Karlsruhe Institute of Technology

Matters of Coercion-Resistance in Cryptographic Voting Schemes

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

von der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Carmen Kempka (geb. Stüber)

aus Karlsruhe

Tag der mündlichen Prüfung:	3. Juni 2014
Erster Gutachter:	Prof. Dr. Jörn Müller-Quade
Zweiter Gutachter:	Prof. Dr. Jeroen van de Graaf

Acknowledgement

On the road which has led to this thesis, I have been inspired, motivated and supported by many people, to whom I am grateful.

First of all, I thank my thesis supervisor Prof. Dr. Jörn Müller-Quade, who has inspired me with his contagious enthusiasm for cryptography. I thank him for all his support, advice, patience and tolerance, especially for making it possible that I could bring my son to the office, and for being able to do research in maximal freedom.

Many thanks go to Prof. Dr. Jeroen van de Graaf, who corefereed this thesis, for inviting me to join research on the taxonomy project, for invaluable input, and for inspiring discussions in an uncomplicated working atmosphere. Besides, without him, this work would be in much worse English.

I express my gratitude to Jun. -Prof. Dr. Dennis Hofheinz, for granting me oracle access to his marvelous knowledge about all kinds of fancy encryption and signature primitives, for sanity-checking my crazy ideas and for all kinds of support.

This work would not be what it is without my coworkers Dr. Christian Henrich, Bernhard Löwe and Dirk Achenbach. Thanks for being with me at all those paper deadline night shifts and especially in the thesis last minute panic period, for proof reading, invaluable discussions and feedback.

I thank Víctor Mateu for taking away the mystery of pairing based cryptography and elliptic curves, and for helpful and inspiring discussions, especially about the ID-comparison with the Elgamal scheme. Thanks to Sven Krohlas for briefing me about delegated voting.

Many thanks to everyone who has made the application of Bingo Voting in the election of the student's parliament possible, especially Björn Tackmann and Michael Bär. Many thanks to Reiner SCT for providing the chip-card readers and the customized random number generators for this election.

I offer my gratitude to Prof. Dr. Bernhard Beckert and Prof. Dr. Ralf Reussner for inviting me to discuss about my work and for important feedback.

This work could be started thanks to Prof. Dr. Jaques Calmet, who welcomed me to IAKS and gave me freedom in my research. I would like to thank Prof. Dr. Thomas Beth (1949-2005), for his inspiring lectures, and for believing in me.

Thanks to all my current and former colleagues at IAKS/IKS/EISS/ITI for the enjoyable working atmosphere, especially Hildegart Kühne for her friendship, Thilo Mie for discussions about basically everything, and Florian Böhl, who shared with me the pain and excitement of the last part of the road towards a doctorate. I could feel this enjoyable working atmosphere already as a student thanks to Stefan Röhrich, Dr. Jens-Matthias Bohli, Prof. Dr. Dominique Unruh, Prof. Dr. Rainer Steinwandt, Prof. Dr. Markus Grassl and many more. Thanks for making me feel welcome at IAKS right from the beginning. Special thanks to Jens and Stefan for

the joint work on Bingo Voting improvements, and for motivating me to work on e-voting.

For her tireless administrative support I would like to thank Carmen Manietta. No less I thank Holger Hellmuth, especially for not doing that server reboot on the day of my thesis deadline. I also thank Audrey Bohlinger for patiently answering all questions concerning the doctorate procedure.

I would like to express my gratitude to Dr. Masayuki Abe for welcoming me to NTT as a postdoc, which gave me a lot of motivation and endurance during the last phase of this work.

Thanks to my dear friends Makitaro, Reiko and Momo Arima for counter-balancing my worktime with their wonderful world of music.

Not least at all I thank my family. I am in great debt to my parents Walter and Anne Stüber for all their support throughout my life, and especially for being there for my son Jonas. I thank my sister Tanja Mohr and her family Marlon, Luisa and Michael Mohr for all their support. I thank my dear husband Matthias Kempka for his love and all his support, and for shielding me from matters of real-life to give me the freedom and a clear head to finish this work. And I thank my son Jonas Nehemias Kempka for his love and patience, and especially for offering me to skip kindergarten to help me with my work instead.

Abstract

Electronic voting has gained more and more interest during the last decades. It promises less error-prone and faster tallying, while saving paper as well as working time. On top of this, voters can be supported by a vote casting interface, reducing the amount of ballots which would otherwise be marked invalid unintentionally. Moreover, the introduction of electronic voting introduces the possibility of conducting an election over the internet, which promises to increase voter turnout.

However, voting machines do not offer the same obvious transparency as traditional paper elections, since they are complex systems which are hard to verify. It became soon apparent that paper ballots cannot be substituted by computers without additional measures.

Cryptographic voting schemes offer the possibility of publicly verifiable elections. This verifiability depends on cryptographic techniques and holds independently of the implementation of the voting computer. This is even more important if elections are held over the internet, where ballots are recorded and counted on one or several distant servers, and the voter does not interact with most of the system components in person. However, the voter's choice is to be secret and free – it should be cast uninfluenced by any adversarial impact. This in fact leads to an even stronger requirement than secrecy, namely *coercion-resistance*. Strongly related to this requirement is *vote-buying*. Coercion and vote-buying can be considered in an equivalent manner, since in both coercion and vote-buying situations the aim of the adversary is to find out if the voter followed certain instructions.

Since cryptographic functions which prove correctness of the tally naturally get as input the cast ballots, care has to be taken that no information is revealed about the voter's choices in any verification data. Verifiable correctness while preserving coercion-resistance becomes even harder to achieve in non-standard elections, like elections which allow write-in candidates or vote delegation. Elections held over the internet are a special challenge, since the privacy of a voting booth is not guaranteed.

In this work, we take a look at cryptographic voting schemes, with focus on their coercion-resistance, from a wide viewpoint. First of all, verifiability and coercion-resistance are not the only design criteria for voting schemes. Measures to achieve those two requirements affect other criteria. The provability of a manipulation makes the voting scheme more *robust* than a mere detection. The early leakage of intermediate results makes an election *unfair*. A too high amount of published data for verification makes the election *non-scalable*. At the same time, there is a huge variety of voting schemes, based on different underlying models, using different cryptographic techniques. A consensus on a formal description of voting schemes is still missing. Therefore, it is hard to consistently analyze or compare this variety of voting schemes. This raises the question of a catalog of suitable design criteria.

The first contribution of this work is a taxonomy which identifies such design

and analysis criteria, together with an analysis roadmap, with which the existing variety of voting schemes can easily be analyzed and compared. The taxonomy is written in natural language and can be applied to any voting scheme, regardless of its underlying model. The taxonomy makes strengths and potential weaknesses of an election scheme apparent rather quickly.

Since the main focus of this work is coercion-resistance, we give an overview over existing definitions. After this, we describe real-world experiences with the voting scheme Bingo Voting, discuss its coercion-resistance and retrospectively analyze it with our taxonomy. We will see that Bingo Voting offers a great amount of flexibility since it provides vote-splitting and cumulative voting without weakening coercion-resistance. Further improvements of Bingo Voting are discussed, and to add to its flexibility, we provide Bingo Voting with support for write-in candidates, which means that instead of choosing a candidate from an existing candidate list, the voter can write in an arbitrary name.

This brings up the first of three special cases considered in this work which offer particular challenges when trying to achieve coercion-resistance. Write-in candidates are problematic since the voter can always be coerced to "vote for" a certain name or even a random string which would otherwise most likely get no votes. In the final tally, the adversary sees if this candidate has gotten one or zero votes. Hence, it seems impossible to achieve coercion-resistance in elections with write-in candidates. We solve this problem by providing techniques to publish the tally in a controlled fuzzy way, such that the tally leak less information while being sufficiently verifiable.

The second challenging case is the possibility to re-cast a vote: in internet voting schemes, it cannot be assumed that all voting processes take place in private. We have the problem of so-called *shoulder voting*: The voter might be observed during her voting process. This opens doors for coercion. An obvious and often implemented solution to this is *revoting*: The voter can cast a ballot as many times as she wants, and therefore overwrite a ballot cast under adversarial observation. However, to make this solution effective, the adversary must not see if the voter has revoted. Even more, the voter must not be able to prove that she has not revoted, since this would open doors for vote-buying and coercion. At the same time, the revoting process must be verifiable: The voter should be able to verify that her last ballot has been counted, and everyone should be able to verify that only everyone's last ballot has been counted. Achieving both properties at the same time has been an open problem, for which we introduce a proof-of-concept solution in this work.

The third challenging election type is *delegated voting*, where the voter can choose between voting by herself or delegating her choice to someone else. The new challenge to coercion-resistance is that the adversary must not be able to coerce the voter to delegate her choice to him. This is also related to revoting since in elections with delegated voting, it is often required that the voter can change her decision between voting and delegating at any time. We introduce a delegated voting scheme, which combines our revoting solution with a new paradigm called *vote fetching*.

Zusammenfassung

Elektronische Wahlen haben in den letzten Jahrzehnten immer mehr Interesse gewonnen. Sie versprechen weniger fehlerbehaftete, schnellere und dabei personalsparende Auszählungen, und können Wähler beim Ausfüllen des Wahlzettels unterstützen, was die Anzahl versehentlich ungültig abgegebener Stimmen sinken lässt. Die Stimmabgabe über das Internet wird möglich, was eine höhere Wahlbeteiligung verspricht. Allerdings bieten Wahlmaschinen nicht die Transparenz herkömmlicher Papierwahlen. Es handelt sich bei Wahlmaschinen um komplexe Systeme, deren korrekte Umsetzung nur schwer zu überprüfen ist. Offensichtlich kann daher die Papierwahl nicht ohne weitere Maßnahmen durch elektronische Wahlen ersetzt werden.

Kryptographische Wahlverfahren ermöglichen öffentlich verifizierbare Wahlen, wobei die Verifizierbarkeit im Idealfall unabhängig von der Implementierung elektronischer Komponenten gegeben ist. Jedoch darf die Verifizierbarkeit das Wahlgeheimnis nicht gefährden. Die Stimmabgabe soll in freier Wahl erfolgen, ohne äußeren Einfluss. Dies verlangt eine noch stärkere Anforderung: die *Nicht-Erpressbarkeit*. Diese ist stark verwandt mit Stimmkauf: in beiden Fällen ist das Ziel des Angreifers, zu erkennen, ob sich der Wähler an bestimmte Anweisungen gehalten hat.

Kryptographische Funktionen, die die Korrektheit einer Wahl, insbesondere der Auszählung, nachweisen sollen, bekommen natürlicherweise die Wählerstimmen in irgendeiner Form als Eingabe. Diese müssen zu lückenlosen Korrektheitsbeweisen verarbeitet werden, die allerdings keine Information über einzelne Wählerstimmen preisgeben. Verifizierbarkeit und Nicht-Erpressbarkeit scheinen deutlich in Konkurrenz zu stehen, umso mehr in Nicht-Standardwahlen, wie etwa Wahlen mit Write-In-Kandidaten oder Wahlen mit Stimmweitergabe.

In dieser Arbeit werden kryptographische Wahlen unter dem Schwerpunkt der Nicht-Erpressbarkeit, jedoch aus einem weiten Blickwinkel behandelt. Maßnahmen zum Erhalt der Verifizierbarkeit und der Nicht-Erpressbarkeit beeinflussen wiederum andere Kriterien, wie die Robustheit, die Fairness oder die Skalierbarkeit einer Wahl. Dem gegenüber steht eine Vielzahl existierender Wahlverfahren, aufbauend auf unterschiedlichen formalen Modellen und unterschiedlichen kryptographischen Primitiven. Diese Vielzahl an Wahlverfahren ist nur sehr schwer zu vergleichen oder einheitlich zu bewerten.

Diese Arbeit stellt eine Taxonomie vor, mit der Wahlverfahren einheitlich und auf einfache Weise untersucht werden können. Die Taxonomie enthält eine Liste von Kriterien, die kryptographische Wahlverfahren erfüllen sollten, und wird vorgestellt zusammen mit einem Analysefahrplan, in Form eines Fragekatalogs mit 1-6 Fragen pro Kriterium. Mit Hilfe dieser Fragen lässt sich für ein beliebiges Verfahren sehr schnell ein Überblick über Stärken und potenziellen Schwächen des Verfahrens gewinnen.

Der weitere Verlauf der Arbeit konzentriert sich auf die Nicht-Erpressbarkeit. Es

werden verschiedene Definitionen der Nicht-Erpressbarkeit verglichen im Bezug auf zwei Soncerfälle, namentlich Write-In-Kandidaten und Internetwahlen, in denen der Wähler die Möglichkeit hat, eine unter Beobachtung abgegebene Stimmabgabe durch eine weitere Stimmabgabe zu überschreiben.

Dieser allgemeinen Betrachtung verschiedener Kriterien von Wahlverfahren und Sicherheitsdefinitionen folgt ein Kapitel über Präsenzwahlen, in welchem Praxiserfahrungen mit dem Wahlverfahren Bingo Voting vorgestellt werden. Bingo Voting hat sich als sehr flexibel erwiesen, es erlaubt Kumulieren und Panaschieren ohne ein Verlust an Sicherheit, was nicht selbverständlich ist. Diese Arbeit stellt weitere Verbesserungen des Verfahrens vor, und eine Möglichkeit, Write-In-Kandidaten zu unterstützen.

Write-In-Kandidaten sind einer von drei in dieser Arbeit betrachteten Wahlformen, die eine besondere Herausforderung für kryptographische Wahlverfahren darstellen. Sind Write-In-Kandidaten erlaubt, so kann der Wähler immer gezwungen werden, einen Kandidaten zu wählen, der sonst vermutlich keine Stimmen bekommen würde, zum Beispiel eine Zufallszeichenkette. Der Angreifer würde im Wahlergebnis sehen, ob dieser unwahrscheinliche Kandidat eine Stimme bekommen hat oder nicht. Aus diesem Grund werden Wahlen mit Write-In-Kandidaten in Definitionen der Nicht-Erpressbarkeit oft ausgeschlossen oder nicht betrachtet. In dieser Arbeit werden Techniken zur kontrolliert verwaschenen Darstellung des Wahlergebnisses vorgestellt, die erpressbarkeitsfreie Wahlen mit Write-In-Kandidaten unter Erhalt der Verifizierbarkeit ermöglichen.

In Internetwahlen ist die Privatsphäre des Wählers nicht gewährleistet, daher kann nicht ausgeschlossen werden, dass er bei der Stimmabgabe beobachtet wird. Eine Standardtechnik, dem zu begegnen, ist es, den Wähler seine Stimmabgabe beliebig oft durch eine weitere überschreiben zu lassen, was unseren zweiten Sonderfall darstellt: Dies ist nur sinnvoll, wenn der Angreifer nicht bemerkt, ob der Wähler von dieser Möglichkeit Gebrauch gemacht hat. Gleichzeitig muss die Wahl verifizierbar bleiben. Den Teilnehmern der Wahl muss bewiesen werden, dass von jedem Wähler nur die neueste Stimme gezählt wird. Bisherige Lösungen erreichen entweder die Verifizierbarkeit des Aussortierprozesses oder das Nicht-Erkennen einer wiederholten Stimmabgabe. Diese Arbeit stellt ein Verfahren vor, welches beide Eigenschaften zugleich erreicht.

Der dritte Sonderfall sind Wahlen mit Stimmweitergabe, auch bekannt unter den Stichpunkten *Liquid Democracy* (flüssige Demokratie) oder *Delegated Voting* (delegiertes Wählen). Hier hat der Wähler die Möglichkeit, sein Stimmgewicht an einen sogenannten Proxy zu delegieren, der an seiner Statt eine Stimme abgibt. Diese relativ junge Art der Wahl stellt neue Anforderungen an die Sicherheit eines Wahlverfahrens. Es reicht nicht mehr aus, die korrekte Auszählung nachzuweisen. Der Wähler muss auch nachprüfen können, dass seine Stimme an den richtigen Proxy delegiert wurde. Um die Nicht-Erpressbarkeit zu gewährleisten, darf der Proxy oder ein Dritter allerdings nicht erfahren, ob oder an wen der Wähler seine Stimme delegiert hat. Eine weitere Anforderung dieser Wahlart ist oft, dass der Wähler sich jederzeit umentscheiden kann zwischen einer Delegation und eigener Stimmabgabe. Diese Arbeit stellt eine Möglichkeiten zur Umsetzung vor, die die oben erwähnte Lösung zur Wahl mit Stimmabgabewiederholung mit einer neuen Idee zum Umsetzen von Delegationen verbindet, dem *Vote-Fetching*. Die Idee des Vote-Fetching ist es, statt dem Delegieren der Stimme Stimmen von einem Proxy zu holen und rerandomisiert abzugeben.

Contents

1. Introduction

An election, in its simplest form, is the process of making a decision together, by jointly choosing one element out of a set of possible choices. The easiest way to do this is that every *voter* who is allowed to participate in the election tells her preferred choice to all others, and in the end, the voters jointly calculate the result. However, with this approach, the voter's choice in not confidential, voters can be manipulated by bribe, hectoring and coercion, rather than making an uninfluenced, free choice. For this reason, secrecy has become one of the basic principles of elections. At the same time, the participants of an election need to be able to convince themselves that the outcome of the election is *correct*, without revealing a single voter's choice.

There is an established method to achieve this: the traditional paper election, which has been used for decades in the German governmental elections, and in similar form with similar principles in various other democratic countries [fSoiE90]. The traditional paper election is generally accepted because of its easily assessable components and its transparency. It offers fundamental properties, making it a prime example of what an election scheme should achieve:

- The voter's choice is kept *secret:* the voter marks her ballot in a voting booth, folds it and enters it into a ballot box. No one can see how she has filled out her ballot. By the time her ballot is taken out for counting, the connection to the voter is no longer deducible.

- The election scheme is *receipt-free:* the voter cannot prove to another person how she voted, which prevents *vote buying* and *coercion*.

- The whole election process, including the tally, is *verifiable:* the election is observable from beginning to end. Observers can check that only eligible voters are allowed access to the voting process, that each voter enters only one ballot into the ballot box and that the ballots in the ballot box are counted correctly.

The traditional paper election seems hard to excel concerning security, understandability and acceptance, for good reason. However, it requires a high capacity in paper and, due to the counting process by hand, a lot of working time of poll

workers. Both the counting process and the voting process itself are error-prone. Especially in complex elections, voters often cast invalid ballots unintentionally. The obvious solution to these problems - computer-aided support and automatization - has led to the introduction of voting machines, and electronic voting in general. An electronic vote casting interface could help the voter to cast a valid ballot. The tally could be done electronically which would be much faster and less error-prone. Moreover, electronic voting introduces the possibility of casting ballots via the internet, which promises to increase voter turnout.

However, electronic voting makes the once so transparent process of voting very opaque. Voting machines are seen as black boxes by voters. It is not obvious to see that a ballot is recorded as it was filled-out by the voter, or that all cast ballots are counted correctly, since the ballot no longer has a physical representation which is accessible for the voter. Malicious software could break privacy, influence voters improperly, manipulate cast votes and output a wrong tally. Testing complex hardware and software systems reliably is very difficult. The situation becomes even more challenging when elections are conducted over the internet, where the voter does not interact with most system components in person, and the correct processing of ballots seems impossible to observe.

All this shows the importance of mechanisms which prove the correctness of an election, independent of their implementation. Rivest and Wack call this *Software independence* [RW06].

Cryptographic voting schemes use cryptographic methods to prove that the election result is correct. These proofs ideally do not depend on any implementation and can be verified by anyone. Also, each voter should be able to convince herself that her choice is included in the counting process as it was intended and filled-out by the voter. This requirement seems to be in competition with secrecy and coercion-resistance, since such proofs of correctness receive the voter's choice as input in some form, and need to prove that this input was processed correctly, without revealing information about it.

In fact, in the e-voting community, there has been a long-standing debate about vote secrecy and certain levels of coercion-resistance versus verifiability. And yet, one cannot be considered without the other. The stronger the level of secrecy and coercion-resistance, the more opaque becomes the voting process, the more convincing proofs of correctness are necessary. The advocates of secrecy argument that, on the other hand, without coercion-resistance, one cannot be sure if the tally really reflects the will of the voters, so without coercion-resistance, we cannot speak of verifiable correctness of the tally.

Moreover, both requirements have impact on other aspects of a voting scheme, like its robustness, or its suitability for large-scale elections. This leads to the question of suitable design criteria for voting schemes. In fact, research on electronic voting has been done for several decades, resulting in a huge variety of voting schemes, designed along heterogeneous aspects, resulting in largely differing concepts. This is reflected in the fact that the widely agreed on requirements of coercion-resistance and verifiability are defined in various ways, under differing underlying models. And still, there are election types for which the task of achieving coercion-resistance is particularly challenging and not yet achieved.

The aim of this work is twofold. First, we introduce a taxonomy with which the existing variety of voting schemes can easily be analyzed and compared. The taxonomy offers design criteria for new voting schemes. It also offers a tool for

analyzing if a new voting scheme fulfills basic requirements, and what its strengths and weaknesses are, independent of its underlying formal model.

The second part works towards achieving coercion-resistance in three special cases. One of them are elections with write-in candidates, where additionally to a candidate list from which the voter can choose, she has the possibility to write a name on her ballot which is not in this list. An adversary can use this to coerce the voter to vote for a recognizable write-in candidate. Another special case occurs in internet elections. As a countermeasure to observation, or simply to get familiar with the voting system, voters are often allowed to cast more than one valid ballot, of which only the last ballot is included in the tally. However, if the adversary learns whether the voter has revoted, he can coerce her to cast a certain ballot and then not cast another ballot afterwards. A rather young challenging paradigm is vote delegation, where the voter can either cast a vote by herself, or delegate her decision to another eligible voter. The correct delegation needs to be proven to the voter without showing the adversary whether the voter has delegated her vote to him.

1.1. Contributions of this Work

The contributions of this work are presented in three parts. The first part discusses requirements of voting schemes. The second and third part discuss coercion-resistance in presential and internet elections, respectively.

1.1.1. Requirements of Voting Schemes

The first two contributions center around requirements of voting schemes:

- We introduce a taxonomy with which voting schemes can be analyzed consistently. This Taxonomy is an advanced version of the work in [CvdGRV07]. Its advancement is joint work with Jeroen van de Graaf, Dirk Achenbach and Bernhard Löwe. Our main contribution is a roadmap which helps analyzing voting schemes according to a list of criteria. We provide a catalog of questions, which allows us to easily and quickly categorize and compare different voting schemes, estimate which criteria they achieve to what extend, and what the strengths and weaknesses of these schemes are. We demonstrate our taxonomy on various voting schemes.

- We give a brief review of definitions of coercion-resistance and discuss how they can deal with write-in candidates, and with the possibility of revoting.

1.1.2. Presential Elections

The second part of this thesis discusses practical experiences as well as several matters of coercion-resistance in presential elections:

- We introduce our implementation and real-world experiences with Bingo Voting. We describe practical experiences and discuss coercion-resistance aspects. Then we analyze this election with our taxonomy and compare the analysis to our experiences.

- Though Bingo Voting has turned out to be very flexible, it does not allow for write-in candidates in its original version. To make the scheme more broadly applicable, we equip it with write-in candidate support.

- Coercion resistance of election schemes which allow write-in candidates is hard to measure, existing definitions are often impossible to fulfill with write-in candidates. But coercion-resistance for write-in votes is possible if the tally is published in a fuzzy way. This work introduces a first definition for a controlled fuzzy tally representation, as well as a technique which meets this definition while maintaining verifiability. The definition is very similar to k-anonymity, a security definition for privacy in databases used for data mining.

- After introducing our general approach, we provide Bingo Voting, as well as its version which supports write-in candidates, with the possibility of a fuzzy but provable tally representation. To our knowledge, the scheme introduced is the first to provide coercion-resistant and publicly verifiable elections with write-in candidates.

1.1.3. Internet Elections

The third part of this thesis discusses coercion-resistance in internet elections:

- In internet elections, we cannot assume that each voting process takes place in private. This opens doors for coercion because it becomes easier for an adversary to observe the voter.

  On a closer look, the seemingly intuitive solution to this problem, allowing revoting, turns out to be more challenging than it seems at first sight. The voter now needs to be proven that her most current ballot is counted even though she has cast provably valid ballots before. Moreover, the public wants to be proven that of each voter, only one vote counts. However, such a proof must by no means yield the information whether the voter has made use of her possibility to revote. Otherwise, the adversary could observe the voter once and then coerce her not to cast a vote again afterwards. This work introduces a voting scheme which allows incoercible revoting with verifiable correctness of the processing of revotes.

- Delegated voting is an election type which implements a so-called *Liquid Democracy*. In elections which allow vote delegation, each voter can choose to either vote by herself or give the weight of her ballot to a so-called proxy, another eligible voter of the voter's choice, who then votes in her stead. This introduces interesting new security challenges, because instead of merely proving tally correctness, the voter also has to be provided with a proof that her vote has been delegated to the right proxy. But to prevent coercion, this proof must by no means enable the voter to convince a proxy that she delegated her vote to him. An additional requirement to such elections is usually that the voter can change her mind at anytime, taking away delegations from proxies and voting by herself instead, or delegating to another proxy. In this work, our solution to the revoting problem is applied to delegated voting, in combination with a new paradigm we call *vote fetching*.

1.2. Structure of this Work

This work is organized as follows. In Chapter 2, we present preliminaries and cryptographic building blocks. Chapter 3 contains our taxonomy for cryptographic voting schemes and a review of existing definitions of coercion resistance with respect to write-in candidates and revoting. Chapter 4 discusses coercion-resistance in

presential elections on the example of Bingo Voting, and describes our practie experiences with the scheme in the student parliament election 2008. The main part of the chapter then concentrates on write-in candidates, including definitions and techniques for a fuzzy tally representation. This is followed by Chapter 5 which discusses coercion-resistance in internet elections while mainly centering around the revoting problem. After introducing our revoting solution, we introduce vote fetching and merge the two concepts to a voting scheme which allows incoercible vote delegation. Chapter 3-5 each open with related work relevant to the respective chapter. Chapter 6 concludes this work with a short summary of our results, open problems and possibilities for future work.

2. Preliminaries

This chapter first gives an intuition of what cryptographic voting schemes are, and how they achieve the two main properties of verifiability and privacy. After this, some known attacks on coercion-resistance are described, to give a first intuition of some possible pitfalls. The rest of this chapter then introduces preliminaries on cryptographic primitives which are important for this work.

2.1. What are Cryptographic Voting Schemes?

Cryptographic voting schemes are voting schemes which use cryptographic methods to achieve certain requirements. Usually, the minimum which is to be achieved is *public verifiability* to a certain extend, and certain *privacy-type properties*. This section gives a short overview over these notions before giving an intuition of how these properties are usually achieved in cryptographic voting schemes.

2.1.1. Privacy-type properties

Privacy in its simplest form means that no one but the voter can see which choice the voter casts. But privacy is often not enough. To prevent voters from improper influence through coercion or vote-buying, it is important that even in full cooperation with the adversary, the voter cannot convince the adversary that she has followed his instructions. If, from the adversary's point of view, a full cooperation of the voter is indistinguishable from the situation where the voter follows her own will, the adversary is likely not willing to pay the voter, and a coercion is useless for the same reason. The following privacy-type notions have been established in the literature:

- **Privacy:** The voter has the possibility to cast her ballot in a way that no third party learns her choice.
- **Receipt-Freeness:** The voter obtains no data which allows her to create a convincing proof that she has cast a certain choice. So even with a fully cooperating voter, the adversary is not able to deduce the voter's choice.
- **Coercion-Resistance:** Even a fully cooperating voter cannot convince the adversary that she has followed his instructions in a way which affects her choice. This property is also called *incoercibility*.

Relations between privacy-type properties

Obviously, receipt-freeness is a stronger notion than privacy. It has been introduced in [BT94] and basically states the requirement that privacy should not be optional. The difference between receipt-freeness and coercion-resistance might not be as obvious. Receipt-freeness is the weaker notion since it only demands that the voter cannot provably reveal her choice. Coercion-resistance demands that even if the adversary does not learn the voter's choice, he cannot take any influence which affects her choice in any visible way. In Juels et al. [JCJ05], the difference between receipt-freeness and coercion-resistance is described via three separating examples:

- **Randomization attack:** The adversary does not learn the voter's choice, but can force her to cast a vote for a random candidate instead of her choice.
- **Forced abstention attack:** The adversary can coerce the voter not to participate in the election. We also speak of forced abstention if the adversary can coerce the voter to cast an invalid ballot.
- **Simulation attack** The adversary coerces the voter to reveal all her keying material to him, so the adversary can impersonate the voter and vote in her stead.

Everlasting privacy

A somewhat orthogonal privacy-type property is *everlasting privacy* [MN06]: privacy holds in an information-theoretic sense and does not depend on cryptographic assumptions. Everlasting privacy is implied by neither above mentioned privacy property and implies neither. It is, however, an important topic when talking about coercion-resistance. It guarantees that even in, say, 20 years or more, voter privacy still holds. In most voting schemes, privacy depends on the encryption scheme which is used to encrypt the ballot, which in turn usually depends on certain cryptographic assumptions. One could argue that everlasting privacy is necessary to achieve incoercibility, since the adversary could threaten to break privacy eventually and then find out the voter's choice. General techniques to achieve everlasting privacy in homomorphic and mixnet-based voting schemes have been presented [DvdGSdSA12, BDvdG13]. In this thesis, everlasting privacy is not considered as a main topic, but is always kept in mind as an important future research direction.

2.1.2. Verifiability

The three most important notions of verifiability are the following:

- **Individual verifiability:** The voter can check that her ballot is recorded correctly and included in the tally.
- **Universal verifiability:** Everyone can check that the tally is computed correctly.
- **End-to-End verifiability:** Every voter can check that her ballot is included in the tally and processed correctly, and that the tally is then done corectly.

End-to-End verifiability (E2E) is believed to hold if both individual and universal verifiability are given. The voter can track her ballot until it enters the tally, where it is unlinked from the voter's identity, but processed correctly together with the other ballots in a provable and verifiable way. E2E is an important paradigm to achieve *software independence*[RW06]: the correctness of the tally can be verified independently of the implementation of any voting computers or election servers.

2.1.3. Types of Cryptographic Voting Schemes

In the literature, there are three main techniques which have been established to achieve privacy-type properties and verifiability simultaneously, dividing existing cryptographic voting schemes mainly into three groups.

1. **Voting schemes based on anonymous channels:** The ballot is unlinked from the voter before casting: she gets an anonymous credential for voting. This credential can have the form of a token [JCJ05] or a blind signature [Oka98] and does not reveal the voter's identity. The credential proves that a ballot has been cast by an eligible voter. Since the ballot is not linked to the voter upon casting, it can be opened and counted.

2. **Voting schemes based on verifiable shuffling:** The second group consists of mix-based voting schemes [SK95, Nef01, Adi08, BMQR07], where voters cast their ballots in form of a rerandomizable ciphertext or a commitment. These ballots are then unlinked from the voter's identity by shuffling and rerandomizing them, and proving the correctness of the shuffling process. After the ballots are shuffled, they can be opened and counted.

3. **Voting schemes based on homomorphic tallying:** The voter casts her ballot encrypted with a homomorphic encryption function. The election result is then computed without opening a single ballot. Instead, the homomorphic property is used to compute the sum on the ciphers. The ciphertext containing the sum is then opened and the correctness of the decryption is proven. Examples for voting schemes in this group can be found in [CGS97, Acq04, DvdGSdSA12].

The separation between the groups is blurry: voting schemes in the first group might use a mixnet to achieve an anonymous channel, and a homomorphic tallying procedure can often be substituted by a mixnet and vice versa. This work mainly introduces voting schemes based on the second group.

2.2. Known Attacks on Coercion-Resistance

There are several known attacks on voting schemes which affect coercion-resistance. Most of them can be met with standard measures. A similar overview over known attacks is given in [Hen12].

Pattern Voting

In some voting schemes the voter can be coerced to fill out her ballot on a certain pattern. This is the case for each election where the voter does not cast a 1 out of n choice. The voter can be coerced to distribute her choices in a certain pattern. In this case, the attack can easily be prevented by vote splitting, as mentioned in [PS07]. In some voting schemes like Punchscan [PH10] or Prêt à Voter [CRS05],

where receipt-freeness relies on the permutation of candidates, the voter can still be coerced to fill out her ballot in a certain pattern, for example by forcing her to mark the first choice regardless of what candidate this choice encodes. This would be an example for the randomization attack mentioned in Section 2.1.

Mixnets and homomorphic encryption

If the ballot is encrypted with Elgamal or another homomorphic scheme, the adversary can encrypt his vote for example as $E(2) \cdot E(v_C) = E(2v_C)$, where v_C is the coerced voter's choice, and see if $2v_C$ appears after mixing and opening. The adversary casts an invalid ballot but breaks voter privacy of this voter. This way, the adversary could coerce $2k$ voters, coerce voter i to vote for candidate v_i for $i = 1, \ldots, k$, and coerce voter $k+i$ to cast $E(i) \cdot E(v_i)$. So the adversary has caused k invalid ballots that contain information about the k other ballots.

To prevent this, upon casting an encrypted vote, the voter has to prove that the contained plaintext is valid, as is usually done in voting schemes with homomorphic tallying, or by letting the voter prove knowledge of the encrypted plaintext.

Babble attack

A so-called *babble attack* on the voting scheme of Moran and Naor [MN06] was described in [BMQR07]. The idea is that the adversary has an audio connection to the voter during the voting phase, and whenever the voter is to input randomness which has recognizable impact on any published data, the adversary dictates the randomness to the voter.

Shoulder Voting

If an election is held over the internet, voter privacy cannot be guaranteed. The voter can be observed during the voting process, for example by family members and friends. This is called *shoulder voting* or *family voting*. Measures against this attack are introduced in Section 5.

2.3. Cryptographic Primitives

In this section, the cryptographic primitives used in this work are introduced.

2.3.1. General Definitions

Before the cryptographic primitives are described, we start with some general definitions and notions.

2.3.1.1. Probabilistic Polynomial Time (PPT) Algorithms

Most algorithms used in this work are *probabilistic*; they can be thought of as Turing machines which can flip coins [Gil74], i. e. choose certain randomness during their runtime. A *probabilistic polynomial time (PPT) algorithm* is a probabilistic algorithm which terminates within *polynomial time*, i. e. its runtime is bounded by a polynomial of its input length. We call such an algorithm *efficient*. We call a problem *computationally infeasible*, if there is no PPT algorithm which can solve the problem. We speak of these problems as *hard* in certain algebraic structures.

Many cryptographic primitives are based on such underlying problems. Two examples are the discrete logarithm problem and the decisional Diffie-Hellman problem described below.

2.3.1.2. Discrete Logarithm Problem

The security of many cryptographic primitives is based on the *discrete logarithm problem*. Examples used in this work are the Elgamal encryption and Pedersen commitments.

Definition 1 (Discrete Logarithm Problem) *Let $\mathbb{G}$ be a cyclic group of prime order p, $a \in \mathbb{G}$. Given an element $c \in \mathbb{G}$, the* discrete logarithm problem (DLog problem) *is to find an exponent $x \in \mathbb{Z}_p$ with $a^x = c$.*

The discrete logarithm problem is generally believed to be hard if the group order p is appropriately chosen.

2.3.1.3. Decisional Diffie-Hellman Problem

Related to the discrete logarithm problem is the *decisional Diffie-Hellman problem* [Bon98]. If the decisional Diffie-Hellman problem is hard, the DLog problem is also hard.

Definition 2 (Decisional Diffie-Hellman (DDH) Problem) *Let $\mathbb{G}$ be a cyclic group of prime order p with a generator $G \in \mathbb{G}$. Given (G, G^a, G^b, G^c), the* decisional Diffie-Hellman problem *is to decide whether $c = ab$.*

2.3.1.4. Negligible and Overwhelming Functions

As usual in the literature, a function $f : \mathbb{N} \to \mathbb{R}$ is called *negligible*, if for every constant $c > 0$ there is an $n_0 \in \mathbb{N}$ such that for every $n > n_0$ it holds that $f(n) \leq \frac{1}{n^c}$.

A function f is *overwhelming* if $1 - f$ is negligible.

2.3.2. Cryptographic Hash Functions

In general, a hash function

$$h : \{0,1\}^* \to \{0,1\}^k$$

maps an element $m \in \{0,1\}^*$ of arbitrary length to an element $h(m) \in \{0,1\}^k$ of fixed length k. We call $h(m)$ the *hash value* of m. A *cryptographic hash function* is a hash function with the following properties:

- **Efficiency:** The function h is efficiently computable, i. e. there exists a PPT algorithm which computes h.
- **Preimage-Resistance:** Given a hash value $c \in \{0,1\}^k$, it is computationally infeasible to compute a message $m \in \{0,1\}^*$ with $c = h(m)$.
- **Collision-Resistance:** It is computationally infeasible to find two messages $m_1, m_2 \in \{0,1\}^*$ with $h(m_1) = h(m_2)$.

We call a pair (m_1, m_2) with $h(m_1) = h(m_2)$ a *collision*. Since $\{0,1\}^k \subsetneq \{0,1\}^*$, h cannot be injective, so there are always collisions, but it should be hard to find them.

2.3.3. Public Key Encryption Schemes

Intuitively, an encryption scheme E defines an *encryption function Enc*, with which messages from a *plaintext space* $\mathcal{M}$ can be *encrypted* with a *public key pk*, also called an *encryption key*, to obtain a *ciphertext c*, also called an *encryption*, in a *ciphertext space* $\mathcal{C}$. The ciphertext c can later be *decrypted* with an *decryption function Dec* and a *secret key sk* to obtain the original message $m \in \mathcal{M}$. The secret key *sk* is often also called a *decryption key* or a *private key*.

2.3.3.1. Definition of Public Key Encryption Schemes

More formal, an encryption scheme $E = (Setup, KeyGen, Enc, Dec)$ consists of four efficiently computable algorithms, which are defined as follows:

- $Setup(k) \mapsto gs$ is a randomized algorithm which takes as input a security parameter k and outputs an algebraic structure which we call a *group setup* gs, in which encryptions can be created. The group setup gs defines a *plaintext space* $\mathcal{M}$, a *ciphertext space* $\mathcal{C}$ and a *key space* $\mathcal{K}$.
- $KeyGen(gs, k) \mapsto (sk, pk)$ is a randomized algorithm which takes as input a security parameter k and outputs a keypair $(sk, pk) \in \mathcal{K}$, whereas sk is referred to as a *decryption key*, and pk is called the *encryption key*.
- $Enc(pk, m) \mapsto c$ takes as input a message $m \in \mathcal{M}$ and a public key pk, and outputs a ciphertext $c \in \mathcal{C}$. We call m a *plaintext* and c a *ciphertext*.
- $Dec(sk, c) \mapsto m$ takes as input a ciphertext $c \in \mathcal{C}$ and a decryption key sk, and outputs a plaintext $m \in \mathcal{M}$ with $Enc(pk, m) = c$, if such an m exists. Otherwise, it outputs $\perp$.

Each algorithm might take additional input. Sometimes, Enc additionally takes as input some randomness r, or is assumed to choose a random parameter r which it uses in the encryption process, such that the same plaintext can be encrypted to different ciphertexts. In this case, we call *Enc probabilistic*.

An encryption scheme must be *correct*, i. e. for all $m \in \mathcal{M}$ and key pairs (sk, pk) generated by $KeyGen$ it must hold that $Dec(sk, Enc(pk, m)) = m$ with overwhelming probability.

2.3.3.2. Reencryption

Some probabilistic encryption functions allow to be *rerandomized*, i. e. there exists an algorithm $Reenc(c, r)$, which takes as input a ciphertext c and a parameter r, which we call the *reencryption randomness*, and creates a new ciphertext

$$c' := Reenc(c, r).$$

The ciphertext c' contains the same plaintext message as c, but $c' \neq c$. In the following we omit the parameter r when it is clear from the context or implicitly chosen by the reencryption algorithm.

Definition 3 (Rerandomizable) *We call an encryption function Enc* rerandomizable*, if a function Reenc exists, such that for each ciphertext c in the underlying ciphertext space, Reenc(c) is indistinguishable from a new ciphertext which encrypts another plaintext. We call Reenc the* reencryption function *of Enc.*

2.3.3.3. The Elgamal Encryption Scheme

The Elgamal encryption scheme [ElG85] is an asymmetric encryption scheme introduced in 1984 by Taher Elgamal. It is IND-CPA-secure [GM84, BDPR98, Bon98] under the decisional Diffie-Hellman assumption.

An Elgamal encryption scheme $(Setup, KeyGen, Enc, Dec)$ is a public key encryption scheme with the following algorithms:

- $Setup(k)$ is a randomized algorithm which on input of a security parameter k chooses a cyclic group $\mathbb{G}$ of prime order $p \geq 2^k$ and a generator $g \in \mathbb{G}$. It outputs the group setup $gs := (p, \mathbb{G}, g)$. The plaintext space is then the group $\mathbb{G}$, and the ciphertext space is $\mathbb{G} \times \mathbb{G}$.

