a voting system. Constructing a voting protocol without the property of integrity is meaningless in itself since it allows unlimited corruption of the election rendering its outcome meaningless.
Protocols should be designed in a way that all actions that compromise the integrity, are at least evident. For example: in the Estonian voting system an insider could corrupt the tallying machine and thus change the outcome of the election without anyone noticing. Understandably it is desirable to implement an on-line election in a way, which makes

this impossible. The voting system should at all times either guarantee integrity or the following surrogate-definition which at least defines a mandatory detection of integrity breaches.

Corruption-evidence: An on-line voting system should be implemented in a way which presents some means to an unspecified instance to undoubtedly verify, if any for the election crucial parts were manipulated or behaved against protocol guidelines.

Essentially this can be done by choosing a suitable voting protocol. Following are some methods for ensuring integrity and exemplary systems using those methods. Note that implementation of software can cause bugs, which void the integrity the protocol provides.

5.1.1 Public auditing

Public auditing, if implemented correctly, allows any interested party to verify if the integrity of the election was maintained. This is typically achieved by obligating every participating instance to publish either the actions undertaken by that instance or a surrogate cryptographic non-interactive zero knowledge proof, if publishing of the action itself would void anonymity or some other crucial property of a voting system.
All auditors, either participating in the election or not, can then use the provided public information to verify, if the election was held correctly and without manipulation. Typically, one has to make sure that all published actions can not be forged, deleted or altered after publishing. If any irregularity is encountered by an auditor, it can be relayed to the election organizer and mathematically proven. Many voting systems, like *Civitas* [CCM07], are using a *bulletin board* to implement public auditing.

5.1.2 Cast-as-intended mechanisms

If the ballots are not posted publicly, because they for instance contain information that would disclose either if someone has voted or how someone has voted, a *Cast-as-Intended* proof should be implemented to ensure that the voter can verify whether her vote was cast and that it was cast as she intended and not altered in the procedure. Such a mechanism also helps in case of a compromised voter device, where the attacker would have the power to alter votes before/during the casting-stage unnoticed. If Cast-as-Intended mechanisms are available the attacker can not perform those attacks unnoticed, and the voter can initiate steps to cancel the faulty vote or inform election officials. Naturally, because of those dangers, the confirmation has to be obtainable in a way which works even with a compromised device. Since a compromised device instantly creates an integrity breach when no detection or prevention measures are in place, a Cast-as-Intended, or equivalent, measure is mandatory.

5.1.3 Securing the Application Code

If the voting software needs to be installed by the voter, one has to make sure that the provided software can be retrieved securely and unaltered. It is not an easy task to achieve, since the typical voter does not know how to check the correctness of software with a checksum or even what a checksum is. Possible solutions could be to create authenticated downloads and secure connections via e.g. TLS/SSL, which has been proven to be exploitable in many instances; see [DAM+15, hea15]. Another possibility would be to distribute the election software physically via postal mail on a CD, which brings its own host of problems like voter using devices without a CD drive, faulty CDs, etc. Therefore, the software should be available as an Internet application to permit access to a wide range of devices and to circumvent the above mentioned problems. This does not add any extra constraints to the system, since the problem of a secure transmission medium has to be tackled either way to enable Internet voting.

5.2 Anonymity

Anonymity of the ballot is one of the most important properties of a voting system. Gaining anonymity by commonly used cryptography is in most cases not possible, because the use of encryption often does not allow for universal verifiability and integrity of the election. A simple example can be made if every voter uses a designated election key, known to the tallying instance, to encrypt her ballot. Now the ballot is anonymous, since only the tally authority can decrypt her, but there is no way for an outside auditor to verify, that the tallying authority is correctly doing its job.
Implementing a voting system with a protocol which guarantees theoretical anonymity, can produce flaws which void said guarantee in reality. Widespread malware on voter devices, for example, would be able to record the vote of every user whose device is infected, no matter if the protocol guarantees anonymity or not. One of the goals is to provide anonymity to the voter while preventing any authority from having the power to introduce non-eligible voters or votes.
Another challenge is building anonymous authentication without allowing to sell credentials. If a voter gets the possibility to anonymously authenticate with the election system, then it is most likely possible for the voter to sell her credentials to a third party, since the authorities can not determine if the credentials are used by the rightful owner. This is unacceptable in a democratic election and should be prevented or at least made unlikely to occur.

5.2.1 Everlasting Privacy

One often overlooked problem with network voting is (the missing) appropriation of everlasting privacy. The anonymity of the ballots in a typical Internet voting system relies on the fact that they are protected by a contemporary encryption, because all data is published, e.g. on a bulletin board, to ensure integrity and audit-ability. Since readily available computational power is steadily rising, it is likely that after some time[1] it will be possible for a standard adversary to break the decryption. This very well may influence people to not vote freely out of fear that their vote is publicly available after e.g. 20

[1]In the magnitude of years.

years. At the same time this opens the doors to long time coercion.
Moran et al. [MN06] explores receipt-free universally verifiable voting with everlasting privacy in traditional paper ballot settings. Arapinis et al. [ACKR13] defines theoretical everlasting privacy in the pi-calculus.
Unfortunately, there is no scientific work solving this problem fully for network based voting systems. It poses an interesting opportunity for future work and should be kept in mind during the design of future network voting systems.

5.3 Robustness / Reliability

A voting system has to be reliable during an election, meaning that it should not fail or have an outage during the election times. Naturally, the servers running the system should be connected to an emergency power grid and be monitored at all times. The main goal here is to have a robust system that also withstands active attacks, the biggest and foremost threat being Distributed Denial of Service (DDoS). DDoS can potentially bring the system to a stand-still for large amounts of time during e.g. high traffic hours. As long as traditional paper voting can still be used in parallel, this does not seem like a big issue, but the voters needing the system the most, e.g. overseas citizens, handicapped voters, etc. are potentially not capable of voting via paper ballot and are out of time to request postal voting.
The system also has to be robust enough to work while potentially malicious participants produce unexpected inputs, or try to fill the ballot boxes with chaff if re-voting is allowed.

5.4 Secure Authentication

The authentication of a voting system is an integral part which supports a lot of the above mentioned goals. An abusable authentication would be equivalent to compromised integrity of the voting hard- and software, because invalid votes could be introduced at will by the attacker.

Authentication credentials naturally have to remain secret, otherwise they have no value. One problem is intentional sharing of authentication credentials. As soon as credentials are constituted by some electronic data, e.g. passwords, cryptographic keys, etcetera, they can be sold and used by everyone that is willing to buy the credentials. Therefore, methods to deter voters from selling their authentication credentials need to exist. Furthermore, the (undetected) theft of credentials may be a possibility and should thus be considered.
Another goal is that an insider attacker shall not be capable of forging authentication credentials or create fake eligible voters. The reason being again the power to introduce arbitrary votes through the fake credentials / voters.

5.5 Coercion Freeness

A voting system is coercion free when a third party can not reliably influence the votes of eligible voters. Receipt freeness goes hand in hand with this goal, since all systems which produce any kind of receipt showing the voter choice, or giving the opportunity to deduce the voters choice can be used as a proof for the coercer that the voter followed her instructions.
If it is possible to deduce that a voter cast a vote, but not what she voted, it is exploitable by a coercer as well. Consider the following: if some voters are well known for always supporting party A the coercer which wants party B to succeed can coerce the supporters of party A to not vote at all. Since the coercer has the ability to monitor which voters followed her instructions, coercion is possible.
It is important to understand that systematic coercion is only possible if the coercer can rely on the fact that her finite resources are spent coercing voters that follow her instructions. Coercion freeness is not as important a goal as anonymity, integrity or robustness but is nonetheless a goal worth aiming for, since (automated) systematic coercion is potentially easier to achieve with network voting systems than with traditional paper ballots.

5.6 Usability

Usability is often overlooked when considering desirable properties for an Internet voting system. Systems providing poor usability lead to frustration and erroneous operation of the system as shown in [Eve07]. Typically, cryptography added after the design process of the software results in poor usability. A wrongly used voting system in turn could lead to lessened security properties, especially when the system needs some action by the voter to operate correctly. An example would be the understanding of the audit function of a *Helios* ballot. The user might not use it correctly and thus not notice any manipulation by an attacker or not understand that the audit option does not cast her vote. This is verified through a usability study of the Helios system by Karayumak et al. [KOKV11].
Usability of Internet voting has of yet not been studied extensively to our knowledge. To do so would be orthogonal to the goals of this thesis and has to remain for future work. We still would like to point out the importance of cryptography with good usability in voting systems and take a look at the impact of our proposed guidelines on usability during the evaluation.

Chapter 6

Guidelines for Secure Internet Voting Systems

This chapter offers guidelines on how to achieve the goals defined in Chapter 5.
The first and probably most important statement is that the aspect of network security has to be present during the design process. It is often possible to add additional security through several measures, e.g. adding cryptography, but it is more secure and desirable to include network security in the design and very core foundation of the voting system.

6.1 Measures towards Integrity

The guidelines for integrity are structured analogous to the parts of a typical voting system. It is not necessary that those parts are physically apart but highly encouraged to exacerbate the risk of compromising the system through e.g. the hardware. Parts each system fundamentally consists of are a bulletin board, a tallying instance and an authentication authority. The reasoning for this partition is given implicitly throughout the chapter.

6.1.1 Bulletin Board

As stated in the goals chapter, one needs public auditing or similar end-to-end verifiability options to make sure the integrity of a system can not be endangered. For both purposes, a bulletin board seems like an easy solution. An attacker should not be in the position to add new messages to the bulletin board in behalf of other participants, to delete already posted messages and to change already posted messages. Note that it is possible for the bulletin board itself to be controlled by the attacker.

Structure

The bulletin board is an append-only storage which publicly displays all actions performed by participants of the election. Actions which would void the anonymity of participants can be revised to produce a cryptographic non-interactive zero-knowledge proof which can be displayed instead of the action itself. Additionally, the bulletin board can and should also be built in a distributed fashion. *Distributed Denial of Service* attacks are harder to execute on a multitude of instances than on one. Lastly, the bulletin board can be implemented through a forward-secure append-only persistent authenticated data structure like Balloon [PP] or similar.

Security

Now we consider the possible attacks on the integrity of the bulletin board. An attacker could try to forge messages on behalf of participants of the election. When all messages posted to the bulletin board have to be signed by the instance appending them, this attack is no longer possible. The attacker can only post messages signed with keys she has control of and not forge messages. This measure prohibits the changing of the messages as well. Data structures[1] as outlined above do not allow for changing of the messages in any way.

[1]Balloon, etc.

Balloon furthermore, prevents deletion of messages. Anyone can challenge the server to proof that a commitment is consistent with all previous commitments. Furthermore, as stated in the assumptions, there is at least one instance constantly auditing the bulletin board and is thus privy to messages being deleted. Distributed bulletin boards should also periodically exchange their contents which will make deletion even harder because the message has to be deleted on several instances at once.