- $KeyGen(gs, k)$ takes as input a security parameter k and a group setup gs, and chooses an $x \in \mathbb{Z}_p$ at random, the *secret key*. It computes a corresponding *public key* $y := g^x \in \mathbb{G}$. It outputs the key pair (x, y).

- $Enc(y, m)$ is a probabilistic algorithm which takes as input a public key $y \in \mathbb{G}$ and a message $m \in \mathbb{G}$, chooses a random $r \in \mathbb{Z}_p$ and outputs the ciphertext

$$c := (g^r, y^r m).$$

- $Dec(x, c)$ takes as input a secret key $x \in \mathbb{Z}_p$ and a ciphertext $c = (a, b) \in \mathbb{G} \times \mathbb{G}$, and outputs the message

$$m = a^{-x} b.$$

The Elgamal encryption scheme has some properties that are very useful for cryptographic voting schemes. One of these properties is that it is *plaintext aware*: the sender of an Elgamal-encrypted message can prove knowledge of its plaintext without revealing information about it, for example by using the Schnorr-protocol [Sch91]. In e-voting schemes, we need this property to prevent the voter from casting a ciphertext given to him by the adversary, or depending her input on the encrypted ballot of another voter. A proof of correct decryption can be done with the *Chaum-Pederson protocol*[CP93], which enables a server which creates an election result from Elgamal-encrypted ballots to prove the correctness of the result.

Homomorphy of the Elgamal encryption

The Elgamal encryption function Enc is multiplicatively homomorphic under componentwise multiplication: given two messages $m_1, m_2 \in \mathbb{G}$, we have that

$$Enc(m_1) \circ Enc(m_2) = Enc(m_1 \cdot m_2),$$

where $\circ$ denotes componentwise multiplication. This holds because for two ciphertexts $c_1 = (g^{r_1}, y^{r_1} m_1)$ and $c_2 = (g^{r_2}, y^{r_2} m_2)$, we have that

$$c_1 \circ c_2 = (g^{r_1} g^{r_2}, y^{r_1} m_1 y^{r_2} m_2) = (g^{r_1 + r_2}, y^{r_1 + r_2} m_1 m_2).$$

Reencryption

Since Elgamal is multiplicatively homomorphic, it allows an easy way of reencryption without knowledge of the encrypted plaintext, by multiplying a ciphertext with an encryption of the neutral element of $\mathbb{G}$:

Let $c = (g^r, y^r m)$ be an Elgamal ciphertext, then we can create a reencryption of c by choosing a random number $r' \in \mathbb{Z}_p$ and calculating

$$c' = (g^r, y^r m)(g^{r'}, y^{r'}) = (g^{r+r'}, y^{r+r'} m).$$

The new ciphertext c' is an encryption of the same message m, but with new randomness $r + r'$.

Threshold Elgamal

The Elgamal encryption scheme can be set up in a way that decryption can be distributed among several parties. We call this *threshold Elgamal*. For a full description of threshold Elgamal, the reader is referred to the original paper of Pedersen [Ped91b], an explanation is also given in [CGS97]. Intuitively, several different parties $P_1, \ldots, P_k$ each create their own key pair (sk_i, pk_i), $i = 1, \ldots, k$. The keys sk_i are kept secret, the pk_i are accumulated to a joint public key pk. The keys are setup in a way that the different parties share a secret s which is the Elgamal decryption key corresponding to pk.

An encryption with threshold Elgamal is an ordinary Elgamal encryption with public key pk. However, decryption can only be done jointly by all k parties who share the secret s. If only one of them is missing, decryption is impossible. In this process of decryption, the secret s is never reconstructed in plaintext, therefore it remains unknown to the participating parties.

In voting schemes, this is used to distribute the task of decrypting ballots to several servers, to prevent a single server from decrypting a ballot while it is still linked to the voter's identity.

2.3.4. Commitment Schemes

The purpose of a commitment scheme is to enable a party to commit to a value m without revealing it, and optionally opening it at a later time to prove the commitment to m.

2.3.4.1. Definition of Commitment Schemes

A commitment scheme $(Setup, Com, Unveil)$ consists of three algorithms $Setup$, Com and $Unveil$ defined as follows:

- $Setup(k) \mapsto gs$ takes as input a security parameter k and outputs a group setup gs for the creation of commitments. The group setup gs implies a *message space* $\mathcal{M}$, a *commitment space* $\mathcal{C}$ and a *randomness space* $\mathcal{R}$.
- $Com(m, r) \mapsto c$ takes as input a message $m \in \mathcal{M}$ and randomness $r \in \mathcal{R}$, and creates a *commitment* $c \in \mathcal{C}$. We call the randomness r the *unveil information* of c.
- $Unveil(c, r) \mapsto m$ takes as input a commitment $c \in \mathcal{C}$ and unveil information $r \in \mathcal{R}$, and outputs a message m with $c = com(m, r)$.

A commitment scheme has the following two properties:

- **Binding:** The commitment function Com is *computationally binding* if for any pair $(m, c) = com(m, r)) \in \mathcal{M} \times \mathcal{C}$, it is computationally infeasible to find another message $m' \neq m$ and unveil information r' with $c = com(m', r')$. We call a commitment function *perfectly binding*, if no pair (m', r') exists with $m \neq m'$ and $c = com(m', r')$.
- **Hiding:** The commitment function Com is *computationally hiding* if for any commitment $c = com(m) \in \mathcal{C}$ it is computationally infeasible to open the commitment c, i. e. find out m without the knowledge of m and r. It is *perfectly hiding*, if opening c is impossible without knowledge of r.

It is proven in [Dam99] that a commitment scheme cannot be both perfectly binding and perfectly hiding. For e-voting schemes, commitment schemes which are perfectly hiding are usually preferred, since the hiding property affects the privacy of the voter's choice. Computationally binding is often sufficient since the binding property only needs to hold until proofs of correctness of the tally are created, which is usually during or immediately after the election, so this property only needs to hold "long enough".

2.3.4.2. Pedersen Commitments

Pedersen commitments were originally introduced by Chaum et al. in [CDvdG87], but have been called Pedersen commitments after they were also described by Pedersen in [Ped91a]. The Pedersen commitment scheme consists of the three algorithms *Setup*, *Com* and *Unveil* which work as follows:

- $Setup(k)$ takes as input a security parameter k and creates a cyclic group $\mathbb{G}$ with prime order p, and two generators g and h of $\mathbb{G}$. It outputs the group setup $gs := (p, \mathbb{G}, g, h)$.
- $Com(m, r)$ takes as input a message $m \in \mathbb{G}$ and randomness $r \in \mathbb{Z}_p$ and outputs the commitment
$$c := g^m h^r \in \mathbb{G}.$$
- $Unveil(c, m, r)$ takes as input a commitment $c \in \mathbb{G}$ with $c = Com(m, r)$ and outputs the unveil information (m, r).

Pedersen commitments are unconditionally hiding and computationally binding: for each commitment $c = Com(m, r)$ and each message $m' \in \mathbb{G}$ there is an $r' \in \mathbb{Z}_p$ with $Com(m', r') = c$, but a discrete logarithm must be computed to find such an r'. For this reason, Pedersen commitments are computationally binding with the binding property depending on the discrete logarithm problem.

Pedersen Commitments are often used in cryptographic voting schemes since they have useful properties. For example, they are multiplicatively homomorphic. Using this property, they can be *masked* without knowing their content, and the correct masking can be proven with merely knowing the masking randomness. Masking for commitment schemes is the analog of reencryption for encryption schemes: it is a rerandomization of a given commitment.

Masking Pedersen commitments

Pedersen commitments can be *masked*, i. e. rerandomized, without knowledge of their contents, in the following way:

Let $(p, \mathbb{G}, g, h)$ be the group setup of a Pederson commitment scheme. A commitment $c = g^m h^r$ of a message $m \in \mathbb{G}$ can be masked by choosing a random $r' \in \mathbb{Z}_p$ and computing the new commitment

$$c' := c \cdot h^{r'}.$$

We have

$$c' = ch^{r'} = g^m h^r h^{r'} = g^m h^{r+r'},$$

so c' is also a commitment to the message m, but with new randomness $r + r'$.

2.3.5. Zero-Knowledge Proofs

Informally stated, *zero-knowledge proofs* [GMR85] are proofs with which a *prover* proves a *statement* A to a *verifier*, without leaking any information other than the fact that A is true.

A zero-knowledge proof has the following properties:

- *Completeness:* An honest prover is always able to convince an honest verifier that the statement A is true.
- *Soundness:* A dishonest prover is not able to convince an honest verifier that the statement A is true. We speak of *computational soundness*, if no dishonest PPT prover can convince the verifier that a false statement A is true. Alternatively stated, a dishonest prover can convince the verifier only with negligible probability. We speak of *perfect soundness*, if this probability is 0, i. e. an unbounded dishonest prover is unable to convince the verifier that A is true.
- *Simulatability:* With only the knowledge of A, the verifier is able to generate a simulated transcript which is indistinguishable from the transcript of an interaction between the prover and the verifier, in which a valid proof of A is created. This is modeled by the existence of a *simulator* which can create such simulated transcripts. This property is often referred to as the *zero-knowledge property*. We speak of *perfect zero-knowledge*, if the transcript created by the simulator is perfectly indistinguishable from a valid proof. We speak of *computational zero-knowledge*, if the transcript of the simulator is *computationally indistinguishable* from a valid proof, i. e. there is no PPT algorithm which can distinguish if the transcript is from an honest prover or a simulator in polynomial time.

If soundness is achieved only computationally, the literature speaks of a zero-knowledge *argument*, and uses the notion *proof* only in the case of perfect soundness. We relax this notion in this work and use the word *proof* also for zero-knowledge arguments. A zero-knowledge proof is usually an interaction between the prover and the verifier, and convinces only the verifier that the statement A is true, since because of the zero-knowledge property, the verifier can *simulate* the proof. We call a zero-knowledge proof a *non-interactive zero-knowledge proof (NIZK)*, if it is non-interactive. Often the statement A depends on a secret *witness* w. A zero-knowledge proof is called a *zero-knowledge proof of knowledge*, if the prover proves that he knows a witness w which makes the statement A true. Weaker forms of zero knowledge are the notions *witness hiding* or *witness indistinguishable*. A proof of a statement A is *witness hiding*, if it hides the witness w but may leak other information. A proof of a statement A is *witness indistinguishable*, if in the case that more than one witness exists which makes the statement A true, the proof does not leak which witness w is used in the proof, but it may leak other information.

2.3.6. Verifiable Shuffling and Mixnets

An important cryptographic primitive often used in cryptographic voting schemes is verifiable shuffling. It is, next to homomorphic tallying and anonymous channels [JCJ05], one of the three most used techniques to unlink a plaintext ballot from the voter's identity, i. e. achieving vote privacy. There are generally two types of shuffles,

namely re-encryption mixes and decryption mixes. In this work, we only use re-encryption mixes, and use the words *shuffle* and *re-encryption mix* interchangeably from now on.

In general, a re-encryption-mix computes a function $shuffle(C) \mapsto C'$, which gets as an input a list

$$C := [c_1, \ldots, c_n]$$

of rerandomizable ciphertexts. Its output is a list

$$C' := [c'_{\pi(1)}, \ldots, c'_{\pi(n)}]$$

of ciphertexts, where π is a random permutation and $c'_{\pi(i)} := Reenc(c_i)$ for $i = 1, \ldots, n$, and $Reenc$ is the reencryption function used for rerandomizing the ciphers $c_1, \ldots, c_n$. Since in the shuffling procedure, a reencryption of each ciphertext is computed, the function *shuffle* also takes as input the randomness used for reencryption. We omit this in the notation for clarity.

For its use in e-voting schemes, we wish a shuffle to have the following properties:

- **Secrecy:** We say that the shuffle is *secret* if it is infeasible to establish a link between an input ciphertext c_i and its corresponding output ciphertext $c_\pi(i)$ better than guessing.
- **Verifiability:** We say that a shuffle is *verifiable*, if there exists a proof that C is indeed a permutation and rerandomization of C', i. e. for each $c' \in C'$ there exists a $c_i \in C$ with $c' = Reenc(c_i)$, and vice versa: For each $c \in C$ there exists a $c' \in C'$ with $c' = Reenc(c)$.

Intuitively, *verifiable* means that an outsider can verify that no element was lost, added or changed in the shuffling process.

Several techniques for verifiable shuffling and mixnets building on them have been introduced: [Nef01, SK95, Gro02, KMW12, Wik04, AH01, GJJS04] to name just a few. A review on mixes was presented in [Adi06]. Verifiable shuffling of commitments or ciphertexts is used in several parts of this work. Two examples of techniques for verifiable shuffling are described below.

2.3.6.1. Mixnets

We speak of a *mixnet*, if a shuffle is done sequentially by several servers, called the *mix servers*, such that each server shuffles its list of input ciphertexts and gives the output of its shuffle to the next server as an input.

2.3.6.2. Proof of a Correct Shuffle with Shadow Mixes

In [Adi08], Adida describes a rather intuitive zero-knowledge proof of a shuffle, following the idea of Benaloh [Ben06] and [SK95], using so-called *shadow-mixes*. The idea of a shuffle proof with shadow mixes is as follows:

Let $shuffle(C, \pi)$ denote a secret shuffle of a list C of ciphertexts with a permutation π, i. e. for $C := [c_1, \ldots, c_n]$ and a permutation π,

$$C' := shuffle(C, \pi) = [c'_{\pi(1)}, \ldots, c'_{\pi(n)}]$$

with $c'_{\pi(i)} := Reenc(c_i)$ for all i and a reencryption function $Reenc$.

A prover who has created a shuffle C' of a list of ciphertexts C, proves the correct shuffling to a verifier as follows: the prover chooses a permutation π_L and creates a *shadow mix*

$$C'' := shuffle(C, \pi_L).$$

Then he chooses a second permutation π_R such that

$$\pi_L \circ \pi_R = \pi,$$

i. e. $\pi_R(\pi_L(i)) = \pi(i)$ for each $i = 1, \ldots, n$. Since the prover has also created the reencryption of each ciphertext, he knows the randomness used for the reencryption and can therefore create reencryption randomness for a second shuffle that leads to C':

$$C' = \mathit{shuffle}(C'', \pi_R).$$

The prover sends C'' to the verifier, but keeps the permutations π_L, π_R and π secret.

The verifier chooses a bit b.

- If $b = 0$, the prover opens π_L and the reencryption randomness used to create $C'' = \mathit{shuffle}(C, \pi_L)$.
- If $b = 1$, the prover opens π_R and the reencryption randomness with which $C' = \mathit{shuffle}(C'', \pi_R)$.

In this proof, the prover has a 50% chance of a manipulation, since if C' is not a correct shuffle of C, then either $C'' \neq \mathit{shuffle}(C, \pi_L)$ or $C' \neq \mathit{shuffle}(C'', \pi_R)$. Therefore, the prover creates several instances of the proof, by creating k shadow mixes $\mathcal{C}''_1, \ldots, \mathcal{C}''_k$ and corresponding pairs of permutations $(\pi_{L,j}, \pi_{R,j})$ with

$$\pi_{L,j} \circ \pi_{R,j} = \pi$$

for $j = 1, \ldots, k$, and answers a challenge of the verifier for each of them.

For the use in e-voting, the challenge bits are usually either created by a set of auditors, or the Fiat-Shamir heuristic (see below) is used. The operation of sending a value to the verifier is usually done by publishing it on a public bulletin board.

Under the assumption that the reencryption is secure, i. e. a reencryption of a ciphertext is indistinguishable from a new encryption of another plaintext, this is a zero-knowledge proof of correct shuffling:

Completeness:

An honest prover can always correctly answer to both possible challenges.

Soundness:

In each proof instance, the prover has a chance of manipulation of $\frac{1}{2}$. With k proof instances, this chance of a single manipulation not being detected decreases to $\frac{1}{2^k}$.

Simulatability:

The simulator can simulate a proof as follows: he can chose a bitstring of "challenge bits" $b = b_1, \ldots, b_k$. For each challenge bit b_i, he does the following:

- If $b_i = 0$, he chooses a permutation π_L and creates a "shadow mix"

  $$C''_i := \mathit{shuffle}(C, \pi_L)$$

  without knowing a corresponding "right side" π_R to link this shadow mix to C'.
- If $b_i = 1$, he chooses a permutation π_R and creates a "shadow mix"

  $$C''_i := \mathit{shuffle}(C', \pi_R^{-1})$$

  without knowing the corresponding left side.

Non-interactiveness with the Fiat-Shamir heuristic

To make this proof non-interactive, the Fiat-Shamir heuristic [FS86] can be used. To do this, the prover proceeds as follows:

1. The prover creates k shadow mixes $\mathcal{C}''_1, \ldots, \mathcal{C}''_k$ and sends them to the verifier. Then a publicly known cryptographic hash function h is applied to the created shadow mixes to create the challenge. Permutations and randomness are opened according to this challenge.

To be sound, the Fiat-Shamir heuristic requires about 80 proof instances, i. e. 80 shadow mixes, as mentioned in [Hen12]

2.3.6.3. Randomized Partial Checking

A more efficient, but less sound and more information-leaking proof is *randomized partial checking (RPC)*. To prove a shuffle $C' := \mathit{shuffle}(C, \pi)$ of a list of ciphertexts $C = [c_1, \ldots, c_n]$, the prover only creates one shadow mix $C'' := \mathit{shuffle}(C, \pi_L) =: (c''_1, \ldots, c''_n)$ with a corresponding permutation pair π_L, π_R as above. He sends C'' to the verifier. The verifier now creates n challenge bits $B_1, \ldots, b_n$ and sends them to the prover. For each challenge bit b_i, the prover does the following:

- If $b_i = 0$, the prover opens the connection between c''_i and $c_{\pi_l^{-1}(i)}$, i. e. he opens $\pi_L(i)$ and the reencryption randomness used for $c'_i = \mathit{Reenc}(c_{\pi_l^{-1}(i)})$.
- If $b_i = 1$, the prover opens the connection between c''_i and $c'_{\pi_R(i)}$ by opening $\pi_R(i)$ and the reencryption randomness that leads to $c'_{\pi_R(i)} = \mathit{Reenc}(c''_i)$.

Pros and Cons

Randomized partial checking is a rather efficient proof of a shuffle since it requires only one proof instance. However, it has several flaws both concerning soundness and information leakage, some of them were presented in [KW13]. A flaw when using only one mix server is quite obvious: of each cipher c''_i in the list C'', *either* the connection to a $c_{\pi_l^{-1}(i)}$ in C is opened, *or* the connection to a $c'_{\pi_R(i)}$ in C', but never both. So if for a c_i, the connection to a c'' in C'' is opened, we know that this is not any of the c'_j in C' for which a connection between c'' and a c'_j in C' was opened, and vice versa. So for each element in either list, we know for about half of the elements in the other list that these are not a reencryption of this element.

2.3.7. Bilinear Groups and Pairings

A bilinear group is a set of three cyclic groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ of order n (often $n = p$ for a prime p), with generators g_1 of $\mathbb{G}_1$ and g_2 of $\mathbb{G}_2$, for which a function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ exists for which the following holds:

- e is bilinear: for all $a, b \in \mathbb{Z}_n$, all $X \in \mathbb{G}_1$ and all $Y \in \mathbb{G}_2$ we have $e(X^a, Y^b) = e(X, Y)^{ab}$ and
- e is non-degenerate: $e(g_1, g_2)$ generates $\mathbb{G}_T$.
- Group operations, e and membership decision are efficiently computable.

In the literature and here, such a bilinear map e is often called a *pairing*. According to [GPS08], there are three types of such pairings:

- **Type 1:** Symmetric setting: $\mathbb{G}_1 = \mathbb{G}_2$ or an efficiently computable isomorphism exists in both directions.
- **Type 2:** $\mathbb{G}_1 \neq \mathbb{G}_2$, an efficiently computable isomorphism $\psi : \mathbb{G}_1 \to \mathbb{G}_2$ exists, but none in the other direction.
- **Type 3:** $\mathbb{G}_1 \neq \mathbb{G}_2$ and there is no known efficiently computable isomorphism between the groups, in neither direction.

Type 3 pairings are the ones for which the most efficient instantiations exist. In the context of this work, Type 3 pairings are used.

2.3.8. SXDH-Assumption

Ateniese et al. [ACHdM05] argued that in an asymmetric bilinear group setup, the decisional Diffie-Hellman problem is hard in both domain groups. They state this in the following assumption:

Assumption 1 (Symmetric external Diffie-Hellman (SXDH) assumption) *Given a setup $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$ of bilinear groups and a Type 3 pairing e, the decisional Diffie-Hellman (DDH) problem is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$.*

The SXDH assumption is needed in an instantiation of the Groth-Sahai proof system, which we will use in our revoting scheme.

2.3.9. The Groth-Sahai Proof System

Groth and Sahai introduced in [GS08] a way to construct efficient non-interactive witness-indistinguishable (NIWI) and zero-knowledge (NIZK) proofs of satisfiability of a set of equations. We will call these proofs *GS-proofs* from now on. An advantage of GS-proofs is that they are non-interactive without relying on the random oracle model and its drawbacks [CGH04]. Instead, they depend on the CRS model and one cryptographic assumption. Groth and Sahai introduce three instantiations of their proof system, depending on different cryptographic assumptions. Of these, the instantiation based on the SXDH assumption leads to the most efficient proofs [EGW09]. We use this instantiation in our work, since we work with asymmetric (Type 3) pairings.

Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be cyclic groups of prime order p, and let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear map. Let G and H be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. In this work, we use Groth-Sahai proofs for the following two equation types, with secret variables $X_1, \ldots, X_m \in \mathbb{G}_1$, $Y_1, \ldots, Y_n \in \mathbb{G}_2$ and $x_1, \ldots, x_m \in \mathbb{Z}_p$. All other variables, namely $A_i \in \mathbb{G}_1$, $B_j \in \mathbb{G}_2$, $c_{i,j}, a_i \in \mathbb{Z}_p$, $T_2 \in \mathbb{G}_2$ and $T \in \mathbb{G}_T$ for all $i = 1, \ldots, n$ and $j = 1, \ldots, m$ are publicly known constants.

- **Pairing product equations:**

$$\prod_{i=1}^{n} e(A_i, Y_i) \prod_{j=1}^{m} e(X_j, B_j) \prod_{i=1}^{m} \prod_{j=1}^{n} e(X_i, Y_j)^{c_{i,j}} = T$$

- **Multi-scalar multiplication equations in $\mathbb{G}_2$:**

$$\prod_{i=1}^{n} Y_i^{a_i} \prod_{j=1}^{m} B_j^{x_j} \prod_{i=1}^{m} \prod_{j=1}^{n} Y_i^{c_{i,j} x_j} = T_2$$

GS-proofs are NIWI-proofs, but in most cases NIZK-proofs can be constructed out of them. For the pairing product equations, this possibility is restricted to the case that $T = 1$ or T can be written as a pairing product. This will be the case for their application in this work.

For a full explanation of how the proof system works, we refer the reader to [GS08] or [EGW09]. But we shortly describe their intuition here, as indicated in the original paper.

Groth and Sahai define commitment functions for each of the groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ and $\mathbb{Z}_p$. These groups can be interpreted as $\mathbb{Z}_p$-modules. The commitment functions map to other $\mathbb{Z}_p$-modules, which have a similar structure, such that the equations can also be expressed in the target groups of the commitments. To create GS-proofs for a set of equations, the prover commits to all secret variables and then has to prove that the committed values satisfy the equations. The commitments are constructed in a way that their committed values *almost* satisfy the corresponding equations in the target groups. To fully satisfy the equations, additional terms are needed that depend on the witness variables and on randomness used for the commitments. The GS-proofs then consist of all commitments to witness variables and information from which these additional terms can be computed.

The commitments used in the GS-proofs can be set up in either a *binding* setting or a *hiding* setting. In the binding setting, the GS-proofs are perfectly sound. In the hiding setting, the GS-proofs are perfectly witness indistinguishable/perfectly zero-knowledge. The setting is given by the CRS, which consists of the group setup $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$ and commitment public keys which determine the setting (binding or hiding). The security of the proof system depends on the indistinguishability of the two settings.

In the SXDH instantiation, i. e. with the indistinguishability of the hiding and the binding setting depending on the SXDH assumption, these commitments in the binding setting are basically Elgamal encryptions, so they allow witness extractability, and we can use them as proofs of knowledge of a witness. We use this fact in our voting scheme to let the voter prove knowledge of a signature created with her signature key, while revealing neither the signature nor her public key, and with that, her identity.

2.3.10. Digital Signatures

Intuitively, a signature scheme S defines a function $Sign$, with which messages from a *message space* $\mathcal{M}$ can be digitally signed, to obtain a digital signature σ of the message $m \in \mathcal{M}$ under the *signing key* sk. The signature σ can be verified with a *verification key* vk.

More formally defined, a digital signature scheme

$$S = (Setup, KeyGen, Sign, Verify)$$

consists of four efficiently computable algorithms $Setup, KeyGen, Sign$, and $Verify$, which are defined as follows:

- $Setup(k) \mapsto (gs, param)$ is a randomized algorithm which takes as input a security parameter k and outputs a group setup gs, in which signatures can be created. The group setup gs defines a *message space* $\mathcal{M}$ and a *key space* $\mathcal{K}$. It might also output additional parameters which are used by the other algorithms.

- $KeyGen(gs, k) \mapsto (sk, pk)$ is a randomized algorithm which takes as input a group setup gs a security parameter k and outputs a keypair (sk, pk), whereas sk is referred to as a *signing key*, and pk is called the *verification key*.
- $Sign(sk, m) \mapsto \sigma$ takes a signing key sk and a message $m \in \mathcal{M}$ as input, and outputs a *signature* σ.
- $Verify(pk, m, \sigma) \mapsto b$ takes as input a verification key pk, a message m and a signature σ, and outputs a bit b. If $Verify(pk, m, \sigma) = 1$, the signature σ is called a *valid* signature of m under the verification key pk. If $Verify(pk, m, \sigma) = 0$, σ is called an *invalid* signature of m under the verification key pk.

Each algorithm might take additional input. The signing key sk is usually kept secret and therefore often referred to as a *secret key*, while vk is usually public and referred to as a *public key*.

2.3.11. Automorphic Structure-Preserving Signatures

Automorphic structure-preserving signatures are signatures in which verification keys, messages and signatures are group elements, and where signature verification consists of checking a set of pairing equations. Automorphic means that the signature scheme can sign its own public keys. An application of this is that the signature scheme can be used to create a certificate for its own public keys. Research on structure-preserving signatures in general was started in [Gro06]. In [AFG+10], Abe et al. developed automorphic structure preserving signatures which are compatible with the Groth Sahai proof system.

We use the signature scheme introduced in [AFG+10] in our revoting solution introduced in Section 5.3, and describe it here. It consists of four algorithms *Setup*, *GeyGen*, *Sign* and *Verify* which are defined as follows:

- $Setup(k)$ takes as input a security parameter k and chooses a triple of bilinear cyclic groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of prime order $p > 2^k$, generators G of $\mathbb{G}_1$ and H of $\mathbb{G}_2$, and a bilinear map and

  $$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$$

  with $\langle e(G, H) \rangle = \mathbb{G}_T$. The message space of the scheme is

  $$\mathcal{M} = \{(G^i, H^i) | i \in \mathbb{Z}_p\}.$$

  Three randomly chosen public parameters $F, K, T \in \mathbb{G}_1$ are chosen. *Setup* outputs the group setup

  $$gs = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, G, H)$$

  and the public parameters (F, K, T).

- $KeyGen(gs, k)$ takes as input the group setup gs and a security parameter k. It chooses $x \in \mathbb{Z}_p$ at random, and computes

  $$pk := (X, Y) := (G^x, H^x) \in \mathbb{G}_1 \times \mathbb{G}_2.$$

  It outputs the key pair (x, pk), consisting of the signing key x and the verification key $pk = (X, Y)$.

- $Sign(x, m)$ takes as input a secret key $x \in \mathbb{Z}_p$ and a message m of the form

  $$m = (M_1, M_2) = (G^a, H^a)$$

  for an $a \in \mathbb{Z}_p$. To sign this message with the signing key x, two random values $r, c \in \mathbb{Z}_p$ are chosen, and as output the signature σ is computed as

  $$\sigma = (A, C, D, R, S),$$

  where $A := (KT^r M_1)^{\frac{1}{x+c}}$, $C := F^c$, $D := H^c$, $R := G^r$, and $S := H^r$.

- $Verify(pk, m, \sigma)$ verifies if $\sigma = (A, C, D, R, S)$ is a valid signature of the message $m = (M_1, M_2)$ with verification key $pk = (G^x, H^x)$, by checking the following conjunction of pairing equations:

  $$e(A, YD) = e(KM_1, H)e(T, S) \wedge e(C, H) = e(F, D) \wedge e(R, H) = e(G, S).$$

  It outputs 1 if the above term is true, 0 otherwise.

As stated in [AFG+10], other messages which are not in the message space can be signed by using a cryptographic hash function h to map them to an $a \in \mathbb{Z}_p$ and transform them into a signable message $m = (G^a, H^a)$.

Compatibility with Groth-Sahai proofs

The signature scheme works with asymmetric (Type 3) pairings, for which the SXDH assumption is believed to hold. Since the verification consists of checking pairing product equations, the scheme is compatible with the SXDH instantiation of the Groth-Sahai proof system. This makes it easy to create NIZK-proofs of knowledge of a signature σ for a message (M_1, M_2) with $Verify(pk, (M_1, M_2), \sigma) = 1$ for a verification key pk, where each of the components σ, pk and (M_1, M_2) can be witness variables as needed. This fact is used by the authors of [AFG+10] to construct a blind signature out of this scheme. It is also used by Ghadafi in [Gha11] to hide the public key and make a group signature out of the blind signature version of the scheme. Inspired by Ghadafi's group signature idea, we use this property to hide the signature as well as the voter's public key to make it publicly verifiable that a vote has been cast with a valid signature key of an eligible voter, without revealing the voter's identity.

3. Requirements of Cryptographic Voting Schemes

During the last 30 years, a huge amount of conceptually very different cryptographic voting schemes has been developed. As a first contribution of this chapter, we introduce a taxonomy for cryptographic voting schemes together with a roadmap for analyzing voting schemes. The aim of the taxonomy is to provide a tool to consistently analyze and compare this diversity of voting schemes.

Apart from the taxonomy, this work mainly concentrates on coercion-resistance. There are several definitions of coercion-resistance, based on different underlying models. The second contribution of this chapter is a brief review of existing definitions of coercion-resistance, with respect to two of our special cases, namely write-in candidates and revoting.

3.1. Related Work

Our taxonomy builds on a list of requirements that was published in [CvdGRV07].

Another review of privacy-type definitions for voting schemes was presented in 2010 by Lucie Langer in her Ph.D. thesis [Lan10]. She did her review from another point of view, concentrated on privacy and did not take into account our cases of revoting and write-in candidates. In her thesis, Langer introduced a taxonomy for privacy and verifiability in electronic voting schemes. Her taxonomy is quite different from ours and constitutes a linkability/unlinkability model, where privacy is defined via the unlinkability of plaintext ballot and voter, and verifiability is defined as the linkability of the set of cast votes with the set of counted votes.

Requirements of electronic voting schemes have been studied in several publications. A survey of electronic voting schemes and their properties was presented in [FDL]. Chevallier-Mames et al. [CMFP+10] analyzed under which assumption certain security properties of voting schemes can be achieved simultaneously. A Common Criteria protection profile [fSidI07] with basic requirements for remote electronic voting was proposed by the German Bundesamt für Sicherheit in der Informationstechnik (BSI). The requirements for the German governmental election can be found in the German Federal Electoral Regulations [dJ13].

Jeremy Clark used a game-theoretic approach in his Ph.D. thesis [Cla11] to analyze coercion and vote-buying possibilities in several election schemes.

Küsters et al. [KTV12] introduced a rather general formal model for voting schemes as well as a rather intuitive and generally applicable formal definition of coercion-resistance. They compared several definitions of coercion-resistance to their own work. Most of these definitions also appear in our review.

In 2013, Jonker et al. [JMP13] made an interesting survey about developments and trends of notions of privacy and verifiability, as well as techniques used in voting schemes to achieve these notions.

3.2. A Taxonomy for Cryptographic Voting Schemes

There is a huge variety of existing voting schemes, for example [CCC+08, PH06, Gjø10, JCJ05, CGS97, FOO93, BMQR07] and many more, each different in focus and conception. To the best of our knowledge, a list of formal definitions of requirements with which existing and newly developed voting schemes could consistently be analyzed and compared is yet missing.

As a step towards solving this problem, we introduce a taxonomy, in natural language, which provides:

1. A list of requirements we have towards voting schemes.

2. A roadmap with which a big variety of voting schemes can be analyzed according to these requirements.

The taxonomy is joint work with Jeroen van de Graaf, Dirk Achenbach and Bernhard Löwe [vdGKAL14]. An earlier version of the list of requirements used in our taxonomy was published in [CvdGRV07], but was not elaborated. We have refined these requirements and identified questions, between 1 and 6 for each requirement, to be able to easily find out in which way and to which extent an arbitrary voting scheme meets these requirements. These questions build the heart of our analysis roadmap. Our taxonomy has then been tested against several known voting schemes and developed further.

In the remainder of this section, we introduce our taxonomy and our analysis roadmap. At the end of this section, to demonstrate the capability of our taxonomy, we summarize the analysis of several voting schemes. As examples for a full analysis, the analysis of the non-cryptographic but generally accepted German paper election can be found in Appendix A, the cryptographic voting schemes Prêt à Voter and Scantegrity II are analyzed in Appendix D and E, respectively. An analysis of an election held with Bingo Voting at the University of Karlsruhe is presented in Appendix B and summarized in Section 4.3. The Analysis of our revoting scheme can be found in Appendix C.

3.2.1. What is an Election?

An election is a process in which a predetermined set of participants, called *eligible voters*, is allowed to take part. A subset thereof consists of the *voters*, which participate in an *election protocol*, by giving a *ballot* as input into a predetermined *election functionality*, which computes an outcome, called the *tally*, as a function of these ballots. The tally is thought of as a joint decision of the voters. Each voter inputs exactly one ballot, by default exactly once.

To avoid the ambiguity of the word *vote*, we use the following vocabulary: a voter has *n votes* which she can distribute among several *candidates* on a *ballot*. The distribution of her votes among candidates is called the voter's *choice*. If the voter

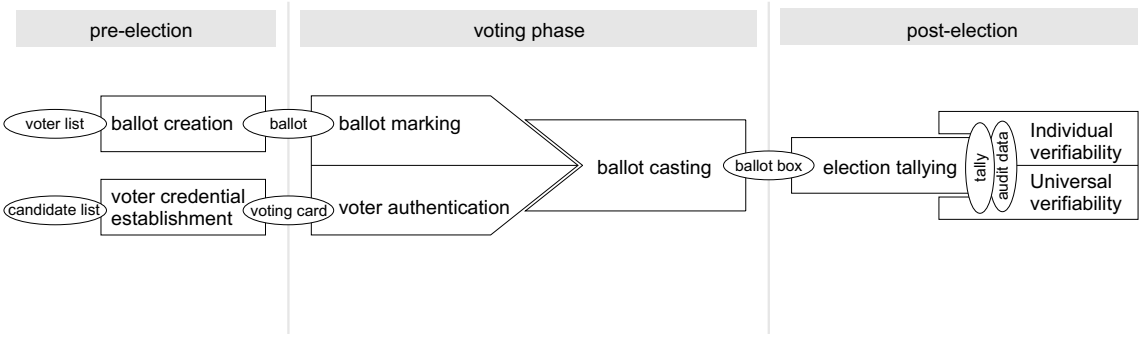

Figure 3.1.: Visualization of a generic voting procedure. Image source: [vdGKAL14]

votes, she *casts* a *ballot* which will be transfered to the *ballot box*. If she *votes for a candidate*, she marks this candidate on the ballot she casts. The winning candidate of an election is *elected*.

Following the vocabulary of the widely known paper election, we speak of *ballots* and *ballot boxes* even in electronic elections, where the choices recorded on the ballots might be stored in an arbitrary form. A ballot can therefore be a paper ballot or electronically processed data which represents the voter's choices. Likewise, a ballot box can refer to a physical ballot box filled with paper ballots, or a digital storage device that contains data from which the tally can be computed.

An election can be roughly divided into three phases: the *pre-election*, the *voting phase* itself and the *post-election*. During the pre-election, the voting phase is prepared, for example ballot boxes are set up, or the voter list is compiled. During the voting phase, voters mark their choices on ballots and cast them. The tallying and auditing are performed in the post-election.

We visualize this procedure in Figure 3.1. Depending on the voting scheme, more steps are necessary. As an example we list the important steps of a conventional paper election, as they are carried out in most democratic countries.

3.2.2. Process of a Paper Election

In this Section, we describe the process of a conventional paper election. This description is an extended version of a description by Chaum et al. [CvdGRV07], which is also the foundation of this taxonomy. The election takes place in a *polling station*, which is equipped with a *voting booth* where voters can cast their ballots in private, a ballot box, and a registration desk.

3.2.2.1. Pre-election

1. A list with the names of all legitimate voters is established, the *voter list*.
2. A candidate list is established.
3. The (empty) ballots are printed.
4. An empty *ballot box* is set up. Before starting the election phase, all present observe that the ballot box is empty.

3.2.2.2. Voting phase

5. Voters present themselves at a registration desk and prove their identity.

6. A legitimate voter (that is, a person whose name (identity) is on the voter list and who has not already voted) receives a fresh *ballot*, enters the voting booth, and fills in her *choice*. Her name (identity) is marked on the voter list, so that she cannot vote again.

7. The voter *casts* her ballot in the ballot box. (From that moment on, the ballot is cast, and she cannot undo or modify her vote.)

8. When the time for voting has expired, no ballots can be cast any more.

3.2.2.3. Post-election

9. Votes are tallied by opening the ballot box and publicly counting the votes. All present observe the opening of the ballot box and tallying of the contained ballots.

10. Those who disagree with the count can request a recount. The votes are recounted (maybe more than once), under the observation of all present, until there is consensus.

11. The *outcome* of the election is published.

12. Ballots and documentation are archived or destroyed.

Conventional elections are usually carried out in democracies for which one single ballot box or one polling stations does not suffice. Therefore, electoral districts are set up containing several polling stations, and the election results of each district are accumulated to obtain the joint election result.

3.2.3. Requirements of an Election Scheme

In this section, we introduce a set of requirements which are desirable for voting schemes. These requirements were motivated by the traditional paper election, which is generally accepted by a broad range of voters. The requirements were then amended in the process of analyzing several cryptographic voting schemes. To motivate the requirements, we state a justification for each of them, and explain how it is implemented in the traditional paper election. Some requirements might not depend on a voting scheme itself but on its implementation in practice; they help find out which requirements the scheme has towards its environment. In fact, it is advisable that descriptions of new voting schemes explicitly state assumptions about their operational environment.

In the following, some requirements refer to an *election*. These are requirements towards an election conducted with the analyzed voting scheme and state that a voting scheme should enable these requirements, or at least not make them unsatisfiable.

Requirement A (Eligibility) *Only persons on the voter list specified by the* Registration Authority*, called* voters*, can cast votes.*

Justification: Any election has a finite set of persons who have the right to participate in that election and only these persons are allowed to cast a vote. We will call the entity which is responsible for maintaining this list the Registration Authority.

How implemented traditionally: In very small paper ballot elections this set of persons can simply be the people present in the room, no actual list is made. In a larger election usually a list is maintained which contains the names of all eligible voters. The voter identifies herself to the poll worker and is granted access to the voting system, which from then on is supposed to provide her anonymity.

Requirement B (Equality) *Each eligible voter can cast a vote, from now on called a* ballot*. Each eligible voter can cast at most one ballot.*

Justification: Each eligible voter has only one vote. Some elections are exceptions to this requirement; for instance, in share holder elections the voting power can be proportional to the amount of shares a person holds. However, these exception are fairly rare and will not be considered in this taxonomy.

How implemented traditionally: In traditional voting this is enforced by handing each voter only one ballot, and marking in a voter roll which voter has already cast a ballot.

Revoting: In some election schemes, the voter is allowed to overwrite a cast ballot with a new ballot. We call this *revoting*. If revoting is allowed, the second sentence of this requirement is to be rephrased as "Of each eligible voter who has cast at least one ballot, only the last cast ballot is included in the tally."

Requirement C (Layout neutrality) *The way the choices are presented to the voters is unambiguously defined by a* Ballot Creation Authority (BCA)*, does not favor any choice beyond what is specified by the law, and is equal for each voter.*

Justification: On the one hand, this requirement is related to equality: each eligible voter should have the same possibilities to cast her choice. The voting process should not depend on the voter's race, gender or political view, i. e. not be made more difficult for some voters and easier for others. On the other hand, Requirement C requires fairness towards the candidates: no candidate should unjustly be disadvantaged. In some elections, the order of candidates on the ballot is specified by law and should be the same on each ballot. In voting schemes that present the choices in random order, care has to be taken that the permutation is chosen uniformly at random and does not put certain candidates more often on top than others.

How implemented traditionally: In simple, informal elections the voter writes her choice on a dedicated space on the ballot by writing a name or number. However, this can be a source of various problems: ambiguity of the preference expressed, loss of ballot privacy, intentional ballot marking, etc. This can be avoided by the creation of preformatted ballots which list all possible options, each accompanied by some identical symbol (like a circle or box). In the German paper election, the order in which choices are presented to the voter is specified by law and equal for each voter. This can be checked by auditing ballots.

Requirement D (Revisable ballot marking) *The voter indicates her preferences which are recorded in some unambiguous representation, called a* ballot*. The voter can verify the validity of her ballot and can revise her preferences before casting the ballot.*

Justification: After marking her choice on a ballot but before actually casting the ballot, the voter should have the right and the possibility to check her choice is

marked (and when voting electronically, recorded) as intended and that it is valid. She should have the opportunity to correct or revise her ballot.

How implemented traditionally: The voter uses a pen or pencil to write down her preference or mark it in the way specified. The voter can visually verify that her ballot is filled out correctly, and that it is valid. In case she made a mistake, she can return her paper ballot to the authority in exchange for an unmarked ballot. She gets no (technical) support for checking the validity of her ballot.

Requirement E (Irreversible ballot casting) *When a voter, in agreement with Requirements A and B, submits a ballot to be included in the tally, this is called* casting *a ballot. The act of casting makes the ballot legitimate, and is irreversible.*

Justification: Adherent to the previous requirement, the voter can fill out and revise her ballot several times. This calls for a mechanism for the voter to determine by herself when her ballot is filled out as intended and ready to be cast. With casting, the voter accepts her ballot as a legitimate representation of her choice that can be counted in the tally.

How implemented traditionally: In the traditional paper election, the act of casting a ballot is performed by putting the ballot into the ballot box. Ideally we have a ballot box which has an opening just large enough to let a paper ballot pass. but making it virtually impossible to retrieve a ballot from it.

Revoting: If revoting is allowed, the last sentence of this requirement is to be rephrased to "The act of casting makes the ballot legitimate, but can be countermanded by the voter through casting a new ballot within a given time, called the *voting period*".

If revoting is allowed, the voter's choice can be updated by a new ballot. There is an important difference to the revision in the ballot marking process: there, the ballot is not yet legitimate. With casting, the ballot becomes legitimate. So revoting countermands a legitimate ballot that would otherwise be counted.

Requirement F (Privacy and incoercibility) *Under no circumstance, not even with the cooperation of the voter, is it possible to link the preferences as recorded on a legitimately cast ballot to the identity of the voter, nor can the voter be coerced to deviate from her choice in any provable way.*

Justification: It is important to realize that this requirement has two sides. First, the voter should have the freedom to express her will without the risk of repercussion. To guarantee this, nobody should be able to discover for whom or what she voted or did not vote.

Second, it is necessary to prevent improper influence of voters, which includes the buying and selling of votes. Consequently it should not be possible, *even with the cooperation or connivance of the voter*, to deduce the vote. For this reason it is of utmost importance that, during the marking and casting of the ballot, no proof or receipt is created which could be linked to a ballot inside the ballot box, since this would permit coercion and the buying and selling of votes.

How implemented traditionally: In order to guarantee ballot secrecy, there exists a private space, the voting booth, where the voter can fill out the ballot. To prevent a voter from linking her ballot to her ID in order to sell her vote, ballots that are marked in a way other than specified are counted as invalid.

Requirement G (Secrecy of intermediate results) *Until the end of the vote casting process (as defined by a* Ballot Casting Authority*) no information can leak about the choices marked on any of the ballots cast already.*

Justification: Revealing partial results early would violate the secrecy of the ballot for those who voted already. Second, knowing the partial result might influence the choice of someone who votes later. Moreover, exclusive access to this information during the voting period could provide advantage in terms of allocation of electioneering resources or even trigger disruption of the voting process.

In fact, any information leaking from the ballot box is undesirable. Ideally, in an election the voters should express their opinion simultaneously. It is only for logistical reasons we make it into a sequential process. So it would be unfair if voters who vote last would know already the content of the ballot box, since they could change their behavior based on additional information, to which the early voters had no access.

How implemented traditionally: In large elections with several ballot boxes in different locations, the announcement of the partial results is coordinated with the end of the voting periods in each polling station, so that no intermediary results become available before all ballot boxes are closed. Also, it is generally prohibited to release any projections or estimates to the public before the last ballot box is closed.

Requirement H (Inviolatability of the ballot box) *It is not possible to modify the set of ballots cast, other than by adding ballots legitimately cast according to Requirements A–F.*

Justification: It is obvious that any possibility to modify the contents of the ballot box would alter the tally.

How implemented traditionally: This requirement explains why the ballot box should remain in a publicly visible place so that everybody present can see no tampering takes place. The ballot box is not opened until the ballots are counted. The counting itself is performed semi-publicly, i. e. all people present observe that no ballots are removed or modified.

Requirement I (Tally integrity) *All ballots contained in the ballot box, and only these, will be included in the tally.*

Justification: Votes not written on a proper ballot, for example, should not be counted as they could represent multiple votes from a single voter. Additionally, ballots that are ambiguous ought not to be counted.

How implemented traditionally: The Tallying Authority makes sure to only regard counts from valid ballot boxes. When counting the ballots from a single ballot box, all people present observe that only the ballots in the box are counted.

Requirement J (Individual verifiability) *Any voter can convince herself that her ballot is included in the tally.*

Justification: We would like to be able to hand a proof/receipt to the voter, allowing her to verify that her ballot is among the set of ballots tallied. However,

conventional wisdom has it that this requirement contradicts the more important requirement of ballot secrecy.

There is a tension between the requirement of verifiability (and auditability, see below) on the one hand, and the secrecy of the ballot on the other, which makes the design of election systems satisfying both requirements without using ballots or other physical objects extremely challenging. Implicit in each election is a sub-procedure that shuffles ballots to destroy the link between the ballot and the voter and protect her anonymity. This procedure, trivial when dealing with physical objects such as ballots or playing cards, is difficult to simulate in the virtual world. The problem is made more difficult by the verifiability requirement: it should be possible to ensure that the virtual shuffle did not add, subtract or replace any of the shuffled items.

How implemented traditionally: In the case of paper ballots, this requirement is achieved in the following way: after the voter has cast her ballot by depositing the ballot in the ballot box, she waits until the closing of the election. When the ballot box is opened for the tallying of the ballots, she is sure that her ballot is among the set, even if she does not know which particular ballot corresponds to the one she filled in. It is interesting to note that, essentially, the voter's faith is based on the common sense notion that an object put in some place stays there and will not disappear by itself, which can be assumed for physical objects, but not for data stored electronically.

Requirement K (Auditability and public verifiability) *Any interested party can verify that the result, published by a* Ballot Tallying Authority (BTA)*, is correct.*

Justification: For higher credibility of the result it is important that voters, party representatives and neutral observers are able to verify the process.

How implemented traditionally: In traditional elections the tallying of the ballots happens in a public session: a person (or group of persons) designated for this task opens the ballot box, takes out all the ballots (making sure none is left inside) and counts the votes. He does so in the presence of observers and in a way to make it easy to these observers to see that the process is executed correctly.

In addition, any person can contest the result and request a recount of the ballots, which also happens in a public session. In principle this process should converge to a result with which everybody agrees.

Requirement L (Robustness) *The voting scheme must be robust towards invalid or malformed data that is submitted intentionally or unintentionally by voters.*

Justification: In traditional paper elections, the presence of invalid ballots is quite common. Especially in complex elections, the casting of invalid or malformed data can easily happen unintentionally, and should therefore not violate the election process in any way. Moreover, everything that can happen unintentionally can also be triggered by the adversary to gain advantage. Therefore it is important to ensure that malformed data cannot abort or invalidate an election, or lead to an illegitimate tally result undetected. An example where this is important is homomorphic tallying, where an invalid ballot could encode $-x$ votes for a candidate for some x, so this ballot would subtract votes from a candidate.

There are two standard ways to meet this requirement: either the validity of a ballot is proven and checked before the ballot is cast, this is important in electronic

elections with homomorphic tallying, as the example above shows. The other possibility is filtering out invalid ballots before or during the tally process, so they are not counted. This can be done in elections where the identity of the voter is unlinked from the ballot and the ballot published in plaintext, like traditional paper elections or voting schemes using mixnets.

How implemented traditionally: In traditional election schemes, each ballot is processed in plaintext (without a link to the voter who cast it), so invalid ballots can be filtered out easily. Ballots that contain malformed data (as defined by rules) are discarded. The error cannot be corrected by voters who have unintentionally cast invalid ballots, but the tally does not have to be aborted and redone because of invalid ballots, and an invalid ballot does not lead to an invalid ballot outcome.

Requirement M (Availability) *The election must be available, i. e. cope with the unavailability of components.*

Justification: This requirement is important for two reasons: for achieving equality, each voter should have access to the voting process. The voting period should be estimated long enough that every voter has a chance to cast a ballot, and the components necessary for the voting process should be available until a predefined end of the voting period. In electronic voting, the capacity of voting machines or vote servers should be high enough to handle the expected number of voters. The other reason is that the breakdown of single components should not enforce a repetition of the whole election. This would invalidate already cast ballots and in many cases change the tally outcome.

How implemented traditionally: In traditional paper elections, each component is a physical object and can be substituted. It has to be made sure before election day that enough poll workers are there to help, that enough ballots and ballot boxes etc. are available, and that the length of the voting period and the number of polling stations is estimated such that each voter has a chance to cast a ballot.

Requirement N (Scalability) *The voting scheme should scale well. That is, it should support an arbitrary number of voters without compromising any of the other requirements.*

Justification: There are many different kinds of elections, ranging from an amount of two to several millions of eligible voters. When judging the scalability of a voting scheme one has to take into account the amount of components needed, the amount of published data needed for proofs of correctness as well as the time that is needed for casting, counting and verification. There are voting schemes that work perfectly well for elections with a few hundred voters but are infeasible to apply to a bigger election with millions of voters. Therefore, before applying an election scheme to an election with a big (or too small) amount voters, one should be aware of how well it scales.

How implemented traditionally: The easiest way to achieve scalability is to use multiple electoral constituencies and to have several instances of an election. The end of the voting phases and the tally procedure is then synchronized and the results are added up and published. Another solution would be to have one central counting process, but this might be hard to achieve without being very error prone and inefficient.

3.2.4. A Roadmap for Analyzing Elections

In this Section, we introduce a roadmap in the form of some questions about the voting scheme, along which voting schemes can be analyzed. The analysis is divided into three parts. The first part consists of questions about general properties of the voting scheme. In the second part, we evaluate if and to which extent the requirements specified above are met. The third part is about a short conclusion of the analysis and asks which requirements are met or could be met with minor adaptions.

3.2.4.1. General Information about the Voting Scheme

To evaluate for which kinds of election a voting scheme is designed, what assumptions it depends on and what underlying concepts it uses, our analysis starts with some general questions about the voting scheme:

Election type

The first two questions are about the type of the election scheme:

(Q1) Is the election scheme ...

- ... paper based, scanner based or computer based?
- ... meant for presential or internet elections?

(Q2) For which kind of elections is the scheme designed? (E.g. governmental elections, non-political elections, etc.)

(Q3) Does the election scheme allow any special election types? (E.g. vote-splitting, write-in candidates, vote delegation etc.)

Question (Q2) is about the intended use of an election scheme. Some elections, like the scheme introduced in [Gjø10], are designed for and used in governmental elections, while others, for example Helios [Adi08], are explicitly designed for elections with a low risk of coercion. Question (Q2) asks about "special features" of the analyzed election scheme. Does it require a special ballot layout which only allows to choose 1 out of n candidates, or is it more flexible?

Preliminaries and Assumptions

The next questions ask about preliminaries and underlying assumptions required by the voting scheme:

(Q4) On what cryptographic assumptions is the scheme based?

(Q5) If the scheme relies on trusted parties, which parties need to be trusted?

(Q6) Which components have to be trusted? Does the scheme rely on trusted hardware?

(Q7) Are there other assumptions? (E.g. the existence of a voting booth, at least one voting process done in private etc.)

What makes this scheme secure?

(Q8) What is the main attribute that leads to the security of the analyzed voting scheme?

Question (Q8) asks for a brief description of the most important feature or underlying concept that is responsible for the security of the scheme. For example, does the scheme use anonymous credentials to unlink the ballot from the voter's identity, as in the voting scheme of Juels et al. [JCJ05]? Or does the security rely on a trusted random number generator, as in Bingo Voting [BMQR07]?

3.2.4.2. Analysis of the Requirements

The next part analyzes to which extent the requirements defined in our taxonomy are met. For the analysis, we derive between 1 and 6 questions for each requirement, which are listed in the following:

Requirement A (*Eligibility*)

(A1) How does the system identify whether persons presenting themselves to vote are eligible?

Requirement B (*Equality*)

(B1) How does the system ensure that each eligible voter can cast at most one ballot?

(B2) How is it made sure that each eligible voter has the possibility to cast her vote?

Requirement C (*Layout neutrality*)

(C1) How are choices presented to the voter?

(C2) How is it made sure that no option is favored more than specified by election rules?

(C3) How is it made sure that choices are presented fairly to the voters?

When answering question (C1), an interesting point would be if and how it is made sure that the way choices are presented to the voter does not depend on the voter's identity. Question (C2) asks for fairness towards the candidates, while Question (C3) asks for fairness towards the voters. Question (C3) could also be thought of as how is it made sure that each voter gets the same ballot or sees the same user interface, often referred to as the *vote casting interface*. On the other hand, Questions (C2) and (C3) are particularly important if the ballot or vote casting interface displays the candidates in random order. In that case, it is interesting to know how it is made sure that the distribution of the displayed order is really random and does not put one candidate more often to the top than others.

Requirement D (*Revisable ballot marking*)

(D1) How does the voter mark her choice?

(D2) How can the voter verify that the ballot she just created contains a faithful representation of her preferences? Under what assumptions?

(D3) How is it made sure that the voter can revise her ballot?

Requirement E (*Irreversible ballot casting*)

(E1) How does the voter cast her ballot?

(E2) What defines when a ballot is cast?

(E3) What makes it irreversible? Under what assumptions is it irreversible?

(E4) What defines the ballot box?

Requirement F (*Privacy and incoercibility*)

(F1) How is individual privacy assured? Under which assumptions?

(F2) How is individual privacy assured in the future? And under which assumptions? (Is it computational? Unconditional? Why?)

(F3) How is receipt-freeness assured? And under which assumptions?

(F4) How is coercion-resistance assured? And under which assumptions? (I. e. how is it ensured that a ballot/voter's choice cannot be marked in a way that shows to the adversary that the voter has been successfully coerced?)

(F5) Who learns the vote or is able to calculate it?

(F6) Who gets critical information? What information is this and in which way is it critical?

Requirement G (*Secrecy of intermediate results*)

(G1) How does the voting scheme guarantee that no information about the contents of the ballot box leaks before the tallying procedure is completed? Under which assumptions?

Requirement H (*Inviolatability of the ballot box*)

(H1) How does the system make sure that no ballots are modified? Under which assumptions?

(H2) How does the system make sure that no ballots are removed? Under which assumptions?

(H3) How does the system make sure that no ballots are added?

Requirement I (*Tally integrity*)

(I1) How is the tallying of the votes implemented?

(I2) How is it made sure that only the ballots correctly cast are counted?

(I3) If not correctly cast ballots are counted, who sees it?

(I4) If not correctly cast ballots are counted, who can prove it and how?

(I5) If not correctly cast ballots are counted, what can be done to correct the error?

Requirement J (*Individual verifiability*)

(J1) How can a voter convince herself that her ballot is included in the tally? Under which assumptions?

(J2) What is the underlying principle used? Physical? Statistical/Probabilistic? Trust in the authorities?

(J3) If the above check fails, how can the voter prove that her ballot is not included in the tally?

(J4) How and by whom can this error be corrected?

Requirement K (*Auditability and public verifiability*)

(K1) How can an interested party be convinced that the result published by the BTA is correct?

(K2) What is the underlying principle used? Physical? Mechanical? Electrical? Electronic? Statistical/Probabilistic? Trust in the authorities?

(K3) If the result is not correct, who can prove it and how?

(K4) If the result is not correct, can the error be corrected? If so, by whom and how?

(K5) Is it traceable who/what has caused the error?

Requirement L (*Robustness*)

(L1) How is invalid or malformed voter data treated?

(L2) What happens when a voter aborts the voting process?

Requirement M (*Availability*)

(M1) How does the system handle the unintentional breakdown of its components?

(M2) How can the system made to gracefully deal with denial of service attacks?

Requirement N (*Scalability*)

(N1) How well does the scheme scale? How big can an electoral constituency be?

3.2.4.3. Conclusion of the Analysis

At the end, there should be a small summary of which requirements are met:

(S1) Which of the requirements listed above are fully met under the underlying assumptions?

(S2) Which requirements are only partly met, and in which way?

(S3) Which requirements could be met with minor improvements of the scheme? What are the suggested improvements?

(S4) Which requirements are not met?

(S5) For which requirements does it depend on a concrete implementation or application if they are met?

Each analysis should close with some concluding remarks:

(S6) Are there any concluding remarks?

3.2.5. Categorizing the Requirements

Analyzing a voting scheme by going through the questions of the roadmap usually leads to several pages of analysis, which can be seen in the Appendix A, B and D, for example. In order to show the benefits and weaknesses of an analyzed voting scheme at a glance, it is often appropriate to summarize the outcome of such an analysis. We have derived five bigger categories from our list of requirements, assigning each requirement to one or two categories. This should make it easy to quickly summarize to which extent requirements related to a category are met, and give an overview over the scheme's benefits and weaknesses.

The first two categories are about the "cryptographic parts" of a voting scheme, *coercion-resistance* and *correctness*. The third category, *fairness*, is about influence on the voter, be it through ballot layout or by leaking intermediate results. The last two categories, *provability of a fraud* and *robustness and scalability* are mainly about the robustness of a scheme when actually applied in practice.

Category 1: Privacy and Coercion-Resistance

This category contains all requirements related to privacy, namely Requirements F (privacy and incoercibility) and G (secrecy of intermediate results). Requirement G belongs here because seeing intermediate results might diminish privacy: the choice of a voter "hides between" less votes. In the extreme case that only one ballot is in the ballot box, leaking an intermediate result would fully break privacy. If k ballots are in the ballot box, the probability of guessing the voter's choice has a lower bound of $\frac{1}{k}$ and increases with pre-knowledge of the adversary of the distribution of these ballots. The adversary might have cast one of these ballots himself, or have knowledge about the contents of some ballots. It is obvious that the chance of correct guessing decreases with the number of ballots in the ballot box.

Category 2: Correctness and Verifiability

Assigned to this category are requirements that are related to the (verifiable) correctness and manipulation resistance of a voting scheme. These requirements are Requirement D (revisable ballot marking), E (irreversible ballot casting) and J (individual verifiability) for individual verifiability, Requirement H (inviolatability of the ballot box) and I (integrity) for integrity and Requirement K (auditability and public verifiability) for universal verifiability.

Category 3: Fairness

Requirements A (eligibility) and B (eEquality make sure that all eligible voters, but only those, can cast the same amount of votes. Requirements C (layout neutrality) and G (secrecy of intermediate results) also have an impact on the fairness of an election. A ballot layout that differs between voters or favors certain candidates leads to an unfair election. Intermediate results influence voters who have not cast a ballot yet. These voters would make their decision with other preconditions than voters who have already cast their ballot – the election would not be fair.

Category 4: Provability of a fraud

This category summarizes which fraud can be detected by whom, and whether these frauds can additionally be proven. For detectable frauds it is also interesting whether errors can be corrected. All requirements related to verifiability, which are those in Category 2, are also related to this category, since a verification might lead to the detection of a manipulation, which rises the question of the provability of a fraud. Questions I4, J3, and K3 ask if attempted or detected frauds can be proven.

This category is closely related to robustness: if a voter can detect but not prove a fraud, she can *in argumentum e contrario* always claim to have detected a fraud, since her wrongness cannot be proven to her. This enables a group of malicious voters who disagree with the tally outcome to spread mistrust in an election and in the worst case enforce a reelection, even if the tally was computed correctly.

Category 5: Robustness and Scalability

This is the category for Requirements which are important for applicability in real world. Analysis outcomes of Requirement L (robustness), M (availability) and N (scalability) the questions I5, J4, and K4 are summarized in this category.

3.2.6. Experiences

To test and refine our Taxonomy, we have analyzed several different voting schemes known to us, presential as well as internet voting schemes, paper-based as well as electronic voting schemes. Among them are the traditional German paper election, Prêt à Voter[CRS05], Bingo Voting[BMQR07], and ScantegrityII[CCC+08]. We exemplarily summarize the analysis of two of these voting schemes here. The full analysis can be found in the appendix.

3.2.6.1. German Paper Election

The German governmental election is usually implemented as a traditional paper election, which takes place in designated polling stations. The election can be observed from beginning to end by everyone who is interested. Eligible voters are informed of the upcoming election several weeks before the election day(s) and get an invitation together with a personalized voting card with which they can prove eligibility at the polling station. Before the voting phase starts the poll workers show to the observers that the ballot box is empty. Then the ballot box is closed and its opening stays covered except when voters put in their ballots. Poll workers maintain a list of eligible voters, a *voter roll*, in which they record which voter has already cast a vote. To cast their vote, voters go to a polling station assigned to them, where they show their identity card and their voting card, are ticked off in the voter roll as present and obtain a ballot. With the ballot, the voter enters a voting booth and marks her choice in private on her ballot, which she then folds to hide her marks. The voter puts her filled-out ballot into a ballot box and is again ticked off in the voter roll to mark that she has cast her ballot. There are several elections run in parallel in different polling stations. At the end of election day, before counting it is made sure that all voters in all other polling stations are finished with casting their ballots, to avoid the leaking of intermediate results before the poll is closed. To count the ballots, they are taken out of the ballot box one at a time and shown to the observers. The content of each ballot box is counted by more than one person, the counting process is repeated until there is a consensus.

The German paper election scheme is rather strong concerning privacy, fairness and verifiable correctness. Provability of a fraud strongly depends on the availability of voluntary election observers. A weak point of the scheme is its lack of redundancy and the fact that its voting processes within one polling station can hardly be parallelized.

The full analysis of this election can be found in Appendix A.

Election type

The voting scheme is paper-based and designed for governmental elections which take place distributed over several polling stations.

Preliminaries and Assumptions

- The poll workers as a group are neutral and trusted and do not mark ballots in an inconspicuous way.
- The ballot box is big enough that the ballots inside are sufficiently shuffled.
- The voter marks her choice in a voting booth which does not contain a camera.
- The election and counting process is observed by sufficiently many witnesses.

What makes this scheme secure?

The main security feature of the scheme is that the whole election and counting process is observable from beginning to end. The observability is supported by the fact that the votes are recorded on physical objects that can not be tampered with without physical access.

Category 1: Privacy and Coercion-Resistance

Voter privacy is ensured by the voting booth. The leakage of intermediate results is prevented as long as the counting procedures of different polling stations are sufficiently synchronized between polling stations. The scheme is not coercion resistant since the voter can be forced to mark her ballot to make it invalid, or not to participate in the election at all. If the voter can mark more than one choice per ballot, the voting scheme is vulnerable to pattern voting attacks.

Category 2: Correctness and Verifiability

Verifiable correctness is ensured by the possibility of observation. A downside of the scheme is that each voter can only observe one polling station, so the election is not exactly universally verifiable. The observation is also very time-consuming.

Category 3: Fairness

The ballot layout is auditable, and the leakage of intermediate results is prevented by observation.

Category 4: Provability of a fraud

Since everything except for the ballot marking process is done in the open, everything can be observed. So provability of a fraud depends on the presence of witnesses.

Category 5: Robustness and Scalability

The limiting factor of the paper election is that voters can only vote sequentially and the ballots are counted by hand, so the whole process takes a lot of time. Since choices are recorded on paper ballots, recounts can be done as often as wished. On the other hand, there is no redundancy in the recording of votes, lost ballots cannot be recovered.

3.2.6.2. Prêt à Voter

This analysis refers to the original scheme Prêt à Voter as introduced in [CRS05], and is done with the purpose of demonstrating and testing the taxonomy, not as an evaluation of Prêt à Voter. The original scheme has some drawbacks which become obvious in the taxonomy. However, it shall be mentioned here that Prêt à Voter has been further developed and improved in both privacy and robustness aspects [RS06, DHvdG+13].

The full analysis is in Appendix D. Prêt à Voter is a paper-based election scheme for presential elections. The voter gets a ballot which has two parts: a right-hand side which displays the candidates in random order, and a left-hand side where the voter can mark her choice. To cast her ballot, the voter makes a mark next to the candidate of her choice, separates the two ballot halves and destroys the part which contains the candidate names. The other half also contains an encryption of the permutation. This part is scanned and recorded and later published on a public bulletin board for verification. The voter can take this part home as a receipt, and check if it is published correctly.

The permutation of each ballot has been created prior to the voting phase by a set of *tellers*, each teller using its own randomness called a *germ* to create its permutation. The permutations are applied subsequently, and ballots are later decrypted by the tellers by sequentially decrypting and applying the permutation to the ballot. This procedure is called an *onion mix*. In newer versions [RS06], a more robust re-encryption mix is used.

Election type

Prêt à Voter is a paper and optical scanner based cryptographic voting scheme designed for presential elections.

Preliminaries and Requirements

- Tallying is performed by a set of tellers which are realized as several hardware devices. Each teller has two key pairs, and creates a permutation of the candidate order for each ballot.

- There is a trusted *election authority (EA)* which generates a random seed of which the tellers calculate their permutations. The EA generates the ballots depending on these permutations.

- The EA generates a *random seed* from which random values, called *germs*, are created for the tellers. The germs are encrypted and hashed to obtain each teller's permutation.

What makes this scheme secure?

- Each ballot has a random candidate order which is invisible but present as an encryption on the voter's receipt.

- The recording device only reads the mark but not the candidate order (only its encryption), so it does not see the voter's choice in plaintext.

- Voters can check their receipts on a public bulletin board.

Trusted Instances/cryptographic assumptions:

- Trusted ballot creators/printers (they see the permutation of each ballot)

- IND-CPA-secure encryption for the permutations

- Encryption acts as binding commitment; Encrypted permutations should not be decryptable to other permutations with some trapdoor.

Category 1: Privacy and Coercion-Resistance

Privacy and receipt-freeness depend strongly on the underlying assumptions, especially the assumption of trusted authorities: each entity which sees the full ballots, i. e. parties who create it, print it, hand it out to voters etc. sees the permutation, and each ballot has a unique identification number which identifies the ballot. Privacy is ensured as long as these parties are trusted. Privacy is not unconditional, it depends on the encryption scheme used for the encrypted permutations that are printed on the ballots. However, Demirel et al. [DHvdG+13] introduced a technique to provide Prêt à Voter with everlasting privacy. Receipt-freeness holds under the given assumptions, but coercion-resistance does not, since the voter can be forced to mark a certain position which can be seen on her receipt. A big advantage of Prêt à Voter is that the device which records the ballots does not learn the voter's choice.

Category 2: Correctness and Verifiability

For individual verifiability, the voter can see on her paper ballot that it is correctly marked and that it is published on the bulletin board correctly. The counting process is publicly verifiable by anyone who is interested, but requires some mathematical background. Verifiable correctness strongly depends on enough voters checking that their ballots are on the bulletin board, or at least the EA not knowing who is going to check their ballot, since not checked ballots can be modified unnoticed.

Category 3: Fairness

Fairness holds under the assumption that the checking of eligibility is performed properly and it is checked that each voter takes part in the voting process only once, and sufficiently many ballots are audited to ensure layout neutrality. For intermediate results to leak, the scanner and either all tellers or the voting authority would have to be corrupted.

Category 4: Provability of a fraud

If a ballot is missing on the bulletin board, the voter can prove this with her receipt. Her ballot can then be included in the tally. It is then unclear though if her ballot is just missing or was substituted by another one. Since the ballots are not signed by the voters, ballot stuffing is only prevented if additional measures are taken, like observability as in the German paper election.

Category 5: Robustness and Scalability

Due to the onion mixing technique, each teller can perform a denial of service attack by refusing to decrypt ballots. Apart from that, similar scalability issues hold as with the paper election: the voting process requires physical presence in a voting booth in a polling station, therefore voters mostly vote sequentially. An advantage of Prêt à Voter is that ballots can be tallied electronically.

3.3. A Review of Definitions of Coercion Resistance

In the literature, coercion-resistance is defined in several different ways, mostly depending on the considered underlying model or type of voting schemes, including symbolic, game-based and UC-based definitions. While some of them are very general, others are very restrictive and only applicable to a small set of voting schemes. In this section, we review these definitions and discuss if they can be applied to our special cases write-in-candidates and revoting. First, we present some general

observations concerning write-in candidates and revoting in Section 3.3.1. In Section 3.3.2, we present the review of definitions of coercion-resistance. There are several definitions that capture the weaker notion of *receipt-freeness*, for example [MN06, BT94, JP06a]. These are not reviewed here.

3.3.1. General remarks

Before we start our review, we discuss some general observations. There are two kinds of information the adversary can use for coercion: the information specific for and leaked by a voting protocol, and information the adversary gets through external parameters, independently of the voting scheme. An example of the latter would be the election result, which can be compared to an expected distribution of the choices of honest voters. As stated in [MN06], it is obvious that if every voter votes for the same candidate, there is neither privacy nor coercion-resistance. The same holds if there is only one voter.

Write-in candidates

Analyzing elections with write-in candidates with respect to existing definitions is a special challenge because it gives two main weak points for coercion, which are in line with the above mentioned different kinds of information:

- Some voting schemes support write-in candidates, but in order to do so, process them differently than the normal candidates. Whether such a scheme can be analyzed by a certain definition strongly depends on the protocol, so we do not consider this issue in this review.

- The adversary learns significant information about the voter's choice from the tally itself: he can always coerce the voter to vote for a candidate that will most likely get no votes otherwise. This is by the way not only a problem of write-in candidates but with all elections that have such unlikely candidates.

The way the protocol-independent information such as the tally is treated strongly differs between existing definitions. Some definitions just ignore these external parameters and the level of coercion-resistance in elections with information-giving external parameters cannot really be told with such a definition. Other definitions take into account this external information but are not achievable by elections with write-in votes, because they give yes/no answers to the question of coercion-resistance, while another group of definitions is more flexible, and measures coercion-resistance by a bound δ. The latter build up the group of definitions that are suitable to measure the coercion-resistance of elections with write-in candidates.

Revoting

Revoting is a counter-measure to observation during the voting process, but also gives the voter the ability to change her mind, as in our application to delegated voting. The challenge that revoting gives us is that the voter can change her input during the protocol, or more precisely, give more than one valid input of which only last one may be included in the final outcome. This possibility is sometimes difficult to model: some definitions allow the voters to only cast a vote once in the protocol, other definitions let the adversary control the network and therefore learn which voter has cast how many ballots.

Delegated voting

With vote delegation, we have two dimensions of coercion: the coercion concerning the voter's choice and the coercion concerning the delegation. To capture vote delegation, a definition would have to consider the possibility of a party depending her input on someone else's without letting this person know and without learning the input of that other party. To the author's knowledge, currently no formal model exists that captures delegated voting. We leave this as an open problem and concentrate on the other two special cases in this review.

3.3.2. Definition Review

In this section, we first review two symbolic definitions which are slightly weaker than the general notion of coercion-resistance. Then we review simulation-based definitions which are related to the UC-framework [Can00], and several game-based definitions.

Symbolic Definitions

- Delaune et al. [DKR09] introduce a model for privacy-type properties of cryptographic voting schemes using the applied pi calculus. They concentrate on blind signature based voting schemes, where privacy is achieved through anonymous channels. The authors state that coercion-resistance cannot hold if the coercer can physically vote on behalf of the voter, and therefore model a voting booth as an anonymous channel. As usual for symbolic approaches, cryptography is assumed to work perfectly.

  In the semantics of [DKR09], the difference between privacy, receipt-freeness and coercion-resistance is defined via the nature of cooperation between the voter and the adversary. In all three definitions, the adversary must not be able to distinguish whether two voters A and B have swapped their votes, whereas in the definition of privacy, the adversary only sees public information, while in the definition of receipt-freeness, the voter gives input to the adversary but not the other way around. In the definition of coercion-resistance, the adversary additionally gives input to the voter. This notion does not capture coercion-resistance in its full generality, forced abstention attacks are not covered, for example.

  Write-in candidates and revoting: Additional information that is leaked by the tally is not taken into account by this definition, so it cannot be applied to write-in candidates. As to revoting, the model makes no restriction. It can be applied if the corresponding voting scheme can be modeled in the applied pi-calculus.

- Jonker and Pieters [JP06b] define an interesting notion they call *strong receipt-freeness* in epistemic logic. Their notion is rather close to coercion-resistance and states that with all information the voter can give to the adversary, the adversary cannot learn whether the voter did *not vote* for a certain candidate X.

  Write-in candidates and revoting: The definition takes into account the information supplied by the voter, as well as public information, but gives a yes/no-answer to the question of (strong) receipt-freeness. Therefore, the notion is not useful for write-in candidates. Revoting is not excluded by the underlying model.

Simulation-based definitions

Simulation-based definitions define security of a protocol via indistinguishability between an ideal world an the real world. In the real world, protocol participants interact with an adversary and the real protocol, while in the ideal world, the desired functionality of the real protocol is emulated by an ideal functionality, and the adversary's role is played by a simulator. These definitions have in common that they are often considered too strong and it is hard to construct protocols that can be proven secure with such definitions. They often require cryptographic primitives that are specially designed to achieve simulatability in all possible situations.

- Canetti and Gennaro [CG96] introduce a definition of coercion-resistance for general multi party computation. An election can be seen as a special kind of multi party computation since each participant wants to keep his input, the ballot, secret, and a joint output - the tally - is computed. So the definition in [CG96] can be applied to voting schemes and is actually listed by its authors as a use case. The definition measures coercion-resistance as the statistical difference δ between an execution of an ideal protocol with a simulator that has blackbox access to the adversary, and the execution of a real protocol with a real adversary. An advantage of this definition is that it is more flexible than giving a yes/no answer to the question of coercion-resistance.

  Write-in candidates and revoting: Write-in candidates would mainly affect the external parameters the adversary sees, which are the same in the ideal and the real world. So a vote for an unlikely candidate would have no influence on simulatability. It is therefore not advisable to measure coercion-resistance of a scheme with write-in candidates with this definition.

  Revoting as a counter strategy to coercion is hard to capture with this definition. The underlying model assumes that the adversary sees all communication between the parties and the input procedure is thought of as parties broadcasting their encrypted input. So the adversary is assumed to see the number of messages each party sends.

- Müller-Quade and Unruh [UMQ10] introduce the notion of *universally composable incoercibility (UC/c)*, an extention of the UC-framework of Canetti [Can00]. Parties are modeled by interactive Turing machines (ITM), coercion-resistance is defined by indistinguishability between a real world and an ideal world. In the ideal world, there is a deceiver that models the voter's free will, which has to be accomplished by the voter in the real world with the help of a deceiver-simulator. A party in the real world can have one of three corruption states: it is either uncorrupted, controlled by the adversary or controlled by the deceiver-simulator. A party controlled by the deceiver-simulator tries to accomplish her own will (modeled by the deceiver) while making the adversary believe she follows his instructions. A real protocol π then UC/c-emulates an ideal protocol ρ, if for every deceivier $\mathcal{D}$ there exists a deceiver-simulator $\mathcal{D}_S$ in the real world such that for each adversary $\mathcal{A}$ there exists an adversary-simulator $\mathcal{S}$ in the ideal world such that no environment $\mathcal{Z}$ can distinguish between the real world that contains the real protocol, the adversary and the deceiver-simulator, and the ideal world that contains the ideal functionality, the adversary-simulator and the deceiver. The definition assumes the adversary to have a chance δ_{min} of coercion in the ideal protocol, depending on

external parameters, while δ_{min} is not explicitly measured. To be coercion-resistant, the adversary's maximal chance of coercion has to be bounded by δ_{min} plus a negligible function.

Write-in candidates and revoting: Coercion by forcing the voter to vote for unlikely candidates would also be successful in the ideal world. Therefore, write-in candidates would increase coercion possibilities captured by δ_{min} in the ideal protocol, but have no effect on simulatability. So coercion-resistance with write-in candidates is hard to measure in the UC/c framework without explicitly measuring δ_{min}.

Modeling revoting is possible but not trivial, since care has to be taken with scheduling issues. Each Turing machine invokes the next in turn by sending it a message on its input tape. If no next Turing machine is specified, it is the environment's turn to invoke an ITM. So without further restrictions it cannot be assumed that each voter-ITM has the possibility to revote.

Game-Based and Computational Definitions

- Juels, Catalano and Jakobsson [JCJ05] were among of the first to give a formal and rather strong definition of coercion-resistance, and clearly differentiate between coercion-resistance and receipt-freeness. Their model is rather restrictive and only applicable to voting schemes with a specific structure which is not met by the voting schemes introduced in our work. Voters are assumed to use anonymous credentials for proving eligibility and casting their votes. The voter's strategy to evade coercion is then assumed to be providing the adversary with a fake credential. The definition takes into account external parameters and assumes the adversary to have knowledge about the expected distribution of votes cast by honest voters.

  Write-in candidates and revoting: The authors state themselves that elections supporting write-in candidates can never be coercion-resistant in their model as long as the election result is published accurately, which was actually the motivation for Section 4.5 of this work.

  The definition by Juels et al. is not designed for revoting as a deceiving strategy. This can be seen in the attack game, where the adversary casts his vote, with the credential obtained by the voter, after all other voters have cast their vote, and right before the tally is computed. They also define an ideal tally function that ignores double votes and therefore excludes revoting.

- Gardner et al. [GGR09] propose a game based definition where the adversary is given oracle access to the vote casting process, but given no access to post-voting outputs. Coercion-resistance is defined via the indistinguishability of public protocol outputs during the voting phase in answer to inputs of the voter's choice versus inputs of the adversary's choice. They model a counter strategy for the voter as a function *GenerateInput* that takes as input the voter's wish and the adversary's input of choice, which for example can be a vote for a certain candidate or certain randomness. It outputs input the voter can use for vote casting that will lead to the voter's wish but deceive the adversary.

  Write-in candidates and revoting: The definition does not take into account information on plaintext ballots, including write-in candidate strings.

The adversary only gets public output specific to each cast vote, it does not see the tally or anything published in the post-voting phase. So with this definition it is hard to measure the impact write-in votes would have on the coercion-resistance of an election.

The definition does not specify what the voter's input looks like, just that the voter sends some input, at least including her final choice, in a series of interactions with the election servers. So revoting can in principle be captured by the definition.

- Küsters et al. [KTV12] provide an intuitive game-based definition of coercion-resistance that is rather general and applicable to a wide range of voting schemes. The idea of their definition is that an adversary can not distinguish between a voter that follows the instructions of the adversary, and a voter that instead uses a deceiving strategy with which the voter accomplishes her own will. An advantage of this definition is that it takes into account both the voting scheme itself and external parameters.

  Coercion-resistance is measured by a parameter δ the calculation of which consists of a combinatorial part, that takes into account only the external parameters that are independent of the voting scheme, and a cryptographic part, that analyzes information specific to the voting protocol. They define an ideal election, which has coercion-possibility δ_{min}, in relation to which the coercion-resistance of voting schemes can be measured.

  The definition is very similar to UC/c (see Section 3.3.2): in this definition, too, there is assumed to be one universal deceiving strategy, determined by the voting scheme and known to the adversary, that can be used against each coercion-strategy. Though while UC/c is very specific concerning the scheduling, in the model of Küsters et al. an appropriate scheduling is assumed without being defined explicitly.

  Write-in candidates and revoting: Write-in candidates would increase δ_{min}, so as with the UC/c definition, judging coercion-resistance of an election with write-in candidates requires explicitly measuring δ_{min}. Our notion of fuzziness introduced in Section 4.5 would in turn decrease δ_{min}, and the two definitions in combination seem to be appropriate to handle write-in candidates.

  Revoting is not excluded by the model and can easily be used as a deceiving strategy.

- In their work about single transferable votes (STV), Teague et al. [TRN08] propose a definition of coercion-resistance that takes into account the information leakage of (intermediate) tally results. The adversary can communicate with the voter before and after, but not during the voting process. The view of an adversary is then defined as the information the adversary learns in this communication together with public information. Coercion-resistance is defined against a certain threshold for the ratio between the likelihood of a certain view given the voter obeyed the adversary and the likelihood of a view given the voter did not obey. They model the voter's choice in a very abstract way, so it can be not only her choice of candidate but also any decision the voter could make, like for example abstaining from voting.

Write-in candidates and revoting: The definition seems suited for elections with write-in candidates since it takes into account all public information, including the tally results. To measure the impact of write-in votes on coercion-resistance, an upper bound for the ratio between the likelihoods of views would be the value of interest.