Ballot Box

A ballot box should not be able to change, drop or delete votes as stated before. All those problems are already solved by the bulletin board.
The only possible attack vector left is the ballot box not accepting the votes. In this case the ballot box is behaving maliciously and the user can try to cast the vote at one of the other ballot boxes and alert official election authorities that monitor the election. It must be said that this opens the gates for false allegations e.g. claiming ballot boxes to be malicious to disturb the election. Solving this is possible by implementing a non-repudiation scheme, e.g. by Coffey et al. [CSB03], which typically produces a proof of receipt and a proof of origin. If the ballot box has to produce the proof of receipt before it receives personalized information, e.g. during the connection buildup, the ballot box can not deny having received a message without denying the receiving to random participants, which can be detected.
We point out, that the implementation of ballot boxes can be skipped if the non-repudiation scheme is implemented for the bulletin board, since then both instances have the same properties and the bulletin board can simultaneously be used as a ballot box. It still might be advisable to have physically separate ballot boxes, since the bulletin board is the part of the system that has to bear the most network traffic load[2].

[2]Since every part of the system constantly posts to it.

6.1.2 Tallying Instance

A voting system, be it digital or traditional, needs an instance to count the votes and announce the results of the election. The tallying instance must provide verifiability in a manner that even insider can not change the votes or the result. Otherwise it would be an easy task to manipulate the election by simply manipulating the tallying instance. There are two possible ways to verifiably implement the tallying instance. Either by the use of homomorphic encryption, where the sum of all votes is produced in encrypted form or a trough a mix-net before the tallying process itself.

Homomorphic Encryption

If homomorphic encryption like multiplicative Elgamal is used, all the encrypted votes have to be anonymously posted on the bulletin board before the election closes. The tallying instance then simply adds all encrypted votes, and decrypts them non-publicly while producing a zero knowledge proof of correct decryption.
Everyone can take the publicly posted (and encrypted) votes and add them herself. Any auditor then can validate if the proof of correct decryption posted by the tallying instance is for the result of the addition and if it is correct. The tallying instance can thus not add superfluous or fake votes to the tally without an auditor noticing. This satisfies the definition of *Corruption-Evidence*.
However, homomorphic encryption only works for elections where there is a definite amount of candidates which can be marked with "yes" or "no"[3]. More complicated ballot types, that allow weighted rankings of candidates or parties or write in candidates, are currently not efficiently possible with homomorphic encryption. The structure of the ballot has to be determined before the election and be the same for every voter to work correctly with a homomorphic tally. Those types of ballots however are not needed for every type of election, thus making homomorphic encryption a viable choice in some legislatures.

[3]If "yes" is expressed as 1 the ballots can be added in cypher-text.

Mix-Nets

The other solution is to use re-encryption mix-nets to separate the identity of a voter from the vote itself. Optimally several tallying instances are implemented, which each perform a mix that re-encrypts all ballots, and a corresponding zero-knowledge proof of a correct mix. If all proofs are in order and posted on the bulletin board the final instance publicly decrypts all the votes and posts them to the bulletin board together with the final result. A corrupt tallying instance can try to manipulate the tally by swapping out votes before applying a mix or by mixing in a certain way which makes it possible to reverse the mix[4]. Everyone can validate the posted zero-knowledge-proofs of the corrupt tallier and report him to the election authorities. Mix-Nets thus satisfy the definition of corruption-evidence.
An article by Ribarski and Antovski [RA12] benchmarks the performance of different mix-nets and can be used as one of several directives.

Homomorphic Encryption vs. Mix-Nets

Both homomorphic encryption and mix-nets before the tally serve the same purpose of keeping the anonymity of the ballot. The homomorphic encryption has the disadvantage of not being able to support elections where more than a yes and no for certain predetermined candidates is needed. Write-ins or vote-rankings of parties and candidates are only supported by the mix-net approach.
While the homomorphic method can be implemented side by side with threshold encryption to achieve a distributed design and multiple tallying instances, the mix-net approach needs to have multiple tallying instances to produce a secure shuffle. We highly recommend using a distributed implementation regardless of which method is chosen. When the tallying instance is distributed, the trust is also distributed amongst all tally authorities and no single authority has the power to deanonymize a ballot.
The mix-net approach produces a zero-knowledge-proof and a new re-encryption of all votes for every shuffle-step while the homomorphic encryption route only produces a sin-

[4]Thus opening a door to break anonymity.

gle zero-knowledge-proof of correct decryption and a decrypted sum of all votes. This means that for *n* shuffles, the storage space for the mix-net is *n* times as much as for the homomorphic approach.
Since there are a magnitude of re-encryption mix-nets and partially homomorphic algorithms, an objective speed-comparison can not be done easily and mainly depends on the kind of election held, ballot being used and several other parameters like the number of tallying authorities and voters. This lends itself to future work in a separate thesis.

6.1.3 Authentication Authority (AA)

Making sure that a voter is eligible to partake in the election and relaying this information to the other parts of the voting system is the task of the authentication authority.

Registration

The main threat from a malicious authentication authority is the registration of non-eligible voters. Unfortunately, there is no sure way to prevent this during an on-line election when the authentication authority is corrupt. Until this problem can be solved, we strongly advise to use off-line or postal registration, to at least make sure, that the fraud potential is not higher than in traditional elections. The perfect solution is an already in place public key infrastructure like in Estonia, although this is a more fundamental problem which supersedes the design of a secure Internet voting system. This is discussed more closely in section 6.4.1.

Vote-Signing

If the system is designed in such a way that the user authenticates herself to the parts of the system[5], then the only job of the authentication authority is to sign the users votes;

[5]This can be the case when voting registration is done off-line before the election and at the start of the election a list with eligible voter credentials or their identifier respectively is published.

otherwise it also authenticates the voter to the other parts of the system. A voter-signed ballot can not be enough to legitimize a cast vote because the bulletin board would have to posses knowledge of all voters and their signatures, which can be used for coercion practices. Letting the authentication authority sign the ballots reduces the knowledge the bulletin board possesses[6].
This in turn allows for the authentication authority to sign ineligible votes, since there is no publicly verifiable way to prove that the signed ballot is eligible. A solution is to publish the public credentials of all eligible voters, but not the link between a voter and her credential at the beginning of the election on the bulletin board. Any authentication authority has now the job of checking if the ballot fulfills all formal standards and can sign it accordingly. Auditors can check any cast and signed ballot for corresponding published credentials. The auditors are not gaining any sensitive information, because they do not posses the link between credentials and the corresponding voter.

Blind Signatures

There exist paper which propose the use of blind signatures signed by an authentication authority to authorize ballots, e.g. [Oka98, OMA$^+$99]. We argue that blind signatures do not satisfy the property of *Corruption Evidence*. To ensure *Corruption Evidence* it should be possible to detect irregularities during the signing process even if the authentication authority is corrupt and e.g. signs ineligible votes. A set-membership-proof would be the weakest form of cryptographic proof the authority can provide to satisfy corruption evidence[7]. There are two cases which allow us to construct a set membership proof:

Case 1: *The set, i.e. a list of all eligible voters, or their credentials respectively, is public.*
If a list of all voters is publicly known, the situation does not differ from the case described in subsection *Vote-Signing* 6.1.3, where the voter can sign her own ballots and

[6]The voter also has to only authorize to the Authentication authority in this case and to no other authority during the election, which is also beneficial.

[7]Since it would be comprehensible if the authority signed the ballot of an ineligible voter, with the set being all eligible voters and the credentials of the ballots owner the element for which membership is to be proven.

signing by the authentication authority is not needed.

Case 2: *The set of all eligible voters is not public*
If the set of all eligible voters is not public the element for which membership has to be shown, i.e. the credential of the potential voter has to be known to construct a membership proof, see the methods such as [CC$^+$08]. A zero-knowledge-proof showing that the credential is in the set of eligible voters can be constructed if the authentication authority makes a statistical commitment to the set of eligible voters[8]. This approach has two problems: Firstly, the authentication authority can introduce ineligible voters to the set of eligible voters before the election begins, thus being able to produce zero knowledge proofs for ineligible voters. Thus not satisfying the *Coercion-Evidence* requirement. Secondly, because of the need to post the credentials of the voter together with her ballot and the zero knowledge proof of authentication certain coercion attacks may be possible[9].

6.1.4 Network Connection

By controlling (parts of) or exploiting the network connection, an adversary may be in the position to manipulate traffic, or attack communications of an election. Following, we consider security of the internal network and the general Internet traffic.

Internal Voting Authority Network

A closed network separating each part of the voting system should be used to secure the system. Every part of the system can be hosted on machines separated by firewalls and commonplace practices like an intrusion detection system, etc. should be deployed inside the network. Without these measures, an attacker can use a compromised machine to attack other devices inside the network. An example would be an attacker compromising

[8]Without proof, since it is analogous to showing a Hamiltonian cycle zero knowledge proof for a large graph.

[9]A coercer can supervise the voter while she casts her vote according to the coercer, afterwards the coercer can automatically check if any re-votes where made using the same credentials, thus not following the coercers instructions.

a bulletin board server which is then used to compromise the tallying servers. If the servers are strictly separated by protective measures and the firewalls only let certain protocols pass, these kinds of attacks naturally becomes harder.
Only the authentication authority and the bulletin board need a connection to the Internet. The tallying server for instance only needs to communicate with the bulletin boards. An Internet connection would only pose unnecessary risk. Encrypted traffic between the election servers should be mandatory to thwart Man-in-the-Middle attacks.

General Internet Traffic

Internet-communication is generally hard to secure, since a typical connection is routed through several ISPs where each hop is possibly compromised by a Man-in-the-Middle. This warrants encryption of all communication between voter and any voting authority. Besides Man-in-the-Middle attacks, there are newly discovered Man-by-the-Side attacks like *Quantum Insert*[10] where the attacker is embedded in the router software and tries to beat the response of any web-server with its malicious packets. A detection for quantum insert attacks is already made public by Fox-IT [IT15].

6.1.5 Voter Device

A voting-system that is secure on the server-side can still have one major flaw: the voter's device is probably one of the easiest attack vectors regarding network voting. This particular problem has been termed *Secure Platform Problem* [Opp02].
Most voters are not technically adept and use computers that are infested with different kinds of malware, which can be an attack vector for an attacker seeking to manipulate an election. An attacker typically has three different kinds of attack she can perform on the voters device:

- The attacker can *change* the ballot before it is sent.

[10]Since the attack is relatively new there are no scientific publications as of yet. One of many good articles about the function of Quantum can be found on-line by e.g. Bruce Schneier [Sch13]

- The attacker can *delete* the vote while pretending to send it.
- The attacker can *send* her own ballot instead of the voters.

Note: not sending the vote altogether is not possible in a system that employs some measure of public auditing like bulletin boards, since the voter can take notice if her vote is not published. That leaves us with the other two attack types.

Code Voting

The usual solution for changed votes / fake votes is the employment of code voting such as proposed by Kutylowski and Zagorski [KZ07]. Code Voting is a process where every voter receives unique coding tables out of band and uses them to communicate with the election servers without having to trust the device they are using.
For example consider an election authority that generates a coding card for every user, where every candidate in the election gets a unique pseudo-random code. Coding cards can be marked with serial numbers or in another way, so the candidate can be deduced from the code by the election authority. Authorities send the cards to the voter out-of-band by e.g. postal mail. The voter proceeds to vote using her serial number and voting code for her preferred candidate. In consequence, the device, even if malicious, is now unable to deduce which candidate has been chosen as long as the election authority does not collude with the malicious device. If the device tries to change the vote it has to guess the correct code out of a multitude of possibilities, which is not feasible for an attacker to do undetected.
The "*Scratch, Click and Vote*" Internet voting system [KZ10] uses a similar procedure to the one described. It is noteworthy that code voting in itself could be hard to use for some voters and should be designed with that thought in mind.
A problem with this approach is that it enables the possibility of vote selling, since it is easy to just sell coding cards. Discussions how to stop vote-selling and coercion are held in Section 6.5.

6.1.6 Cast-as-intended

As pointed out in the goals chapter, a cast-as-intended mechanism can be beneficial in raising trust and security, especially in systems that do not have a public auditing mechanism.

Confirmation codes

One possible method for a cast-as-intended mechanism is to deliver an unique cryptographic code out-of-band to the voter after the vote is cast. The Norwegian e-voting pilot [Gjø10] generates a unique code per voter per candidate, which is delivered to the voter on a sheet via postal mail. After the voter casts her ballot normally, a code confirmation is received via SMS. If a malicious program or a Man-in-the-Middle changes the ballot, the voter can see this in her confirmation code and re-vote or report it to the authorities. Since it is theoretically possible that both the machine used to vote and the voter's mobile phone are infected with cooperating malware, this does not provide a completely secure solution.

Test-audits

Another possibility is to give the voter the option to audit "test-ballots" after they are encrypted by the voter's device, like used in the Helios system [Adi08]. In principle test-audits work as follows: The voter fills out a ballot and lets her device encrypt it. Only after the encryption, the voter chooses if she wants to cast the ballot or if she wants to audit it. In case the voter chooses the audit option, the device has to reveal the cryptographic secret and randomness used to encrypt the ballot. Following, the voter can then verify if the correct ballot was encrypted. This method can only be used securely when all encrypted ballots are publicly available. If this is not the case the following can happen: a voter audits multiple ballots and decides to finally cast one after all audits were successful. Subsequently, the device pretends to cast the ballot, but in reality generates another ballot instead, which is being cast. When all cast ballots are publicly available,

the voter has the guarantee that the device encrypts correctly and that the correctly encrypted ballot was sent to the ballot box.
A device can not know beforehand if the ballot gets audited or not and has therefore to commit to change the vote or not if it is malicious. Since it is statistically unlikely that the device can guess reliably if the ballot gets audited, the device is unlikely to commit fraud undetected. In principle this method is similar to commitments during cryptographic zero knowledge proofs. This method is an adequate cast-as-intended mechanism.

Confirmation Codes vs. Test-audits

Since both practices allow for a cast-as-intended verification, there is no need to deploy both. Confirmation codes have the drawback of being prone to vote selling, since they produce a receipt together with the code-sheet. Confirmation codes also need additional infrastructure in form of servers in charge of code delivery and creation and (physical) dispatch of code sheets. Test-audits on the other hand are possibly to complicated for the average voter, since she has to understand what they do and how they deliver a cast-as-intended proof for them to be effective. A usability study by Karayumak et al. [KOKV11] found that most voters where unfamiliar with the technical procedure of auditing the ballot and were unable to cast the ballot correctly.
This implies that the choice of implementing test-audits or confirmation codes is a trade-off between coerce-ability and usability which should be thoroughly evaluated before choosing one.

6.2 Measures Towards Anonymity

Usually, ballots are encrypted in some way to protect the link between vote and voter, thus enforcing anonymity[11]. The possibility of deanonymization exists during the tallying phase and decryption of the votes for the purpose of tallying.

[11] Although this does not anonymize whether someone voted, just what her choice was.

6.2.1 Distributed Election Keys

Typically, ballots are encrypted with an election key, where the private part is known by the tallying instance and used to decrypt the ballots for the final tally[12]. If the decryption keys to the ballots lie in the hand of a single authority, it poses a risk to the election. Imagine said authority is compromised, e.g. the tallying server is hacked, then the anonymity is lifted and all voters can be linked to their votes, because usually the ballots can be traced to certain credentials to guarantee auditability. The anonymity is only protected by the encryption of the ballot, but since a malicious entity now has the power to decrypt said ballots without adhering to any anonymity-preserving procedures, the anonymity is voided.
The problem in this scenario is the centralized trust. It is possible to decentralize trust by using threshold encryption as shown by different schemes like [BF97, CDN01]. If the trust is decentralized, a threshold of parties has to be compromised in order to void the anonymity of the voter.
To be more specific: the parties holding the trust-shares should be separated as much as possible to harden attempts to compromise them.

6.2.2 Everlasting Privacy

Everlasting privacy is not being implemented in contemporary systems, but nonetheless should be if the possibility is given. Some recent work shows that there is the possibility to guarantee everlasting privacy with the trade-off of achieving computational correctness instead of unconditional correctness. This should in theory be no problem.

Everlasting Privacy for Homomorphic Encryption

Demirel et al. [DVDGA12] propose the use of Pederson commitments. They show how the commitments guarantee everlasting privacy as long as the discrete log problem cannot be solved. Briefly worded: During the casting of the ballot, the voter generates a

[12]Regardless of how the specifics of the tally-process are designed.

Pederson commitment of the form $\alpha^s \times \beta^t$ and sends s and t over an encrypted channel to the tallying server. Since Pederson commitments are homomorphic themselves, they can be added up and the result can be used to uncommit the homomorphic sum of all votes. Everyone can reproduce the tally since the homomorphically added commitment factors are publicly posted.
One major problem of this method is, that if an attacker can somehow intercept the private channel between voter and server, she can still decrypt the ballot in the future as soon as the encryption used for the ballot is broken. Thus, it may be advisable to search for a better solution to the everlasting privacy problem.

Everlasting Privacy for Mix-Nets

Demirel et al. [DVDGA12] also developed a similar approach that works with mix-nets through re-encryption of the commitment values, which can be revealed after the mix without revealing the link between ballot and voter. The major flaw is again the needed private channel as with the homomorphic encryption approach.

6.3 Measures Towards Robustness

A voting system has to be built robust enough to withstand active attacks and not produce failures during unexpected events. The following sections take a look at how to realize robustness.

6.3.1 Invalid Input

A fairly minor threat regarding robustness is invalid input. The system must deal with all invalid input without failure. Simple routines checking all input for validity should filter out all injection attacks. SQL-injections for example are a fairly common threat for web applications, which could be exploited in a web based voting system to gain access to or

manipulate a database.
Other injections or malformed input can damage the system as well if for instance ballot parsers can be made to produce errors or behave unexpectedly. The solution is a rigorous check of all system parts that interact with any input made to the system. Even a missing file-ending check can compromise the whole voting system as shown by Wolchok et al. [WWIH12].

6.3.2 Risk of centralized Components

In this section we concentrate more on the robustness of the system's implementation. Internet voting systems typically rely on heavily centralized components, e.g. [CCM07, Adi08, SFD$^+$14, WWIH12].
A large-scale DDoS-attack can disturb one of the centralized components long enough to prohibit a certain part of the constituents from voting. Network-level DDoS can be combated by pre-filtering traffic and deflecting obvious attacks like SYN floods or amplification attacks. Note that those deflective measures can naturally be overwelmed themselves, if the attacker has enough resources. Several types of network-layer DDoS and possible mitigations are described by Zargar et al. [ZJT13].
An obvious, although not trivial, solution would be to decentralize the voting system as much as possible. It becomes exponentially more difficult for the attacker to flood several targets at once. Unfortunately, a completely decentralized voting system has not been developed as of yet[13]. Future work in this regard would yield high improvements regarding Internet-voting when successful.

6.3.3 Application Layer (D)DoS

Another matter entirely and potentially more easily preventable is application-layer DDoS. Typically, the voting infrastructure has to perform a lot of work for every cast vote, since ballots typically have to be validated for their correct cryptographic signatures, check credentials and maybe form a proof of correct storage. This asymmetry can be used as

[13]Civitas is in that regard the most advanced, but is still only highly decentralized and not completely.

an attack vector, because the process of casting a vote costs considerably less computational power then validating, storing and tallying the vote. Such an asymmetric workload is the typical flaw abused by DoS attacks.
A solution would be to require the voter to compute some proof of work, which has to accompany the cast ballot and which would eliminate the asymmetry in turn stem application layer DDoS. This methodology is e.g. used by the *Hashcash* procedure [B$^+$02] for E-mail.

6.3.4 Spam / Chaff

Even if straight up DoS is not possible, other similar attacks have to be considered. Civitas e.g. accepts votes encrypted with faux keys to prevent coercion and as a consequence has to filter out those votes before the tally. It is an easy and cheap task to generate a lot of faux votes which bloat up the time it takes to tally. While not rendering the election useless, it can lengthen the election process as a whole and thus heighten the voters dissatisfaction with the system, which in turn lowers the acceptance rates. Since we also considered re-voting earlier in this chapter, the possibility of deliberately introduced chaff is quite likely. Faux votes blow up the time to tally because there is no way to sort them out before the tallying process without allowing certain coercion practices.
Same as with application level DDoS, it would be beneficial for the voting system to force the voters machine to solve cryptographic puzzles. This would also remove the majority of chaff votes. Voters with older or mobile devices would have to wait longer for the vote to be cast, since a cryptographic puzzle needs some amount of computational power to be solved quickly. This may be an undesirable side effect. Other solutions against chaff are not properly researched and remain for future work.

6.4 Measures Towards Secure Authentication

Every voting system faces two major tasks. Firstly, how to securely authenticate the voter and secondly how to do so without allowing for vote selling by vending the credentials. The following subsections take a look at different solutions to those challenges.

6.4.1 Credential Types

Two types of credentials can be used for authentication during an election. The following sections take a look at their suitability.

Expensive Credentials

One method to prevent the selling of credentials is using "expensive credentials". Expensive credentials hold a certain power for the owner and or a penalty for the loss of the credentials. The prime example for expensive credentials is the Estonian electronic ID (eID). The eID is used for nearly all everyday transactions with authorities and governmental agencies. The eID also provides legally binding electronic signatures and is used for voting in the Estonian I-Voting system.
Every voter selling her ID-Card, and the corresponding PIN, gives the buyer the power to sign documents in her stead; from credit loans to tax reports. This disincentivizes voters from selling their credentials since the drawbacks are significant and in the case of the Estonian system are equivalent to almost selling an identity.
Note that expensive credentials typically can not be manufactured on demand with the sole purpose to be used for an election, if the jurisdiction, where the election is held, does not have a credential infrastructure already in place. Most countries, such as Germany or USA, have no legally binding electronic identification and authentication structure in place.

Election Specific Credentials

Election specific credentials are credentials which are created for every eligible voter with the specific purpose to be used in an upcoming election. The first challenge is to transfer those credentials to the voter securely. All digital communication between the voter and the authority issuing the credentials can be potentially infiltrated at several points, e.g. the voter device or any point of the connection, and thus should be

avoided for the purpose of sending credentials over the network[14]. As shown previously in [HT15], Man-in-the-Middle attacks can be used to steal authentication credentials transferred through a network connection. The logical consequence is that credentials should not be transfered over the network, because there is the inherent risk of a man in the middle stealing them.
The solution is to use an out of band channel like postal mail to achieve the, imperfect, security level of traditional postal voting.