The definition does not put restrictions on the casting process, so revoting is not excluded, but as the definition assumes a private channel for vote casting, it makes little sense to analyze revoting as a counter-measure to adversarial observation in this model.

3.3.3. Conclusion

We conclude our review with a brief summary of the suitability of the reviewed definitions for measuring the coercion-resistance of election schemes which support write-in candidates and revoting.

Write-in candidates

We have seen that the definitions of Delaune et al. [DKR09], Canetti et al. [CG96] and Müller-Quade et al. [UMQ10] are not suited to measure coercion-resistance with write-in candidates, since they do not take into account non-protocol specific information, and write-in candidates would have no impact on coercion-resistance. The definitions of Gardner et al. [GGR09] and Juels et al. [JCJ05] do explicitly not take into account write-in votes. As the definition of Jonker et al. [JP06a], they cannot be fulfilled with voting schemes that allow write-in candidates if the tally is published accurately. The definitions of Küsters et al [KTV12] and Teague et al. [TRN08] are suited to measure coercion-resistance in the presence of write-in candidates, though they would correctly show a rather low level of coercion-resistance if the tally is published accurately.

Revoting

The definition of Canetti et al. [CG96] is not applicable to revoting since the adversary controls the network. Revoting is also not captured by the definition of Juels et al. [JCJ05], which is designed for a different, specific strategy to evade coercion. The definition of Teague et al. [TRN08] assumes a private channel during the voting process, while in a szenario where revoting is used as a counter-measure against coercion, it is assumed that the voter can be observed during the voting process. The definition of Gardner et al. [GGR09] is applicable to revoting, but gives the adversary no access to data published in the post-voting phase. Both reviewed symbolic definitions are applicable to voting schemes which use revoting as a counter-measure. However, these definitions only capture a weaker notion of coercion-resistance. The definitions of Unruh et al. [UMQ10] and Küsters et al. [KTV12] seem best suited to capture revoting, though scheduling issues need to be kept in mind.

4. Coercion Resistance in Presential Elections

In presential elections, voters cast their choices in a designated polling station. Since voters are present in person, their eligibility can be checked on-site by poll workers. Therefore, election schemes for presential elections do often not specify how voters are authenticated and how their eligibility is checked. Secrecy is usually provided by a voting booth which the voter enters alone to fill out her ballot. Depending on the voting scheme, this can be done on paper or with the help of a so-called *vote casting interface*, the user interface of a voting machine.

A weakness presential elections have in common is that forced-abstention attacks are always possible: the adversary can observe the polling station and see who has cast a vote. This attack is very expensive and uncomfortable for the adversary: he has to be present at the polling station, or observe it during the whole election time with a camera. Therefore, it is still important to prevent other forced abstention attacks in a voting scheme, where the adversary could perform such an attack comfortably from at home by analyzing published data. Bingo Voting is a cryptographic voting scheme which was proven to be as coercion-resistant as an ideal voting scheme in [KTV12]. In this Chapter, we introduce real-world experiences with Bingo Voting, introduce possible improvements and discuss under which assumptions the scheme is coercion resistant. To review our practical experiences, we analyze the conducted election with our taxonomy. To add more flexibility to Bingo Voting, we provide it with the possibility to allow for write-in candidates.

As we have seen in Section 3.3, most definitions of coercion-resistance are not suited for elections with write-in candidates. The reason for this is that the adversary can use information leaked by the tally to coerce the voter. Motivated by this issue, we introduce a definition for a controlled fuzzy tally representation, to enable coercion-resistant elections with write-in candidates. We introduce tallying techniques for mix-based schemes and schemes with homomorphic tallying, which fulfill the new definition. After introducing the general approaches, we apply it to Bingo Voting and show that the modified scheme is secure under our new definition.

This Chapter is structured as follows: after introducing related work relevant for this chapter in Section 4.1, we introduce Bingo Voting and its improvements in Section 4.2. Practice experiences with Bingo Voting are discussed in Section 4.3. To

make Bingo Voting more flexible, we provide it with write-in candidates in Section 4.4. After discussing the additional impact write-in candidates have on coercion-resistance, we introduce our notion of *fuzziness* in Section 4.5. Section 4.6 shows how fuzziness can be realized in cryptographic voting schemes, and applies those techniques to Bingo Voting as an example.

4.1. Related Work

4.1.1. Related Work on Presential Elections

During the last three decades, many voting schemes for presential elections have been introduced which provide public verifiability while protecting privacy. They can be divided into two groups: paper-based schemes, where the voter filles out a paper ballot, and computer-based schemes, where the voter casts her choice using a vote casting interface. Most paper-based schemes use a hybrid approach: the voter casts her choice on a paper ballot, which is then scanned and processed electronically.

With Three Ballot [Riv06], Rivest introduced a paper-based scheme which is publicly verifiable without using any cryptography. The scheme is easy to understand, but allows pattern-voting attacks in elections with many candidates, and gives the adversary a $\frac{2}{3}$ chance of manipulation for each ballot. Punchscan [PH10] is a receipt-free paper based scheme which is not coercion-resistant, it is vulnerable to a randomization attack [JCJ05]. ScantegrityII [CCC+08] is a paper-based scheme, which has been successfully applied in a real world election [CCC+10]. Prêt à Voter [CRS05] is a paper-based scheme in which the scanner does not learn the voter's choice. It has recently been further developed to achieve everlasting privacy [DHvdG+13].

Research on cryptographic electronic voting in general has started with the work of Chaum [Cha81]. One of the first electronic cryptographic voting schemes was the scheme of Benaloh and Tuinstra [BT94], where the voter enters a voting booth for registration, but later casts her vote over a public channel. A cryptographic voting schemes that uses a voting machine is introduced by Moran and Naor [MN06], it gives a receipt to the voter with challenges for zero-knowledge proofs, which prove for each candidate that this candidate is voted and included in the tally, whereas the proofs are simulated for the not voted candidates. Only the voter, who took part in the voting process, knows which of the proofs is not simulated. Motivated by this scheme was Bingo Voting [BMQR07], which hides the voter's choice between dummy votes. Bingo Voting is used throughout this chapter to introduce and demonstrate our techniques.

There are many more voting schemes for presential elections, those in [AR06, Cha04, BJR10, CEC+08, ACvdG10, RS07, AN09] still build up an incomplete list, but provide an overview over their variety. Reviews on several voting schemes were done in [Hen12] and [JMP13].

4.1.2. Related work on Bingo Voting

Bingo Voting was originally introduced in 2007 by Bohli, Müller-Quade and Röhrich in [BMQR07]. Improvements regarding both voter privacy and verifiability were done by Bohli et al. in [BHK+09], they will be discussed in Section 4.2.2. Bingo Voting was implemented and applied in the Student's Parliament's election of Universität Karlsruhe (TH), now Karlsruhe Institute of Technology, in 2008. The experiences of the event together with a comparison of the scheme with three others, regarding usability, were published in [BHMQ+08]. We discuss those practical experiences in Section 4.3. Additional improvements were introduced in the dissertation

of Christian Henrich [Hen12], who, among other things, used a proof technique of Groth [Gro02] to minimize the size of the correctness proofs of the scheme.

Küsters et al. [KTV12] introduced a definition of coercion resistance under which they proved Bingo Voting as secure as an ideal voting scheme.

4.1.3. Related work on write-in candidates

A review of definitions of coercion-resistance and their relation to write-in candidates can be found in Section 3.3. Several voting schemes exist which efficiently include write-in votes in elections based on homomorphic encryption [Acq04, KY04]. While forced-abstention attacks are not considered in [KY04], Acquisti [Acq04] prevents forced abstention attacks based on forcing the voter to "vote for" a random string by only allowing certain permissible choices. This does not keep an adversary from forcing the voter to vote for a valid, rarely elected candidate.

As stated in [Adi06], mixnet-based schemes like Neff's [Nef01] naturally support write-in votes as they support free-form ballots. Scantegrity II [CCC+08] has been used in a real election [CCC+10] providing the possibility to write in candidates in a way which is practical and straightforward for the voter. Their technique is similar to ours in that it also tallies in two steps: first the candidates from the list and then the write-in candidates. However, the write-in solution of Scantegrity II has some drawbacks. As stated in [CCC+10], the election authority can modify the write-in names. Furthermore, any observer auditing the resolution of write-in votes sees the handwritten choices. The two-step idea of counting write-in votes separately has also been proposed in [Adi06] for paper-based schemes. All schemes allowing write-in votes, including those mentioned above, are naturally vulnerable to forced-abstention attacks as long as their proofs of correctness yield the exact tally.

4.2. Bingo Voting

In this Section, we briefly review Bingo Voting and its development during the last few years. Bingo Voting provides verifiable correctness even if the voting machine is corrupt. If the voting machine is uncorrupted, it additionally provides coercion resistance. The scheme's verifiability relies on a trusted random number generator. A voting machine is usually a rather complex system, so its hardware and software are hard to test. As opposed to this, the random number generator can be very simple hardware, ideally without any software component, like for example a bingo cage.

Bingo Voting was successfully applied in the election of the student parliament in the University of Karlsruhe (now Karlsruhe Institute of Technology) in 2008, Section 4.3 gives some insight into our practical experiences. The scheme is rather flexible since it is applicable to elections which allow *vote-splitting*, i. e. voters can distribute their votes among several candidates, and *cumulative voting*, i. e. voters can give more than one vote to a candidate. Furthermore, it makes no restrictions towards the vote casting interface. Choices can be presented to the voter in an arbitrary way. In Section 4.4, we introduce a technique to provide Bingo Voting with write-in candidates.

4.2.1. The Original Bingo Voting Scheme

In this section, we describe the original Bingo Voting scheme, as introduced in [BMQR07]. The idea of Bingo Voting is similar to the voting scheme of Moran

and Naor [MN06]: after casting her vote, the voter gets a receipt with which she can later verify that her vote is correctly included in the tally. Since this receipt must not yield information about the voter's choice, the chosen candidate is hidden between so-called *dummy votes*, and only the voter, who has been present in the voting booth during the voting process, can differentiate between her choice and the dummy votes, but not prove her knowledge to others.

In Bingo Voting, votes are represented by random numbers. Dummy votes are random numbers that are precalculated in a *pre-voting phase*, and commitments to them are published on a public bulletin board before the election starts. In the *voting phase*, after the voter has entered her choice, the voting machine prints a receipt that contains the candidate of the voter's choice, together with a fresh random number created and displayed by the trusted random number generator. To conceal this choice, the receipt also contains dummy votes for each not voted candidate. By comparing the display of the random number generator with the receipt, the voter is convinced that her chosen candidate is represented by a fresh random number and not a dummy vote. In a *post-voting phase*, all receipts, the tally and its proofs of correctness are published for verification. It is proven that each receipt contains only one fresh random number, and apart from that only dummy votes. That the fresh number is assigned to the right candidate could be verified by the voter in the voting booth. The three phases will be described in more detail below, after all necessary assumptions have been explained.

4.2.1.1. Notation

Let com be the commitment function of a Pedersen commitment [Ped91a] (see Section 2.3.4.2). With $com_r(m)$ we denote a commitment to a message m with randomness r. We omit the parameter r if it is clear from the context or irrelevant and write $com(m)$ instead. A commitment to a tuple (m_1, m_2) denoted by $com(m_1, m_2)$ is performed componentwise: $com(m_1, m_2) = (com(m_1), com(m_2))$. Please note that this differs from the usual notation that denotes the randomness used for the commitment in the second parameter.

Bingo Voting uses Pedersen commitments because they are unconditionally hiding, which allows it to provide everlasting privacy. The implied computational binding property is sufficient since it only needs to hold until all proofs of correctness are computed. Pedersen commitments are also maskable through rerandomization, which allows to shuffle a set of commitments with common mix-techniques as explained in Section 2.3.6.

4.2.1.2. Preconditions

Coercion-resistance and verifiable correctness hold under the following assumptions:

Preconditions for verifiable correctness

- The trusted random number generator is uncorrupted, and its output is displayed to the voter unchanged.
- The probability of collisions between dummy random numbers and fresh random numbers is negligible.
- There is a public bulletin board to which anyone has read access, and an election authority has write access. This election authority does not need to be trusted for correctness.

213	683	172
769	579	413
...	...	...
145	123	756
Alice	**Bob**	**Carol**

Figure 4.1.: Dummy votes are created in the pre-voting phase and stored on the voting machine. They have to be kept secret. Each candidate gets the same number of dummy votes.

- A mechanism is provided with which the voter can log in at the voting machine and cast exactly one ballot.

We assume that the random number generator has its own display, so the fresh random numbers are not displayed to the voter by a possibly corrupted device. If a fresh random number collides with a dummy random number of the voted candidate, the voting machine can substitute the fresh random number by the colliding dummy random number and assign another new random number it creates by itself to a candidate of its choice. Therefore, collisions should happen only with negligible probability. A discussion about an appropriate choice of random numbers and their probablity of collisions is presented in the thesis of Christian Henrich [Hen12].

Preconditions for privacy

- The voting machine is uncorrupted.
- The distribution of random numbers created by the trusted random number generator and the distribution of the dummy votes are indistinguishable.

All calculations, including the creation of dummy votes and all proofs of correctness, can be executed by the voting machine. The voting machine needs to know the dummy votes in plaintext so it can print them on the receipts. Therefore, it has to be trusted for privacy. We model the trustworthiness of the voting machine as an election authority that has full access to the voting machine and write access to the bulletin board. This election authority has to be trusted for secrecy but not for correctness. Fresh random numbers should not be identifiable on the receipts, so their distribution has to be the same as the distribution of the dummy votes.

4.2.1.3. Pre-Voting Phase

In the pre-voting phase, dummy votes for each candidate are created which are later used on the voter's receipt to conceal her choice. Let n be the number of candidates, and $\mathcal{C} = \{C_1, \ldots, C_n\}$ the set of candidates. Furthermore, let l be the number of eligible voters, and s the number of votes a voter can give to a single candidate. As indicated in Figure 4.1, each candidate gets the same amount k of dummy votes, where $k = l \cdot s$ is the number of votes a candidate can maximally get in the tally.

For the sake of convenience we assume that each voter can cast exactly one vote per candidate, so each ballot encodes a one out of n choice and $k = l$. Other ballot formats that allow vote-splitting and cumulative voting are discussed in Section 4.3. If the exact number of expected voters is unclear, more dummy votes can be created, as long as this is later accounted for in the calculation of the tally.

$(com(Bob), com(123))$	$(com(Bob), com(683))$	$(com(Carol), com(172))$
$(com(Alice), com(769))$	$(com(Carol), com(413))$	$(com(Alice), com(145))$
...	...	...
$(com(Carol), com(756))$	$(com(Alice), com(213))$	$(com(Bob), com(579))$

Figure 4.2.: Commitments to dummy votes are published before the voting period starts. The commitments are published in random order so they cannot be assigned to candidates.

The dummy votes are created as follows: for each candidate $C_i \in \mathcal{C}$, k random numbers $r_{C_i,1}, \ldots, r_{C_i,k}$ are drawn. These random numbers are the dummy votes for candidate C_i. The random numbers have to be mutually distinct and indistinguishable from those drawn by the trusted random number generator during the voting phase, so they are ideally drawn out of the same source.

For each candidate C_i, commitments $com(C_i, r_{C_i,j}) = (com(C_i), com(r_{C_i,j})$ to each of his dummy votes $r_{C_i,j}$ are created. The commitments are published on the public bulletin board in random order, as shown in Figure 4.2. The random numbers themselves must be kept secret by the voting authority. Additionally, a proof that each candidate has the same amount of dummy votes is published on the public bulletin board.

In the original scheme [BMQR07] these proofs are suggested to be performed using randomized partial checking (RPC) [JJR02], but the authors of [BMQR07] also state that other proof techniques can be used, for example a shuffle using shadow mixes as explained in Section 2.3.6.2. In fact, RPC should not be used since it yields too much information [KW13]. In the election of the student parliament we performed an interactive proof with shadow mixes.

4.2.1.4. Voting Phase

In the voting booth, the voter uses the vote casting interface of the voting machine to enter her choice. The representation of the choices to the voter is arbitrary and not specified. After confirming her choice, the vote casting process is initiated. The voting machine prints a receipt that contains a list of all candidates with a random number next to each candidate's name. Each random number next to a not-voted candidate is one of this candidate's dummy votes. The voting machine internally marks these dummy votes as used and does not use them again on another receipt. The fresh random number from the trusted random number generator is written next to the voted candidate.

The receipt creation is shown in Figure 4.3. In the privacy of the voting booth, the voter can compare the number next to the elected candidate's name with the fresh random number shown on the trusted random number generator's display. If the two numbers are equal, the voter can be convinced that the number next to her chosen candidate is not a dummy vote. She can take the receipt with her to later check if her vote is included in the tally. When the voter leaves the voting booth, the random number generator's display is cleared, and the receipt does not show which of the numbers were fresh. This information is only known to the voter and the voting machine, and cannot be proven by the voter to an adversary. The receipt is also stored digitally on the voting machine to create the proofs of correctness in the post-voting phase.

Dummy votes stored by the voting machine:

213	~~683~~	~~172~~
769	579	413
...	...	...
145	123	756
Alice	**Bob**	**Carol**

Voter view:

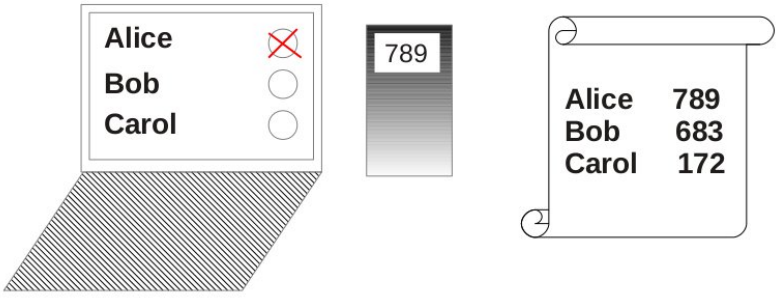

Figure 4.3.: In the voting booth, the voter votes for Alice. The fresh random number 789 is created by the trusted random number generator. Bob and Carol both "lose" a dummy vote which is printed on the receipt.

4.2.1.5. Post-Voting Phase

After the voting period is over, all commitments to unused dummy votes are opened by publishing their unveil information on the public bulletin board. The unused dummy votes mirror the tally, since each time a candidate gets a vote, he does not lose a dummy vote. Therefore, in theory, it is thus sufficient to publish the unused dummy votes as the tally. The tally and all receipts are published on the public bulletin board, and the voters can check the presence of their receipts.

A proof is published that each receipt contains a) exactly one fresh random number and b) for each not elected candidate a dummy vote of which a commitment was published in the pre-voting phase. The proof takes as input a new commitment to the fresh random number and the commitments to the dummy votes on the receipt which were published in the pre-voting phase. It is then proven that there is one commitment for each candidate and that the content of the commitments correspond to the random numbers on the receipt, without revealing any associations between commitments and random numbers. The details of this proof are again omitted here since there are different proof techniques [BMQR07, JJR02] that work here as well as with our adapted fuzzy scheme which is described in Section 4.6.3.

4.2.2. Improvements of Bingo Voting

In joint work with Jörn Müller-Quade, Jens-Matthias Bohli, Christian Henrich and Stefan Röhrich, several improvements of Bingo Voting have been suggested since its introduction in 2007, most of them can also be applied to other voting schemes.

This section shortly summarizes some of the improvements which were suggested in [BHK+09] and [Hen12]. Additional improvements motivated by our practice experiences in the student parliament election are discussed in [BHMQ+08, Hen12] and

Section 4.3. A way to let Bingo Voting support write-in candidates is introduced in [Kem12] and Section 4.4.

Early deletion of secrets

The voting machine stores all dummy votes in plaintext, since they are needed for the generation of the receipts and their proofs of correctness. If an adversary gets access to these dummy votes, he can identify fresh random numbers and with that break voter privacy. However, after generating a receipt, the dummy votes used on this receipt are only needed for the receipt's proof of correctness, the tally is calculated from the unused dummy votes. There is one such proof per receipt, which takes as input the dummy votes used on the receipt, their corresponding commitments (with unveil information) and the fresh random number. If these proofs are created by the voting machine non-interactively, directly after each voting process, the used dummy votes and the unveil information of their commitments can be deleted immediately [BHK+09]. This way, all secrets which could break a voter's privacy would be deleted directly after she leaves the voting booth, and cannot be derived by a malicious voting authority who has access to the voting machine after the voting phase.

Hash Chains

If the adversary knows which receipts are not checked, it can manipulate the tally by manipulating these receipts. To prevent this attack, the voting machine can build up a so-called *hash chain*: each receipt is hashed after creation and its hash value is then included on the receipt of the next voter. This way, to manipulate a receipt after the voting phase, the adversary either needs to find a collision with the receipt's hash value or also manipulate all receipts that were created after the manipulated receipt, which would significantly increase the chance of being detected. Therefore, the hash chain makes manipulating receipts after the voting phase very hard. The voting machine can still manipulate receipts upon their creation, but to this end it needs to know in advance if the voter is going to check her receipt. This is discussed in more detail in [BHK+09] and [Hen12].

Proving a fraud in the voting booth

In the voting booth, if the fresh random number does not appear next to the voter's chosen candidate, it is hard to prove a discrepancy without giving up voter privacy. Moreover, if the fresh random number appears on the receipt but in the wrong place, it is hard for the voter to prove that she did not vote for the other candidate in the first place. In [BHK+09, Hen12], a solution is suggested in which the voter casts her choice by filling out a paper ballot, which acts as forensic evidence of the voter's choice. This ballot is scanned to record the vote and create the receipt. To be able to prove a fraud without breaking privacy, alignment information is printed both on the paper ballot and on the receipt. Using two different kinds of privacy sleeves, the voter can prove a discrepancy: with one privacy sleeve, the voter can prove that the fresh random number is not in the same row as where she has put her mark. With the other privacy sleeve, the poll worker can check whether the alignment information is correctly printed on both ballot and receipt.

Resolving a dispute after the voting phase

After all receipts and proofs of correctness are published, the voter detects a fraud if her receipt is not published or a proof of correctness is incorrect. In the latter case, there is nothing to prove since everyone can check the proofs. If the voter's

receipt is missing or not correctly published, the voter can prove this by showing her receipt. This implies that it has to be ensured that the receipt is unforgeable. On way to achieve this is letting the voting machine or the printer digitally sign the receipt, or using unforgeable paper. However, if the signing component is corrupt or the printer uses the wrong paper, the voter cannot prove that she did not forge her receipt. Moreover, using these techniques it is hard for the voter to convince herself in the voting booth that her receipt is created with the correct forensic evidence. So it would be of advantage to have more than one evidence to prove the authenticity of the receipt, created by different instances [BHK+09, Hen12].

Convenient comparison of random numbers

In the voting booth, the voter has to compare the random number on the trusted random number generator to the random number next to the chosen candidate on her receipt. To make this comparison more convenient, the random numbers should be presented to the voter in a way such that they can be easily processed by the human brain. Following Ryan and Wickelgren [Rya69, Wic64], groups of 3 or 4 digits of easily distinguishable characters seems to be a suitable representation. This is described in more detail by Christian Henrich [Hen12], who also discusses the necessary length of random numbers and techniques to decrease their probability of collision, which in turn allows for shorter random numbers.

4.2.3. A discussion on Coercion-Resistance

In this Section, we discuss the privacy properties of Bingo Voting. First, we show why the choice of the used shuffle proofs is crucial. Then we show that Bingo Voting offers more privacy and coercion-resistance than a paper election if cumulative voting and vote-splitting are allowed. After that, we argue that under given assumptions, Bingo Voting offers everlasting privacy.

Importance of the shuffle proofs

In [KTV12], Bingo Voting is proven as coercion-resistant as an ideal voting scheme. This holds if all used shuffle proofs are zero knowledge proofs.

Bingo voting uses proofs of correctness of a shuffle on two occasions: in the pre-voting phase, to prove that each candidate has got the same amount of dummy votes, and in the post-voting phase, to prove that each receipt contains exactly one fresh random number and otherwise dummy votes. In these proofs, Bingo Voting shuffles and opens Pedersen commitments, and then proves the correctness of the shuffles.

For the proofs of correctness of the receipts, the used shuffle proof technique is critical: it is not sufficient to hide the link between the fresh random number and the chosen candidate, since this would only provide receipt-freeness. To provide coercion-resistance, it is important that the adversary cannot see that a candidate is not voted, or that one candidate is voted with a higher probability than another. Therefore, a shuffle proof needs to be computed in a way that the fresh commitment, which encodes the fresh random number, could in principle belong to each opened random number, with equal probability.

However, the original paper [BMQR07] suggests randomized partial checking (RPC)[JJR02] as a proof technique, which is not zero-knowledge and in fact has several flaws, both concerning privacy as well as concerning soundness [KW13]. It is easy to see that if only one mix server is used, which is the case if the voting machine creates the proofs by itself, RPC reveals significant information as shown in Section

2.3.6.3: if an RPC proof is performed in the naive way, by shuffling the commitments twice and then for each commitment in the middle reveal either the link to the original commitment or the link to the opened commitment, the adversary would know for half of the candidates that the voter did not vote them.

A better, but less efficient way is the zero-knowledge proof of a shuffle using shadow mixes, as introduced in [Adi08, Ben06] and explained in Section 2.3.6.2. For more efficiency, Henrich [Hen12] suggests using a proof technique of Groth [Gro02], which is a honest-verifier zero knowledge argument.

As a side note, an additional improvement would be following the example of a real world use of Helios [AdM09] and opening shuffled commitments only after giving voters a chance to complain. This way, voters can complain if a shuffle is not computed correctly and reveals information about links between shuffle input and output.

Resistance against pattern voting

Assuming the used shuffle proofs are zero knowledge proofs, Bingo Voting is even more coercion-resistant than a paper election, or other elections where the plaintext ballots are revealed for counting after being unlinked from the voter, since it reveals less information. The reason for this is that the tally is computed by counting the unused dummy votes, which are not linked to any voter. Therefore, Bingo Voting has no chance to reveal any voting pattern, except in the proof of correctness of each receipt. So if each receipt contains the same amount of fresh random numbers, no pattern is revealed by these proofs. For the adversary it is indistinguishable for any s if a candidate got s votes from one voter, or one vote from s voters, or any combination in between.

To give an example, imagine a voting scheme where vote-splitting or cumulative voting is allowed. If plaintext ballots are revealed, the voter could be coerced to fill out her ballot in a certain pattern. However, Bingo Voting only reveals the information that on the voter's ballot, k votes were given, where k is the number of fresh random numbers on the voter's receipt. These k votes could be distributed to candidates in any possible way. If a special abstention candidate is used as was done in the student parliament election (see Section 4.3), a voter can even undervote without being detected.

Everlasting privacy

To break voter privacy, the adversary needs to identify the fresh random numbers on each receipt. He can do this by opening the commitments to the dummy votes published in the pre-voting phase, which are unconditionally hiding. Without giving a formal proof, we argue that Bingo Voting offers everlasting privacy under the following conditions:

- The distribution of fresh and dummy random numbers is indistinguishable.
- The voting machine is trusted.
- All dummy votes and the unveil information of all published commitments is stored only on the voting machine.
- All proofs of correctness of a shuffle computed in the pre-election and post-election phases are perfectly zero-knowledge.

If these conditions hold, the adversary can only identify the voted candidate by opening the published Pedersen commitments, which are unconditionally hiding. The first three assumptions are requirements of the Bingo Voting scheme. The fourth requirement is important and achievable as discussed above.

4.3. Bingo Voting in the Student Parliament Election

In January 2008, we had the opportunity to apply Bingo Voting in the election of the student parliament of the University of Karlsruhe, now Karlsruhe Institute of Technology. The complexity of the election allowed us to show the flexibility of the scheme: fifteen polls were run in parallel, and the election allowed vote-splitting and cumulative voting. Voters could choose between casting a paper ballot or using the Bingo Voting scheme. Many voters decided to try out the new scheme and cast their votes electronically.

In the remainder of this section, we describe how we adapted and implemented Bingo Voting to meet the requirements of the student parliament election, and share our practice experiences with the Bingo Voting scheme. We first describe election details and adaptions we made to meet the election's requirements. After that we describe the implementation of our voting machine. We then analyze the election with our taxonomy to compare the analysis with experiences gained in the student parliament election.

The realization of Bingo Voting and its application in the student parliament election was joint work with Christian Henrich. It is also discussed in [BHMQ+08] and Christian Henrich's dissertation [Hen12]. This work concentrates more on implementation and security issues, but for completeness, some election details that were already described in [Hen12] are also described here.

4.3.1. About the Election

The election of the student parliament was held in January 2008 on five consecutive days. The election took place in several polling stations, Bingo Voting was available in one of them. In the same polling station, as well as in all others, students could cast their choices on a paper ballot, similar to the paper election described in the introduction. In the following, we concentrate on the part of the election that was performed with Bingo Voting. The voting machine was treated as one ballot box. Students could choose between casting a ballot either with Bingo Voting or on paper. Revoting was not allowed, an electronically cast vote could not be overwritten with a paper ballot.

The election was rather complex: fifteen different polls were run in parallel, each student could cast a vote in up to five out of these fifteen polls. The largest of the polls was the election of the student parliament itself, which actually consisted of two polls: in the first poll, the voter could vote for one out of nine lists. In the second poll, the voter could then distribute 9 votes among the 72 candidates on the lists, at most five votes per candidate. Each student could additionally take part in the election of the student council of his faculty, there were eleven such polls. Female students could additionally take part in a poll for the women's representative. Foreign students could additionally vote for the representative for foreign students.

4.3.2. Special Requirements of the Student Parliament Election

For its application in the student parliament election, the Bingo Voting scheme had to be adapted to meet all the requirements of the election. As implied by the description above, the voting scheme had to support vote-splitting and cumulative voting. This can easily be done with Bingo Voting by adjusting the number of dummy votes. A more challenging requirement was that everything that was possible with the paper election also had to be possible in the electronic election. This implied that voters had to be able to cast invalid ballots, or cast empty or only partially filled-out ballots. Voters were also allowed to take part in different polls on different days.

The original Bingo Voting scheme as described in [BMQR07] does not specify how the voter proves eligibility, and how she logs in at the voting machine. So our voting machine had to be equipped with a login mechanism that allowed each voter to cast only one ballot per poll, and only in these polls she was eligible for, while allowing her to leave the voting booth after one poll and cast a ballot in another poll at a later time.

4.3.3. Implementation and Application

This section describes how Bingo Voting was adapted to meet all requirements of the student parliament election, and how it was applied in practice. The software for the voting machine was implemented by the author and Michael Bär in the context of Bär's diploma thesis [BÖ8]. The implementation was specifically designed for the election of the student parliament. The implementation was written for Java 1.6 and has about 8000 lines of code. It consists of three sub-programs which can be executed separately:

- The **pre-election** software that creates and commits to dummy votes and proves the correct number of dummy votes for each candidate,
- the software for the **voting process** that is responsible for voter-login at the voting machine, ballot marking and casting, and
- the **post-election** software that computes the tally and all proofs of correctness.

4.3.3.1. Used Hardware

In the election of the student parliament, we used a standard PC with a linux system (SUSE 3.2) as a voting machine. For printing the receipts, we used a laser printer, the receipts were printed on A4 paper. Connected to the voting machine were the printer, a mouse, a chip-card reader and a random number generator, the latter two provided by Reiner SCT[1]. Voters could log in at the voting machine with a memory card they inserted into the chip card reader. It consists of a certified smart card in a smart card reader. The card is protected by a seal so it cannot be taken out and exchanged unnoticed. The smart card was originally designed for digital signing and has an integrated hardware-based random number generator. The smart card reader was specifically customized for our requirements and has special firmware to display the random numbers as a string of hexadecimal digits.

[1] Website of Reiner SCT: `http://www.reiner-sct.com/`

4.3.3.2. Pre-Voting Phase

In the pre-election phase, the dummy random numbers were created, and commitments to them were published on the website of the voting authority, from here on called the *bulleting board*. Proofs were created to show that each candidate had the proper amount of dummy votes. Details are described in the following paragraphs.

Cumulative voting and vote-splitting

The election of the student parliament allowed cumulative voting and vote-splitting. cumulative voting means that the voter could give more than one vote to a single candidate. Vote-splitting allowed the voter to distribute her votes among different candidates. In the following, let n be the number of votes a voter could distribute in a poll and k the maximum number of votes she could cast per candidate in that poll. In the most complex election the voter had $n = 9$ votes which she could distribute among 72 candidates, whereupon she could give at most $k = 5$ votes to a single candidate.

In Bingo Voting, one fresh random number represents one vote. So each voter got n fresh random numbers in the voting phase, which were assigned to the candidates according to how the voter distributed her votes. At most k votes, i. e. fresh random numbers, could be assigned per candidate. To represent this on the receipt, there were k random numbers written behind each candidate. For the pre-election, this implied that k dummy random numbers per candidate had to be created for each voter.

Invalid ballots and abstention

To prevent coercion and preserve verifiability, intentionally invalid, empty or only partially filled-out ballots had to be indistinguishable from a valid, fully filled-out ballots. For this purpose, we introduced two special candidates: an *invalid* candidate, and an *abstention* candidate. To cast an empty ballot or undervote, a voter could give all or some of her votes to the abstention candidate. To cast an invalid ballot, all of the voter's votes were given to the invalid candidate. Because a ballot is either invalid or not, but never partially invalid, a voter could only give either all or no votes to the invalid candidate. However, this was only enforced by the software and not publicly verifiable. Since the voter could give all n of her votes to the invalid or abstention candidate, n dummy votes per voter were created for the invalid and the abstention candidate each.

Dummy vote generation and the pre-election proof

In the biggest poll, voters could distribute 9 votes among 72 candidates, maximum 5 votes per candidate. So in this poll, for each voter, we had to create 5 dummy random numbers per candidate plus 9 dummy votes for the abstention and invalid candidate, resulting in a total number of 378 dummy votes per voter. Since this was one of the two polls for the student parliament itself, this poll also had the highest amount of eligible voters. We prepared dummy votes for 4000 voters for the student parliaments election and for 1000 to 2000 voters for the other polls. This led to an overall number of dummy votes of a little more than a million. Since a paper election was run in parallel and Bingo Voting was only available in one out of several polling stations, we did not have to cover the full amount of eligible voters. To commit to the dummy votes, we used Pedersen commitments. The commitments to the dummy votes were published on the bulleting board.

For the pre-election proof, we used the shuffling technique with shadow mixes as explained in Section 2.3.6.2: we rerandomized and shuffled the candidate parts of the commitments twice, and then opened the shuffled and masked commitments resulting from the second shuffle. For the proof of correct shuffling, a challenge has been created together with a member of the voting authority. According to the challenge, either the permutation of the first shuffle or the permutation of the second shuffle was opened and proven. The pre-election calculations took about ten days. With appropriate parallelization the calculations could have been completed within several hours.

4.3.3.3. Election Phase

On election day, there were several desks with two poll workers each, who maintained an electoral register and handed out memory cards which allowed the voters to log in at the voting machine. The login cards were customized since each voter was eligible for different polls, but the cards contained no information about the voter other than for which poll she was eligible. The registration software with which the poll workers encoded these cards was provided by the voting authority on bootable CD-ROMs. Each poll worker desk had a laptop which was booted from one of these CD-ROMs to decrease the risk of malicious software running on the laptops. The laptops contained an electronic voter roll with eligibility information about each voter, which was also used for the registration process of the paper election.

To prevent voters from exiting the vote casting interface and accessing the underlying system of the voting machine, no keyboard was connected to it except when administration was necessary. The voting machine was not connected to the internet or any other network. The only hardware in the voting booth was the voting machine, the random number generator, a Mouse, a chip card reader and the receipt printer. For administration, a special administration card could end the voting software (without triggering the tally) to access the underlying system. The voting software could later be restarted to go on with the voting process. For administration, no person was allowed access to the voting machine alone. At least one administrator (who was in possession of the administration card) and one member of the voting authority had to be present to guarantee proper mutual observation.

Voter Registration

Before starting the voting process, the voter first had to register at a poll worker's desk. There, she presented her student identity card to a poll worker and her eligibility was checked. The poll workers prepared a memory card for her on which they encoded for which polls she was eligible. The registration procedure had to be designed in a way that voters could not forge, copy or modify cards in order to vote more than once in one poll, or cast a vote in a poll they were not eligible vor. Therefore, each login card contained a unique ID, the voting machine's device name as well as status bits which encoded for which poll a voter was eligible and in which polls she has already cast a vote. There were also status bits which were used during the voting process, these were set to an initial state when the card was handed out. The device name was encoded on the chip card because we originally planned to use two voting machines, and we wanted to prevent voters from copying a login card and casting a ballot on both voting machines. The data written on the memory card was digitally signed by the software of the voting authority. The voter was handed out the memory card in exchange for her student's ID card, to make sure she gave

back the memory card after the voting process. The poll workers documented in the electoral register that the voter has obtained a login card.

The voter could then enter the voting booth and cast her vote as described below. After the voting process was finished, the voter gave back her login card and in turn got back her student identity card. The poll workers checked the voting machine's signature on the card, the status bits and the bits that indicated in which poll the voter has cast a ballot. This made it possible to let voters vote in different polls on different days.

Voting Process

After registration, the voter entered the voting booth, where she was prompted by the vote casting interface to insert her login card. After the login card was inserted, the voting machine checked the device name encoded on the card, and whether a card with the same ID has already been used. The card's ID was then stored on the voting machine so the card could not be copied for voting a second time. The voting machine also checked if the status bits were in a valid initial state and if the data was signed with a valid signature from the voting authority. If one of the above mentioned checks failed, the voting machine displayed an error message, and after a few seconds returned to prompting for a valid login card.

As we will see in the description below, during the voting process, the voting machine used the status bits of the login card to encode and check the current state of the voter's voting process, more precisely, if she was still editing her ballot, or if she had already cast it. This was necessary because the chip card could have been taken out of the card reader at any time. Therefore, we had to prevent voters from pulling out their login cards too early after casting a ballot and then return a chip card to the poll workers that indicated that the voter has not voted yet in that poll. With each write access, the data on the card was digitally signed by the voting machine, and with each read access, this signature was tested by the voting machine.

After successful login, the voter could choose with which poll to start.After choosing a poll, the voting machine displayed the corresponding "ballot", consisting of a list of candidates with plus and minus buttons next to them. where the voter could distribute her votes among the candidates. The voter could cancel the voting process of her current poll at anytime, proceed with another poll, and return to the first poll later. The vote casting interface displayed to the voter at any time how many votes the voter had already distributed and how many she had left to give. It did not allow the voter to create an invalid ballot through an invalid distribution of votes, so the voter could not unintentionally cast an invalid ballot. Instead, the voter could give all her votes to a designated invalid candidate as described above. Since a ballot could either be valid or not, the voter could give either all her votes to the invalid candidate or none. The voter could also give some or all of her votes to an abstention candidate.

When the voter indicated that she was finished, the voting machine checked validity of the ballot. If the voter had not yet distributed all her votes, she was informed about this and could choose between going back to the ballot screen and distributing the remaining votes or giving them to the abstention candidate. This was necessary to not leak information in the post election proof about how many votes the voter had actually cast. Therefore, each voter had to get the same amount of fresh random numbers.

After all votes were distributed, the user interface displayed a confirmation screen where the voter could double check her choices. After confirmation, the voting machine checked the presence of the login card and its status bits before starting the ballot casting process. The status bits of the chip card were then set to a state that indicated that the voter has cast her ballot in this poll. Then the casting process was run: dummy random numbers for the receipt were chosen at random and marked as used, and the random number generator generated the fresh random numbers for the chosen candidates.

The voter's receipt was created: next to each candidate were as many random numbers as the voter could give votes to one candidate. Each candidate got as many fresh random numbers as he got votes, the rest was filled up with dummy votes. The receipt also contained a hash of the so far created receipt content as a unique ID with which the voter could later find her receipt on the bulletin board. The receipt content was then digitally signed by the voting machine and the signature also put on the receipt, to enable the voter to prove a fraud, should her receipt be missing on the public bulletin board. The hash-chain described in Section 4.2.2 was not implemented in this election, but later included in our prototype. The receipt was then printed on a sheet of A4 paper with the laser printer. This size of the receipt was necessary for the most complex poll which as explained above contained 378 random numbers. The user interface asked the voter to compare the numbers on the random number generator with the numbers on the receipt, indicating the positions of the fresh random numbers on the screen to assist the voter. After checking the receipt, the voter could clear the display of the random number generator. Otherwise, the display was cleared after a given time if the voter did not press any buttons on the random number generator, to hide the voter's fresh numbers from the next voter. To complete the vote casting process, the voter was asked to confirm the correctness of her receipt. After this, a completion flag for this poll was set on the login card. The voter could then either proceed with another poll or end the voting process. Please note that the voter obtained one receipt per poll. After the voter was finished, she gave back her login card to the poll workers who checked the completion flag for each poll and documented this in the electoral register. If the voter had not cast a ballot in each poll she was eligible for, she could come back later to obtain another login card which would unlock only the remaining polls.