6.4.2 Credential Creation

It must not be possible for a malicious entity to reproduce voter credentials. If a system for instance uses coding cards for code voting, those cards have to be printed as they are usually delivered via postal mail. The printing services should not have the potential link between voter codes and voters. One can prevent this by separating the process of printing the cards and preparing them for sending by applying a real world double envelope procedure.
We emphasize that not only the authentication credentials have to be kept secret but also all steps and procedures that could lead to reproduction of credentials. Alternatively the procedures have to be designed in a way that makes them non-replicable.

6.5 Measures Towards Coercion Freeness

To design a coercion-free system, one needs to make sure the system does not produce any kind of receipt of how one voted, i.e. there is no way, optimally even with the help of the voter herself, to deduce how anyone voted. Systematic coercion only works if the coercer can be sure that the voter followed her instructions with a reasonably high chance, since otherwise the coercer would have spent her resources without effect.

[14]Consider that we can not assume there is already some authentication in place, as it is during the election, so we can not reliably encrypt the communication at this point.

6.5.1 Privacy Preserving Communication Medium

To preserve all aspects of coercion freeness, the voter needs to communicate over a medium that is privacy preserving and anonymous. The guidelines developed in section 6.2 do not apply to this section, since they only help preserve the anonymity of the ballot, but not any other information, which may help a coercer.
Some work done regarding a privacy preserving communication medium suggests using low-latency networks like *Tor* during ballot transmission to solve this requirement. However, as shown in [SMHM15], low latency networks are not enough for use during an election. A powerful attacker can use pattern matching to deduct if and when a voter cast a ballot, despite the use of the low-latency network. This clearly breaks parts of the coercion freeness definition. For example, the coercer can check that voters, who were instructed to abstain[15], voted against her instructions.
A solution could be to use high-latency networks, since they possibly deny patten-matching attacks. Please note, that there is no scientific work extensively studying this claim and thus future work regarding privacy preserving communication during ballot casting is needed.

6.5.2 Receipt Freeness

Other aspects of coercion freeness include properties such as receipt freeness. Receipt freeness defines that the use of end-to-end verifiability measures, should not allow the voter to use the measure as a receipt.
The possibility of re-voting should introduce enough uncertainty for the coercer to not know if she is paying resources for canceled votes. If the voter has the ability to re-vote, the following situation is likely even when the system is not receipt free.

- Voter votes according to the coercers instructions
- Voter obtains receipt and shows it to the coercer

[15]Either to prevent voters of a certain party from voting, or after handing the credentials to the coercer and not being allowed to re-vote.

- Voter uses re-voting to change vote to personal preference after some time during the election

The coercer can not be sure that the voter did not re-vote and thus has no real means of enforcing the coercion.
For this to hold true the system needs to not give away which votes belong to the same voter or credentials. This should be kept in mind when implementing re-voting. The benefits of re-voting should thus be strongly considered when designing a voting system.
Another possibility explored by Oppliger et al. [OSH08] is to provide multiple code sheets with only one being the code sheet which is able to cast valid votes that are counted at the end of the election. There are even more complex variants of this scheme explored in the same paper. We argue that multiple code sheets with different credentials would be too complex for the typical voter and are thus not realistically applicable.
The possibility of re-voting seems to be the easiest way to solve this problem and should be applied. However, the complete elimination of coercion is an open problem not only in network voting systems.

6.5.3 Alternative Measures

One detrimental approach is to allow coercion but make it evident as coined by the *Caveat Coercitor* voting system [GRBR13]. The approach is to use the amount of re-votes to estimate an upper bound on coercion and not count votes which re-vote for different candidates, because this indicates possible coercion. If the estimated margin is bigger than the margin between two candidates in the election, the election can be declared invalid. This disincentivizes coercion, since the coercer can only cancel votes and not change the outcome of an election.
One should keep in mind that this exact procedure can be abused as well by deliberately heighten the upper bound with re-votes to sabotage the election every time.
Although the *Caveat Coercitor* approach is not a perfect solution, the general concept should be researched further. Coercion does not necessarily has to be prevented if it can be detected, similar to integrity and corruption-evidence.

6.6 Combination of Introduced Measures

Now that we discussed measures that should be undertaken separately from each other, a look at the synergies or lack thereof is in order.

6.6.1 Homomorphic Encryption + Code Voting

During the guidelines homomorphic encryption is proposed as well as code voting to solve different problems. These properties clash together, since homomorphic ballots need to be designed in a way where the vote for a candidate is always a well defined numerical value, which allows for the addition of votes in cyphertext. A ballot prepared for homomorphic voting is a vector where each entry represents a candidate and where the voter puts a one for the chosen candidate and a zero for everyone else.
Code voting uses a separate code per voter per candidate, which obviously can not be used to add ballots in encryption. Code voting codes can not follow any pattern, since they would be vulnerable to reconstruction, which defeats the purpose of the codes as explained in 6.1.5. Because the codes are practically random it does nothing to add the codes. With code voting every vote has to be decrypted individually and then matched to a candidate before tallying.
The alternative for homomorphic encryption is the use of several mix-nets performed by the tallying authorities before the tallying process and is thus the only option which does not clash with other security guidelines[16]. The form of the ballot does not matter for a tallying implementation which uses mix-nets.

6.6.2 Blind Signatures + Re-Voting

Blind signatures used to sign ballots clash with the ability to re-vote. When blind signatures are used the authentication authority signs blinded ballots after checking the credentials of the voter. The vote itself is unblinded, stripped of the credentials and posted.

[16]Mix-Nets also allow for ballots which are not used for single candidate elections but other forms like multiplicity rankings, etc.

This way the tallying authority has no possibility to detect re-votes, since only the authentication authority would know if someone voted multiple times. There is no way to determine which votes in the ballot box were casted by the same person, since the purpose of blind signatures is removing the link between voter and ballot. We can not let the authentication authority check for re-votes, because there would be no way to find the previous casted vote and mark it invalid.
A possible solution would be to let the authentication authority derive a random token for every user which is attached to the ballot before signing. This would potentially make the ballot vulnerable to coercion. The coercer could remember the token used to cast the coerced vote and monitor the bulletin board for another vote with the same token, which would tell the coercer that her instructions were ignored by the coerced voter. Thus, blind signatures can not be used together with the other proposed guidelines and we recommend using threshold encryption of the ballot to preserve ballot anonymity.

All other proposed building blocks should be implementable side by side without any problems. A table showing all proposed measures in short is shown in figure 6.2.

6.7 General Security

Apart from special precautions, voting systems should share a few general measures with other software in need of security. A few of the most important practices are described in the following section.

6.7.1 Formal Training

Security is only effective if all parties concerned with the development and execution understand its meaning. Formal training for all personnel, from system administrator to developer and election officials, should be mandatory and extensive. The best security

practices will only function sub-par as long as the person handling it has no full understanding of what is going on. A negative example is the Estonian voting system. As already mentioned a study by Springall et al. [SFD$^+$14] shows that the Estonian administrators behaved in a way that suggests they were not aware of the security risks they were imposing on the voting system. One instance is staff using private, previously used USB drives to transfer votes to the air-gapped tallying server or updating the voting servers ad-hoc during the election without testing and knowledge whether the update will break election software.
An attacker can use this human failure as an attack vector. Uploading manipulated software packages into the repositories that are used by the election officials or compromising the private devices of the administrators in a targeted attack is not an impossible task.

Another possible attack vector leveraging the lack of knowledge would be the following: If election officials are to generate a distributed threshold election key, with which all ballots are encrypted, there is the possibility that the officials chose a faulty generation method[17] which would allow to an attacker to reconstruct the private key.

6.7.2 Layering the Software

Another commonplace practice is the layering of software. Every part of the voting system should be encapsulated in an abstract layer which can only access functions strictly needed for its functioning. A layer always depends on the functions of the "lower" layers but never the other way around. Typically, only the highest layer has an interface to the user. An attacker trying to compromise the system by abusing programming flaws can get access to the highest layer and can not cause more damage without further attacks. Would the software not be layered getting access through any flaw would mean that the attacker has full control over all components of said software.

[17]By e.g. choosing broken algorithms, key-length which are too short or too small prime numbers, which can speed up the factoring of the key.

6.7.3 Threat Modeling

During the design phase continuous threat models should be created and evaluated for each iteration of the system design. Early threat models highlight possible attack vectors during a design stage where they can be eliminated intrinsically rather than hot-fixed during or after the implementation. Threat models can be conducted with a multitude of techniques, e.g. with attack trees [Sch99].
A threat model can be used to create security requirements as shown by Myagmar et al. [MLY05]. The threat model helps to identify and methodically weigh the risks a system faces to create requirements, which should be adhered to during the implementation.
Microsoft, for instance, uses threat modeling as an inherent part of its software design process. A threat modeling guide by Torr [Tor05] has been published in the past.

6.7.4 Further Analysis

After the design is complete, the implementation can begin. During the implementation *static code analysis* tools, that search for possible threat vectors should be used frequently to find programming errors which open up attack vectors. Static analysis tools help sift out all common security issues without much overhead work.
It may be worth considering incorporating *model checking* or *data-flow analysis* into the implementation process. Unfortunately, there is currently no work analyzing possible advantages and downsides of using those methods during the design of voting systems and would be a good subject for further work.

Penetration Testing

When the system implementation nears completion, public and private penetration testing should be conducted. Although penetration testing will most likely not find all security flaws, it will most certainly find issues that would otherwise not come up. A penetration test that comes up without any findings does not mean that the system is secure; it simply means no flaws where found during the test. The test is rather representative instead of confirmational. As pointed out in an article by Arkin et al. [ASM05], penetration testing helps develop mitigation techniques which in turn help securing the software.

Fuzzing

Simultaneously to penetration testing, fuzzing can be used to find rare security flaws, that would not come up organically. During fuzzing, automated tools use invalid data for all inputs of the system and observe the systems reaction to it. The data is either generated by the fuzzing tool, if the syntax of the input is known, or valid data is permuted. The goal of fuzzing is to generate input-data that is valid enough to pass some of the parsing of the test system to observe if the invalid parts cause problems. Fuzzing can be seen as a tool to test border-cases. Its benefits are illustrated comprehensively in an article by Oehlert [Oeh05]. A use case in voting systems would be for example generating filled out ballots through fuzzing, to test whether there is any possibility that malformed submitted ballots can cause a disruption in the system.

6.7.5 Further Resources

A lot of the above methodology is well-tried by different organizations with security backgrounds and summarized in several documents like the *Microsoft Security Development Lifecycle* [Lip04], *Methodology for Secure Software Design* by Fernandez [Fer04] and *SAFECode Security Development Practices* [SHR^{+}11].

6.8 Summary

The above defined guidelines can be found in tabular form in figure 6.2. Furthermore, a score-chart has been developed which can be used during development or rating of a system by marking guidelines that have been fulfilled. By taking a look which categories have the most unfulfilled criteria, the main flaws of the system become apparent. This is tested further in chapter 8. The score-chart template can be found in figure 6.1.

Verifiability	**Anonymity**	**Secure Implementation**	**Coercion**	**DDoS / Chaff**
☐ Bulletin Board	☐ Mix-Nets	☐ Network Security	☐ Re-Voting	☐ Crypto-puzzles
☐ Offline-Registration	☐ Homomorphic Encryption	☐ Code-Voting	☐ Expensive Credentials	☐ Heavily decentralized components
☐ Public Voter-List	☐ Threshold Encryption		☐ Coercion Detection	
☐ Confirmation Codes				
☐ Test-Audits				

Figure 6.1: Template score-chart regarding the developed guidelines

Measure	**Description**	**Gained Property**
Mandatory Bulletin Board. See 6.1.1	All parties of the election post messages to the bulletin board. All parties sign their messages. May be distributed.	Public audit-ability, may be used for end-to-end verification.
Ballot mix-net before tally. See 6.1.2	Tallying authority mixes ballots before decrypting and tallying. Zero Knowledge Proof is generated. Tallying authorities should be distributed.	Link between voter and ballot is obscured during tally.
Homomorphic encryption of ballots. See 6.1.2	Ballots are encrypted with a homomorphic encryption, enabling calculating the tally in cypher-text and decrypting the final result.	Link between voter and ballot is obscured during tally.
Threshold Encryption for ballots. See 6.2.1	The elections private key for ballots is distributed among n authorities. at least k out of n needed to decrypt ballots.	Harder deanonymization by corrupted tallying authorities.
Offline Voter registration. See 6.1.3	If no authentication structure in place, the registration of voters and obtaining of credentials should be done offline.	Reduced chance of credential theft.
Publication of eligible credentials. See 6.1.3	Publish credentials of eligible candidates on bulletin board.	Impossible to add faux voters during election.
Regulate Network of Servers. See 6.1.4	Use IDS, Firewalls, restricted access to the Internet, layered network, etc. for the network containing the election servers.	Secured authority network.
Code Voting. See 6.1.5	Coding cards or similar mechanisms work around the secure platform problem.	Voters may vote securely despite corrupt devices.
Ability to re-vote. See 6.1.5	To disincentivize vote selling re-voting should be possible.	Less coercion.
Cryptographic Puzzles. See 6.3	To prevent voter from conducting DoS attacks and generating chaff.	DDoS protection.
Expensive Credentials. See 6.4	If authentication structure in place make the credentials expensive to contain vote selling.	Less coercion.
Coercion Detection Capabilities. See 6.5	*Caveat Coercitor* or similar measures to determine level of coercion in the election.	Coercion Detection. Less coercion.
Confirmation Codes. See 6.1.6	Codes representing candidates are sent out of band as cast-as-intended proof to the voter after casting the ballot.	Cast-as-intended. Heightened trust in system.
Test Audits. See 6.1.6	The Voter has the ability to audit a ballot after the device has committed to the encryption.	Cast-as-intended.
Decentralize components. See 6.3.2	Every component of the voting system should be as decentralized as possible.	Better (D)DoS protection.

Figure 6.2: A condensed form of the guidelines derived in Chapter 6.

Chapter 7

Evaluation

In the previous chapter we introduced several guidelines which should be used to attain security properties or mitigate flaws a network-voting system might experience. This chapter evaluates those guidelines in a self-reflective manner. Following, we assess the complexity of implementation, efficiency and usability of the proposed measures. Because of the theoretical nature of the thesis, we are restrained to a theoretical evaluation. Some guidelines depend on the specific implementation and are therefore not analyzed in this chapter, but considered in the coming chapter 8. An overview of the results from this chapter is given in the figure 7.1. Future work would consist in implementing a prototype and evaluating the system in a real-world election or simulation.

7.1 Ranking Criteria

This section details how each of the criteria is assessed. Since there is no quantifiable way to put numbers on the evaluated parts, we use gradations which are defined as well. The gradations assist in displaying the evaluation clearly arranged. All gradations are color-coded for visual clarity.

7.1.1 Efficiency

Efficiency measures the effectiveness in combating the flaws a guideline is designed for. The efficiency of a guideline can be rated as one of the following.

- *perfect* - A flaw is completely mitigated by the proposed method.
- *mainly solved* - A flaw is mainly mitigated and will statistically only appear rarely after implementation of the proposed method.
- *appeasing* - The appearance rate of severity of a flaw is strictly less than before the implementation of the proposed method.
- *ineffective* - The flaw was not or only slightly mitigated by the proposed method.

7.1.2 Usability

The best guidelines will not be beneficial to a voting system if they eliminate a flaw but in the same turn complicate the usability of a system in a manner which impairs the typical voter. The usability is rated in one of the following categories.

- *enhanced* - The usability of the system is enhanced after implementing the proposed method.
- *unchanged* - The usability of the system was not affected by the proposed method.
- *worsened* - The usability of the system is impaired after implementation of the proposed method.
- *disruptive* - The usability of the system is strongly impaired by the implementation of the proposed method.

7.1.3 Complexity of Implementation

A heightened complexity of implementation entails harder maintenance and the potential for more bugs in the source code. It is therefore desirable to keep the complexity of the source code as low as possible. We rate the increase of complexity in one of the following categories.

- *substantial* - Major substantial parts have to be added to the system or reworked for the proposed method. Should be avoided if possible.
- *minor* - Some new modules, procedures or code has to be added to the system.
- *unchanged* - The complexity does not change, happens when e.g. some part of the system is replaced by another equivalent complex part.
- *improved* - The proposed method leads to a lowered complexity. Very desirable.

7.2 Evaluating Parts

Now we evaluate the guidelines in reference to the defined evaluation criteria. The summary of the evaluation is appended in tabular form at the end of the section.

7.2.1 Mandatory Bulletin Board

Complexity: *substantial*
Adding a bulletin board (section 6.1.1) to a network voting system requires completely new server-side parts. Furthermore, all other existing parts of the system have to be adapted to continuously communicate with the bulletin board, which has to be done through adjustments in the voting protocol as well as the software.
Efficiency: *perfect*
The bulletin board is a prerequisite for most public auditing schemes, as well as other

guidelines proposed in the previous chapter. The purpose of enabling public auditing techniques is completely satisfied.
Usability: *unchanged*
The existence of the bulletin board demands no special actions or other adaptation from the voter. One could even argue, that the added ability to voluntarily use the bulletin board and audit the election is an usability improvement.

7.2.2 Tallying Mix-Nets

Complexity: *minor*
Implementation takes several modifications of the tallying authorities protocol, including implementing a mix-net algorithm and the creation of a zero knowledge proof, which are fairly common methods.
Efficiency: *mainly solved*
The goal of tallying mix-nets (section 6.1.2) is to guarantee anonymity of the ballot by breaking the link between a voter and her ballot. Correctly executed mixes completely achieve this goal, however there is still the possibility for all mixes to be corrupt, which would allow deanonymization. This possibility is small but nonetheless exists.
Usability: *unchanged*
Mix-nets are implemented as a protocol change for the tallying authorities and no changes on the user end have to be made.

7.2.3 Homomorphic Encryption

Complexity: *minor*
To implement the homomorphic encryption (section 6.1.2) the voter client needs to be able to encrypt the ballots using a homomorphic algorithm. This should be a fairly standard task. Apart from that only minor changes, like switching the decryption algorithm, need to be made to the tallying protocol.
Efficiency: *appeasing*
Homomorphic encryption has the goal to provide public auditability without forfeiting

the anonymity of the ballot. This is achieved as long as the tallying authority is honest. Since only one authority has to be compromised, an attack seems not impossible. Either way the audit-ability is never forfeit, only the anonymity.
Usability: *unchanged*
It does not make any difference to the end-user whether homomorphic or conventional cryptography is used, since it only affects the tallying method.

7.2.4 Threshold Encryption

Complexity: *minor*
The election key generation algorithm needs to be replaced for this method. Furthermore, a few minor changes in the tallying protocol, i.e. the step where the tallying authorities assemble the key.
Efficiency: *mainly solved*
Threshold encryption (section 6.2.1) tries to mitigate the fact, that a single tallying authority possibly can posses the link between voter and ballot, which breaks ballot anonymity. As long as at least k out of n tallying authorities are honest this works as intended for a (k,n)-homomorphic algorithm. For a well chosen k and n it is very unlikely that k instances are corrupt at the same time, but still theoretically possible.
Usability: *unchanged*
No changes are made for the voter.

7.2.5 Offline Voter Registration

Complexity: *improved*
An online registration feature that otherwise is implemented does potentially not have to be realized.
Efficiency: *perfect*
Offline voter registration (section 6.1.3) tries to circumvent the theft of credentials that is possible during online registration. Since the credentials are handed to the registrant in person, no theft is possible.

Usability: *disruptive*
An Internet voting systems aims to be a complete on-line substitute to traditional paper ballots, which is disrupted by the off-line procedure of registering for the election.

7.2.6 Code Voting

Complexity: *substanial*
The codes have to be generated and delivered out of band to the user. The tallying instance has to be adapted to decipher the codes uniquely for each ballot before tallying as well.
Efficiency: *mainly solved*
Code voting (section 6.1.5) tries to mitigate the secure platform problem. By giving the voter the chance to communicate through code, the voter's device does not have the chance to manipulate the votes or to eavesdrop as long as the authority holding the link between the code and its meaning is not malicious.
Usability: *disruptive*
Code voting requires the entry of multiple codes, which can be mixed up, and is prone to error.

7.2.7 Re-voting

Complexity: *minor*
Server-side changes have to be made to accept only the last valid vote from each user. This should not take too much work since it is only the collection of all votes and dismissing every vote that is not the most recent.
Efficiency: *appeasing*
Re-Voting (section 6.5.2) is meant to provide disincentives to coercers as they can not be sure if their resources are well spent, because the voter can disobey at every point of the vote-process by voiding previous votes. If voters would really use the possibility to re-vote and whether the coercer would abstain from coercion when re-voting is possible, has not been scientifically shown yet. The disincentive for coercion is provided, whether

it is relevant to the process is unclear.
Usability: *unchanged*
The voter can just re-vote by casting a vote as she would do otherwise.

7.2.8 Cryptographic Puzzles

Complexity: *minor*
The implementation of cryptographic puzzles (section 6.3.4) needs changes on the server side to only accept clients or ballots with solved puzzles and on the client side to accept and solve puzzles which was implemented many times for non voting systems.
Efficiency: *appeasing*
Preventing DDoS and chaff is well done by cryptographic puzzles. It however does not prevent all kinds of DDoS.
Usability: *unchanged*
Usability should not be affected, since the voter casts her ballot as before. The cryptographic operations are done unnoticed by the voter.

7.2.9 Expensive Credentials

Complexity: *substantial*
Expensive credentials (section 6.4.1) have to be deployed for the whole voter-base and exceed the scope of every voting system. If expensive credentials are to be deployed specially for voting purposes, the implementation effort is huge, because more systems than the voting system have to adapt to the credentials for them to be considered expensive credentials.
Efficiency: *mainly solved*
It is very unlikely, but still possible for voters, to sell their expensive credentials.
Usability: *enhanced*
Voters can use the same credentials they are accustomed to using somewhere else in contrast to special credentials they have to manage.