4.3.3.4. Post-Voting Phase

The post-voting phase was performed together with the voting authority. To calculate the tally, the unused commitments were opened. Calculating the tally took about five minutes.

To prove that on each receipt there were only n fresh random numbers, where n is the number of votes each voter was allowed to cast, n fresh commitments were created for the voted candidates and their random numbers. These were shuffled together with the commitments to the dummy votes on the receipt, to create a prove as described in Section 2.3.6.2. For the proofs, challenges were created jointly with the voting authority. After about three hours the calculation of the proofs was finished. Checking the proofs successfully showed correctness of the tally.

4.3.4. Experiences

The election was conducted without any critical incidents. Some voters pulled out their login cards in the middle of the voting process, causing an erroneous state on

the login card. This happened a few times, but the state on the card could always be retraced and corrected without breaking privacy, and the voting machine itself was not put into an incoherent state by this. Many voters wanted to try out the new scheme, despite the parallel paper election which had a much faster voting process. The feedback of voters who voted electronically could be summarized as follows: the voter interface itself was not hard to use but the voting process could have been more comfortable.

Many voters did not bother checking the random numbers. The largest receipt contained 378 random numbers of which nine had to be compared to numbers on the random number generator. However, the display of the random number generator was very small and could only display two random numbers at a time, so the voter had to scroll to find the other numbers.

Some voters stated that the voting process took too long and that the receipt printer was too slow, or that they would have preferred a touchscreen.

The voting process took several minutes due to the slow printer and the necessary write accesses to the memory card. This problem with the memory cards could be solved with a chip card reader that pulls in the login card completely and releases it only after the voter ends or aborts her voting process.

A rather funny incident was that the voting booth was setup in front of a glass wall, which was perfectly fine for the election with paper ballots, but the glass reflected the screen of the voting computer. Fortunately, this problem was noticed before the election started, and could easily be solved by putting a movable wall between the glass and the voting booth. This showed us that it is hard to actually define all requirements towards the operational environment of a voting scheme in advance, or consider all possible side channels. It also showed us that an environment that provides security for one voting scheme is not necessarily suitable for another.

4.3.5. Analyzing this Election with the Taxonomy

Our experiences with Bingo Voting showed us that the scheme is rather flexible and, aside from solvable usability issues, can successfully be applied to a real world election. But this was only one application. For a deeper insight, we analyze the election with our taxonomy as a direct comparison to our practice experiences.

The full analysis can be found in Appendix B, this section summarizes our results. An election run with the same setup as the student parliament election would have the following properties:

Election type

Bingo voting is designed for presential elections using voting computers.

Preliminaries and Assumptions

The security of Bingo Voting relies on the following assumptions:

- The parameters for the Pedersen commitments must be created by a trusted authority and the discrete logarithm problem must be hard in the used group.
- The voting machine and its administrators need to be trusted for privacy.
- Verifiability and privacy rely on a trusted random number generator.
- A voting booth is provided as a private channel between the voter and the voting computer.

- The probability of collisions between random numbers is sufficiently small.
- Dummy random numbers are indistinguishable from numbers generated by the trusted random number generator.

What makes this scheme secure?

Verifiability of Bingo Voting relies on a trusted random number generator which can be very simple hardware and is therefore easier to audit than a more complex voting machine. To achieve individual verifiability, the voter gets a receipt with which she can immediately check the correctness of her ballot. But to achieve privacy, the voter's choice is hidden between dummy votes which conceal the real vote.

Category 1: Privacy and Coercion-Resistance

Provided that the underlying assumptions hold, Bingo Voting achieves everlasting privacy and receipt-freeness. Coercion-resistance is met except for the scheme's vulnerability to forced abstention attacks, which it has in common with any presential election.

Category 2: Correctness and Verifiability

As long as there are no collisions between random numbers, a manipulation can always be detected.

Category 3: Fairness

Assuming that enough auditors test the user interface, the election is fair.

Category 4: Provability of a fraud

Under the assumption that there are no collisions between random numbers, all manipulations are always detectable and provable at most times, but the cause of an error is not always tracable and error correction is not always possible. Approaches to solve this problem are discussed in [BHK+09] and [Hen12].

Category 5: Robustness and Scalability

With the voting machine as a single point of failure, the lack of error correction possibilities in some cases and the big amount of published data for verification, robustness and scalability are clearly the weak points of the scheme. Some of these problems could be met as discussed in the full analysis in Appendix B, Section B.3.

4.3.6. Discussions about the Election's Security

In the biggest poll, the voter could distribute nine votes among 72 candidates, distributed to several lists. Due to the many possibilities to fill out a ballot, pattern voting attacks could have been done easily in the paper election, where the full plaintext ballots were visible in the tallying process. This was not possible with the electronic election because votes were not counted ballot-wise and the voting pattern was not visible on the ballots. Even undervotes were not visible since non-distributed votes were given to the abstention-candidate automatically, so there was the same amount of fresh random numbers on each receipt. Therefore, the electronic election with Bingo Voting actually offered more coercion-resistance than the paper election.

The receipt-proofs were zero-knowledge, so they did not yield any information about the voters' choices.

Had the voting machine put an invalid signature on the receipt, the voter would have had no proof of manipulation. In theory, she could have checked the signature immediately, but in practice, this would have required additional designated hardware in the voting booth. Other methods of achieving the provability of manipulations are discussed in [Hen12] and [BHK+09].

4.3.7. Conclusion and Possible Improvements

The application of Bingo Voting showed that the scheme can handle complex elections in a real world environment. It also showed that steps that are often assumed as given in the description of a voting scheme, like the login process of the voter at the voting machine, are not necessarily trivial.

While the scheme worked as supposed, usability was far from perfect. By now, a new prototype exists that has a touchscreen and a much faster printer. A new user interface has been implemented in the context of a software engineering course held at the Institute of Cryptography and Security, now part of the Institute of Theoretical Informatics, of the Karlsruhe Institute of Technology. The new software supports write-in candidates using the technique described in Section 4.4. Techniques to decrease the amount of published data, mostly consisting of the proofs of correctness, were discussed in [Hen12]. For a use of the scheme in a bigger election it would be preferable to provide error traceability and provability of a fraud for each possible manipulation or error while protecting privacy. As mentioned above, approaches for this were discussed in [BHK+09] and [Hen12].

The assumption of a trusted voting machine for privacy is a rather strong one. It can barely be achieved for a complex system like an ordinary PC without making sure that the voting machine is only accessed under observation of independent parties. To improve trustworthiness of the voting machine, a modular design would be of advantage where tasks are distributed among components, such that no component with a significant amount of memory learns any secrets.

Another step towards trustworthiness would be following the suggestion of Bohli et al. [BHK+09], who suggested creating receipt proofs immediately after each voting process and deleting the according dummy votes afterwards.

4.4. Bingo Voting with Write-in candidates

The previous sections showed that Bingo Voting allows cumulation and vote-splitting without being vulnerable to pattern voting attacks.

However, for some elections, this flexibility is not enough. There are elections, like the municipal election in Takoma Park [CCC+10], where it is required by law that a voter has the chance to cast a vote for a candidate that is not on a predefined candidate list, a so-called *write-in candidate*. In this section, we adapt Bingo Voting to allow write-in candidates.

The construction introduced here has been published in [Kem12]. It does not require new techniques and is actually quite generic: the basic idea is to add an additional candidate called “Write-In”, which is treated as a regular candidate. The “normal” then yields the overall number of write-in votes, as well as the tally for the list candidates. The write-in votes are then counted in a separate tally. Similar approaches for other voting schemes have been suggested in [KY04], [CCC+10] and [Adi06].

As in [KY04], our scheme does not show whether a voter has voted for a write-in candidate or chosen a candidate from the candidate list. At the same time, our

C_1	$\mathtt{R}_1$
C_2	$\mathtt{R}_2$
...	...
C_n	$\mathtt{R}_n$
$WriteIn$	$\mathtt{R}_{n+1}$
com("Na Me")	

Figure 4.4.: Scheme of a receipt with write-in support: the with write-in candidate "Na Me" was chosen.

scheme is maximally flexible: we can easily allow more than one write-in candidate, or allow vote-splitting and cumulative voting while allowing the voter to vote for both write-in candidates and candidates from the list in one voting process, with one ballot and one receipt.

This is achieved with almost no additional effort for the voter. In fact, a voter who does not write in a candidate can do exactly the same as in the original Bingo Voting scheme. Apart from a voluntary testing of the voting process her only additional effort compared to a pen-and-paper election is the also voluntary comparison of a random number. As mentioned above, an example implementation of Bingo Voting with write-in support was produced by students in a software engineering course held at the Karlsruhe Institute of Technology.

4.4.1. Preconditions

The same preconditions hold as in the original Bingo Voting scheme, which is described in Section 4.2.1.

4.4.2. Pre-voting Phase

To be able to include a write-in candidate into the Bingo Voting election process, we use an additional candidate named "Write-In" as a wildcard. The pre-voting phase is performed as in the original scheme, described in Section 4.2.1. The new candidate "Write-In" gets as many dummy votes as voters are allowed to cast votes for write-in candidates, multiplied by the number of eligible voters. For the sake of simplicity we describe our technique for one write-in candidate and a one out of n election.

4.4.3. Voting Phase

In the voting booth, the voter can mark her choice as she would in the original scheme, except that she now has the additional possibility to write in a candidate. In the original scheme, the voter's receipt is printed and the vote casting process is started immediately after the voter has marked and confirmed her choice. This is different here. To be able to count the write-in votes, a Pedersen commitment to the write-in candidate's name is created and included in the voter's receipt, as sketched in Figure 4.4. Apart from the commitment, the receipt looks like a receipt in the original scheme. To conceal the fact whether a voter has voted for a write-in candidate, we include such a commitment in any case. If no write-in candidate is chosen, the voting machine commits to "No write-in" instead.

Before generating the final receipt, we include an additional testing phase in which the voter can check the content of the commitment, following the idea of voter initiated auditing [Ben06]: the voter can either test or accept the commitment.

Only after accepting a commitment, the vote casting process is initiated, which is performed as in the original scheme.

The following steps describe the voting process in more detail:

1. **Login:** The voter enters the voting booth and logs in at the voting machine (we assume an existing login mechanism).

2. **Ballot marking:** The voter marks her choices, the ballot can be revised as often as the voter wishes. The user interface additionally provides the possibility to write in a candidate. How this is implemented is arbitrary.

3. **Confirmation:** After the voter is finished marking her ballot, she confirms her choice.

4. **Commit:** The voting machine creates a commitment to the write-in candidate's name. If the voter's choice is not a write-in candidate, the commitment is to the string "No Write-In". The printer prints the commitment.

5. **Test:** Following the idea of voter-initiated auditing [Ben06], the voter can now choose to either test the commitment or cast her vote.
   - **Voter chooses to test:** In the case of testing, the vote does not count, the unveil information and the supposed content of the commitment is printed and the voting process begins anew at Step 2.
   - **Voter chooses to cast:** The printed commitment is not opened, the voting process proceeds with step 6.

6. **Cast:** The voting casting process is performed as in the original scheme:
   a) The trusted random number generator creates the fresh random number representing the vote.
   b) Dummy votes for the not-voted candidates are chosen for the receipt.
   c) The rest of the receipt (additionally to the unopened commitment) is printed.
   d) The voter can compare the random number on the trusted random number generator's display with the random number next to her chosen candidate. If the voter voted for a write-in candidate, the fresh random number appears behind "Write-In".

7. **Receipt verifiation:** The voter leaves the voting booth and takes her receipt home for later checking, see the description of the post-voting phase. She can also take home all printed commitments from the testing phases together with their printed unveil information, to test them later.

The printed commitments created in Step 5 and their corresponding unveil information can be checked offline, either by the voter herself or by a competent person or institution of her choice. Please note that we do not have to make any additional assumptions about the printer. The dummy votes are chosen and sent to the printer only after the voter has accepted the commitment. Furthermore, the voter does not have any influence on the receipt or any published data after accepting a commitment. Therefore, unlike in the scheme of Moran and Naor [MN06], the voter can see what is printed anytime and take away the print-out anytime.

Main Tally:

Candidate	Number of votes
John	9
Joe	7
Write In	3

Figure 4.5.: Main tally: There are 3 votes for write-in candidates.

Tally of the write-in candidates:

Write-in candidate	Number of votes
Alice	2
Bob	1
No Write-In	16

Figure 4.6.: Tally of the write-in candidates: The number of "votes" for No Write-In has to match the number of non-write-in votes.

4.4.4. Post-voting Phase

After the voting phase, the results of the main tally and the tally of the write-in candidates are published, as sketched in Figure 4.5 and Figure 4.6, together with a proof of their correctness.

The tally consists of two parts: a tally of the list candidates and a tally of the write-in votes. The tally of the list candidates also contains the overall number of write-in votes. It is computed as in the original scheme (see Section 4.2.1) by counting the unused dummy votes for each candidate. The correctness of this tally part is proven as in the original scheme: all receipts are published, and for each receipt, a proof is computed that it contains exactly one fresh random number. The Tally of the write-in candidates is computed by shuffling and opening the write-in commitments on the published receipts, and proving the correctness of the shuffle. For this, the same shuffle technique can be used as for the proofs of correctness of each receipt. The number of commitments to "No Write-In" has to equal the number of all votes minus the number of votes for write-in candidates.

4.4.5. Privacy and Coercion-Resistance

Bingo Voting with write-in candidates provides the same amount of voter privacy as the original scheme, with the exception that the voter can be coerced to write in a certain random string, or write a certain candidate in a distinguishable way. This is a problem of all voting schemes that allow write-in candidates if the tally is published one-to-one. This problem is addressed in the following two sections. An advantage of our scheme is that the voter's receipt as well as any published information does not reveal whether a voter voted for a write-in candidate or a list candidate.

Everlasting privacy holds under the same assumption as in the original scheme, since for the commitments to the write-in names, Pedersen-commitments are used, which are unconditionally hiding.

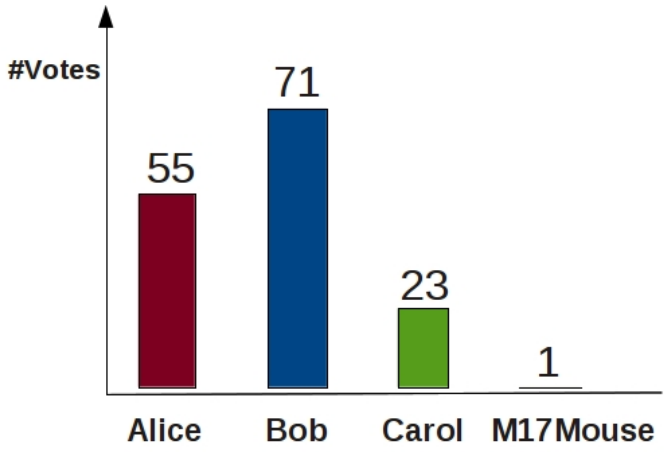

Figure 4.7.: Accurate tally representation: Everyone sees that M17Mouse got exactly one vote.

4.5. Fuzziness: Coercion-Resistant Elections with Write-In Candidates

In some elections, it is required by law that voters can vote for a write-in candidate. However, of the security definitions reviewed in Section 3.3, only the definitions of Küsters et al. [KTV12] and Teague et al. [TRN08] can be reasonably applied to elections with write-in candidates, since only these definitions measure external information that the adversary learns regardless of the used election scheme. But these definitions would most likely show for most election schemes that they are not very coercion-resistant when applied to elections with write-in candidates. The reason for this is that the external information leaks too much about the voter's choice, as can be seen in Figure 4.7: the voter can be coerced to write in a certain rarely elected candidate or a random string chosen by the adversary, who will see this in the election result. Most existing definitions of coercion-resistance disregard or even explicitly exclude write-in candidates, arguing that an election allowing them can never be fully coercion-resistant. The tally is usually considered public, and as stated by Juels et al. [JCJ05], the above mentioned attacks cannot be prevented as long as the tally is published directly.

But it is possible for write-in supporting schemes to be coercion-resistant if only a fuzzy version of the tally is published. In real world elections a fuzzy representation of the tally is often sufficient. Examples for such fuzzy representations are the representation of the tally in percent, as in Figure 4.8, or the resulting number of parliamentary seats for each candidate. In this case, if a candidate only got zero or one votes, it is sufficient that the representation of the tally shows that the number of votes for this candidate is less than a certain threshold, instead of showing whether this candidate got a vote or not. However, the added fuzziness should not be greater than necessary. This Section takes a closer look at this fuzziness. An arising problem is that if the tally is published fuzzy, a verifiable election scheme has to prove the correctness of the fuzzy tally without revealing the exact tally.

We provide a formalization of fuzzy tally representations that enables reasonable definitions of coercion-resistance to be achieved by write-in supporting verifiable election schemes without weakening that definition.

Our notion can be seen as an add-on to existing and upcoming models, bridging

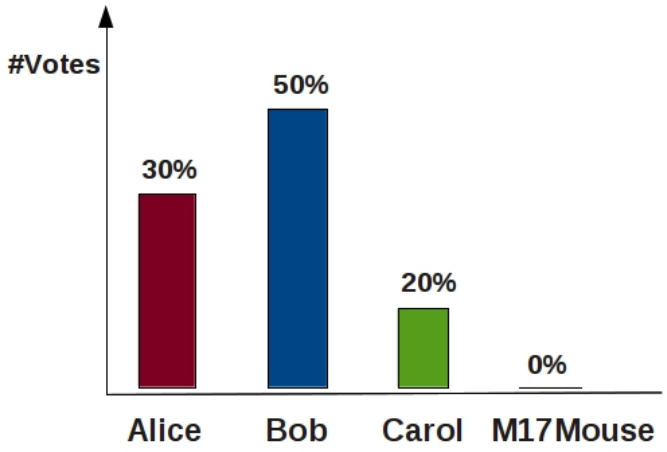

Figure 4.8.: Tally representation that does not show whether M17Mouse got a vote, or if someone just asked the voting authority to list it in the tally while voting for another candidate.

the gap between write-in candidates and coercion-resistance. The idea of its usage is as follows: consider a voting scheme with verifiable correctness that is coercion-resistant (by any definition), and provides a proof of correctness of the tally which can be adapted such that it proves the correctness of the fuzzy tally representation without revealing the exact tally. Applied to an election where the tally can be published fuzzy according to our definitions of fuzzability, the adapted scheme should still achieve the same coercion-resistance while additionally being resistant against forced abstention caused by coercing voters to vote for a rarely elected candidate. This holds even if the scheme supports write-in candidates, assuming that a voter can add names to the representation of the tally without voting for (i. e. losing her vote for) them.

Some remarks: on a first glance, there seems to be a simple countermeasure to the attack that a voter is forced to vote for an unlikely or fictional write-in candidate in opening and counting write-in votes only if they could make a difference in the result. First of all, this is also a fuzzy representation, but without any formalization. Second, on a closer look, this is not as trivial as it seems. To prevent voters from being coerced to write-in a name instead of voting for a list candidate, the voting process must not leak the information whether a voter voted for a write-in candidate. At the same time, to preserve verifiable correctness, there needs to be a proof that these unopened votes are really for write-in candidates and that they really make no difference in the result. This is especially a problem if voters can validly write in names of list candidates. By contrast, our techniques prove the correct counting of write-in votes without revealing information about a single voter's choices.

The attack of forcing the voter to vote for a certain string cannot only be used as a forced abstention, but also as identifying a ballot that contains a vote for more than one candidate. The problem is more crucial when the voter does not cast a one out of n choice: the voter can encode her ID with the write-in candidate on the ballot, and the adversary see how the rest of the ballot is filled out. It is hard to prevent this when opening plaintext ballots, for example when the tally is computed by a mixnet. We leave this as an interesting open problem, our techniques concentrate on one out of n choices.

4.5.1. A Definition of Fuzziness

In this section, we introduce two concrete approaches to blurr the tally result in a way that the voter's choice is sufficiently hidden, without giving up too much accuracy of the real tally. Our definition of fuzziness is motivated by k-anonymity [CdVFS07] and l-diversity [MKGV07], two notions of database privacy, in which an entry is hidden between k elements, that have at least l different values. We do the same with the voter's choice: it leads to one out of a set of k possible tally results, in which the number of votes for each candidate can have l different values. This section first provides a rather strong definition of fuzziness and then a weaker version of this definition in which the fuzziness/accuracy-trade-off is adjustable in a more fine-grained way. The formalization of fuzziness introduced here is seen as an add-on to existing (or upcoming) models. Therefore it is phrased very general, without commiting to a specific model of voting schemes.

We differentiate between the *tally* and its *representation*. We assume the tally itself to remain secret, while instead a representation of the tally is published.

We denote the set of all possible tally results of a given election by $\mathcal{T}$. This set strongly depends on parameters given by the election, like the number n of eligible voters, the set of candidates that can be voted, both list candidates and write-in candidates, and on election rules like the number of votes each voter can cast.

A *representation* R can be seen as a view that represents a certain subset $T_R \subset \mathcal{T}$ of tallies. The possible representations of the tally are also co-determined by the election since the election defines the needed accuracy with which the tally has to be represented.

Definition 4 (complete) *We call a set $\mathcal{R}$ of representations* complete *for a set of tallies $\mathcal{T}$ if $\bigcup_{R\in\mathcal{R}} T_R = \mathcal{T}$, where T_R denotes the set of tallies represented by a representation $R \in \mathcal{R}$.*

Hence a set of representations is complete if each possible tally has a representation.

We henceforth assume that the tally is a vector of length n where n is the number of candidates. The ith entry in this vector corresponds to the number of votes for the ith candidate.

Definition 5 (δ-neighbored) *Let A and B be two vectors of length n. We call A* δ-neighbored *to B, if a vector $X = (x_1, x_2, \ldots, x_n)$ exists with $\sum_{i=1}^{n} |x_i| \leq \delta$ and $A = B + X$.*

Definition 6 (μ, δ-fuzzability) *Let $\mathcal{T}$ be the set of all possible tallies of a given election. We call this election μ, δ-*fuzzable *if there is a set $\mathcal{R}$ of representations which is complete for $\mathcal{T}$, and for each representation $R \in \mathcal{R}$ the following holds for its set of represented tallies $T_R \subseteq \mathcal{T}$:*

1. *For each pair of tallies $(T_1, T_2) \in T_R \times T_R$ it holds that T_1 is δ-neighbored to T_2.*
2. *Let E_i be the set of all entries that occur at position i within any tally-vector in T_R. Then $|E_i| \geq \mu$ for $i = 1, \ldots, n$.*

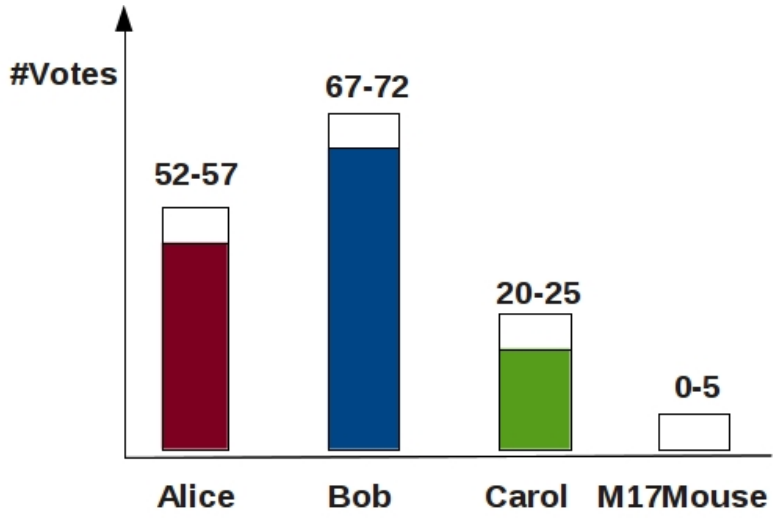

Figure 4.9.: Fuzzy tally representation for $\delta = 20, \mu = 5$

Intuitively, the second requirement states that the number of votes of each candidate can have at least μ different values encoded in each representation, as sketched in Figure 4.9.

Let in the following an *election protocol* denote the protocol that is specified by an *election scheme* to execute an election, i. e. the election protocol specifies all subprotocols for running the pre-voting phase, all voting processes and the post-voting phase. A *protocol run* of an election protocol denotes the execution of one whole election with this election protocol, i. e. a pre-voting phase, a voting period where a set of more than μ voters cast their ballots, and a post-voting phase where the tally and its representation is computed and its proofs of correctness published. Let the *public view* $\mathcal{V}$ of a run of the election protocol denote the public information the election protocol creates during that protocol run. It includes the tally representation, all public proofs of correctness etc.

Definition 7 (μ, δ-fuzzy election scheme) *We call an election scheme μ, δ-*fuzzy*, if applied to a μ, δ-fuzzable election, for each possible protocol run of the election leading to a public view $\mathcal{V}$ and a tally T with representation R which represents a set T_R of tallies, the following holds:*

1. *The proofs of correctness in $\mathcal{V}$ prove that $T \in T_R$.*
2. *There is a subset $M \in T_R$ such that*
   a) *For each pair of tallies $(T_1, T_2) \in M \times M$ it holds that T_1 is δ-neighbored to T_2.*
   b) *Let E_i be the set of all entries that occur at position i within any tally-vector in M. Then $|E_i| \geq \mu$ for $i = 1, \ldots, n$.*
   c) *For each $T_i \in M$, there is a possible election protocol run which leads to tally T_i and representation R, and a public view that is indistinguishable from $\mathcal{V}$.*

Particularly, if $\mu > 0$, the proof of correctness must not yield the tally itself. This means that voting schemes whose proofs of correctness can only be verified with knowledge of the exact tally cannot comply with the above definition unadapted.

4.5.2. Weak Fuzziness

To prevent forced abstention, full μ, δ-fuzzability is not generally required. We usually do not have to conceal whether a candidate got, say 150 or 151 votes. In some cases we even must not conceal this because the order of the most voted candidates is an important tally outcome. If we just want to conceal if a candidate got zero votes or more, a weaker definition as follows might be more suitable.

Definition 8 (weak μ, δ-fuzzability) *Let $\mathcal{T}$ be the set of all possible tallies for a given election. We call this election* weak μ, δ-fuzzable*, if there is a set $\mathcal{R}$ of representations that is complete for $\mathcal{T}$ and the following holds for each tally $T \in \mathcal{T}$: let $T = (x_1, \ldots, x_n)$. For each representation $R \in \mathcal{R}$ that represents T, meaning $T \in T_R$, the following holds:*

1. *For each pair of tallies $(T_1, T_2) \in T_R \times T_R$ it holds that T_1 is δ-neighbored to T_2.*

2. *For each $i \in \{1, \ldots, n\}$ with $x_i < \mu$: let E_i be the set of all entries that occur at position i within any tally-vector in T_R. Then $|E_i| \geq \mu$.*

Intuitively this means that if a candidate got less than μ votes, then each representation of the tally indicates μ other possible numbers of votes for this candidate.

Please note that μ, δ-fuzzability implies weak μ, δ-fuzzability: given μ, δ-fuzzability, for each representation R there occur μ different outcomes in T_R for each candidate, in particular for these having less than μ votes.

Definition 9 (weakμ, δ-fuzzy election scheme) *We call an election scheme* weak μ, δ-fuzzy*, if applied to a μ, δ-fuzzable election, for each possible protocol run of the election leading to a public view $\mathcal{V}$ and a tally T with representation R which represents a set T_R of tallies, the following holds:*

1. *The proofs of correctness in $\mathcal{V}$ prove that $T \in T_R$.*

2. *There is a subset $M \in T_R$ such that*

   a) *For each pair of tallies $(T_1, T_2) \in M \times M$ it holds that T_1 is δ-neighbored to T_2.*

   b) *For each $i \in \{1, \ldots, n\}$ with $x_i < \mu$: let E_i be the set of all entries that occur at position i within any tally-vector in M. Then $|E_i| \geq \mu$.*

   c) *For each $T_i \in M$, there is a possible election protocol run that leads to tally T_i and representation R, and a public view that is indistinguishable from $\mathcal{V}$.*

The intuition behind this definition is that if an entry in the tally vector is less than μ, it "hides between" at least $\mu - 1$ other values that are all mutually different. Since we only need to hide whether a voter voted for a certain candidate or not, the two parameters δ and μ should be a little bit greater than the number of voters under control of the adversary. For real world elections, parameters δ and μ less than 10 seem reasonable. However, if to prevent group coercion, a higher value for these parameters should be considered.

4.6. Including Fuzziness in Election Schemes

This section introduces a general construction of μ, μ-fuzzy verifiable election schemes from voting schemes based on mix-based or homomorphic tallying. A former version of these schemes is published in [Kem12]. Our construction only affects the post-voting phase of the voting scheme that is to be made fuzzy. Precalculations and vote casting are performed as in the original schemes. The only additional assumption our construction requires is a set of trustees that can jointly compute and learn the exact tally, which in the original schemes would have been public.

The descriptions below assume that the voting schemes are applied to a μ, μ-fuzzable election with n candidates $C_1, \ldots, C_n$ and one vote per voter. For the sake of clarity, our techniques are described without considering a special treatment of write-in candidates. But we do assume that if write-in candidates are supported, the voter has the possibility to ask a voting authority via a private channel to include candidates in the tally representation, so she can vote for another candidate while the candidate she was coerced to vote still appears in the representation of the tally. The reason for this is that public data should not reveal a voter's choice by including a name in the representation that had otherwise not been there. Whenever a zero-knowledge proof of correct shuffling is computed, the shuffle technique with shadow mixes described in Section 2.3.6.2, [Ben06] and [Adi08] can be used.

Since vote-casting and pre-election phases stay unaffected, we only describe the new post-voting phases.

4.6.1. General Construction of μ, μ-Fuzzy Voting Schemes with Homomorphic Tallying

In this section, a verifiable μ, μ-fuzzy election scheme is constructed from an arbitrary verifiable election scheme that is based on homomorphic tallying, provided that the original scheme publishes all cast ballots in encrypted form on a public bulletin board and homomorphically adds these ballots to compute the tally.

The idea of our technique to blur the tally is to assign each possible tally result to a bucket of size $n \times \mu$, where n is the length of the tally vector. Let E be the additive homomorphic encryption function used to encrypt the ballots in the original scheme. We assume E to be probabilistic[2], so implied by its homomorphic property it supports re-encryption. For each candidate C_i that is to appear in the tally, the trustees create μ fake ballots $B_{C_i,0}, \ldots, B_{C_i,\mu-1}$, where for all $i = 1, \ldots, n$ and $j = 0, \ldots, \mu - 1$, $B_{C_i,j}$ is a "ballot" that gives $-j$ votes to candidate C_i and 0 votes to all other candidates. These ballots are encrypted to optain n sets of ciphers $\{c_{i,1}, \ldots, c_{i,n}\} := \{E(B_{C_i,0}), \ldots, E(B_{C_i,\mu-1})\}$ for $i = 1, \ldots, n$. The fake ballots and their encryptions are published with a proof of correct encryption. We call these encrypted ballots $c_{i,j}$ the *blurrers* of candidate C_i. Each candidate's blurrers are then shuffled and reencrypted, and one blurrer of each candidate is included in the final result, to blurr the tally in a controlled way. The blurring and tallying process is now described in more detail. In the following, the trustees are assumed to depend the blurrers, and all additional steps performed to blur the tally (i. e. shuffle proofs of the reencrypted blurrers etc.), solely on the tally vector, and not on individual ballots or any other data. They are also assumed to do all reencryptions honestly in the

[2]This is not a strong assumption. Ballots generally constitute a very small plaintext space. Would they be encrypted deterministically, everyone could find out their contents by encrypting all possible ballots and testing the outcome.

sense that they do not encode information about the exact tally into any published data. After all votes are cast, the trustees compute the tally representation and its proof of correctness as follows:

1. The trustees compute the tally $T = (T_1, \ldots, T_n)$ in secret (including an entry for each candidate they were asked by voters to include in the tally representation, should write-in votes be supported).

2. For each candidate that is to appear in the tally representation, the trustees create μ blurrers $\{c_{i,0}, \ldots, c_{i,\mu-1}\} := \{E(B_{C_i,0}), \ldots, E(B_{C_i,\mu-1})\}$, as described above, publish each set $\{c_{i,0}, \ldots, c_{i,\mu-1}\}$ together with $\{B_{C_i,0}, \ldots, B_{C_i,\mu-1}\}$, and prove that $c_{i,j} = E(B_{C_i,j})$ for all $j = 0, \ldots, \mu - 1$. The trustees are assumed to depend the calculation of the blurrers solely on the list of candidates that are to be included in the tally.

3. For each candidate C_i, the trustees do the following:
   a) They secretly compute $k_i := T_i \mod \mu$, where T_i is the entry in the tally vector that corresponds to candidate C_i. Candidate C_i's real tally outcome T_i will later be blurred by k_i.
   b) They compute a shuffle of the blurrers of Candidate C_i by permuting them with a permutation π and reencrypting them, resulting in a list of shuffled blurrers $(E'(B_{C_i,\pi(0)}), \ldots, E'(B_{C_i,\pi(\mu-1)}))$, where for each j, $E'(B_{C_i,\pi(j)})$ is a reencryption of $E(B_{C_i,j})$.
   c) They publish the list $(E'(B_{C_i,\pi(0)}), \ldots, E'(C_i, B_{\pi(\mu-1)}))$ with a zero knowledge proof of correct shuffling.
   d) They pick out $e_{C_i} := E'(B_{C_i,\pi(k_i)})$ to later include it in the tally. This will subtract k_i votes from candidate C_i. Only the trustees know the permutation, and therefore know which blurrer is chosen. Everyone else only sees that e_{C_i} contains a ballot that subtracts between 0 and $\mu - 1$ votes from candidate C_i.

4. The trustees include the blurrers $e_{C_1}, \ldots, e_{C_n}$ for all candidates $C_1, \ldots, C_n$ into the set of cast ballots, by publishing them. Everyone can check that there is exactly one blurrer of each candidate.

5. The tally is computed by homomorphically adding the encrypted real votes and the chosen blurrers $e_{C_1}, \ldots, e_{C_n}$.

6. The sum is decrypted to optain the representation $R = (R_1, \ldots, R_n)$, which is published as the tally representation.

7. The correct decryption of the sum is proven, as would have been done in the original scheme.

Verifiable Correctness

The representation of the tally is the result $R = (R_1, \ldots, R_n)$ of the decryption of the sum of all ballots, including the blurrers. Everyone can check that in Step 4/5, there is one blurrer included in the tally for each candidate, by checking the shuffle proofs of the blurrers for each candidate. So anyone can check that the number of votes for each candidate results from subtracting a value between 0 and $\mu - 1$ from

the candidate's actual number of votes. Therefore, everyone is convinced that the real tally T lies between $(R_1, \ldots, R_n)$ and $(R_1 + \mu - 1, \ldots, R_n + \mu - 1)$, without knowing the exact tally T.

Individual verifiability is as in the original scheme, since the vote casting process stays the same, and the cast votes are published on the bulletin board as in the original scheme.

Coercion-resistance

Since per definition, the blurring procedure solely depends on the tally and not on any other data created in the election process, especially not on individual votes, it can not yield more information about them than the tally would. Therefore, the fuzzy scheme is at least as coercion-resistant as the original scheme.

Fuzziness

The above described construction provides μ, μ-fuzziness: each entry in R is a multiple of μ, and per construction each tally result T' that lies between $(R_1, \ldots, R_n)$ and $(R_1 + \mu - 1, \ldots, R_n + \mu - 1)$ has the representation $(R_1, \ldots, R_n)$. So R implies μ different possible values for each entry in T, and all tallies represented by R are μ-neighbored. The corresponding views are indistinguishable for the adversary: for each possible tally $T' = (T'_1, \ldots, T'_n)$ represented by R, it holds that

$$R_i \leq T' \leq R_i + \mu - 1$$

for all $i = 1, \ldots, n$. In Step 2, a set of blurrers $\{c_{1,0}, \ldots, c_{1,\mu-1}, \ldots, c_{n,0}, \ldots, c_{n,\mu-1}\}$ is created. For each possible tally T' there is a subset of blurrers $\{c_{0,x_1}, \ldots, c_{n,x_n}\}$ that would lead to the representation R when chosen in Step 3: the set $c_{0,x_1}, \ldots, c_{n,x_n}$ with

$$x_i := T'_i \mod \mu$$

for each $i = 1, \ldots, n$. Since the proves of correct shuffling in Step 3 are zero knowledge proofs, the adversary cannot distinguish which blurrers were chosen. Therefore, these proofs contain no information about T and are indistinguishable for all possible tallies T' represented by R. Therefore, μ, μ-fuzziness holds.

Weak fuzziness

A weak μ, μ-fuzzy version of this scheme can be created by only including blurrers of "critical" candidates in the tally, i. e. candidates that have less than μ votes. Please note that if the overall number of cast votes is public, at least two candidates have to be blurred.

Write-in support

Write-in support can be included straightforwardly by using a hybrid approach, which tallies the list candidates homomorphically and the write-in candidates with a mix-based scheme, as in [KY04]. Fuzziness can be provided by combining the homomorphic construction described above with the mix-based construction described below.

4.6.2. General Construction of μ, μ-Fuzzy Mix-Based Voting Schemes

The construction of a μ, μ-fuzzy mix-based scheme is a bit more tricky. In schemes with mix-based tallying, ballots are published on a public bulletin board in some encrypted form. These ballots are then shuffled with a proof of correct shuffling

to be unlinked from the voter's identity, and decrypted to obtain the tally. Some voting schemes use a decryption mix where decryption and mixing is computed in one procedure. For our construction, these mixes have to be substituted by a mix that only permutes and rerandomizes the ballots, but does not decrypt them, for example a re-encryption mix as described in [Ben06].

A naive way to construct a μ, μ-fuzzy version of a mix-based voting scheme is to follow the technique of the construction above for homomorphic schemes, and instead of subtracting blurrers, take out votes for each candidate. But since the number of overall ballots is known, the number of taken out ballots would also be known and reveal information about the tally. To achieve representations where each entry is a multiple of μ, it could happen that less than μ ballots are taken out, which would prevent μ, μ-fuzziness. Instead of using this bucket approach, we subtract a "fuzziness vector" $f = k_1, \ldots, k_n$ from the tally by taking out ballots, but the number of taken out votes per candidate does not equal $T_i \mod \mu$ for all entries T_i in the tally. Therefore, in this construction, the same tally can lead to different representations.

For the algorithm to terminate, we require that more than 2μ votes are cast. Our blurring technique is described for elections where each ballot contains a one out of n choice, i. e. gives one vote to one out of n candidates. Since plaintext ballots are revealed, the problem of pattern voting remains, and the construction would not make sense for other ballot formats.

To compute the representation of the tally, the trustees do the following:

1. The trustees publicly shuffle all cast encrypted ballots, but without opening them, resulting in a set L of shuffled and reencrypted ballots. We assume that the voter's identities are detached from the encrypted ballots before mixing, i. e. the elements in L are unlinked from the voter's identities.

2. The correctness of the shuffle is proven with a zero knowledge proof, for example with the technique introduced in [Ben06].

3. The trustees then open the ballots in secret and compute the tally $T = (T_1, \ldots, T_n)$, including an entry for each candidate name that should be included in the representation. Entry T_i equals the number of votes for candidate C_i.

4. To choose a fuzziness vector $f = (k_1, \ldots, k_n)$, the trustees do the following (in secret) for each candidate C_i, $i = 1, \ldots, n$:
   - If $T_i < \mu$, they compute $k_i := T_i \mod \mu$.
   - If $\mu \leq T_i < 2\mu$, they choose a random k_i with $0 \leq k_i \leq T_i \mod \mu$.
   - If $T_i \geq 2\mu$, they choose a random k_i with $0 \leq k_i < \mu$.

5. If $\Sigma_{i=1}^{n} k_i < \mu$ or $\Sigma_{i=1}^{n} \mu - k_i < \mu$, the trustees repeat Step 4. Otherwise, they proceed with Step 6.

6. For each candidate C_i, $i = 1, \ldots, n$, the trustees do the following (in secret):
   a) They take out k_i encrypted ballots of the set L that contain a vote for candidate C_i.

b) They create $\mu - k_i$ encrypted ballots for candidate C_i, such that the created plaintext ballots are indistinguishable from real ballots.

7. Let M be the set of encrypted ballots that where taken out in the previous step, and L_R the resulting reduced set $L_R = L \setminus M$ of encrypted ballots. The trustees publish M, without showing which or how many of these ballots were for which candidate.

8. Let N be the set of newly created ballots from Step 6. b). The trustees publish N, again without revealing which or how many ballots belong to which candidate.

9. The trustees shuffle $M \cup N$, to obtain a new set of encryptions, in which it cannot be seen which ballots are cast ballots and which ones were newly created. The correctness of the shuffle is proven with a zero knowledge proof.

10. The shuffled ballots from Step 9 are opened, there should be μ ballots for each candidate. This proves that of each candidate, no more than μ ballots were taken out of L.

11. The ballots in the reduced set L_R are opened, the resulting tally is the representation of the real tally. The resulting representation is $(R_1, \ldots, R_n) = (T_1 - k_1, \ldots, T_n - k_n)$.