7.2.10 Confirmation Codes

Complexity: *substantial*
Implementation of confirmation codes (section 6.1.6) needs a similar logistical effort as code voting.
Efficiency: *mainly solved*
Confirmation codes strive to serve as a cast-as-intended-proof which is fulfilled, but can be hijacked by e.g. compromising the medium which is used as the out-of-band channel to deliver the codes as well as being in possession of the code sheets. The event that both happens is very unlikely.
Usability: *unchanged*
Pilot projects of the Norwegian system [SB12] have shown that confirmation codes work and possibly heighten the voters trust in the system. Voters participated voluntarily in the confirmation process and did not lose trust even when the system acted incorrectly. The codes are furthermore not mandatory for the voter.

7.2.11 Test Audits

Complexity: *minor*
The audit option (section 6.1.6) has to be implemented inside the voting client, whereby the client reveals randomness factors and other information needed for decryption. This is a fairly minor effort, since the client only has to store those values and wait for the user to decide to challenge the encryption or to send the ballot.
Efficiency: *mainly solved*
Test-Audits serve as a cast-as-intended-proof and will statistically succeed if the voter uses them a few times before casting her vote. The audits purpose can be defeated by wrong use.
Usability: *worsened*
The Helios system uses test audits. Many users of Helios are unclear on how to correctly use the audits and are sometimes confused by the option as shown by a usability study by Karayumak et al. [KOKV11].

Summary

The results from this chapter are compiled in figure 7.1 as an overview and for reference purposes.

Method	**Efficiency**	**Usability**	**Complexity**
Mandatory Bulletin Board	*perfect*	*unchanged*	*substantial*
Tallying Mix-Nets	*mainly solved*	*unchanged*	*minor*
Homomorphic Encryption	*appeasing*	*unchanged*	*minor*
Threshold Encryption	*mainly solved*	*unchanged*	*minor*
Off-line Voter Registration	*perfect*	*disruptive*	*improved*
Code Voting	*mainly solved*	*disruptive*	*substantial*
Re-Voting	*appeasing*	*unchanged*	*minor*
Cryptographic Puzzles	*appeasing*	*unchanged*	*minor*
Expensive Credentials	*mainly solved*	*enhanced*	*substantial*
Confirmation Codes	*mainly solved*	*unchanged*	*substantial*
Test Audits	*mainly solved*	*worsened*	*minor*

Figure 7.1: An overview of the results of the evaluation chapter 7.

Chapter 8

Application

This chapter visits existing voting systems in the light of the developed guidelines. The important issues are whether those systems take some of the proposed methods into account. If the guidelines were not followed, we research whether the systems are therefore vulnerable for the corresponding issues. For this we take a look at Civitas, Helios, the Estonian I-Voting system, Washington DVBM and the Norwegian I-Voting pilot.
The sections concentrate more on the unimplemented measures and its implications on the system, since the effect of the guidelines when implemented already have been discussed in the guidelines chapter 6.
Every section is accompanied by a score-sheet, where the guidelines have been sorted in one of the five groups: verifiability, anonymity, secure implementation, coercion and DDoS / chaff. Every guideline is color-coded with either green, meaning that the guideline is implemented by the system, yellow, signifying either the explicitly stated possibility to implement the guideline or a solution with a similar goal is already implemented and red, meaning that the guideline has not been implemented. The purpose is to help identify in which categories the system may be the most unsafe. In every section we try to determine if the impression the score-chart is giving is accurate or rather is a good estimation of the systems issues.
By pitting the score-chart against results from actual research on the system, we try to determine if the score chart and guidelines posses any meaningful predictive power.

8.1 Civitas

Civitas is one of the most cited I-Voting systems implemented to date and is therefore reviewed according to our guidelines. It implements five out of fifteen guidelines and has similar systems or the option to upgrade to six guidelines.
Furthermore, Civitas does not use homomorphic encryption for the tally, but uses re-encryption mix-nets instead, which serve the same purpose of preserving ballot anonymity before tallying.
The issue of building a secure network between the election authority servers is not addressed by Civitas. This may be because Civitas was not deployed for any real elections. Nevertheless at least a schematic of how to securely deploy Civitas in a real-world network would help mitigate risks during deployment. As long as Civitas does not address this, it keeps a potential security flaw as predicted by the chart.
A major pitfall is that Civitas does not use Code Voting or similar measures, because the voter device is implicitly trusted. As shown by the score-chart this trust is misplaced and can realistically not be assumed. Voter devices are highly susceptible to malware and are thus potential attack vectors.
Civitas is also vulnerable to chaff votes since cryptographic puzzles or similar measures are not implemented. Confirmation codes and test audits have not been implemented as well, but are not necessary since the voter can check that her ballot was cast correctly after it appears on the bulletin board.
Overall, Civitas is partly insecure against chaff votes, attacks on the voters end and has no concept of how to deploy a secure instance in a network. This does correspond to the issues made evident by the score-sheet in figure 8.1.

Verifiability	**Anonymity**	**Secure Implementation**	**Coercion**	**DDoS / Chaff**
🟩 Bulletin Board	🟩 Mix-Nets	🟥 Network Security	🟩 Re-Voting	🟥 Crypto-puzzles
🟨 Offline-Registration	🟨 Homomorphic Encryption	🟥 Code-Voting	🟥 Expensive Credentials	🟩 Heavily decentralized components
🟩 Public Voter-List	🟨 Threshold Encryption		🟨 Coercion Detection	
🟨 Confirmation Codes				
🟨 Test-Audits				

Figure 8.1: Civitas score-chart regarding the developed guidelines

8.2 Helios

Helios scores fairly badly with only fulfilling three out of the fifteen guidelines and having one additional mechanism that serves a similar purpose as another guideline.
The system is developed as a coercion-susceptible system that tries to be as simple as possible while still at least achieving verifiability in the case of a system-compromise. This is also shown by the score-chart, where Helios does not fulfill any of the guidelines in the coercion category.
Furthermore, the score-chart indicates that Helios is vulnerable to DoS, which is accurate, because Helios runs on a singular server, and does not use any protective measures against DoS attacks. Several more flaws are indicated by the score-chart in the category of secure implementation, which is the case since Helios does nothing to counter insecure voter devices and has no concept regarding the deployment of the system.
All of those issues can be identified well with a glance at the Helios score-sheet, although the sheet may also suggest that bad anonymity is provided, which is only the case when the server is malicious. The score-chart in figure 8.2 for Helios thus seems a good estimate of possible issues.

Verifiability	**Anonymity**	**Secure Implementation**	**Coercion**	**DDoS / Chaff**
🟩 Bulletin Board	🟩 Mix-Nets	🟥 Network Security	🟥 Re-Voting	🟥 Crypto-puzzles
🟥 Offline-Registration	🟨 Homomorphic Encryption	🟥 Code-Voting	🟥 Expensive Credentials	🟥 Heavily decentralized components
🟥 Public Voter-List	🟥 Threshold Encryption		🟥 Coercion Detection	
🟥 Confirmation Codes				
🟩 Test-Audits				

Figure 8.2: Helios score-chart regarding the developed guidelines

8.3 Estonian I-Voting

The Estonian I-Voting system is widely used in national elections for several years. It fulfills three guidelines to the fullest extent and has two measures similar to guidelines provided by us. A glance at the score-chart 8.3 suggests that the system has massive potential vulnerabilities in every category with exception of coercion and to a lesser degree verifiability.

A security analysis by Springall et al. [SFD+14] confirms that possible attacks on anonymity and verifiability can be conceived at least in a lab environment. In the analysis Springall et al. make it evident that the Estonian system can have a complete breach of anonymity trough a singular insider attacker, which is also shown by the score-chart where none of the guidelines in the anonymity category is fulfilled.
The Estonian system has no precautions whatsoever against Denial-of-Service attacks which is also highlighted by the score-chart. It is unknown how and if the system deals with possible chaff-votes, made possible through re-voting. Moreover, the systems seems to be operated in an insecure manner, which is mentioned in the security report by Springall et al., where e.g. singular administrators are updating the system ad-hoc without the presence of another administrator.
The one positive aspect is the ability to re-vote and the Estonian ID system being an exemplary case for expensive credentials, which are great for reducing coercion.
Again, the score-chart is a good starting point for locating possible threats in the system and highlights the main flaws which also surfaced in real-world studies.

Verifiability	**Anonymity**	**Secure Implementation**	**Coercion**	**DDoS / Chaff**
🟥 Bulletin Board	🟥 Mix-Nets	🟥 Network Security	🟩 Re-Voting	🟥 Crypto-puzzles
🟨 Offline-Registration	🟥 Homomorphic Encryption	🟥 Code-Voting	🟩 Expensive Credentials	🟥 Heavily decentralized components
🟥 Public Voter-List	🟥 Threshold Encryption		🟥 Coercion Detection	
🟩 Confirmation Codes				
🟨 Test-Audits				

Figure 8.3: Estonian I-Voting score-chart regarding the developed guidelines

8.4 Washington DVBM

The Washington Digital-Vote-By-Mail (DVBM) system was one of the first to conduct a public trial before launch. In their paper, Wolchok et al. [WWIH12] describe several severe flaws which made it possible for them to compromise the system and the outcome of the mock-election.
The system only fulfills one out of fifteen guidelines. One glance at the score-chart in figure 8.4 shows that the system is unfit for any type of integrity-preserving election. The system has no concept of a secure network during deployment. In the previously

cited paper Wolchock et al. use a security flaw to compromise the election web server and use it as a starting point to compromise the rest of the election network. They can furthermore read and manipulate votes at will, which is a huge integrity and anonymity breach. They did not perform coercion or DoS attacks, but those would have been easily possible as well. All of the above mentioned flaws are clearly visible in the score-chart. It does not make sense to further speak of the singular flaws of the system, since the only guideline it does fulfill is an optional off-line registration. This is another example of the predictive nature of the score-chart by lack of accompanying guidelines.

Verifiability	**Anonymity**	**Secure Implementation**	**Coercion**	**DDoS / Chaff**
Bulletin Board	Mix-Nets	Network Security	Re-Voting	Crypto-puzzles
Offline-Registration	Homomorphic Encryption	Code-Voting	Expensive Credentials	Heavily decentralized components
Public Voter-List	Threshold Encryption		Coercion Detection	
Confirmation Codes				
Test-Audits				

Figure 8.4: Washington DVBM score-chart regarding the developed guidelines

8.5 Norwegian I-Voting Pilot

The system fulfills two guidelines completely and three guidelines either partially or has an equivalent solution in place.
A look at the score chart in figure 8.5 suggests that the system probably has problems in the areas of anonymity, DDoS-vulnerability and potentially coercion. The system ran multiple ballot box servers but was otherwise relatively centralized. Furthermore it had no protection against vote-spam, since re-voting was allowed and could have been abused. Both facts suggest that the system indeed is prone to be target of successful DoS-attacks as predicted.
The only measure implemented against coercion is re-voting, which goes a long way in coercion-prevention but would work better in tandem with other measures like the use of expensive credentials and coercion detection. This again is correctly indicated by the score-chart.
The provided anonymity of the ballot seems fine at first sight, since the votes are not publicly published. But since the power of the authorities is fairly centralized in the system,

they themselves have the capability to void the anonymity of the ballot and gain knowledge about the voters choices. Any insider attacker can void the anonymity as clearly shown by the score-chart.
Lastly, the system does not follow a lot of the guidelines regarding verifiability but has nonetheless enough equivalent solutions[1] for the system to be considered verifiable, which is reflected by the score chart with its two yellow and one green entry in the verifiability section.

Verifiability	**Anonymity**	**Secure Implementation**	**Coercion**	**DDoS / Chaff**
🟨 Bulletin Board	🟥 Mix-Nets	🟨 Network Security	🟩 Re-Voting	🟥 Crypto-puzzles
🟨 Offline-Registration	🟥 Homomorphic Encryption	🟥 Code-Voting	🟥 Expensive Credentials	🟥 Heavily decentralized components
🟥 Public Voter-List	🟥 Threshold Encryption		🟥 Coercion Detection	
🟩 Confirmation Codes				
🟥 Test-Audits				

Figure 8.5: Norwegian I-Voting score-chart regarding the developed guidelines

8.6 Summary

This chapter applied the guidelines score-chart from figure 6.1 to five in the real world relevant systems. The results are consistent in that the score-chart is successfully predicting open security threats the systems possess. No to date known vulnerabilities of the system were omitted by the score chart.

[1]The system uses several cryptographic proofs which can be conducted by the election authorities chained together to produce a verifiable tally.

Chapter 9

Blueprint for an Attack

This chapter serves as an illustration of a hypothetical attack based on flaws highlighted by the guidelines and the resulting score-chart. As the target we chose the Estonian voting system because it is the currently most relevant network-voting system. A corresponding score-chart can be seen in figure 8.3.
The attack-concept is multi-pronged and tries to present multiple threats at once. Attacker-types and needed resources are described ad-hoc when needed. For better understanding we created the figure 9.1, which is a schematic overview of the Estonian system.

9.1 Denial of Service

Since the system is fairly centralized the threat of (distributed) denial of service is present and can be abused. Just a raw attack with enough attacker controlled machines on the *vote forwarding server* could be successful, since all votes have to pass through this server. This however is an attack that needs a lot of resources on the attacker side, since it is symmetrical.
For a better attack we abuse asymmetry in the system. The Estonian systems allows unlimited re-voting, but does not require the solving of cryptographic puzzles or similar measures. The attacker needs Estonian ID-cards and the corresponding PINs from a

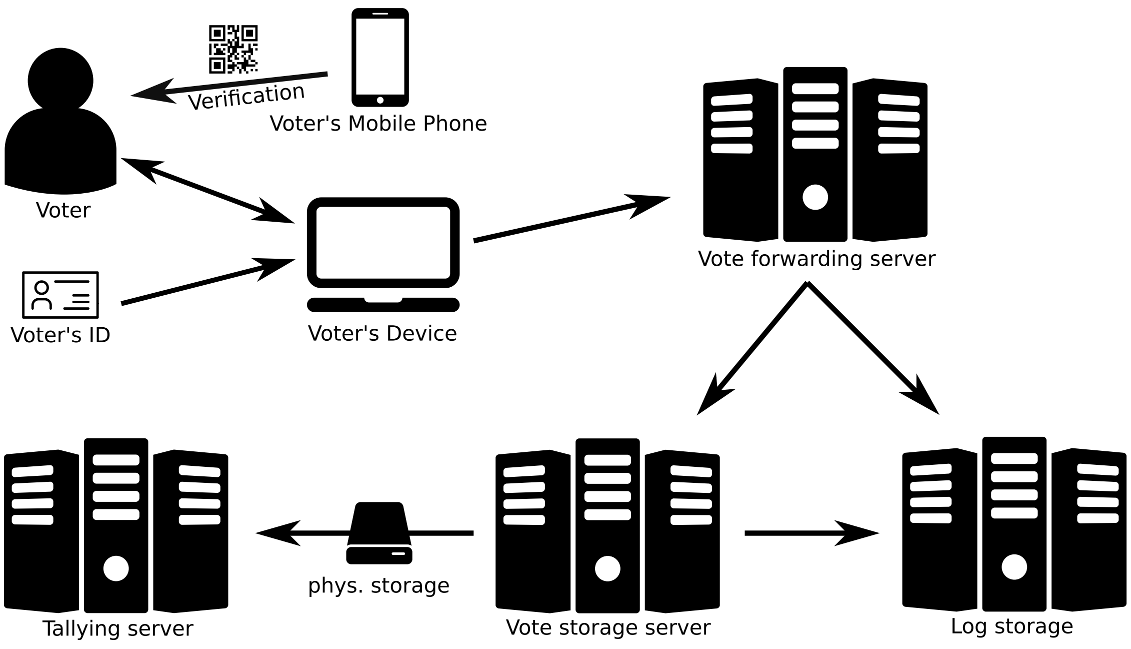

Figure 9.1: A schematic overview of the Estonian voting system.

number of eligible voters[1]; not an impossible feat for a criminal organization or intelligence agency. Now the cards can be used to generate valid ballots, that are sent as fast and as many as possible to the vote forwarding server. For every ballot generated by the attacker the vote forwarding server checks the ballot's credentials, creates and forwards log-messages to the log-server and signs the ballot before forwarding it to the vote storage server.

The attack now requires considerably less resources than a brute-force distributed DoS attack to overload the vote forwarding server. Since the vote forwarding server is the chocke-point which all votes must pass, legitimate voters are forced to wait for the attack to be over to cast their vote electronically.

The attack can be continuous or executed periodically, during times which are the most probable for the main chunk of eligible voters to cast their vote. While this attack, depending on length, would prevent electronic voting, the voters could still opt to cast their vote at a physical ballot box. It does therefore not prevent the election, but rather turns

[1]The hardware-specifications of the Estonian system are not known, but assuming that they use common hardware for their servers, a *few hundred* simultaneous ballots sent repeatedly should suffice. Keep in mind, that a single eID and PIN can produce several votes, since unlimited re-voting is allowed.

the network voting structure into a non-functioning money and power sink.

9.2 Hijacking Votes

Another attack vector is hijacking the voters device coupled with manipulation of the vote. The Estonian voting system uses confirmation codes, which are displayed as a QR code, which in turn can be scanned with a special mobile phone application for 30 minutes after the vote is cast. This does not hinder an attacker in manipulating a vote on the voter device. Especially since code-voting is not implemented, an attacker can use this to her advantage. Prerequisite for any attack is that the voters device needs to be infected with any attacker controlled malware, which can be achieved by specially tailored malware or by for example renting existing botnets located in Estonia.
Springall et al. published a paper [SFD$^+$14], where they propose two attacks that defeat the QR code confirmation and are based on a compromised voter device. The first attack is termed **Ghost Click** attack, where the malware sniffs the PIN-code of the Estonian ID-card and then re-votes either after 30 minutes are over or if the voter closes the window with the verification code. As long as the ID-card is then still present in the machine, the malware can revote without the possibility for detection.
The other method from the same paper was termed **Bad Verify Attack** and requires the mobile phone to be compromised in tandem with the voters device. This is not unlikely since the mobile phone is usually often connected to a person's computer and / or synchronized through different online services. The malware on the mobile phone manipulates the verification app to display a predetermined vote instead of the voter's choice, which is encoded in the QR-code. That allows the malware on the device to change the vote during the casting-stage without detection.
A less severe, but still intolerable, attack would be the sole sniffing of voters choice. The voting app requires the entry of the PIN-code for the ID-card, which displays at least the voters full name[2]. The malware can now sniff which candidate is chosen by which voter and thus void the anonymity of the ballot by sending the aquired data to the attacker. Automated coercion can be easily carried out if the lastly described attack is possible.

[2]The client-side application is not fully open source and only screenshots are available for study.

9.3 Infiltrating the Vote Infrastructure

A detrimental approach to the above would be compromising the server side. This however requires some kind of insider-attacker. More precisely, this means either one of the administrators is corrupt, or hard- or software is in advance; which is not impossible, but could be hard to achieve. Malware hidden in e.g. the firmware of hard drives has been used for targeted attacks by, allegedly, intelligence agencies as discovered in February 2015 [Men15].
A corrupt administrator can also be enough to compromise the servers; as Springall et al. report [SFD$^+$14] administrators were sometimes alone, while performing updates and other maintenance tasks, leaving the security of the system to their discretion.
The easiest attack would be attained by deploying code inside the log and storage server. Malicious code would then selectively remove votes, effectively manipulating the election, since the log server seems to be the only form of audit trail. This attack however, could be discovered if the total number of all cast votes can be counted and compared to the tally total. Note that there is no information if there are other logs besides the log storage which would allow for detection of this attack.
Another possible attack would be to deploy malware on to the tallying server. During the tally phase, the server does not seem to produce any form of audit trail, which can be abused. The malware infiltrates the system on an OS-level, thus being able to change the result of any decryption performed on the server, effectively changing all votes without a trace. Conclusively, the server then counts the counterfeit votes normally and outputs a wrong tally.
This is a serious problem and shows, that without universal audit-trails or public verifiability the election integrity can depend on a singular election worker, which should never be the case.

Chapter 10

Conclusion

This thesis **examines attacks** against network voting systems and proposes design and implementation guidelines to mitigate said attacks as much as possible.
For an advanced insight on possible attacks, we investigated obviously flawed voting systems and took a look at historical attacks against electronic voting systems. Following the results, goals were defined, that should be fulfilled by any secure network voting system. Subsequently, the goals were used as a template to construct **guidelines**, which help designers and developers of network voting systems during the inception and implementation phase. Conclusively, the main contribution was identifying the appropriate methods needed for well designed guidelines. Guidelines were compressed into a **checklist** which can be used during development as a quick reference. They were furthermore evaluated for different criteria like usability change or added complexity during implementation.
In addition to the evaluation, the guidelines-checklist was **applied to existing systems**. During the application chapter, one could see that most issues predicted by the score-chart did indeed surface historically within the systems examined. There were no previously found flaws of the examined systems that the score-chart did not hint at. Thus, the checklist works in a predictive manner as well, strongly implying that the checklist can be used effectively during design and implementation to mitigate the most common flaws of a network voting system by trying to fulfill as many items on the checklist as possible.
Lastly we developed a **blueprint for an attack** on the Estonian voting system, which is

based loosely on the potential faults found by the checklist. Three possible attack vectors were described, of whom two severely endanger the election integrity.

Future Work

Since the subject of the thesis is fairly broad, there is a lot of possible future work to be done. The evaluation of the guidelines can be re-executed by implementing the guidelines in open-source voting systems and then comparing efficiency, usability change and complexity of implementation of the modified and original system.
Another field of improvement could be research that examines if certain programming languages / frameworks are inherently better suited in implementing network voting systems. It would also be interesting to study if types of programming languages, like functional or logical, have any edge over traditional iterative languages.
A bigger next step would be the development of framework-software, inspired by the developed guidelines, which aids in the development of network voting systems.
The maybe most obvious and ambitious extension would be to implement a working network voting system, while following the guidelines in hopes of creating a new standardized, secure, open source network voting system.
More future work was suggested in section 5.6 regarding usability in Internet voting system and section 5.2.1 touching on everlasting privacy. Furthermore there is the possibility to study the effects of different kinds of mix-nets on electronic elections as suggested in section 6.1.2.
More future work was already suggested in the sections 6.3.2, 6.3.4 and 6.5.1.