Verifiable Correctness

The representation of the tally is $R = (R_1, \ldots, R_n) = (T_1 - k_1, \ldots, T_n - k_n)$. In steps 7-10, everyone can verify that of each candidate, at most $\mu - 1$ votes were taken out. So anyone can see that the exact tally is a value between $(R_1, \ldots, R_n)$ and $(R_1 + \mu - 1, \ldots, R_n + \mu - 1)$, without knowing the exact tally T.

Individual verifiability is as in the original scheme. The process of vote casting and publishing encrypted votes on the bulletin board is not changed by the blurring process.

Coercion-resistance

The blurring process only depends on the tally, and on information that would be published in the original voting scheme. So the fuzzy version is at least as coercion-resistant as the original voting scheme.

Fuzziness

The tally T lies between $(R_1, \ldots, R_n)$ and $(R_1+\mu-1, \ldots, R_n+\mu-1)$. The possible tallies represented by R are all μ-neighbored. There are at least $n \cdot \mu \geq \mu$ tallies that could have let to this representation. The overall number of taken out votes is visible, but the choice of k_i in Step 4 is constructed in a way that μ different values are possible for each candidate. The corresponding views are again indistinguishable for the adversary: the only difference to the original scheme is the taking out of ballots and the creation of new ballots, that are per construction indistinguishable from cast ballots. The adversary only sees a zero knowledge proof that of each candidate, between 0 and $\mu-1$ ballots were taken out. Because of the zero knowledge property, these proofs contain no information about k_i itself.

Therefore, μ, μ-fuzziness holds.

Write-in support

Write-in support can be included analogous to the write-in supporting fuzzy Bingo Voting scheme, which will be described in Section 4.6.3.2 using an additional "regular" candidate "Write-In" and two tallies with their own representations.

4.6.2.1. Weak μ, μ-fuzzy mix-based voting schemes

A weak μ, μ-fuzzy election scheme can be created in the same way by only taking out ballots of "critical" candidates, i. e. those who have less than μ votes. Since the overall number of cast votes is public, at least two candidates have to be blurred. The vector f determines in which way the tally is made fuzzy, and is kept secret.

Like in the previous section, consider a μ, μ-fuzzable election with one vote per voter, and n candidates $C_1, \ldots, C_n$.

Again, the blurring only effects the post-voting phase of the election. It is done as above, except that instead of filling up the taken out ballots to μ per candidate, we fill them up with less ballots if we want to blurr the result of certain candidates less. The difference is that to obtain weak μ, μ-fuzziness, a second vector $d = (d_1, \ldots, d_n)$ is chosen. Let $T = (T_1, \ldots, T_n)$ be the tally vector. The post-voting phase of the weak μ, μ-fuzzy voting scheme is performed as follows:

1. As above, the trustees shuffle and reencrypt the ballots, but open them in secret to compute the tally $T = (T_1, \ldots, T_n)$. Entry T_i equals the number of votes for candidate C_i.

2. The trustees decide which candidates need to be blurred how much, and which entries in the tally are to be published accurately.

3. A vector $f = (k_1, \ldots, k_n)$ is chosen by the voting machine: for each candidate C_i, $i = 1, \ldots, n$, the trustees do the following (in secret):
   - If $T_i < \mu$, they compute $k_i := T_i \mod \mu$.
   - If $\mu \leq T_i < 2\mu$, they choose a k_i with $0 \leq k_i \leq T_i \mod \mu$.
   - If $T_i \geq 2\mu$, they choose a k_i with $0 \leq k_i < \mu$.
   - If T_i must be published accurately, $f_i = 0$.

4. If $\Sigma_{i=1}^{n} k_i < \mu$ or $\Sigma_{i=1}^{n} \mu - k_i < \mu$, the trustees repeat Step 4. Otherwise, they proceed with Step 6.

5. The trustees choose a second vector $d = (d_1, \ldots, d_n)$ with the following properties:
   - $k_i + d_i \leq \mu$ for all i
   - $k_i + d_i = \mu$ for each i where $T_i \leq \mu$,
   - if an i exists with $T_i \leq \mu$, then
     a) $\sum_{i=1}^{n} d_i \geq \mu$
     b) $f + d$ must have at least two entries that are greater than zero.

6. For each candidate C_i, $i = 1, \ldots, n$, the trustees do the following (in secret):
   a) They take out k_i ballots for candidate C_i of the set L.

b) They create only $d_i - k_i$ (not $\mu - k_i$) encrypted ballots for candidate C_i, such that the created plaintext ballots are indistinguishable from real ballots.

7. The trustees publish the set M of ballots that were taken out of L, without showing which of these ballots were for which candidate. Let L_R be the resulting reduced set of ballots.

8. The trustees also publish the set N of newly created ballots from Step 6. b), again without revealing which or how many ballots belong to which candidate.

9. The trustees shuffle $M \cup N$ and prove the correctness of the shuffle with a zero knowledge proof.

10. The shuffled ballots from Step 9 are opened, there are $k_i + d_i$ ballots for each candidate C_i. Everyone can check that there are less than μ ballots for each candidate.

11. The ballots in the reduced set L_R are opened, the resulting tally is the representation of the real tally. The resulting representation should be $(R_1, \ldots, R_n) = (T_1 - k_1, \ldots, T_n - k_n)$.

As in the scheme above, for each candidate C_i, k_i ballots are taken out. But instead of filling them up with the difference to μ, the trustees fill them up with d_i encrypted ballots. The range in which each entry T_i lies is of size $k_i + d_i$, which is public. In Step 5, the authorities decide with the vector d how strong each entry is blurred. We could call d the *accuracy vector*. If $d_i = 0$, T_i can be computed from the representation of the tally. Of cource, if $d_i = 0$, it makes no sense to choose an $k_i \neq 0$.

Coercion-resistance

Coercion-resistance is not weaker as in the original scheme, as argued in the μ, μ-fuzzy scheme above.

Verifiability

The representation of the tally is $R = (R_1, \ldots, R_n) = (T_1 - k_1, \ldots, T_n - k_n)$. The vectors f and d must be kept secret, but $f + d$ becomes public with step 7 and 8, after which everyone can see that at most $k_i + d_i$ ballots where taken out from the ballots of each candidate C_i. So after the representation is published, everyone is convinced that the tally lies between $(T_1 - k_1, \ldots, T_n - k_n)$ and $(T_1 - k_1 + d_1, \ldots, T_n - k_n + d_n)$.

Weak μ, μ-fuzziness

The tally lies between $(T_1 - k_1, \ldots, T_n - k_n)$ and $(T_1 - k_1 + d_1, \ldots, T_n - k_n + d_n)$. By definition, $k_i + d_i = \mu$ for each candidate with less than μ votes, and the overall number of taken out ballots is greater than μ. So for each candidate C_i with less than μ votes, there are μ possibilities for T_i, which are all indistinguishable, with a similar argumentation as in the μ, μ-fuzziness of the construction described above.

Remarks

If a single candidate needs blurring, at least two entries in T need to be blurred, because the overall number of ballots is publicly known. Therefore, the blurring has to be in a way that the taken out ballots could as well have been taken out from other candidates. However, this “counter-blurring" can be distributed among several candidates that are blurred a little less, with $k_i + d_i < \mu$.

4.6.3. Bingo Voting with Fuzziness

As an example, we adapt Bingo Voting to our notion of fuzziness, using the techniques described above. Bingo Voting uses a voting machine that is trusted for privacy and can do all calculations, including the blurring process. Therefore, the necessity of a trusted entity who can learn the tally does not imply an additional assumption.

For the sake of simplicity, we first describe a μ, μ-fuzzy version of the original Bingo Voting scheme. After that, we describe a μ, μ-fuzzy Bingo Voting scheme with write-in support. In both schemes, the pre-voting phase and the voting phase are exactly the same as in their non-fuzzy versions. Therefore, we only describe the post-voting phases. To allow for a more precise tally representation, we also describe a weak μ, μ-fuzzy version of both variants.

As opposed to our general approaches, the situation in Bingo Voting seems to be a little different, since the tally itself is neither computed homomorphically nor by shuffling ballots. Instead, Bingo Voting calculates the tally reversely: for each voter, instead of (actively) giving a vote to one candidate, one dummy vote is *taken away* from each *not voted* candidate. For tallying, the left-over, unused dummy votes are opened, which do not depend on any individual ballots. Therefore, they can be opened directly and do not have to be unlinked from voters first. On a first look, it seems that an elegant solution would be to blur the tally beforehand, in the pre-election phase, by randomly adding between 0 and μ dummy votes for each candidate. The tally representation would then be calculated by opening the unused dummy votes, where each candidate would have between 0 and μ too many of them. So the representation $R = (R_1, \ldots, R_n)$ would yield an upper bound of each candidate's number of votes, and the real tally would lie between $(R_1 - \mu, \ldots, R_n - \mu)$ and $R = (R_1, \ldots, R_n)$. However, to prevent the adversary from seeing an upper bound less then μ, each candidate should have at least μ votes in the representation of the tally. Since it is not known beforehand which candidates will get less than μ votes, this is hard to achieve. Instead, we use the construction used for the mix-based schemes, and take out unused dummy votes in the post-voting phase.

Apart from that, there is an important difference between Bingo Voting and mix-based schemes: in Bingo Voting, pattern voting is not a problem since no plaintext ballots are published. Ballot formats are regulated via the number of dummy votes created in the pre-voting phase, and their arrangement on the receipt. No pattern from individual ballots becomes visible when opening the unused dummy votes. The reason for this is that each unused dummy vote naturally corresponds to a one out of n ballot, regardless of the voter's ballot format. Multiple choices on the voter's ballot simply result in more unused dummy votes of the corresponding candidates. So unlike in the mixed-based schemes, no restriction to one out of n choices is necessary.

4.6.3.1. Construction of μ, μ-Fuzzy Bingo Voting

The construction of μ, μ-fuzzy Bingo Voting is an instantiation of the general construction for mix-based voting schemes described in Section 4.6.2. Consider a μ, μ-fuzzable election with n candidates $C_1, \ldots, C_n$. Like in the description of the original Bingo Voting scheme, we assume a trusted voting authority that has read access to the voting machine and write-access to the public bulletin board. The post-election phase of μ, μ-fuzzy Bingo Voting consists of the following steps:

1. The voting authority publishes all receipts on the public bulletin board.

2. For each receipt, the voting authority publishes a proof that it contains the correct amount of fresh random numbers and dummy votes. With these proofs, it is also published which commitments correspond to unused dummy votes, and which are commitments of dummy votes that have been used on the receipts.

3. The voting machine opens the unused dummy votes (in secret) to compute the tally $T = (T_1, .., T_n)$. There are T_i unused dummy votes for each candidate C_i.

4. A vector $f = (k_1, \ldots, k_n)$ is chosen by the voting machine: for each candidate C_i, $i = 1, \ldots, n$, the voting machine chooses k_i in the following way:
   - If $T_i < \mu$, compute $k_i := T_i \mod \mu$.
   - If $\mu \leq T_i < 2\mu$, choose a random k_i with $0 \leq k_i \leq T_i \mod \mu$.
   - If $T_i \geq 2\mu$, choose a random k_i with $0 \leq k_i < \mu$.

5. If $\Sigma_{i=1}^{n} k_i < \mu$ or $\Sigma_{i=1}^{n} \mu - k_i < \mu$, the voting machine repeats Step 4. Otherwise, it proceeds with Step 6.

6. Let L be the set of commitments to the unused dummy votes. For each candidate C_i, $i = 1, \ldots, n$, the voting machine does the following (in secret):
   a) It takes k_i commitments of candidate C_i out of the set L.
   b) It creates $\mu - k_i$ new dummy votes for candidate C_i, such that the created dummy votes are indistinguishable from the other dummy votes, and calculates a Pedersen commitment to each new dummy vote.

7. Let L_R be the resulting reduced set of commitments, $M := L \setminus L_R$ the set of all commitments that were taken out in iterations of Step 6. b), and N the set of all new commitments created in Step 6. c). The voting machine publishes M and N.

8. The voting machine publicly opens all commitments in the reduced set L_R by publishing their unveil information on the public bulletin board, resulting in a set of unused plaintext dummy votes. The representation of the tally is the vector
$$R = (R_1, \ldots, R_n),$$
where each entry R_i equals the number of unused dummy votes of candidate C_i that are in L_R.

9. The commitments in $M \cup N$ are masked and shuffled to obtain a set S of shuffled commitments.

10. The set S is published together with a zero knowledge proof that it is a correct shuffle of $M \cup N$.

11. The commitments in S are publicly opened, there should be μ for each candidate. This shows that no more than μ commitments were taken out for each candidate, and that therefore the tally lies between $(R_1, \ldots, R_n)$ and $(R_1 + \mu, \ldots, R_n + \mu)$.

Coercion-Resistance

The blurring process depends solely on the tally and affects only the unused dummy votes, which in turn are independent of each single voter's actions. Therefore, the blurring process does not weaken coercion-resistance.

Verifiable correctness and μ, μ-fuzziness

Verifiable correctness and μ, μ-fuzziness can be shown by analogy with the mixed-based approach.

Bingo Voting with Weak Fuzziness

A weak μ, μ-fuzzy version of Bingo Voting applied to a (not weak) μ, μ-fuzzable election can be setup by instantiating the general construction of weak μ, μ-fuzzy mix-based voting schemes, described in Section 4.6.2.1.

4.6.3.2. Fuzzy Bingo Voting with Write-In Support

The Bingo Voting version with write-in support described in Section 4.4 consists of two tallies: the tally of the list candidates, which is computed as in the original Bingo Voting scheme, and the tally of the write-in candidats, which is performed by mixing and opening commitments to write-in candidates. A μ, μ-fuzzy version of this scheme can be created straightforwardly by computing the tally representation of the list candidates with the μ, μ-fuzzy scheme described in Section 4.6.3, and the representation of the tally of the write-in candidates with the construction of μ, μ-fuzzy mix-based voting schemes described in Section 4.6.2.

To prevent forced abstention that is caused by forcing the voter to "vote" for a certain random string it is crucial here that the voter may write in names without voting for them, so that these names appear in the representation of the tally. Without this possibility, a voter can be forced to not vote for a certain write-in candidate by forcing him to write in another. So we have to provide the voter with the possibility to somehow whisper an arbitrary number of names to the voting authority that are later included in the representation even though they got no votes. Since in Bingo Voting for secrecy we have to assume an honest voting computer anyway we can as well provide the voting computer with an additional interface with which the voter can secretly enter names that she wishes to be included in the representation.

4.6.4. Discussion

The schemes described above were designed to fulfill the definition of fuzziness. However, they can be used in a more general way: both techniques show for each candidate that his number of votes lies between x and $x + \mu$ for some x. If we take a closer look at the situation, we can as well generalize our definition to different bucket sizes, i. e. different values of μ for each candidate. This way, we can easily adapt the schemes introduced above to prove that a candidate has reached a certain quota, or that he has earned a certain number of parliament seats.

4.6.5. Fuzziness and Coercion-Resistance

To show how our definition of fuzziness can be used together with definitions of coercion resistance, we informally apply our work to the definition of Küsters et al. [KTV12] and the μ, μ-fuzzy Bingo Voting scheme described in Section 4.6.3. Küsters et al. prove that Bingo Voting achieves the same level of coercion resistance as an ideal voting scheme except for forced abstention. They state that Bingo Voting

is vulnerable to this attack because the adversary sees the receipts of all voters. We argue that a voter cannot by any feasible means prove that she has obtained no receipt, so she cannot prove to an adversary that she did not vote. We do not regard this attack any further. What remains is forced abstention through voting for an unlikely or fictional candidate.

We informally define μ-coercion-resistance as follows:

Definition 10 (μ-coercion-resistance) *A voting scheme applied to a μ, μ-fuzzable election is μ-coercion resistant if it is coercion-resistant according to the definition of Küsters et al. [KTV12] and μ, μ-fuzzy.*

It is easy to see that if a μ, μ-fuzzy election scheme that achieves coercion resistance according to the definition of Küsters et al., it also achieves the definition of μ-coercion-resistance above and is resistant against forced abstention caused by forcing to vote for unlikely candidates.

Since the μ, μ-fuzzy Bingo Voting scheme does not differ from the original scheme except in its published data and the fact that μ more dummy votes are created per candidate, and since its published data leaks less information than that of the original scheme, the μ, μ-fuzzy Bingo Voting should achieve coercion-resistance according to the definition of Küsters et al. As stated above the voter cannot possibly prove that she did not vote by proving that she has no receipt. Therefore the scheme seems to be resistant against forced abstention.

The μ, μ-fuzzy Bingo Voting scheme with write-in support yields with its published data no more information as is yielded by an election performed with the original scheme and all occurring write-in candidates on the list of regular candidates. The additionally published list of names does not tell whether a candidate on this list got a write-in vote or not. So this scheme, too, should be μ-coercion-resistant and additionally resistant against forced abstention.

5. Coercion Resistance in Internet Elections

The concept of conducting elections via the internet promises new possibilities: voters can cast their ballots comfortably from at home, which might increase voter turnout, and the time and effort of overseeing a polling station is saved. However, the absence of a polling station also brings new challenges: as opposed to a presential election, where the voting process usually takes place in the privacy of a voting booth, voting authorities have no influence on the computational platform on which the ballot is generated and cast, or on the location where this process takes place. Privacy during the vote casting process cannot be assumed. A second problem is that the eligibility of voters has to be checked remotely, without interacting with the voter in person. Similarly, the voter does not have any physical access to the device which records her ballot. This, too, is done by a distant server, as is the counting of the cast ballots. Therefore, voter-privacy is harder to protect, coercion-resistance much harder to guarantee. At the same time, the opposing requirement of providing both individual and universal verifiability becomes a more demanding task as well.

The main topic of this chapter is *revoting*: the possibility for voters to overwrite an already cast ballot. This possibility promises to solve the problem of being observed during the voting process. However, it complicates providing public verifiability, since now the counting of the right ballots needs to be proven. At the same time, these proofs must not provide the adversary with the possibility to find out if the voter has revoted, and by doing so has overwritten a ballot cast under adversarial observation. Coercion is not the only motivation for revoting. As stated in [AdM09], a voter might not be familiar with internet voting, and it would be of advantage if she could ask a friend to go through a voting process with her, and then cast a vote in private later.

Though it has become a standard technique to overcome the problem of being observed during the voting process, there is no existing practical solution that provides revoting with full verifiability and coercion-resistance at the same time. In this chapter, we offer a solution which provides revoting while achieving both public verifiability and strong privacy properties. We apply our solution to a relatively young voting paradigm for which revoting is essential: delegated voting. In elections which allow delegated voting, the voter can choose between casting a ballot herself

or delegating her vote to another eligible voter of her choice. In such elections, it is usually required that a voter can revise her choice of delegation at anytime. This Chapter provides a voting scheme which allows delegated voting by combining our revoting solution with a new paradigm we call *vote fetching*.

This Chapter is structured as follows: Section 5.1 discusses related work on internet elections, revoting and alternative solutions to the observation problem. After discussing the revoting problem in general in Section 5.2, we introduce our solution in Section 5.3. We apply our solution to Delegated Voting in Section 5.4, where we first introduce additional requirements for a voting scheme allowing delegations, discuss an existing scheme and its drawbacks, and then introduce our idea of vote fetching from which we design a voting scheme which allows vote delegation.

5.1. Related Work

Many voting schemes for remote internet voting have been proposed during the last years, some of them were introduced in [SK95, HS00, Cha01, Nef01, RT09, AHL+09, HLVL10, KKW06, Adi08, MR10, OKNV12, Gjø10, Lip11, ZCC+13, JCJ05, MCC08, CH11a, ECH12].

5.1.1. Remote Voting Schemes Used in Practice

Helios [Adi08] is a publicly verifiable election scheme which has mainly been designed for educational purposes and elections with a low risk of coercion. There are several versions of Helios, both mix-based and with homomorphic tallying. The scheme has been used in several smaller elections [AdM09]. Everlasting privacy was added to Helios in [DvdGSdSA12].

POLYAS[1] is a voting scheme that has been designed to comply with the Common Criteria Protection Profile for Online Voting Schemes [MR10, OKNV12, fSidI07], and has been applied, among others, in elections of the Gesellschaft für Informatik[2] (German Society of Informatics). POLYAS uses separate servers to ensure integrity and voter privacy. In its original version, it does not provide public verifiability. Olembo et al. [OKNV12] analyze some modifications to POLYAS that introduce partial verifiability. These modifications allow for a verification of the correct computation of the tally from the stored encrypted votes, but do not provide a possibility to verify the correct recording of the ballots.

Online voting has been used as a trial and in real parliamentary elections in several European countries, for example in Estonia and Norway. Like POLYAS, both the Estonian [HLW12] and the Norwegian voting scheme [Gjø10] are based on the physical separation of vote servers. The Estonian scheme introduces several security measures like audit logs, checksums and tamper resistant hardware to protect integrity and voter privacy. However, a corrupted voting computer is able to modify the voter's choice undetected, as attacks mentioned in [HLW12] have shown. Neither the Estonian nor the Norwegian voting scheme allows for a public verification of the tally.

An approach to evade the risks of coercion and manipulation coming from malware on the voter's PC is to use code voting [Cha01, HSS08]. The voter inputs her choice by typing a code, so her choice is concealed from her computer. The voter then gets back a return code with which she can check that her ballot has been transmitted to

[1] POLYAS website: `https://www.polyas.de/`

[2] Website of the German Society of Informatics: `en.gi.de/`

the voting server correctly. The Norwegian voting scheme provides return codes, but the voter's choice is entered in plaintext for usability reasons. The voter can detect a manipulation with the help of the return codes, but the manipulation cannot be proven [GSB]. The Norwegian scheme [Gjø10] has later been improved by Lipmaa [Lip11].

Remotegrity [ZCC+13], a remote version of Scantegrity II, has been applied in the municipal election of the city of Takoma Park, Maryland. With Remotegrity, a voter can detect and prove manipulations of her ballot. Revoting is not provided, a voter can be coerced through observation during the voting process.

5.1.2. Related Work on Revoting

There are several internet voting schemes which allow revoting. However, in each of these schemes, either the revoting process is not secret, or it is not publicly verifiable. A more advanced version of Helios [AdM09] has been used in the election of the university president of the Université catholique de Louvain in 2008, which was estimated to be an election of low coercion risk by the authors of [AdM09]. Voters could revote, to give voters who are not familiar with internet voting to do a voting process together with a friend to get to know the system, and then vote again in private. Revoting was implemented by overwriting the voter's encrypted ballot on the bulletin board. Therefore, the revoting process was verifiable but not coercion-resistant. In the Norwegian scheme [Gjø10] mentioned above, the voter can revote as many times as she wants and gets a receipt code for each cast ballot. Multiple votes by the same voter are sorted by serial numbers on the ballot box server, only votes with the newest serial number are counted. However, the scheme offers no public verifiability of the revoting process.

Alternative approaches to solve the problem of observation during the voting process are usually based on fake credentials. The voting scheme proposed by Juels et al. [JCJ05] and its advanced version Civitas [MCC08] fall into this category. In these schemes, upon coercion the voter provides the adversary with a fake credential for vote casting, which is indistuinguishable from the real credential. In 2011, Clark and Hengartner introduced Selections [CH11a, CH11b], in which the voter eludes coercion with the help of a panic password, also a form of a fake voter credential. The idea is that a voting process done with a panic password is indistinguishable from a voting process done with the real password. Under observation, the voter can use a panic password to "cast" a vote that will not count, and in a private moment cast a vote with her real password. In 2012, Clark et al. introduced [ECH12], which is also based on fake credentials in the form of panic passwords, but with a more efficient ballot authorization technique.

An advantage of the panic password approach is that the voter can cast a vote bare-handed: the construction of a panic password is easy, and can be done by the voter without any computer support. Another advantage is that, unlike with the revoting approach, the voter does not have to do his real voting process after the voting process under adversarial observation, so it overcomes the problem of being coerced one minute before the end of the voting phase. However, one big disadvantage is that the voter gets no confirmation when voting with her real password (as is the goal of the scheme). So if she votes with a panic password by accident, she might not be aware that she has actually not cast a ballot. Conversely, the voter might not be sure if she has entered her real password unintentionally under observation. Therefore, we propose that this approach should be combined with revoting, to let

the voter cast her ballot again if she is unsure, just in case.

5.1.3. Related Work on Delegated Voting

Basic rules for delegated voting were introduced by Bryan Ford in [For02], suggestions for designing delegated voting schemes were made by Greem-Armytage in [GA14]. Liquid Feedback[3] is a voting scheme designed for polls in which voter's choices are opened directly after all ballots are cast. It is used by the German Pirate Party. It uses standard techniques for authorization, but is not a cryptographic voting scheme. It is not designed to provide coercion-freeness, the only privacy provided is through pseudonyms. Another non-cryptographic project is Adhocracy [4], where security measures are kept low-level for better transparency and usability.

The Agora Ciudadana project[5] is an open source project of a voting scheme which provides vote delegation. It uses cryptographic techniques to provide vote secrecy and verifiability. The implementation can be seen in action on the website `agoravoting.com`.

5.2. Revoting

The work introduced in this chapter is joint work with Jörn Müller-Quade, Bernhard Löwe, and Dirk Achenbach [AKMQL14].

A lot of coercion risks, like family voting or cameras observing the voter, may be resolved by allowing the voter to recast her vote. Therefore, revoting has become a desirable feature in most online voting schemes. However, to achieve end-to-end-verifiability, the voters must be convinced that of each voter only the most current ballot was counted and that the ballot counted was cast by the voter herself and not inserted by an election server. But any information that proves if, how often or when a voter has recast her vote leads to a second problem: an adversary observes the voter once, and coerces her not to cast a vote again afterwards, effectively preventing revoting. Existing schemes either provide verifiability or secrecy of the revoting process, as was seen in the related work section. To the best of our knowledge, the voting scheme introduced in this chapter is the first to achieve both end-to-end-verifiability and coercion-resistance simultaneously.

5.2.1. Requirements for Revoting

As stated above, the strategy of evading coercion by revoting is only effective if the adversary cannot learn whether the voter has used her ability to revote. In particular, the adversary must not learn how many votes a voter has cast. In the following, we state in more detail how a revote could potentially be recognized by an adversary. After this we discuss which additional verifiability issues are introduced by revoting.

5.2.1.1. Information which Betrays a Re-Vote

We identified three sources of information the adversary can use to deduce whether a voter has revoted.

[3] Liquid Feedback website: `http://liquidfeedback.org/`

[4] Adhocracy website: `https://adhocracy.de/`

[5] Website of the Agora Ciudadana project: `https://github.com/agoraciudadana/agora-ciudadana`

Number of ballots of the same voter

The number of valid ballots a voter has cast may lead to a coercion, even if this number is not linked to the voter's name – it can identify the voter. If the number of revotes of the same voter is deducible, the adversary can cast exactly k votes with the coerced voter, and then check if k appears as a number of someone's revotes. Hence, it should be impossible to infer how many ballots have been cast by any voter, and the voter must not be able to prove how many times she has cast a ballot. For this reason, an algorithm which sorts out overwritten votes must not leak any chains of revotes which belong to the same voter.

Timestamps

If an adversary can determine at which time a counted ballot was cast, he can also determine whether his coercion attempt was successful. He simply needs to note the time during his coercion attempt. Therefore, timestamps must not be visibly linked to cast ballots during the tallying process.

Tagging a Ballot

Tagging is the third possibility to detect whether a certain ballot has been counted. To achieve verifiability, voting schemes usually publish all counted ballots on a bulletin board. An adversary might try to "tag" the ballot that is cast during his coercion attempt. If this tag is not removed in the casting and counting process, the adversary can recognize it on the bulletin board, and track it to see if "his" ballot enters the counting process. The simplest way to tag the ballot is to do nothing—just remember the cast ballot. If cast ballots contain static elements like a signature or an ID, the adversary can figure out if the voter revoted after the coercion attempt. This leads to the requirement that ballots, signatures or other data which will be part of the tallying process must not be linkable to any cast ballot. Achieving this while maintaining verifiability is a quite a challenge.

5.2.1.2. Tallying Recast Votes Correctly

In addition to the verifiability requirements any cryptographic voting scheme must comply with, schemes that allow for revoting face at least two additional challenges.

One Ballot per Voter

It must be proven that at least one ballot of each voter (who cast at least one ballot) has been counted. At the same time it must be shown that only one ballot has been counted of each voter. Using digital signatures for these proves is delicate, as explained above.

Tallying the last Ballot

In a voting scheme without revoting it is sufficient to prove that a ballot of a voter has been counted. In a revoting scheme it also has to be proven that the last one, and only the last one, has been counted. For the above mentioned reasons, this proof must be done without revealing how many ballots the voter has cast, the time when the ballot has been cast, or any other information which helps identifying a revote.

5.2.2. An Approach: Revoting in Five Phases

Our solution for secret but verifiable revoting can be described in five phases, of which the first phase consists of ballot casting and takes place in the voting phase, while the other four phases take place in the post-voting phase, in which old ballots

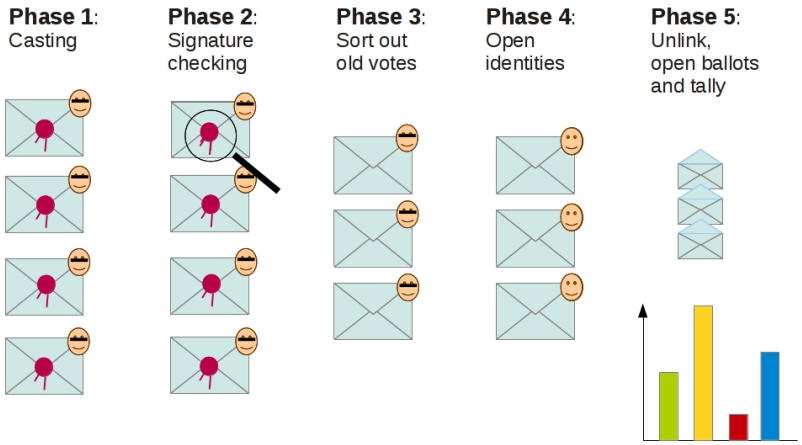

Figure 5.1.: Revoting phases

are verifiably sorted out and counted. The phases are described in more detail below, our detailed voting protocol is described in Section 5.3. After sketching the idea of our approach, we will describe the five phases in more detail below.

5.2.2.1. The Idea of our Approach

The idea of our scheme is that the voter signs her encrypted ballot, which is to be published on a public bulletin board, in a way that links it to her identity while hiding her identity at the same time, even in the process of signature checking. Therefore, instead of using the actual signature, she appends a zero knowledge proof of knowledge of her signature. These proves are then checked to prove that there are only ballots from eligible voters on the public bulletin board, which enter the process of sorting out old ballots and selecting the newest ballots for counting. This process is publicly verifiable, i. e. everyone can check that of each voter who has cast a ballot, a ballot is selected for counting, and this selected ballot is the last ballot this person has cast. This is done without revealing which voter has cast which or how many ballots, or if a voter has revoted at all. After that, it is proven that of each voter exactly one ballot remains. This is done by unlinking the identity of the voter from her ballot and opening it. After this step, the selected ballots, which are in turn unlinked from the voter's identity, are opened and tallied. The phases are sketched in Figure 5.1.

5.2.2.2. Overview over the Five Phases

We now provide a high level description of the five phases used to implement verifiable and incoercible revoting. An instantiation of these five phases is described in Section 5.3.

In the following description, let E be a multiplicatively homomorphic encryption function, i. e. $E(m_1) \cdot E(m_2) = E(m_1 \cdot m_2)$ for two messages m_1 and m_2. We suggest using the Elgamal encryption. Let E_A be an additively homomorphic encryption function, i. e. $E_A(m_1) \cdot E_A(m_2) = E_A(m_1 + m_2)$. We suggest using the Paillier cryptosystem, in which ciphers are multiplied to obtain an encrypted sum of their plaintexts, which is the reason for the above notation.

In our voting protocol, all components of the ballot will be encrypted with one of these homomorphic schemes, and can therefore be rerandomized. In the description below, we will shuffle encryptions on several occasions. Each time encryptions are shuffled, a zero-knowledge proof of correct shuffling is published. In our high level

description we will not always state this explicitely. The shuffling and its proof can be done with standard techniques, for example by using a re-encryption mixnet with shadow mixes, following the ideas described in [Ben06] and Section 2.3.6.2.

Phase 1: Vote Casting

The voter creates a ballot $B := (pk, ts, v)$, where v is the voter's choice, ts is the current timestamp, and pk is the public verification key corresponding to the voter's signature key. The public key pk uniquely identifies the voter. Therefore, we also refer to pk as the voter's *identity*. The voter encrypts her ballot componentwise to obtain an encryption $(E(pk), E_A(ts), E(v))$. The voter's identity pk and the ballot's timestamp are encrypted with the joint public key of a set of *sorting servers*, while the choice v is encrypted with the joint public key of a set of *tellers*. This encrypted ballot will later appear on the public bulletin board.

To publicly prove eligibility, the voter has to sign her ballot. But this signature would reveal the voter's identity, and therefore show if or how often a voter has revoted. Instead, the voter creates a non-interactive zero-knowledge (NIZK) proof π of knowledge of a valid signature of her encrypted ballot. This proof π consists of several sub-proofs, one of which proves that the signature is created with the signing key corresponding to the public key encoded in her ballot, to prevent that a voter signs and casts ballots for other voters. Another sub-proof proves that the voter owns a certificate for this signing key, which proves her eligibility. To cast her ballot, the voter sends the encrypted ballot $(E(pk), E_A(ts), E(v))$ as well as the zero-knowledge proof π to a *casting server*, to which she proves that the encrypted timestamp encodes the current time. Her encrypted ballot and the proof π are published on a *public bulletin board*, the voter can check that it appears there unchanged. Everyone can check the NIZK proofs without learning the voter's identity.

At the end of Phase 1, the bulletin board contains a set of encrypted ballots with proofs allowing a public verification that each ballot has been cast by an eligible voter.

Phase 2: Verifying the signatures

In Phase 2, it is checked which ballots have been cast by eligible voters. It is crucial in this step that the checking of signatures does not reveal the identity of a voter, and with this, if a voter revoted. This is done by checking the NIZK proofs of knowledge of the signature, which are created in a way that the public keys of the voters are not needed for checking. The sorting servers also check the signatures, mark ballots with invalid proofs as invalid and do not process them any further. Since signature checking can be done by everyone, everyone can check that the sorting servers marked the right ballots.

At the end of this phase, the bulletin board contains a set of ballots which are from eligible voters. Only these ballots are further processed and enter the process of sorting out old ballots in the next phase.

Phase 3: Sorting out old ballots

After the signatures are verified, the sorting servers detach the ballots cast by eligible voters from their signatures by mixing and rerandomizing them without the signature proofs, to obtain a list L of ballots.

To sort out old ballots, the sorting servers have to be able to decide and prove whether two ballots belong to the same voter, and if they do, which of the two ballots is older. This needs to be done without building a chain of ballots of the same voter,

because this would reveal the number of revotes of each voter. The process must also not reveal any voter's identities or timestamps. To avoid chains, we sort out ballots in several rounds.

Each round starts with a list M of ballots. The first round starts with the list $M := L$, each subsequent round with the reduced list of ballots that were not thrown out in the previous round. In each round, the first ballot is taken out of the list and its identity part is subsequently compared with that of the other ballots in the list until another ballot with the same identity is found. If no match is found, we put this ballot into the list N of ballots that is to enter the next round, and proceed with the next ballot in the current list M. If a match is found, the timestamps of the two ballots are compared, both ballots are deleted from M. The newer ballot is put into N, and we proceed with the next ballot in M. We repeat this until M is empty, and enter the next round with a shuffled version of the list $M := N$. Please note that in each round, each ballot is compared with at most one other ballot with the same identity.

Now we describe the sub-protocols of identity and timestamp comparison. For both we basically compute the difference on the ciphers. When comparing identities, we check if the difference is the neutral element of the underlying group. When comparing timestamps, we can just decrypt the difference and check whether it is greater or less than zero. However, we must not reveal the difference of any two identities: the difference uniquely determine which pair of identities is compared, so revealing it would show how often two identities are compared with one another, which would reveal information about the number of revotes of these two voters. Therefore, we mask the difference of the identities with a random exponent, which will not affect the neutral group element. This masking is done jointly by the sorting servers to ensure that no exponent is chosen which maps the difference to the neutral element. The masked difference of the identities is decrypted. If it is the neutral element, the two identities are the same, i. e. the ballots belong to the same voter.

At the end of this phase, the bulletin board contains a set of ballots selected for tallying, together with all the necessary proofs (i. e. shuffle proofs, decrypted masked differences etc.) to check that the sorting was done correctly. So it contains all data that is needed to publicly verify that of each voter, a ballot has been selected for tallying, and that this is the newest ballot of this voter.

Phase 4: Proving one ballot per voter

To prove that of each voter, only one ballot is left, the sorting server shuffles the part of the ballots that contain the voter's encrypted public key and proves the correct mixing. The encryptions are then opened to prove that of each voter, there is one public key, and with that, for each voter, only one ballot enters the tallying process.

At the end of Phase 4, the bulletin board contains all necessary information to verify that of the ballots selected for tallying, each belongs to a different voter. Remember that from the previous phase it also contains all information to verify that of each voter who has cast a ballot, the newest one is selected for counting.

Phase 5: Tallying

From here, our scheme follows standard procedure: the parts of the ballots that encrypt the voter's choices are shuffled, decrypted and counted by the tellers, with proofs of correct decryption. This is a standard procedure and requires no new techniques.

At the end of this phase, the tally is published with all necessary proofs of correctness.

5.3. An Instantiation of our Revoting Scheme

A big advantage of our protocol is that it can be set up in a way that no single instance has to be trusted for privacy and coercion-resistance. Each critical calculation like sorting out old ballots or tallying can be distributed over a set of trustees. At the same time, our scheme is publicly verifiable, under the assumption that a common reference string (CRS) is created honestly and the PKI only certifies signature keys of eligible voters.

For the sake of clarity, we describe our protocol with one single server per task, whereas each server which learns critical data can be distributed. A way to distribute each trusted server is discussed below the description of our voting scheme.

Apart from that, our scheme does not depend on any trusted authority for verifiability or voter privacy.

5.3.1. Overview over the Used Techniques

Our instantiation of the revoting scheme builds on several building blocks. Groth and Sahai introduced a way to efficiently instantiate NIWI and NIZK proofs for pairing product equations (*GS-Proofs*). We will use these proofs in Phase 1 and check them in Phase 2. To be able to do this, we need a signature scheme and a ballot encryption method that is compatible with the GS proof system. Also, our building blocks have to enable us to unlink a ballot's identity from the signature while proving correctness, in particular being able to check that each voter has cast her own ballot and not voted in the name of someone else. Our main building block is inspired by a construction of Ghadafi[Gha11], who constructed a blind group signature out of a structure preserving signature scheme.

GS-compatible structure preserving signature schemes were introduced by Abe et al. in [AFG+10]. The aim of their work was to provide building blocks for a modular protocol design. Compatibility with the Groth-Sahai proof system is given because the verification function of the signature scheme consists of checking a set of pairing product equations. This provides the possibility of creating NIZK-proofs of knowledge of a signature σ for a message m that verifies to 1 with a verification key pk, where each of these components can be witness variables, i. e. kept secret, as needed. This fact is used by the authors of [AFG+10] to construct a blind signature out of their signature scheme: they just let the signer sign an also GS-compatible commitment to the message and prove that he knows a signature on the message under verification key pk. This scheme was later transformed to a blind group signature scheme by Ghadafi in [Gha11], who also hides the signer's public key in the witness of the GS-proofs.

In our work, we adapt the non-blind version of the signature scheme in [AFG+10] to be kind of a group signature. Group signatures provide the possibility to trace and open the identity of a signer with additional information. A naive way to realize our scheme would be to use a group signature for ballot signing, check the signature without revealing the voter's identity, then sort out old ballots and then use the open algorithm to reveal the identities of the remaining ballots to prove that only one ballot per voter is left. But there is an important difference between our application and the usual use of group signatures: the open algorithm in a group signature scheme is usually provided to reveal someones identity in the case

of fraud, where it is usually wanted that the signature can be linked to a certain signing process. In our voting scheme, the situation where we need the opening is not a fraud but a standard case, and a link to the signing process would link the signature to a certain ballot. We need unlinkability to remain intact even with the revealed identity.

What we do instead is attach an encryption of the voter's public key to the ballot, and add a GS-proof that this encryption contains the same verification key as is used in the GS-proof of knowledge of the signature.

We first describe the scheme with trusted servers and one server per task, to make it more clear. Later we discuss about which servers need to be trusted and how they can be distributed to implement a trusted instantiation.

5.3.2. Participants

The participants in our voting scheme are the following:

- **Issuer:** The issuer acts as a public key infrastructure (PKI), which certifies the voter's public keys in the registration phase. Eligibility is verified against this list, so the Issuer manages the list of eligible voters and their public keys for voting.
- **Voters:** The voters encrypt, cast and sign their ballots and create proofs of knowledge of a valid signature.
- **Voter-PC:** This is the PC used by the voter to cast her vote. It learns the voter's choice in that voting process. The voter can perform several voting processes with the same or different PCs.
- **Casting Server:** The voter connects to this server to cast her ballot. The server is responsible for checking that the timestamp on the ballot is feasible and the ballot is valid
- **Sorting Server:** The sorting server sorts out the old ballots and selects the newest ballot of each voter for tallying.
- **Tally Servers:** The tally servers mix and decrypt the selected ballots and compute the tally.
- **Auditors:** The auditors jointly create public parameters like the common reference string crs for the Groth-Sahai proofs.
- **Public Bulletin Board:** As common to most voting schemes, our voting scheme uses a public bulletin board. Everyone has read access, the casting server, the sorting server and the tally server have append access. Once on the bulletin board, data cannot be deleted or modified.

5.3.3. Assumptions about the Setup

Our voting scheme requires the following assumptions to hold:

- *Existing PKI:* We assume a PKI from which eligibility can be tested. The PKI maintains a public list of public keys and certificates. A voter is eligible if her public key is in this list and has a valid certificate. The PKI is managed by the

Issuer, who issues certificates on public keys to eligible voters. We assume an existing procedure for key generation, authentication and proving eligibility to the issuer, and do not specify this step in our protocol.

- The casting server might get some information about the PC the voter casts her ballots with (IP address, OS, Browser, etc.). This is not a problem specific to our voting scheme. But it might enable the casting server to recognize a new ballot which comes from the same PC. Therefore, in our protocol description, we assume the casting server as trusted. However, the voter could use an anonymization network, and the casting server could be distributed to weaken this problem (see the discussion below).

- A dishonest sorting server is able to decrypt the timestamps and voter identities of single ballots, but not the voter's choice encoded on this ballot. However, the sorting server does not need to decrypt any identities or timestamps to perform its task of sorting, and our sorting algorithm can be distributed among several servers. We assume the sorting server as trusted, and implement it by distribution as discussed below.

- The tally server is able to decrypt the choices, which are part of the ballots. Therefore, it has to be trustworthy. However, it cannot decrypt the corresponding public keys, i. e. it cannot identify the voters who have cast these ballots. This server is naturally implemented by distribution by using a threshold scheme for encrypting votes.

- The Voter-PC learns the voter's choice and therefore needs to be trusted for privacy. This assumption is not as critical as it looks: since we allow revoting, if the voter does not trust her PC anymore, she can use another one even if she has already cast a vote. The voter's signing key should be stored on trusted hardware so a malicious PC does not sign and cast ballots on its own.

5.3.4. Protocol Description

Our voting scheme consists of three parts which will be described below: a *pre-voting phase*, in which keying material is created, a *voting phase*, in which voters cast their votes, and a *post-voting phase*, in which old votes are sorted out and the newest votes are counted. Our five revoting phases appear in the voting phase and the post-voting phase, we indicate them as "Revoting-Phase 1-5".

5.3.4.1. Pre-Voting Phase

In this phase, all public parameters are setup and the necessary key pairs for the encryption and signature schemes are generated.

Setup of the bilinear groups and public parameters

Before the election starts, the encryption and signature schemes are setup. Let $gs := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$ be a group setup with bilinear groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ of prime order p, a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, and generators G and H of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, where $e(G, H)$ generates $\mathbb{G}_T$.

Let $Sign$ and $Sign_{CERT}$ be the signature function of the automorphic structure preserving signature scheme described in [AFG+10] (see Section 2.3.11), both with the group setup gs chosen above. We use different function names for clarity. The function $Sign$ is used by the voters to sign ballots, while $Sign_{CERT}$ is used by the

issuer to certify the public keys. Actually, $Sign_{CERT}$ does not need to be automorphic, so it can be substituted by another GS-compatible signature scheme based on Type 3 pairings.

The public parameters $F, K, T \in \mathbb{G}_1$ for the signature scheme $Sign$ are chosen at random by the auditors. The auditors jointly create the common reference string crs in the binding setting for the GS-proofs.

In the following, let $E : \mathbb{G}_2 \to \mathbb{G}_2 \times \mathbb{G}_2$ be the multiplicatively homomorphic Elgamal encryption function. We encrypt in $\mathbb{G}_2$ with generator H, so for each $x \in \mathbb{G}_2$, we have $E(x) := (H^r, xY^r)$ where Y is the public key used for encryption and $r \in \mathbb{Z}_p$ a random number. Let E_A be the Paillier encryption, which is additively homomorphic: $E_A(x_1) \cdot E_A(x_2) = E_A(x_1 + x_2)$ for all $x_1, x_2 \in \mathbb{G}$, where $\mathbb{G}$ is an appropriate group used for the encoding of the timestamps with the Paillier encryption. The group $\mathbb{G}$ is also setup by the auditors.

The setup $(gs, crs, F, K, T, \mathbb{G})$ is published.

Encryption is always done componentwise. When not clear from the context, we denote the used public key pk for the encryption by E^{pk}.

Key Generation

1. The Issuer creates a secret key sk_I and a corresponding public key pk_I for the scheme $Sign_{CERT}$. He publishes his public key pk_I.

2. The casting server does not need any keys except those which are public.

3. The sorting server creates a key pair (sk_S, pk_S) for the Elgamal encryption scheme: he randomly chooses $sk_S \in \mathbb{Z}_p$ and computes $pk_S = H^{sk_S}$, and publishes his public key pk_S. He also creates a key pair (pk_{AS}, sk_{AS}) for the additive scheme E_A and publishes pk_{AS}.

4. The Tally Server creates a key pair (sk_T, pk_T) for the Elgamal scheme, i. e. he randomly chooses $sk_T \in \mathbb{Z}_p$, and computes $pk_T = H^{sk_T}$. The key pk_T is published.

Voter registration

Let ET be the public key table in the PKI managed by the Issuer. Before the registration starts, the key table ET is empty. Each eligible voter creates a key pair (sk_i, pk_i) for the signature scheme $Sign$. To this end, the voter chooses a random $s \in \mathbb{Z}_p$ and calculates her public key $pk_i := (S_1, S_2) := (G^s, H^s)$. The voter's key pair is $(sk_i, pk_i) = (s, (S_1, S_2))$. The voter proves her identity and eligibility to the Issuer. How this is implemented is not specified by this work, we assume this as given.

The Issuer verifies the voter's identity and eligibility, and upon success signs pk_i with the scheme $Sign_{CERT}$ with his secret issuing key sk_I. He sends the resulting signature $cert_{pk_i}$ to the voter. The tuple $(ID_i, pk_i, cert_{pk_i})$ is published in the table ET, where ID_i is the voter's name.

The voter can verify the Issuer's signature on her key and complain and create another key pair if the verification check fails.

5.3.4.2. Voting Phase

Revoting-Phase 1: casting

In the voting phase, the voter creates a ballot together with the casting server. Casting a ballot is done in three steps:

1. First, the ballot is created.
2. To prove the legitimacy of her ballot while hiding her identity, the voter creates a proof of knowledge of a signature of her ballot.
3. The ballot and the proof of knowledge are published.

The three steps are now described in more detail.

Step 1: Ballot creation

The Voter creates $E^{pk_S}(pk_i)$ and $E^{pk_T}(v)$ and sends them to the casting server. To prevent common attacks, the voter must prove knowledge of the contents of $E^{pk_T}(v)$. This can be done using the technique of Schnorr [Sch91] and is not described here. The casting server verifies this proof and aborts the voting process if this verification fails. Otherwise, he proceeds with as follows: the casting server creates $E_A^{pk_{AS}}(ts)$ and sends $(ts, E_A^{pk_{AS}}(ts))$ with a proof of correct encryption to the voter. The timestamp can actually be created by the voter or the server, as long as the encryption of the current time is proven to the other party.

The voter's ballot is then

$$B := (E^{pk_S}(pk_i), E^{pk_T}(v), E_A^{pk_{AS}}(ts)).$$

Step 2: Creating a proof of knowledge of a signature

After the ballot is created, the voter computes a GS-proof which shows that:

1. The voter knows a valid signature of her ballot.
2. The public key corresponding to the signing key used to create this signature is the one encrypted in $E^{pk_S}(pk_i)$. This ensures that she does not cast ballots for another voter.
3. The voter owns a certificate $cert_{pk_i}$ for pk_i, i. e. she is eligible.

The GS-proof can only be created if the voter is able to sign her ballot. Therefore, it can act as a signature by itself. The proof must be publicly verifiable without revealing the voter's identity. Therefore, the signature itself, as well as the certificate $cert_{pk_i}$ of the voter's public key, must not be revealed by the proof, since they link the ballot to the voter's identity. The GS-proof is created as follows, using the common reference string crs created in the pre-voting phase:

1. **Signing:** The voter transforms B into an element $m \in \mathbb{Z}_p$ using a cryptographic hash function. Let $(M_1, M_2) := (G^m, H^m)$. The voter chooses $r, c \in \mathbb{Z}_p$ at random and computes the signature

   $$C_1 := F^c, C_2 := H^c, A := (KT^r M_1)^{\frac{1}{s+c}}, R_1 := G^r, R_2 := H^r,$$

   where $F, K, T \in \mathbb{G}_1$ are the public random parameters of the signature scheme, which were created in the pre-voting phase. The resulting signature is

   $$\sigma := (C_1, C_2, A, R_1, R_2).$$

   The components (A, C_1, C_2) contain information about the voter and must be hidden. Therefore, σ is kept secret by the voter. It can be deleted after the proof is created.

2. **Sub-proof of knowledge of a signature:** The voter creates a GS-proof Ω_1 of satisfiability of the signature verification equations

$$e(X, H) = e(G, Y), \tag{5.1}$$
$$e(C_1, H) = e(F, C_2),$$

and

$$e(A, YC_2) = e(KM_1, H)e(T, R_2). \tag{5.2}$$

The witness variables in this proof are $pk_i =: (X, Y)$, and the signature components A, C_1 and C_2.

3. **Sub-proof of eligibility:** The voter proves that she knows a certificate $cert_{pk_i}$ of her public key, by creating a GS-proof Ω_2 of knowledge of a witness $(cert_{pk_i}, pk_i)$ which together with the issuer's public key pk_I satisfies the verification equations of the signature scheme $Sign_{CERT}$. The GS-commitments of the components X and Y of pk_i from the first sub-proof are used, to show that the same key is involved as in the Equations 5.1 and 5.2.

4. **Sub-proof of identity:** To prove that the voter has attached her own identity to the ballot, the voter proves that the public key used in the two sub-proofs above is the one encrypted in $E^{pk_S}(pk_i) =: (U, V)$. For this, she creates a GS-proof Ω_3 for the equation

$$V = (H^r, Ypk_T^r).$$

The public key component Y and the encryption randomness r are treated as witness variables. Again the same GS-commitment of Y is used as in the other two sub-proofs.

Since for Y the same GS-commitment is used in all three sub-proofs, we see that the public key used to sign the ballot is the same as the one encrypted on the ballot, and the certificate used in sub-proof Ω_3 is for this very public key.

Step 3: Ballot casting

The voter casts her ballot

$$ballot := ((E^{pk_S}(pk_i), E^{pk_T}(v), E_A^{pk_{AS}}(ts))$$

together with the GS-proof $\Omega = (\Omega_1, \Omega_2, \Omega_3)$ by sending it to the casting server, who publishes $(ballot, \Omega)$ on the public bulletin board. The voter can check that her ballot appears.

Please note that the fact that the voter knows her encryption randomness is not a problem since she can always revote. So even if she proves a certain vote, this only proves that one such vote signed by her is on the bulletin board. There could be others.

5.3.4.3. Post-voting phase

Revoting-Phase 2: Verifying the signatures

After all ballots are cast, the GS-proofs Ω_i of all ballots,

$$(E^{pk_S}(pk_i), E^{pk_T}(v_i), E_A^{pk_{AS}}(ts_i)),$$

$i = 1, \ldots, n$, are verified by the sorting server, where n is the number of votes cast. The sorting server marks ballots of which the signature check fails as invalid. Everyone can verify this step since the proofs are publicly verifiable.

Revoting-Phase 3: Sorting out old ballots

To sort out old votes, the sorting server takes all ballots

$$(E^{pk_S}(pk_i), E^{pk_T}(v_i), E_A^{pk_{AS}}(ts_i))$$

with valid signatures from the bulletin board as input without the GS-proofs, which are no longer needed. The sorting server shuffles and reencrypts all the ballots, obtaining a new list L of ballots, and proves the correct mixing. The sorting server initializes a new empty list N, which is later filled with already compared ballots. Then the sorting procedure starts as follows:

The sorting server takes the first two elements

$$ballot_1 = (E^{pk_S}(pk_1), E^{pk_T}(v_1), E_A^{pk_{AS}}(ts_1))$$

and

$$ballot_2 = (E^{pk_S}(pk_2), E^{pk_T}(v_2), E_A^{pk_{AS}}(ts_2))$$

out of the list L and compares their IDs (i. e. their public keys, which identify the voters):

First, the server computes

$$E^{pk_S}(div) = \left(\frac{E^{pk_S}(pk_1)}{E^{pk_S}(pk_2)}\right)^r = E^{pk_S}(\left(\frac{pk_1}{pk_2}\right)^r).$$

The division as well as the exponentiation is done componentwise and the server proves with the Chaum-Pedersen-protocol [CP93] that he used the same r for each component. If $pk_1 = pk_2$, we have

$$div = \left(\frac{pk_1}{pk_2}\right)^r = (1,1)^r = (1^r, 1^r) = (1,1).$$

The value div is opened with a proof of correct decryption. If the decryption of $E^{pk_S}(div)$ results in $div = (1,1)$, the server opens r in order to show that $r \not\equiv 0 \mod p$. If the IDs do not match, $ballot_1$ is compared to the next ballot in the list L. This is continued until the end of the list is reached or a ballot is found whose identity matches the one of $ballot_1$.

If the IDs match, the sorting server calculates:

$$E^{pk_S}(diff) = \frac{E_A^{pk_{AS}}(ts_1)}{E_A^{pk_{AS}}(ts_2)} = E^{pk_S}(ts_1 - ts_2).$$

Then $diff$ is decrypted with a proof of correct decryption. If $diff > 0$, $ballot_1$ is older.

Both ballots are deleted from L, the newer ballot is appended to N and not compared again in this round. The procedure is repeated until L is empty. Please note that each element is compared only once.

When L is empty, the remaining ballots in N are shuffled, $L := N$, and N is reinitialized as an empty list. The whole procedure is repeated until N does not get shorter anymore.

Revoting-Phase 4: Proving one ballot per voter

The sorting server shuffles the $E^{pk_S}(pk_i)$ parts of the ballots, and proves the correct shuffling. Then he opens the ciphers and proves correct decryption. All proofs and the plaintexts are published on the public bulletin board. Everyone can check that each pk_i occurs only once.

Revoting-Phase 5: Tallying

The tally server shuffles the $E^{pk_T}(v_i)$ parts of the ballot, proves the correct shuffling and publishes the shuffled ciphers and the proof on the public bulletin board. Then the tally server decrypts these shuffled ciphers and publishes their content with proofs of correct decryption. The tally can be computed by everyone from the set of decrypted choices v_i. At this point, the bulletin board contains all necessary information to verify that the tally is calculated correctly, i. e. only ballots from eligible voters were processed, and of each voter who has cast ballots the newest ballot is counted.

5.3.5. Discussion

We now discuss some aspects of our scheme, including the distribution of servers and other possible improvements. The author is aware that this scheme, especially the sorting algorithm, can be improved. This scheme is supposed to be a proof of concept.

Homomorphic tallying

The encryption function with which the voter's choice itself is encrypted, is, in fact, arbitrary. Therefore, instead of using a shuffle in the tally phase, the tally can be computed homomorphically: the voters encrypt their choice v_i for example with exponential Elgamal, which is additively homomorphic, and prove validity of v_i upon casting. This would make our scheme more efficient and stronger against coercion, since no plaintext ballots would be decrypted, and therefore, pattern voting would not be a problem. In its current version, our scheme is only secure for 1-out-of-n choices.

Distribution of servers

Since the only critical operation of the sorting server and tally server is shuffling and decrypting Elgamal encryptions, we can use a threshold version of Elgamal (see Section 2.3.3.3 and [Ped91b]), where the decryption key is distributed among several servers and decryptions are done without ever reconstructing the decryption key (it is used in the exponent only). So we can implement both the sorting server and the tally servers as a set of servers which share the corresponding secret keys for decrypting differences of timestamps and the voter's choices, respectively. For the voters, this makes no difference since encryption with threshold-Elgamal works exactly the same way as with ordinary Elgamal: there is one joint public key. Only the decryption process is different. The shuffling of encrypted votes can be done jointly by these servers as well.

The sorting server also needs to choose randomness for masking the difference of identities. The masking can be done jointly by several servers, by exponentiating consecutively. The masking randomness of each server is then opened if the difference is the neutral element.

The casting server might learn the number of revotes of a single voter from her IP-adress, but no other critical information. It has no decryption keys. To hide the number of revotes, there could be more than one such server from which the voter can choose, or the voter can use an anonymization network.

The CRS used for the GS-proofs is basically the public key of an Elgamal encryption, since we use the SXDH instantiation of GS-proofs in the binding setting. Therefore, the CRS can be jointly created by a set of trustees. Since we do not need

to actually extract any witnesses, all data which is used to create the CRS can be deleted immediately after its creation.

Everlasting Privacy

Vote privacy in our voting scheme is computational, it depends on the encryption function used for encrypting the votes. In 2012, Demirel et .al.[DvdGSdSA12] showed how to introduce everlasting privacy in voting schemes based on homomorphic tallying. The ballot part encrypting the votes is not used in any algorithm of our voting scheme except the tallying, so its encryption is in fact arbitrary, and we can as well use for example exponential Elgamal (i. e. encrypting H^m instead of m with normal Elgamal) and let the voter prove that her ballot contains a valid vote upon casting. This would additionally require a trusted authority to which the voter proves this validity. This authority would learn the encryption of the vote, which would otherwise, in our current voting scheme, be seen by anyone. However, the authority which learns the Elgamal encryptions would also need to track which ballot is sorted out, i. e. which encryption is not to be included in the tally. We leave a secure realization as an interesting open problem.

Belated coercion

A remaining attack, loosely related to everlasting privacy, would be that the adversary coerces the voter after the voting phase in order to retroactively break privacy, and forces her to open one of the votes. Without a warning of the adversary's intentions of coercion, the voter cannot use revoting as a deceiving strategy anymore. We consider this attack harmless. At this stage, such a coercion would not have an impact on the tally anymore, and the voter can always claim she has forgotten her randomness used for ballot creation since she was not asked in time to keep it.

Forced Abstention

In the above described version of our voting scheme, the identities of the voters who have cast a vote is opened in the end. Therefore, a voter could be coerced not to participate in a vote. This problem could be solved by proving that of the remaining ballots, each contains an encryption of a different public key. For this we could use the same techniques for identity comparison as we used in the sorting algorithm.

Manipulating timestamps

A voter can cast a ballot with a wrong timestamp in cooperation with the casting server. Therefore, a corrupted casting server could coerce the voter to cast a ballot with a future timestamp. However, the timestamp can also be checked by a third party without learning the voter's identity.

5.3.6. Security Discussion

In this section, we discuss the security of our voting scheme. As we shall see, it is not yet perfect since our sorting algorithm leaks some information about the number of voters who revoted. However, if more than one voter revotes, the algorithm does not leak information about which voter revoted how often.

5.3.6.1. Privacy Properties

Secrecy of the vote

The vote is encrypted with Elgamal, and not touched during the election process except for re-encryption and shuffling until the tally is computed with standard

techniques. Before the tally is computed, the vote is detached from the rest of the ballot, in particular, the voter's identity, by shuffling and proving the correctness of the shuffle with a zero-knowledge proof. Therefore, vote secrecy holds depending on the security of the Elgamal encryption.

Secrecy of the act of revoting

The identity of the voter is never shown on the public bulletin board until all but her newest ballots are sorted out. By then, her ballot is unlinked from her cast ballot: the plaintext timestamp is never revealed, and neither is the signature. The GS-proofs do not enter the sorting process. The only information the ballots contain about their voter is the voter's encrypted public key, which is encrypted with different randomness for each ballot, so under the assumption that the encryption scheme is secure, this encryption cannot be used to identify ballots of the same voter. All possible tags, i. e. randomness used to create encryptions and proofs, are destroyed by rerandomization. Therefore, the only possibility to link a counted ballot to a cast ballot is information leaked by the sorting algorithm. We take a closer look at this algorithm now.

In the sorting process, we see that two ballots belong to the same voter, but not which ones. A ballot is only compared once with another ballot of the same voter in one round, i. e. between two rerandomization procedures. Therefore, a chain of ballots of the same voter cannot be deduced. But we do in fact see that there are revotes, but not how many of them are done by a single voter. The sorting algorithm is not perfect yet however. It leaks some information about the number of voters who have revoted: if in the beginning of a round some ballots are processed that do not have any matches, we know that these ballots are from different voters, so the algorithm leaks an upper bound of the number of voters who revoted. We leave the creation of a better sorting algorithm as an open problem, but strongly believe that the problem is solvable.

Receipt-Freeness and Coercion-Resistance

The only data the voter can influence, i. e. use to encode information, is the first appearance of her ballot on the bulletin board, which can be invalidated by a revote. This first appearance is rerandomized and shuffled with other ballots with any further processing, so it cannot be linked to a ballot which is sorted out, or to a ballot which is included in the final tally. The encrypted ballot included in the final tally does not reveal any information about the form of the vote. Moreover, concerning the adversary's view on published data, until the end of the sorting process, each processed vote could belong to any initially cast ballot and have been cast by any eligible voter. In particular, the voter does not have any material to prove that a certain encryption included in the tally encrypts her choice. However, we count by shuffling and opening ballots, so plaintext ballots are revealed. So coercion might be possible if we allow the voter to cast too flexible ballot formats, for example with vote splitting or write-in candidates, in which a voter could encode her identity.

5.3.6.2. Verifiability

Individual verifiability

The voter can check that all her ballots, especially her newest ballot, appear on the public bulletin board. She can then check that all her ballots, most importantly her newest ballot, enter the sorting procedure. From there on, the voter can check

that the ballots of all voters are processed correctly (see universal verifiability). At the same time, since this ballot does not visibly overwrite a particular older ballot (ballots are rerandomized and shuffled before they are compared), privacy is assured.

Universal verifiability

Universal verifiability can be divided into three aspects in our voting scheme. Everyone can check that only ballots cast by eligible voters are counted. Everyone can check that these cast ballots are sorted out correctly, and that the votes selected for tallying are tallied correctly.

- **Verifiability of eligibility** Everyone can check that each ballot on the bulletin board belongs to an eligible voter, by checking the GS-proofs of signature knowledge attached to the ballot. These proofs contain a part which proves that a certificate of the key used for signing exists. Therefore, under the assumption that the signature scheme used for certification is unforgeable, only an eligible voter can have created the signature.

- **Voter authentication** The proof of knowledge of a signature created in Phase 1 binds the ballot to a certain voter's signing key. It is proven that this key corresponds to the public key encrypted on the ballot. So under the assumption that the used GS-proofs are sound, the ballot is signed by the voter who's public key is encoded in the ballot encryption.

- **Verifiability of the Sorting Process** As mentioned above, it is proven that each ballot is signed by the voter who's public key is encoded in the ballot encryption. This public key is used to sort out old ballots. Everyone can verify that whenever two ballots are compared, one of them is thrown out iff they belong to the same voter, and if so, it is the older ballot that is thrown out. At the end it is proven that of each voter, the newest ballot is among these which will be counted.

- **Verifiability of the Tally** Since eligibility is publicly verifiable, everyone can check that only votes from eligible voters enter the sorting process. Since the process of sorting out old votes is publicly verifiable, and it is proven that of each voter, only one ballot is selected for counting, everyone can verify that of each voter who has cast a ballot, a ballot will be counted, and that this ballot is the newest ballot of this voter. Altogether this proves that only the newest ballot of each voter enters the tallying process. The tallying process itself can be verified in a standard way by checking the proof of correct shuffling and decryption of the encrypted ballots.

5.3.7. Analysis with the Taxonomy

An analysis of our revoting scheme with our taxonomy can be found in Appendix C. The analysis showed that our scheme is rather strong regarding most requirements, but it is not suitable yet for large-scale elections because of its lack of efficiency. Apart from the scalability requirement, most requirements are fully met, one could be met with minor adjustments and two requirements are almost met. The two latter requirements are coercion-resistance (because of the minor information leakage of the sorting algorithm), and provability of a fraud if the voter's ballot is not published.

5.4. Delegated Voting

The idea of delegated voting is that a voter can choose between casting her own vote or instead giving the weight of her vote to another eligible voter of her choice. We call the person to whom votes are delegated a *proxy*. By default, the weight of a voter's vote is 1. This weight increases with delegations this voter gets in the case that she acts as a proxy.

The concept of vote delegation poses some interesting challenges to the construction of a cryptographic voting scheme. Not only has the tally to be proven correct, a voter also needs to be given a proof that her vote was delegated to the right proxy. At the same time, the proxy must not know if a voter delegated her weight to him, for this might open possibilities for coercion. This makes balancing voter privacy and verifiability an even harder challenge than in ordinary cryptographic voting schemes.

The possibility of verifiable revoting is essential for delegated voting, since it is often required that a voter can change her mind between delegating and voting by herself at anytime during the voting phase.

In this chapter, we first introduce the new requirements and challenges which delegated voting gives to us cryptographers. Then we review an existing approach, Agora, and discuss how these challenges are met. Agora is a cryptographic voting scheme whose original idea was to enrich Helios with the feature of vote delegation. Agora does not provide secrecy of the revote, revoting is implemented by substituting encrypted ballots on a public bulletin board. We do not evaluate Agora – we use the scheme as an example to show some pitfallss.

After having looked at an existing scheme, we present our own solution which combines our revoting solution with a new paradigm we call *vote fetching*. The idea of vote fetching is that instead of delegating, the voter fetches a vote which was provided by the proxy without letting the proxy know that she is doing so. As we will see, this has several advantages over the delegation approach, since it is more flexible in many ways.

5.4.1. Liquid Democracy and Delegated Voting in a Nutshell

Delegated Voting is an implementation of a so-called Liquid Democracy. The underlying ideas of vote delegation were introduced by Miller in the 1960s [Mil69]. In 2002, Ford introduced some basic rules for delegated voting [For02], Green-Armytage suggests a design for a vote delegation system in [GA14].

Liquid democracy can be seen as a hybrid between direct and representative democracy, where vote delegation acts as a tool for overall decision making. Instead of having a parliament take all the decisions, voters have the choice of giving their own opinion in a poll or delegating the weight of their vote to another eligible voter, a so-called *proxy*. The proxy himself can also delegate his vote to yet another proxy, and Ford even suggests delegating delegation decisions [For02]. It is usually required that a voter can change her mind anytime, as stated for example in [GA14], so after delegating, the voter can take back her weight from the proxy and vote by herself or delegate to another proxy. Another non-cryptographic requirement is that a delegation stays intact for several polls so the voter does not have to be involved in each poll in person.

5.4.1.1. Bryan Ford's Rules for Delegated Voting

In [For02], Ford introduced some basic rules for delegated voting. Interpretation and implementation of these rules differ between existing schemes.

Choice of Role

Every voter has the right to refuse to be a proxy and instead vote as an individual only.

Low Barrier to Participation

In principle, every voter has the chance to become a proxy, but a low barrier is allowed. Ford explicitly states that becoming a proxy shall "not require campaigning or winning a competitive election".

Delegated Authority

The weight of the vote of a proxy is the sum of his own weight an the weight of the voters who delegated their vote to him.

Privacy of Individuals

Voter privacy holds for voters who are not proxies. Voter's choices of candidates as well as delegations to proxies remain secret. A proxy does not know which voter has delegated her vote to him.

Accountability of Delegates

Proxies are called *delegates* by Ford. Each proxy has to account for his choice, therefore Ford demands that their choices are made in public.

Specialization by Re-Delegation

Like voters, a proxy can delegate his choice to another proxy, who gets the full weight of the delegating proxy. This allows several levels of specialization on different topics.

5.4.1.2. Anomalies in Elections with Vote Delegation

The possibility of delegation introduces some known anomalies and peculiarities. Two of them are important for this work:

Delegation Cycles

A delegation cycle occurs when proxy A delegates to proxy B, who delegates to proxy C, who in turn delegates to proxy A. If this happens, the weight of all votes delegated to a proxy in the cycle is lost since it is not given to any actual candidate. Especially if proxies vote in secret, this can happen without anyone noticing. We will see that cycle resolution without introducing possibilities for coercion is a challenging problem.

Unknown Weight

Proxies are not aware of their own weight upon vote casting. Therefore it is difficult for a proxy to cast protest votes since his choice might have significant influence on the outcome. As we will see, our solution with vote fetching solves this problem.

5.4.2. Requirements Specific to Voting Schemes with Vote Delegation

In addition to the usual requirements for internet voting schemes, delegated voting has some requirements which come from the possibility of delegation. In this section we will briefly introduce such additional requirements, which have been identified in the context of Jonathan Bickel's bachelor thesis [Bic12]. The list of requirements listed here is a slightly advanced version of the list in Bickel's thesis. As we shall

see, the listed requirements are not accomplishable concurrently; some are excluding each other. A more detailed analysis of the dependencies between the requirements can be found in [Bic12]. Several requirements might be desirable in one election, whereas in another election the reverse requirement needs to be achieved. Examples for such requirements are *splitting*, *transitivity*, or *vote secrecy for proxies*.

The additional requirements are:

Choice of mode

During the whole voting phase the voter shall be able to choose between voting herself and delegating to a proxy. She can take away her vote from the proxy at any time during the voting phase and vote by herself or delegate to another proxy.

Choice of role

Each voter can choose if she wants to act as a proxy or not. A weaker form is that she can refuse to be a proxy. This is especially important if voter privacy for proxies does not hold.

Unknown weight

Proxies should not know the weight of their vote before they cast their decision. Otherwise they might be vulnerable to social pressure and favored victims of bribe and coercion.

Coercion-resistance: No forced delegation

The adversary must not be able to coerce the voter to delegate to a certain proxy. There's a similar problem here as with write-in candidates: the adversary could force the voter to delegate to an unlikely proxy and then look at this proxy's weight.

Coercion-resistance: No forced delegation to the adversary

This can is a special case of the requirement above. However, it is harder to achieve if the voting scheme warns the voter from delegation cycles. Therefore, we state it as an extra requirement. The adversary must not be able to coerce a voter to delegate her vote to him. So the adversary must not see who, or even whether someone has delegated a vote to him. Please note that if the voting scheme gives warnings if a voter induces a delegation cycle, the adversary can test who delegated their vote to him simply by trying to delegate to these voters.

Vote secrecy for proxies

Privacy can be seen twofold in delegated voting, since we have two different roles. This requirement asks for vote secrecy for proxies and is in competition with the accountability of proxies. While some elections might wish for vote secrecy for proxies, others might explicitly want proxies to vote openly.

Coercion resistance of proxies

The proxies shall not be coercible in a certain way. This is a stronger version of the requirement above, and is impossible to achieve if the proxies have no voter privacy.

Unawareness of role

The voter does not know if she is a proxy or not. As stated in Bickel's thesis, it can prevent the voter from voting "unfree", i. e. being influenced by the knowledge of her power. Maybe the role can be revealed after the voting phase. This requirement contradicts with *Choice of role* if the voter cannot deny to be a proxy. If she can, this is just a special case of *unknown weight*.

Verification of delegation

The voter can verify that her vote was delegated to the right proxy.

Accountability of proxies

The voter can verify what has become of her vote: what her proxy voted for or where her vote was delegated to.

Vote splitting

If a proxy has weight more than one, he can split (or, not split) the weight to different choices or further delegations. This prevents concentration of power that could be caused by specialization through repeated re-delegation [For02].

Transitivity

A proxy can further delegate to another proxy.

5.4.3. Agora: An Existing Solution

As an example, we now take a look at an existing cryptographic voting scheme which allows vote delegation, and discuss its coercion-resistance. The purpose of this section is not an evaluation of this voting scheme, but a demonstration of potential difficulties.

The Agora Ciudadana project [6] is an open source project which implements a cryptographic voting scheme with support for vote delegation. It offers voter privacy and end-to-end public verifiability while using easy to understand cryptographic techniques. The project's original intent was to extend Helios with the possibility to delegate votes, to obtain an easy to use voting scheme. We briefly describe the voting scheme in a simplified version we call *simplified Agora*. Our version is close to the original version, but we abstract from the fact that the voter's choices are actually preference choices. We will shortly review the ideas of the scheme to show some pitfalls one has to take care of when designing a fully coercion-resistant voting scheme with vote delegation.

Agora depends on a trusted voting authority, consisting of a set of trustees, for privacy. The trustees share an ElGamal secret key for the decryption of ballots. Vote delegation is implemented by having two elections in parallel: an ongoing election in which voters choose their proxies, and in which tallies are done periodically, and a direct election in which each voter can cast a direct vote, which will then be counted instead of her delegation. We call the first election the *proxy election* and the second election the *direct election*.

5.4.3.1. Pre-Voting Phase

Before an election starts, the members of the voting authority, consisting of a set of trustees, jointly set up parameters for a threshold ElGamal encryption: each trustee creates his own key pair, the public keys are then combined to build the joint public key used for the encryption of ballots. Remember that we have two elections, which can be set up with a different set of trustees and a different public key. The two joint public keys are published on a public bulletin board, together with the individual public key of each trustee. Eligible voters can register themselves to be proxies. They can use pseudonyms as proxy names to hide their identity.

[6] Website of the Agora Ciudadana project: `https://github.com/agoraciudadana/agora-ciudadana`

5.4.3.2. Proxy's Voting Phase

Prior to the voting period, the proxies cast their choices. Those choices are public, they are published on the bulletin board in plaintext before the actual voting phase of the direct election starts. So the only secrecy a proxy has is by hiding his identity behind a pseudonym. Each proxy can also cast a direct vote in secret in his role of an eligible voter.

5.4.3.3. Voter's Voting Phase

To cast a ballot, the voter encrypts it with the joint public key of the trustees, using the ElGamal encrytion scheme, and digitally signs her vote with her electronic identity card for authentication. The encrypted, signed ballots are published on a public bulletin board to provide individual verifiability. Revoting is implemented by overwriting the voter's choice on the public bulletin board with her new ballot. Everyone can check that there is at most one ballot per voter since the ballots are signed, and the voter can check that her newest ballot is online.

5.4.3.4. Vote Delegation

As stated above, vote delegation is implemented as a separate proxy election which takes place in parallel to the direct elections. It is implemented as a regular election, except that the voting phase never ends and the tally is repeated periodically, the delegations are evaluated for each related direct tally. The choices on the ballot, consisting of the names of the proxies, are updated when new proxies join the system. The idea is that the voter can have a permanent delegation in the proxy election which she can overwrite anytime by casting a new ballot in the proxy election. For each direct election which is linked to this proxy election, the voter can cast a direct vote, in which case her choice of delegation in the permanent election is be ignored. Since cast ballots are published with a signature of the voter, this can be publicly verified. Delegations can be transitive, i. e. a proxy can delegate his vote to another proxy.

5.4.3.5. Post-Voting Phase

There are two tallies: the tally of the direct votes and the tally of the delegations chosen in the ongoing election. The two tallies are basically done in the same way: the corresponding trustees build up a re-encryption mixnet to anonymize the ballots, i. e. each trustee reencrypts and shuffles the set of ballots, and proofs the correctness of the shuffle with a zero knowledge proof. The shuffled ballots are then jointly decrypted using the threshold ElGamal system.

The tally of the delegation is done as in the direct election, but ignores ballots of voters who have cast a direct vote. This tally result determines the weight each proxy vote gets in the direct election: the weight of the proxy's choice equals the number of delegations given to this proxy. The overall tally is then done by summing up the result of the direct election and each proxy's choice multiplied by his weight according to the current tally of the proxy election.

5.4.3.6. How does Simplified Agora Fulfil the Additional Requirements?

Choice of mode

The voter can have a permanent delegation in the proxy election, but cast a direct vote in the real election at any time. This becomes apparent to other voters.

Choice of role

The voter can be both at the same time: She can create a proxy under a pseudonym, and additionally cast a private vote in her role of a voter (which she can, of course delegate to herself). A proxy with no delegations has weight 0.

Unknown weight

Proxies do not know their weight until the tally of the proxy election is done. However, since delegations persist if not updated, a proxy might infer her weight approximately by observing the overall changes of delegations and taking into account past tallies of the proxy election.

Coercion-resistance: No forced delegation

Each voter can create a proxy under a pseudonym, and everyone sees this proxy's weight after the election. So the adversary can create a pseudonym and coerce the voter to cause at least one delegation to this pseudonym.

Coercion-resistance: No forced delegation to the adversary

See above.

Vote secrecy for proxies

The proxy's choice as a proxy is not secret, however, the proxy has a secret vote as an eligible voter, and can hide her proxy behind a pseudonym.

Coercion resistance of proxies

See above.

Unawareness of role

The voter knows if she has created a proxy or not. She also knows that her vote in the role of the voter has only weight one.

Verification of delegation

End-to-end verifiability is provided.

Accountability of proxies

Since proxies vote openly, accountability holds.

Vote splitting

This depends on the rules and implementation of a given election.

Transitivity

This depends on the rules and implementation of a given election.

5.4.3.7. Coercion-Resistance of Simplified Agora

In this section, we introduce some coercion attacks on the Agora voting scheme.

Recognizable revoting

Revoting is implemented by updating the voter's encrypted ballot. Therefore, a voter can be coerced not to revote. Moreover, it becomes apparent whether the voter casts a direct vote, so she can be coerced not to do so.

Evaluation of voting history

There is an attack[7] on coercion-resistance using the results of past elections: the tally of the election for the choice of proxy is repeated periodically. Since ciphers are only updated if the voter changes her delegation, it is easy to deduce information about a voter's choice of proxy by comparing the new distribution of the weights among proxies with the information which voter has updated her ballot in the election of proxies. In principle, a voter could cast a new delegation for the same proxy, but it is still public which voters cast a new ballot and which do not.

Forced delegation

As described when listing the requirements, a voter can be coerced to delegate her weight to a proxy which would otherwise most likely get no delegations. This is similar to the write-in problem described in Chapter 4 and not a problem specific to this voting scheme. It is, however, an attack with more impact, because it is not merely a forced abstention, but a vote which the adversary can direct to a particular candidate, provided the corresponding proxy (which has no privacy) is under the adversary's control.

Proxy Secrecy

Proxies vote openly, before the election phase. So a voter can either use the vote as a suggestion, delegate to another proxy, or decide that she is content with her current delegation. Since the choices of the proxies are not secret, proxies can always be coerced towards a certain choice, giving the adversary influence in the decision making of voters: even if he might not have direct influence on a voter's choice through coercion, because the voter sees his proxies choice and decides by himself if she follows this choice or not, the adversary still has influence on the provided suggestions. However, the openness also implies that accountability holds. Everyone in the election is free to cast a vote of her own choice after looking at the proxie's choices, which might be biased by the adversary.

5.4.4. Vote Fetching

In this section, we introduce our new paradigm called *vote fetching*, which takes a somewhat reverse approach to vote delegation: the voter does not delegate her choice. The idea of vote fetching is that proxies provide an amount of ballots. Then, instead of delegating her choice, the voter fetches a ballot from the proxy, reencrypts it and casts it as her own. As is the idea of delegated voting, the voter can fetch as many ballots from different proxies as she wishes, changing her mind anytime. This approach is more flexible than delegation in several aspects.

The idea of vote fetching was developed in joint work with Jörn Müller-Quade, Dirk Achenbach and Bernhard Löwe. In this section, we first describe the idea of vote fetching, and then introduce a voting scheme that combines the revoting idea with the vote fetching solution.

5.4.4.1. Motivation for Vote Fetching

In schemes with vote delegation, a proof has to be given that the choices of each proxy are counted with the correct weight. A usual way to do this is that, in addition to the tally, the weight of each proxy is published. In this case a voter could be coerced to delegate to an unlikely proxy, which leads to a similar problem as with write-in candidates. Moreover, group coercion is possible. An adversary

[7] This attack has been presented on `https://blog.agoravoting.com/`.

could provide a proxy for a certain group of people, coerce the group to delegate to this proxy as a whole, and punish the whole group if even one delegation is missing. With vote fetching, the weigh of each single proxy stays hidden, while a proof of correct counting can be given. Moreover, the voter does not have to be provided with a proof of correct delegation, and the proxy does not see who fetches votes.

Another advantage of vote fetching is that it provides an easy way of vote splitting. Proxies can define their choice more fine-grained, for example by providing a certain amount of ballots for each candidate, according to a chosen distribution. This type of vote splitting is explicitly wished for by Ford [For02].

With the vote fetching approach that we now can choose freely between letting the proxies vote openly by providing plaintext ballots or letting them provide ciphertexts. So the choice between secrecy of the proxy's choice and accountability is not restricted by the scheme.