Bibliography

[ACKR13] ARAPINIS, Myrto; CORTIER, Véronique; KREMER, Steve; RYAN, Mark: Practical everlasting privacy. In: *Principles of Security and Trust*. Springer, 2013, pp. 21–40.

[Adi08] ADIDA, Ben: Helios: Web-based Open-Audit Voting. In: *USENIX Security Symposium* Bd. 17, 2008, pp. 335–348.

[ASM05] ARKIN, Brad; STENDER, Scott; MCGRAW, Gary: Software penetration testing. In: *IEEE Security & Privacy* 3 (2005), Nr. 1, pp. 84–87.

[B$^+$02] BACK, Adam u. a.: *Hashcash-a denial of service counter-measure*. 2002.

[BF97] BONEH, Dan; FRANKLIN, Matthew: Efficient generation of shared RSA keys. In: *Advances in Cryptology—CRYPTO'97*. Springer, 1997, pp. 425–439.

[CC$^+$08] CAMENISCH, Jan; CHAABOUNI, Rafik u. a.: Efficient protocols for set membership and range proofs. In: *Advances in Cryptology-ASIACRYPT 2008*. Springer, 2008, pp. 234–252.

[CCJC14] CHEN, Chin-Ling; CHEN, Yu-Yi; JAN, Jinn-Ke; CHEN, Chih-Cheng: A Secure Anonymous E-Voting System based on Discrete Logarithm Problem. In: *Appl. Math* 8 (2014), Nr. 5, pp. 2571–2578.

[CCM07] CLARKSON, Michael R.; CHONG, Stephen; MYERS, Andrew C.: Civitas: A secure voting system / Cornell University. 2007. Technical Report.

[CDN01] CRAMER, Ronald; DAMGÅRD, Ivan; NIELSEN, Jesper B.: *Multiparty computation from threshold homomorphic encryption.* Springer, 2001

[Com15] COMMISION, United States Election A.: *TGDC Recommended Guidelines.* `http://www.eac.gov/vvsg/`, 2015.

[CSB03] COFFEY, Tom; SAIDHA, Puneet; BURROWS, Peter: Analysing the security of a non-repudiation communication protocol with mandatory proof of receipt. In: *Proceedings of the 1st international symposium on Information and communication technologies* Trinity College Dublin, 2003, pp. 351–356.

[DAM$^+$15] DURUMERIC, Zakir; ADRIAN, David; MIRIAN, Ariana; BAILEY, Michael; HALDERMAN, J. A.: *FREAK Vulerability.* `https://freakattack.com/`, 2015.

[DiP15] DIPIERRO, Massimo: *E-Vote.* `https://github.com/mdipierro/evote/commit/8b48a6d449bbc1f10f34e94aafee660bf9197435`, 2015.

[DVDGA12] DEMIREL, Denise; VAN DE GRAAF, Jeroen; ARAÚJO, Roberto: Improving helios with everlasting privacy towards the public. In: *EVT/WOTE* (2012).

[Eur11] EUROPE, Council of: *Electoral assistance and Census.* `http://www.coe.int/t/dgap/democracy/activities/ggis/E-voting/E-voting%202010/Biennial_Nov_meeting/ID10322%20GBR%206948%20Evoting%20handbook%20A5%20HD.pdf`, 2011.

[Eve07] EVERETT, Sarah P.: *The usability of electronic voting machines and how votes can be changed without detection*, RICE UNIVERSITY, Diss., 2007

[Fer04] FERNANDEZ, Eduardo B.: A Methodology for Secure Software Design. In: *Software Engineering Research and Practice*, 2004, pp. 130–136.

[Gjø10] GJØSTEEN, Kristian: Analysis of an internet voting protocol. In: *IACR Cryptology ePrint Archive* 2010 (2010), pp. 380.

[GRBR13] GREWAL, Gurchetan S.; RYAN, Mark D.; BURSUC, Sergiu; RYAN, Peter Y.: Caveat coercitor: Coercion-evidence in electronic voting. In: *Security and Privacy (SP), 2013 IEEE Symposium on* IEEE, 2013, pp. 367–381.

[hea15] HEARTBLEED.COM: *Heartbleed SSL Vulerability*. `http://heartbleed.com/`, 2015.

[HT15] HALDERMAN, J A.; TEAGUE, Vanessa: The New South Wales iVote System: Security Failures and Verification Flaws in a Live Online Election. In: *arXiv preprint arXiv:1504.05646* (2015).

[IT15] IT, Fox: *Quantum Insert detection for Snort*. `https://github.com/fox-it/quantuminsert/tree/master/detection/snort`, 2015.

[Kha14] KHARE, Aparna: Scrutiny of the Indian attempt to Internet Voting. In: *International Journal of Engineering* 3 (2014), Nr. 4.

[KOKV11] KARAYUMAK, Fatih; OLEMBO, Maina M.; KAUER, Michaela; VOLKAMER, Melanie: Usability analysis of helios-an open source verifiable remote electronic voting system. In: *Proceedings of the 2011 USENIX Electronic Voting Technology Workshop/Workshop on Trustworthy Elections. USENIX*, 2011.

[KZ07] KUTYŁOWSKI, Mirosław; ZAGÓRSKI, Filip: Verifiable internet voting solving secure platform problem. In: *Advances in Information and Computer Security*. Springer, 2007, pp. 199–213.

[KZ10] KUTYŁOWSKI, Mirosław; ZAGÓRSKI, Filip: Scratch, click & vote: E2E voting over the Internet. In: *Towards trustworthy elections*. Springer, 2010, pp. 343–356.

[Lip04] LIPNER, Steve: The trustworthy computing security development lifecycle. In: *Computer Security Applications Conference, 2004. 20th Annual* IEEE, 2004, pp. 2–13.

[LRWW04] LEVINE, Brian N.; REITER, Michael K.; WANG, Chenxi; WRIGHT, Matthew: Timing attacks in low-latency mix systems. In: *Financial cryptography* Springer, 2004, pp. 251–265.

[Men15] MENN, Joseph: *Russian researchers expose breakthrough in U.S. spying program*. `http://www.reuters.com/article/2015/02/17/us-usa-cyberspying-idUSKBN0LK1QV20150217`, 2015.

[Mil07] MILLER, Charlie: The legitimate vulnerability market: Inside the secretive world of 0-day exploit sales. In: *In Sixth Workshop on the Economics of Information Security* Citeseer, 2007.

[MLY05] MYAGMAR, Suvda; LEE, Adam J.; YURCIK, William: Threat modeling as a basis for security requirements. In: *Symposium on requirements engineering for information security (SREIS)* Bd. 2005, 2005, pp. 1–8.

[MN06] MORAN, Tal; NAOR, Moni: Receipt-free universally-verifiable voting with everlasting privacy. In: *Advances in Cryptology-CRYPTO 2006*. Springer, 2006, pp. 373–392.

[NRRK] NIKAM, Rutuja; RANKHAMBE, Monika; RAIKWAR, Diksha; KASHYAP, Atharv: Secured E-Voting Using NFC Technology.

[Oeh05] OEHLERT, Peter: Violating assumptions with fuzzing. In: *Security & Privacy, IEEE* 3 (2005), Nr. 2, pp. 58–62.

[Oka98] OKAMOTO, Tatsuaki: Receipt-free electronic voting schemes for large scale elections. In: *Security Protocols* Springer, 1998, pp. 25–35.

[OMA$^+$99] OHKUBO, Miyako; MIURA, Fumiaki; ABE, Masayuki; FUJIOKA, Atsushi; OKAMOTO, Tatsuaki: An improvement on a practical secret voting scheme. In: *Information Security*. Springer, 1999, pp. 225–234.

[Opp02] OPPLIGER, Rolf: How to address the secure platform problem for remote internet voting. In: *Sis* 2 (2002), pp. 153–173.

[OSH08] OPPLIGER, Rolf; SCHWENK, Jörg; HELBACH, Jörg: Protecting Code Voting Against Vote Selling. In: *Sicherheit* Bd. 128 Citeseer, 2008, pp. 193–204.

[PP] PULLS, Tobias; PEETERS, Roel: Balloon: A Forward-Secure Append-Only Persistent Authenticated Data Structure.

[RA12] RIBARSKI, Pance; ANTOVSKI, Ljupcho: Mixnets: Implementation and performance evaluation of decryption and re-encryption types. In: *CIT. Journal of Computing and Information Technology* 20 (2012), Nr. 3, pp. 225–231.

[SB12] STENERUD, Ida Sofie G.; BULL, Christian: When reality comes knocking Norwegian experiences with verifiable electronic voting. In: *Electronic Voting* 205 (2012), pp. 21–33.

[Sch99] SCHNEIER, Bruce: Attack trees. In: *Dr. Dobb's journal* 24 (1999), Nr. 12, pp. 21–29.

[Sch13] SCHNEIER, Bruce: *How the NSA Attacks Tor/Firefox Users With QUANTUM and FOXACID*. `https://www.schneier.com/blog/archives/2013/10/how_the_nsa_att.html`, 2013.

[SFD$^+$14] SPRINGALL, Drew; FINKENAUER, Travis; DURUMERIC, Zakir; KITCAT, Jason; HURSTI, Harri; MACALPINE, Margaret; HALDERMAN, J A.: Security Analysis of the Estonian Internet Voting System. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* ACM, 2014, pp. 703–715.

[SHR+11] SIMPSON, Stacy; HOWARD, M; RANDOLPH, K; GOLDSCHMIDT, C; COLES, M; BELK, M; SAARIO, M; SONDHI, R; TARANDACH, I; YONCHEV, Y u. a.: Fundamental practices for secure Software Development Software Assurance Forum for Excellence in Code, 2011.

[SMHM15] SCHNEIDER, Alexander; METER, Christian; HAGEMEISTER, Philipp; MAUVE, Martin: *Tor is not enough: Coercion in Remote Electronic Voting Systems*. 2015.

[Tor05] TORR, Peter: Demystifying the threat modeling process. In: *Security & Privacy, IEEE* 3 (2005), Nr. 5, pp. 66–70.

[Vir15] VIRGINIA INFORMATION TECHNOLOGIES AGENCY: Security Assessment of WINVote Voting Equipment for Department of Elections / Virginia Information Technologies Agency. 2015. Technical Report.

[WWIH12] WOLCHOK, Scott; WUSTROW, Eric; ISABEL, Dawn; HALDERMAN, J A.: Attacking the Washington, DC Internet voting system. In: *Financial Cryptography and Data Security*. Springer, 2012, pp. 114–128.

[ZJT13] ZARGAR, Saman T.; JOSHI, James; TIPPER, David: A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. In: *Communications Surveys & Tutorials, IEEE* 15 (2013), Nr. 4, pp. 2046–2069.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 06. August 2015 Alexander Schneider

Hier die Hülle

mit der CD/DVD einkleben

Diese CD enthält:

- eine *pdf*-Version der vorliegenden Bachelorarbeit
- die LaTeX- und Grafik-Quelldateien der vorliegenden Bachelorarbeit samt aller verwendeten Skripte
- die Websites der verwendeten Internetquellen