Another advantage is that delegation cycles are not a problem. A proxy A who would delegate to another proxy B, now simply fetches ballots from B. If B also fetches ballots from A, these ballots are still counted when fetched by voters. A cycle without vote splitting would lead to the situation that neither proxy provides any ballots, which they would be aware of, so no weight is lost by an unintended cycle.

5.4.4.2. Possible Difficulties with Vote Fetching

One advantage of delegated voting is that the voter does not have to participate in each election in person. She can have a permanent delegation instead, which stays intact in following elections until the voter revises her choice. With the vote fetching approach, this can be achieved by a script which automatically fetches the votes. A problem is then, how is eligibility proven? How is the ballot signed? By combining vote fetching with our revoting solution, we can solve this problem as discussed below.

5.4.5. Fetch-and-Cast: A Delegated Voting Scheme with Vote Fetching

Fetch-and-Cast is a voting scheme which combines our revoting solution with the idea of vote fetching. The idea of our scheme is as follows: the proxy provides encrypted ballots. The voter fetches these ballots (with a script that does this for each election), reencrypts them and casts them. For the voter being able to change her mind whenever she wants, we use our revoting technique. The proxy will not recognize her ballot later, because it is rerandomized.

Please note that we do not specify what a choice looks like. In fact, delegated voting schemes often implement something similar to STV votes, where voter's choices are preferences, and tallied for example with the Schulze-method [Sch11]. Cryptographic counting methods for STV votes are introduced by [TRN08] and can be used on top of this scheme, since in our revoting scheme, the encryption of the vote itself is arbitrary.

In the following, we describe the basic protocol Fetch-and-Cast. Several possible extensions are discussed below. Since it is very close to the revoting protocol described in Section 5.3, we only describe the changes to this protocol. The participants are the same as those in the revoting scheme, plus the proxies and an additional server, called a *reencrypter*, which rerandomizes ciphertexts. The votes which can be fetched from the proxies are published on the public bulletin board

which is also used for vote casting. We prefer this to letting proxies provide ballots on their own webpage since this way they cannot track who fetched ballots from them.

5.4.5.1. Setup and Pre-Voting Phase

The setup is done as in the revoting scheme. Additionally, eligible voters can create a proxy.

5.4.5.2. Proxy's Vote Generation Phase

Before the voting phase starts, the proxies provide an equal amount n of encrypted votes for the voters. For this, each proxy chooses his preferred candidate(s) in the form of a set of votes $v_1, \ldots, v_n$, encrypts them and sends them under his name to a reencrypter. The proxy does not create a full ballot. Timestamp, public key and proof of knowledge of a signature will later be added by the voter. To prevent the proxy from encoding his identity or other information in provided ballots, he has to prove the validity of his encryptions. Since the encryption of the vote is an ElGamal encryption, this can be done with a standard proof as described in [CGS97].

If the proof is correct, the reencrypter rerandomizes the ciphers and proves correct reencryption. After this, both the proxy and the reencrypter sign the encrypted ballot. The encryption and the signatures are published on a public bulletin board under the proxy's name. If vote splitting is allowed, all proxies should provide the same amount of ballots to prevent coercion attacks like forcing a proxy to only provide one ballot.

Our scheme is trivially adaptable to letting the proxies vote openly: they publish their choice and the voters just cast a vote for what their proxy publishes.

5.4.5.3. Voter's Voting Phase

After the proxies have published their ballots, the voters can cast their ballots, as described in our revoting scheme. To do this, each voter either encrypts her own choice or uses one of the encryptions provided by her proxy of choice. If she uses a vote from a proxy, it must be reencrypted before usage. Encryptions that are already on the bulletin board are not accepted. As in the revoting scheme, the voter attaches an encryption of her public key to the ballot, and the casting server encrypts the current timestamp. The voter then signs the ballot as in the revoting scheme and the ballot is processed as in the revoting scheme.

5.4.5.4. Post-Voting Phase

The post voting phase is done exactly as in the revoting scheme.

5.4.5.5. Possible extensions

In this Section, we discuss some possible extensions to Fetch-and-Cast.

Transitive Delegations

Vote fetching does not directly support transitive delegations. One way to implement these is to let proxies fetch and rerandomize votes from other proxies. Another would be to allow proxies to encrypt names of proxies in their ballot. If such a ballot is opened in the tally, a vote from the proxy indicated in the ballot is randomly chosen from the votes this proxy has provided. All such chosen votes are then mixes and rerandomized again and opened. This offers less privacy for the proxies, however, and it would give proxies partial information about their weight.

Permanent delegations

Permanent vote delegations that stay in tact for more than one election could be realized in the following way: the voter casts as a “permanent delegation" in the permanent election a ballot that instead of a vote contains the name of a proxy. The ballot contains a timestamp. Therefore, because of the sorting process, the delegation is naturally overwritten by a direct vote. Whenever such a vote is opened in the tally, a random vote from the corresponding proxy can be chosen, similar to the solution to transitive delegations. As above, this offers less privacy for the proxies and rings back the problem of coercing a voter to delegate to an unlikely proxy with such a permanent delegation. This would give proxies partial information about their weight.

Accountability

The voting scheme could also be setup in a way that voters can ask their proxies for ciphers directly if they want accountability. This way, the proxy could prove the content of the encryption per designated verifier proof [JSI96] or cut-and-choose [Ben06]. Please note that a proxy cannot coerce a voter to cast his vote that way, since the voter can always revote. This cannot be done with votes on the bulletin board since otherwise an adversary could just let the proxy open all his votes, so votes have to be fetched directly from the proxy. Since voters can revote and do not have to cast fetched votes, the proxy does not know if he voter actually casts a vote she fetches this way. Even if the proxy recognizes a vote on the bulletin board, he does not know if the voter revoted afterwards.

5.4.5.6. How does the scheme meet the requirements above?

Choice of mode

The voter can choose between fetching a vote from a proxy and casting her own. She can overwrite her choice anytime with a revote. She can even overwrite her own vote with a delegation.

Choice of role

Voters can choose if they want to provide votes which can be fetched. So they can choose their role.

Unknown weight

Proxies do not know how many votes of them were fetched, cast and not overwritten.

Coercion-resistance: No forced delegation

The basic scheme is resistant against this form of coercion, but the adaptions for permanent delegation and transitivity are not.

Coercion-resistance: No forced delegation to the adversary

See above.

Vote secrecy for proxies

Proxies have vote secrecy in the basic scheme.

Coercion resistance of proxies

Proxies cannot revote since their ballots are linked to their identity. So if the adversary is present while the proxy creates his encrypted votes, he is coercible. However, this attack is more expensive to the adversary than with open votes of proxies.

Unawareness of role

The voter knows if she provides votes for fetching or not, so she knows her role, but not her weight.

Verification of delegation

Since the voter fetches votes, no delegation needs to be proven.

Accountability of proxies

Accountability can be achieved as discussed above.

Vote splitting

Vote splitting is trivially possible by providing different votes according to a certain distribution.

Transitivity

Transitivity can be provided as discussed above.

5.4.6. Discussion

We have seen two extremes here: in the Agora voting scheme, proxies vote openly and can therefore be influenced by the adversary. Therefore, one cannot know if a suggestion by a proxy comes from the proxy himself or from an adversary. On the other hand, Agora provides full accountability. In Fetch-and-Cast, the adversary has as little influence on the proxy's choices as on the voter's choices. The problem with this approach is that there is no proxy accountability at all, the voter has to trust his proxy blindly. We have discussed how to weaken this trade-off and make a compromise by letting the voter get proxy votes which she can open. So it seems that in delegated voting, there is a big trade-off between incoercibility and accountability of proxies. We will not discuss here which of these properties is more important. However, we have offered techniques with which this trade-off can be overcome to some extend.

6. Conclusion and Future Work

In this work, we discussed aspects of coercion-resistance from a wide viewpoint. We introduced new solutions in some aspects, and identified interesting open problems in some others. We introduced a taxonomy and an analysis roadmap with which even the most different voting schemes can easily be analyzed and compared. Answering the questions provided by the roadmap gives a quite accurate idea about what is ensured by an analyzed voting scheme, which assumptions it is based on, and what its strengths and weaknesses are. Future work would be a wider range of requirements, including the important requirements of usability and efficiency. Another important and interesting future work would be generally applicable formal definitions of the identified requirements. The model of Küsters et al. [KTV12] is rather general and seems to be a promising candidate for an underlying model.

We showed the flexibility of Bingo Voting in its application in a real-world election. We provided Bingo Voting with write-in support, and showed the impact of this voting type on coercion-resistance. To capture coercion-resistance caused by information-leakage of the tally result, we provided definitions and techniques for a fuzzy tally representation. With these definitions, we showed that there is a relation between vote privacy and database privacy. Future research could show if there are more such relations. We introduced techniques to present the tally in a fuzzy way, while preserving a suitable amount of verifiability. A more fine-grained blurring technique which allows for more sophisticated representations like the distribution of seats in parliament might be interesting.

For a publicly verifiable revoting scheme to offer coercion-resistance, it is important that the act of revoting is secret. We introduced a proof-of-concept revoting solution which is publicly verifiable and coercion-resistant to a suitable amount. However, our sorting process leaks a minimal amount of information about the participating voters' overall revoting behavior. A next step would be research on a better sorting algorithm which provably leaks less information, to be able to formally prove security of the revoting scheme in a suitable model. Another direction of improvement of the revoting scheme is working towards more efficiency. Introducing everlasting privacy in our revoting scheme seems possible but not trivial, and presents another interesting open problem.

Delegated voting is a relatively young election type, which has barely been addressed in cryptographic literature but poses interesting challenges for cryptography.

We introduced a voting scheme which captures vote delegation while achieving a suitable amount of coercion-resistance for the voters, while at the same time providing techniques for three different stages between proxy secrecy and proxy accountability, respectively. We gave a first idea to realize a permanent delegation. However, this solution does not offer full coercion-resistance. A fully coercion-resistant way to realize permanent delegations is an open problem. A formal model for vote delegation from a cryptographic viewpoint is still missing. Such a model, as well as a voting scheme with a rigorous proof of security, would be interesting future research.

Bibliography

[ACHdM05] Giuseppe Ateniese, Jan Camenisch, Susan Hohenberger, and Breno de Medeiros. Practical group signatures without random oracles. Cryptology ePrint Archive, Report 2005/385, 2005. `http://eprint.iacr.org/`.

[Acq04] Alessandro Acquisti. Receipt-Free Homomorphic Elections and Write-in Ballots. Technical Report 2004/105, International Association for Cryptologic Research, 2004.

[ACvdG10] Roberto Araújo, Ricardo Felipe Custódio, and Jeroen van de Graaf. A verifiable voting protocol based on farnel. In David Chaum, Markus Jakobsson, RonaldL. Rivest, PeterY.A. Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida, editors, *Towards Trustworthy Elections*, volume 6000 of *Lecture Notes in Computer Science*, pages 274–288. Springer Berlin Heidelberg, 2010.

[Adi06] Ben Adida. Advances in Cryptographic Voting Systems. PhD Thesis, MIT, 2006.

[Adi08] Ben Adida. Helios: Web-based open-audit voting. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, pages 335–348, Berkeley, CA, USA, 2008. USENIX Association.

[AdM09] Ben Adida and Olivier de Marneffe. Electing a university president using open-audit voting: Analysis of real-world use of helios. Electronic Voting Technology Workshop/Workshop On Trustworthy Elections, EVT/WOTE 2009, 2009.

[AFG+10] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In *Proceedings of the 30th Annual Conference on Advances in Cryptology*, CRYPTO'10, pages 209–236, Berlin, Heidelberg, 2010. Springer-Verlag.

[AH01] Masayuki Abe and Fumitaka Hoshino. Remarks on mix-network based on permutation networks. In Kwangjo Kim, editor, *Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 317–324. Springer, 2001.

[AHL+09] Arne Ansper, Sven Heiberg, Helger Lipmaa, Tom André Øverland, and Filip van Laenen. Security and trust for the norwegian e-voting pilot project *e-valg 2011*. In *NordSec*, pages 207–222, 2009.

[AKMQL14] Dirk Achenbach, Carmen Kempka, Jörn Müller-Quade, and Bernhard Löwe. How to (secretly) recast a vote. unpublished manuscript, 2014.

[AN09] Ben Adida and C. Andrew Neff. Efficient receipt-free ballot casting resistant to covert channels. In *Proceedings of the 2009 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, EVT/WOTE'09, pages 11–11, Berkeley, CA, USA, 2009. USENIX Association.

[AR06] Ben Adida and Ronald L. Rivest. Scratch & vote: self-contained paper-based cryptographic voting. In Ari Juels and Marianne Winslett, editors, *WPES*, pages 29–40. ACM, 2006.

[BÖ8] Michael Bär. Analyse und vergleich verifizierbarer wahlverfahren. Diploma thesis, Universität Karlsruhe (TH), 2008.

[BDPR98] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45. Springer, 1998.

[BDvdG13] Johannes Buchmann, Denise Demirel, and Jeroen van de Graaf. Towards a publicly-verifiable mix-net providing everlasting privacy. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography*, volume 7859 of *Lecture Notes in Computer Science*, pages 197–204. Springer, 2013.

[Ben06] Josh Benaloh. Simple verifiable elections. In *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, EVT'06, pages 5–5, Berkeley, CA, USA, 2006. USENIX Association.

[BHK+09] Jens-Matthias Bohli, Christian Henrich, Carmen Kempka, Jörn Müller-Quade, and Stefan Röhrich. Enhancing Electronic Voting Machines on the Example of Bingo Voting. In *IEEE Transactions on Information Forensics and Security Vol. 4*, pages 745–750, 2009.

[BHMQ+08] Michael Bär, Christian Henrich, Jörn Müller-Quade, Stefan Röhrich, and Carmen Stüber. Real World Experiences with Bingo Voting and a Comparison of Usability. Workshop On Trustworthy Elections, WOTE 2008, 2008.

[Bic12] Jonathan Bickel. Sicherheitsanforderungen an delegated voting und analyse existierender ansätze. Bachelor thesis, Karlsruhe Institute of Technology, 2012.

[BJR10] Shuki Bruck, David Jefferson, and Ronald L. Rivest. Towards trustworthy elections. chapter A Modular Voting Architecture ("Frog Voting"), pages 97–106. Springer-Verlag, Berlin, Heidelberg, 2010.

[BMQR07] Jens-Matthias Bohli, Jörn Müller-Quade, and Stefan Röhrich. Bingo Voting: Secure and Coercion-Free Voting Using a Trusted Random Number Generator. In A. Alkassar and M. Volkamer, editors, *VOTE-ID 2007*, volume 4896 of *Lecture Notes in Computer Science*, pages 111–124. Springer-Verlag, 2007.

[Bon98] Dan Boneh. The decision Diffie-Hellman problem. In *Proceedings of the Third International Symposium on Algorithmic Number Theory*, ANTS-III, pages 48–63, London, UK, UK, 1998. Springer-Verlag.

[BT94] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, STOC '94, pages 544–553, New York, NY, USA, 1994. ACM.

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `http://eprint.iacr.org/`.

[CCC+08] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, and Alan T. Sherman. Scantegrity II: End-to-End Verifiability for Optical Scan Election Systems using Invisible Ink Confirmation , 2008. `http://www.usenix.org/event/evt08/tech/full_papers/chaum/chaum.pdf`.

[CCC+10] Richard T Carback, David Chaum, Jeremy Clark, John Conway, Aleksander Essex, Paul S. Herrnson, Travis Mayberry, Stefan Popoveniuc, Ronald L. Rivest, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II Municipal Election at Takoma Park: The First E2E Binding Governmental Election with Ballot Privacy. Proceedings of the 19th USENIX Security Symposium, 2010.

[CDvdG87] David Chaum, Ivan Damgård, and Jeroen van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In Carl Pomerance, editor, *CRYPTO*, volume 293 of *Lecture Notes in Computer Science*, pages 87–119. Springer, 1987.

[CdVFS07] Valentina Ciriani, Sabrina De Capitani di Vimercati, Sara Foresti, and Pierangela Samarati. *k*-anonymity. In *Secure Data Management in Decentralized Systems*, pages 323–353. 2007.

[CEC+08] David Chaum, Aleks Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan Sherman, and Poorvi Vora. Scantegrity: End-to-end voter-verifiable optical- scan voting. *IEEE Security & Privacy*, 6(3):40–46, 2008.

[CG96] Ran Canetti and Rosario Gennaro. Incoercible multiparty computation. Cryptology ePrint Archive, Report 1996/001, 1996. `http://eprint.iacr.org/`.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, July 2004.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. pages 103–118. Springer-Verlag, 1997.

[CH11a] Jeremy Clark and Urs Hengartner. Selections: Internet voting with over-the-shoulder coercion-resistance. In George Danezis, editor, *Financial Cryptography*, volume 7035 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2011.

[CH11b] Jeremy Clark and Urs Hengartner. Selections: Internet voting with over-the-shoulder coercion-resistance. Cryptology ePrint Archive, Report 2011/166, 2011. `http://eprint.iacr.org/`.

[Cha81] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981.

[Cha01] David Chaum. Surevote: Technical overview. WOTE 2001, 2001.

[Cha04] David Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy*, 2(1):38–47, 2004.

[Cla11] Jeremy Clark. Democracy enhancing technologies: Toward deployable and incoercible e2e elections, 2011.

[CMFP+10] Benoît Chevallier-Mames, Pierre-Alain Fouque, David Pointcheval, Julien Stern, and Jacques Traoré. On some incompatible properties of voting schemes. In David Chaum, Markus Jakobsson, RonaldL. Rivest, PeterY.A. Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida, editors, *Towards Trustworthy Elections*, volume 6000 of *Lecture Notes in Computer Science*, pages 191–199. Springer Berlin Heidelberg, 2010.

[CP93] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 89–105, London, UK, UK, 1993. Springer-Verlag.

[CRS05] David Chaum, Peter Y. A. Ryan, and Steve Schneider. A practical voter-verifiable election scheme. In *Proceedings of the 10th European Conference on Research in Computer Security*, ESORICS'05, pages 118–139, Berlin, Heidelberg, 2005. Springer-Verlag.

[CvdGRV07] David Chaum, Jeroen van de Graaf, Peter Y. A. Ryan, and Poorvi L. Vora. Secret ballot elections with unconditional integrity. Cryptology ePrint Archive, Report 2007/270, 2007. http://eprint.iacr.org/.

[Dam99] Ivan Damgård. Commitment schemes and zero-knowledge protocols. In *Lectures on Data Security*, pages 63–86. Springer, 1999.

[DHvdG+13] Denise Demirel, Maria Henning, Jeroen van de Graaf, PeterY.A. Ryan, and Johannes Buchmann. Prêt à voter providing everlasting privacy. In James Heather, Steve Schneider, and Vanessa Teague, editors, *E-Voting and Identify*, volume 7985 of *Lecture Notes in Computer Science*, pages 156–175. Springer Berlin Heidelberg, 2013.

[dJ13] Bundesministerium der Justiz. Federal electoral regulations (Bundeswahlordnung). `http://bundeswahlleiter.de/en/bundestagswahlen/downloads/rechtsgrundlagen/bundeswahlordnung_engl.pdf`, 1985, last revised 2013.

[DKR09] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, 2009.

[DvdGSdSA12] Denisa Demirel, Jeroen van de Graaf, and Roberto Samarone dos Santos Araújo. Improving Helios with Everlasting Privacy Towards the Public. Electronic Voting Technology Workshop/Workshop On Trustworthy Elections, EVT/WOTE 2012, 2012.

[ECH12] Aleksander Essex, Jeremy Clark, and Urs Hengartner. Cobra: Toward concurrent ballot authorization for internet voting. In *Presented as part of the 2012 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections*, Berkeley, CA, 2012. USENIX.

[EGW09] N.P. Smart E. Ghadafi and B. Warinschi. Groth–sahai proofs revisited. Cryptology ePrint Archive, Report 2009/599, 2009. `http://eprint.iacr.org/`.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In David Chaum George Robert Blakley, editor, *Advances in Cryptology Proceedings of CRYPTO 84*, pages 10–18. Springer-Verlag, Berlin, Heidelberg, 1985.

[FDL] Laure Fouard, Mathilde Duclos, and Pascal Lafourcade. Survey on electronic voting schemes. Available at `http://www-verimag.imag.fr/~duclos/paper/e-vote.pdf`.

[FOO93] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, ASIACRYPT '92, pages 244–251, London, UK, UK, 1993. Springer-Verlag.

[For02] Bryan Ford. Delegative democracy, 2002. Draft available at `http://www.brynosaurus.com/deleg/deleg.pdf`.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

[fSidI07] Bundesamt für Sicherheit in der Informationstechnik. BSI protection profile - basic requirements for remote electronic voting systems, 2007.

[fSoiE90] Organization for Security and Co operation in Europe. Document of the copenhagen meeting of the conference on the human dimension

of the csce. http://www.osce.org/odihr/elections/14304, Juni 1990.

[GA14] James Green-Armytage. Direct voting and proxy voting, 2014. Available at http://inside.bard.edu/~armytage/proxy.pdf.

[GGR09] Ryan W. Gardner, Sujata Garera, and Aviel D. Rubin. Coercion resistant end-to-end voting. In *Financial Cryptography*, pages 344–361, 2009.

[Gha11] Essam Ghadafi. Formalizing group blind signatures and practical constructions without random oracles. Cryptology ePrint Archive, Report 2011/402, 2011. http://eprint.iacr.org/2011/402/.

[Gil74] John T. Gill, III. Computational complexity of probabilistic turing machines. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 91–95, New York, NY, USA, 1974. ACM.

[GJJS04] Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson. Universal re-encryption for mixnets. In Tatsuaki Okamoto, editor, *Topics in Cryptology – CT-RSA 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 163–178. Springer Berlin Heidelberg, 2004.

[Gjø10] Kristian Gjøsteen. Analysis of an internet voting protocol. *IACR Cryptology ePrint Archive*, 2010:380, 2010.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[GMR85] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 291–304, New York, NY, USA, 1985. ACM.

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Appl. Math.*, 156(16):3113–3121, September 2008.

[Gro02] Jens Groth. A verifiable secret shuffe of homomorphic encryptions. In Yvo Desmedt, editor, *Public Key Cryptography ï¿½ PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 145–160. Springer Berlin / Heidelberg, 2002. 10.1007/3-540-36288-6_11.

[Gro06] Jens Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *In proceedings of ASIACRYPT '06, LNCS series*, pages 444–459. Springer-Verlag, 2006.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *Proceedings of the Theory and Applications of Cryptographic Techniques 27th Annual International Conference on Advances in Cryptology*, EUROCRYPT'08, pages 415–432, Berlin, Heidelberg, 2008. Springer-Verlag.

[GSB] Ida Sofie Gebhardt Stenerud and Christian Bull. When Reality Comes Knocking - Norwegian Experiences with Verifiable Electronic Voting. EVOTE2012.

[Hen12] Christian Henrich. *Improving and Anaysing Bingo Voting*. PhD thesis, Karlsruhe Institute of Technology, Karlsruhe, Germany, 2012.

[HLVL10] Sven Heiberg, Helger Lipmaa, and Filip Van Laenen. On e-vote integrity in the case of malicious voter computers. In *Proceedings of the 15th European Conference on Research in Computer Security*, ESORICS'10, pages 373–388, Berlin, Heidelberg, 2010. Springer-Verlag.

[HLW12] Sven Heiberg, Peeter Laud, and Jan Willemson. The application of i-voting for estonian parliamentary elections of 2011. In *Lecture Notes in Computer Science, 2012, Volume 7187/2012, 208-223*. Springer, 2012.

[HS00] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *Proceedings of the 19th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'00, pages 539–556, Berlin, Heidelberg, 2000. Springer-Verlag.

[HSS08] J. Helbach, J. Schwenk, and S. Schage. Code voting with linkable group signatures. *Proceedings of Electronic Voting*, 2008, 2008.

[JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *WPES '05: Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, pages 61–70, New York, NY, USA, 2005. ACM.

[JJR02] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making Mix Nets Robust For Electronic Voting By Randomized Partial Checking. In *USENIX Security Symposium*, pages 339–353, 2002.

[JMP13] H. L. Jonker, S. Mauw, and J. Pang. Privacy and verifiability in voting systems: Methods, developments and trends. *Computer Science Review*, 10:1–30, 2013.

[JP06a] H.L. Jonker and W. Pieters. Receipt-freeness as a special case of anonymity in epistemic logic. In *Proceedings of the IAVoSS Workshop On Trustworthy Elections (WOTE 2006)*, Cambridge, 2006. Robinson College.

[JP06b] H.L. Jonker and W. Pieters. Receipt-freeness as a special case of anonymity in epistemic logic. In *Proceedings of the IAVoSS Workshop On Trustworthy Elections (WOTE 2006)*, Cambridge, 2006. Robinson College.

[JSI96] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In Ueli Maurer, editor,

Advances in Cryptology - EUROCRYPT '96, volume 1070 of *Lecture Notes in Computer Science*, pages 143–154. Springer Berlin Heidelberg, 1996.

[Kem12] Carmen Kempka. Coercion-resistant electronic elections with write-in candidates. In *Proceedings of the 2012 international conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, EVT/WOTE'12. USENIX Association, 2012.

[KKW06] Aggelos Kiayias, Michael Korman, and David Walluck. An internet voting system supporting user privacy. In *ACSAC*, pages 165–174. IEEE Computer Society, 2006.

[KMW12] Shahram Khazaei, Tal Moran, and Douglas Wikström. A mix-net from any cca2 secure cryptosystem. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 607–625. Springer Berlin Heidelberg, 2012.

[KTV12] R. Küsters, T. Truderung, and A. Vogt. A Game-Based Definition of Coercion-Resistance and its Applications. *Journal of Computer Security (special issue of selected CSF 2010 papers)*, 20(6/2012):709–764, 2012.

[KW13] Shahram Khazaei and Douglas Wikström. Randomized partial checking revisited. In *Proceedings of the 13th International Conference on Topics in Cryptology*, CT-RSA'13, pages 115–128, Berlin, Heidelberg, 2013. Springer-Verlag.

[KY04] Aggelos Kiayias and Moti Yung. The Vector-Ballot E-Voting Approach. Financial Cryptography 2004, 2004.

[Lan10] Barbara Lucie Langer. *Privacy and Verifiability in Electronic Voting*. PhD thesis, Techniche Universität Darmstadt, 2010.

[Lip11] Helger Lipmaa. Two simple code-verification voting protocols. *IACR Cryptology ePrint Archive*, 2011:317, 2011.

[MCC08] Andrew C. Myers, Michael Clarkson, and Stephen Chong. Civitas: Toward a secure voting system. In *IEEE Symposium on Security and Privacy*, pages 354–368. IEEE, May 2008.

[Mil69] III Miller, JamesC. A program for direct and proxy voting in the legislative process. *Public Choice*, 7(1):107–113, 1969.

[MKGV07] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. *L*-diversity: Privacy beyond *k*-anonymity. *TKDD*, 1(1), 2007.

[MN06] Tal Moran and Moni Naor. Receipt-Free Universally-Verifiable Voting With Everlasting Privacy. In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 373–392. Springer, August 2006.

[MR10] Niels Menke and Kai Reinhard. Compliance of polyas with the common criteria protection profile - a 2010 outlook on certified remote electronic voting. In *Electronic Voting*, pages 109–118, 2010.

[Nef01] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. CCS '01 Proceedings of the 8th ACM conference on Computer and Communications Security, 2001.

[Oka98] Tatsuaki Okamoto. Receipt-free electronic voting schemes for large scale elections. In Bruce Christianson, Bruno Crispo, Mark Lomas, and Michael Roe, editors, *Security Protocols*, volume 1361 of *Lecture Notes in Computer Science*, pages 25–35. Springer Berlin Heidelberg, 1998.

[OKNV12] Maina M. Olembo, Anna Kahlert, Stephan Neumann, and Melanie Volkamer. Partial verifiability in polyas for the gi elections. In Melanie Volkamer Manuel J. Kripp and Rüdiger Grimm, editors, *5th International Conference on Electronic Voting 2012 (EVOTE2012)*, volume 205 of *LNI - Lecture Notes in Informatics*, pages 95–109. Co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC, Gesellschaft für Informatik, Jul 2012.

[Ped91a] Torben Pryds Pedersen. Non-interactive and Information-Theoretic Secure Verifiable Secret Sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91: Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[Ped91b] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *Proceedings of the 10th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'91, pages 522–526, Berlin, Heidelberg, 1991. Springer-Verlag.

[PH06] Stefan Popoveniuc and Ben Hosp. An Introduction to Punchscan. IAVoSS Workshop On Trustworthy Elections, WOTE 2006, 2006. `http://punchscan.org/papers/popoveniuc_hosp_punchscan_introduction.pdf`, online version dated 2006-10-15.

[PH10] Stefan Popoveniuc and Benjamin Hosp. An introduction to punchscan. In David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter Y. A. Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida, editors, *Towards Trustworthy Elections*, volume 6000 of *Lecture Notes in Computer Science*, pages 242–259. Springer, 2010.

[PS07] Stefan Popoveniuc and Jonathan Stanton. Undervote and pattern voting: Vulnerability and a mitigation technique. In *In Preproceedings of the 2007 IAVoSS Workshop on Trustworthy Elections (WOTE 2007*, 2007.

[Riv06] Ronald L Rivest. The threeballot voting system. *Unpublished draft*, 2006.

[RS06] Peter Y. A. Ryan and Steve A. Schneider. Prêt à voter with re-encryption mixes. In Dieter Gollmann, Jan Meier, and Andrei Sabelfeld, editors, *ESORICS*, volume 4189 of *Lecture Notes in Computer Science*, pages 313–326. Springer, 2006.

[RS07] Ronald L. Rivest and Warren D. Smith. Three voting protocols: Threeballot, vav, and twin. In *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, EVT'07, pages 16–16, Berkeley, CA, USA, 2007. USENIX Association.

[RT09] Peter Y. A. Ryan and Vanessa Teague. Pretty good democracy. In Bruce Christianson, James A. Malcolm, Vashek Matyas, and Michael Roe, editors, *Security Protocols Workshop*, volume 7028 of *Lecture Notes in Computer Science*, pages 111–130. Springer, 2009.

[RW06] Ronald L. Rivest and John P. Wack. On the notion of "software independence" in voting systems, 2006.

[Rya69] Joanna Ryan. Grouping and Short-Term Memory: Different Means and Patterns of Grouping. The Quaterly Journal of Experimental Psychology 21, p.137-147, 1969.

[Sch91] C.P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[Sch11] Markus Schulze. A new monotonic, clone-independent, reversal symmetric, and condorcet-consistent single-winner election method. *Social Choice and Welfare*, 36(2):267–303, 2011.

[SK95] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme: a practical solution to the implementation of a voting booth. In *Proceedings of the 14th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'95, pages 393–403, Berlin, Heidelberg, 1995. Springer-Verlag.

[TRN08] Vanessa Teague, Kim Ramchen, and Lee Naish. Coercion-resistant tallying for stv voting. In *EVT*, 2008.

[UMQ10] Dominique Unruh and Jörn Müller-Quade. Universally composable incoercibility. In *Crypto 2010*, volume 6223 of *LNCS*, pages 411–428. Springer, August 2010. Preprint on IACR ePrint 2009/520.

[vdGKAL14] Jeroen van de Graaf, Carmen Kempka, Dirk Achenbach, and Bernhard Löwe. A taxanomy for analysing and comparing voting protocolos. unpublished manuscript, 2014.

[Wic64] Wayne A. Wickelgren. Size of Rehearsal Group and Short-Term Memory. Journal of Experimental Psychology, Vol. 68, No. 4, 1964.

[Wik04] Douglas Wikström. A universally composable mix-net. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 317–335. Springer, 2004.

[ZCC+13] Filip Zagorski, Richard T. Carback, David Chaum, Jeremy Clark, Aleksander Essex, and Poorvi L. Vora. Remotegrity: Design and use of an end-to-end verifiable remote voting system. Cryptology ePrint Archive, Report 2013/214, 2013. `http://eprint.iacr.org/`.

Appendix

A. Analysis of the German Paper Election

A.1. General Information about the voting scheme

Election type

(Q1) Is the election scheme ...

- ... paper based, scanner based or computer based?
- ... meant for presential or internet elections?

The scheme is paper based and meant for presential elections.

(Q2) For which kind of elections is the scheme designed?

(Q3) Does the election scheme allow any special election types? (E.g. vote-splitting, write-in candidates, vote delegation etc.)

The scheme is designed and used for governmental elections.

Preliminaries and Assumptions

(Q4) On what cryptographic assumptions is the scheme based?

There is no cryptography involved.

(Q5) If the scheme relies on trusted parties, which parties need to be trusted?

Poll workers need to be trusted as a group to not mark ballots in an inconspicuous way.

(Q6) Which components have to be trusted? Does the scheme rely on trusted hardware?

The scheme relies on some physical assumptions: the ballot box has to be big enough that the ballots inside are sufficiently mixed – the order of ballots would break privacy since it is observable in which order the voters cast their ballots.

(Q7) Are there other assumptions?

The scheme relies on a voting booth and the election and counting process being observed by volunteers.

What makes this scheme secure?

(Q8) What is the main attribute that leads to the security of the analyzed voting scheme?

The whole election and counting process is observable from beginning to end. The voter's will is marked on a physical object that cannot be tampered with without physical access, and which the voter throws into an observable ballot box by herself.

A.2. Analysis of the Requirements

Requirement A (*Eligibility*)

(A1) How does the system identify whether persons presenting themselves to vote are eligible?

Poll workers maintain a list of eligible voters, called a voter roll. *Voters have to present their identity card and their voting card.*

Requirement B (*Equality*)

(B1) How does the system ensure that each eligible voter can cast at most one ballot?

After presenting her identity card, the voter is given only one ballot, may only throw one ballot into the ballot box and is only allowed to enter the polling station once. The voter's name is ticked off in the voter roll to mark her as "has voted". Observation ensures that this is done correctly. Each voter gets only one ballot at a time (it can be exchanged for revision). The opening of the ballot box is covered by the poll workers unless a voter puts in her ballot.

(B2) How is it made sure that each eligible voter has the possibility to cast her ballot?

By law every eligible voter is notified of an upcoming election several weeks in advance. The election is also announced by media. Everyone can observe that no eligible voter is denied access.

Requirement C (*Layout neutrality*)

(C1) How are choices presented to the voter?

Choices are presented on a paper ballot. The exact layout differs, depending on election rules. Generally, candidates are listed in a predefined order, equally to each ballot, with a box next to it where the voter can put her mark.

(C2) How is it made sure that no option is favored more than specified by election rules?

Each ballot is supposed to look the same. This can be audited by persons present in the polling station.

(C3) How is it made sure that choices are presented fairly to the voters?

See the answer to question C3.

Requirement D (*Revisable ballot marking*)

(D1) How does the voter mark her choice?

The voter obtainss her ballot, goes into the voting booth and marks the box next to her chosen candidate.

(D2) How can the voter verify that the ballot she just created contains a faithful representation of her preferences? Under what assumptions?

The voter can check that her mark is in the right place.

(D3) How can the voter revise her ballot?

The voter can ask the poll workers for a new ballot, the old ballot is then destroyed.

Requirement E (*Irreversible ballot casting*)

(E1) How does the voter cast her ballot? *The voter puts her ballot into the ballot box.*

(E2) What defines when a ballot is cast?

The ballot being in the ballot box.

(E3) What makes it irreversible? Under what assumptions is it irreversible?

Under physical assumptions, ballots cannot be taken out of the ballot box unnoticed or modified once they are inside.

(E4) What defines the ballot box?

There are one or several actual ballot boxes that are big enough to sufficiently shuffle the ballots. The opening of the box is covered by the poll workers unless a voter puts in her ballot, and small enough that nothing but the ballot can be put inside.

Requirement F (*Privacy and incoercibility*)

(F1) How is individual privacy assured? Under which assumptions?

The voter's privacy while marking her choice is protected by a voting booth under the assumption that there is no camera inside. Before leaving the voting booth, the voter has to fold her ballot, hiding her marks, and put it into the ballot box. It is assumed that ballots in the ballot box are sufficiently shuffled before they are taken out for counting.

(F2) How is individual privacy assured in the future? And under which assumptions?

As soon as the ballot is in the ballot box, it is unlinked from the voter's identity. So if the ballot is not marked, privacy is preserved unconditionally, except maybe for DNA-tests on the ballots. Marked ballots count as invalid, so the voter can be coerced not to vote for a candidate.

(F3) How is receipt-freeness assured? And under which assumptions?

Receipt-freeness is not perfectly ensured. Poll workers might be able to secretly and inconspicuously mark ballots to recognize them later, without the mark being noticed (otherwise, the ballot would be counted as invalid). In polls where the voter can distribute more than one vote among candidates, pattern voting attacks are possible.

(F4) How is coercion-freeness assured? And under which assumptions?

The voter can mark her ballot. It is then counted as invalid, so the scheme is not fully coercion resistant.

(F5) Who learns the vote or is able to calculate it?

No one but the voter herself learns her own choice. After the voter's ID is unlinked, everyone sees the plaintext ballots.

(F6) Who gets critical information? What information is this and in which way is it critical? *Critical information would be a link between the contents of the ballots and the voter's identity. This is why it is important that ballots are sufficiently shuffled.*

Requirement G (*Secrecy of intermediate results*)

(G1) How does the voting scheme guarantee that no information about the contents of the ballot box leaks before the tallying procedure is completed? Under which assumptions?

Observation of the ballot box guarantees that no ballots are taken out and looked at. The ballot box is opaque, so no ballot's contents can be seen from the outside. Under the assumption that there is no camera in the voting booth or in the ballot box, no one but the voter who has cast it sees a ballot's content before it is thrown into the urn.

Requirement H (*Inviolatability of the ballot box*)

(H1) How does the system make sure that no ballots are modified? Under which assumptions?

The voter is the only one who can modify her ballot before casting it. Physics and observation of the ballot box make sure that ballots in the ballot box cannot be modified. No other item but a ballot may be put into the ballot box. The reason for this is that an additional item could spoil or mark some ballots. The

small size of the opening of the box and covering it ensures that only ballots are put inside.

(H2) How does the system make sure that no ballots are removed? Under which assumptions?

Observation of the ballot box.

(H3) How does the system make sure that no ballots are added? *Before the voting phase starts, it is shown to everyone present that the ballot box is empty. After that, each eligible voter have only one ballot at a time and can enter exactly one ballot.*

Requirement I (*Tally integrity*)

(I1) How is the tallying of the votes implemented?

The ballot box is opened. Then each ballot is taken out, shown to the observers and counted.

(I2) How is it made sure that only the ballots correctly cast are counted?

The ballots in the ballot box have been put there by eligible voters, observation of the polling station ensures this. The counting is done openly, everyone sees that invalid ballots are not counted. Objects other than ballots that indicate choices are not counted.

(I3) If not correctly cast ballots are counted, who sees it?

Everyone who observes the process can see it.

(I4) Who can prove it and how?

Since everyone sees it there is nothing to prove.

(I5) What can be done in this case to correct the error?

Recount without the incorrect ballot.

Requirement J (*Individual verifiability*)

(J1) How can a voter convince herself that her vote is included in the tally? Under which assumptions?

The voter throws her ballot into the ballot box by herself and can observe the ballot box and the counting process afterwards.

(J2) What is the underlying principle used?

Physics and observation.

(J3) If the above check fails, how can the voter prove that her ballot is not included in the tally?

If the voter observes manipulation, probably others have seen it also.

(J4) How and by whom can this error be corrected?

This strongly depends on the error. Since paper ballots are used, a recount is always possible as long as no ballots were removed. Since the voter's choices are only recorded on the paper ballots, votes on lost ballots cannot be recovered.

Requirement K (*Auditability and public verifiability*)

(K1) How can an interested party be convinced that the result published by the BTA is correct?

Everyone can observe the voting and counting process. However, everyone can only be present at one polling station at a time, so if the election takes place at more than one polling station in parallel, there is no true universal verifiability.

(K2) What is the underlying principle used?

Physics and observation.

(K3) If the result is not correct, who can prove it and how?

See the answer to question J3.

(K4) If the result is not correct, can the error be corrected? If so, by whom and how?

See the answer to question J4.

(K5) Is it traceable who/what has caused the error?

If it was observed, yes.

Requirement L (*Robustness*)

(L1) How is invalid or malformed voter data treated?

Invalid ballots are not counted. Invalid voter rolls can be corrected.

(L2) What happens when a voter aborts the voting process?

The ballot is either thrown into the ballot box or not. If it is inside, it counts as cast. If not, this is documented in the voter roll.

Requirement M (*Availability*)

(M1) How does the system handle the unintentional breakdown of its components?

All physical objects except for marked ballots can be substituted. Lost or destroyed ballots or ballot boxes cannot be recovered, there is no redundancy in the vote recording process.

(M2) How can the system made to gracefully deal with denial of service attacks?

Mutual observation assures that no present person can file a denial of service attack unnoticed. There are no electronic systems involved which could be attacked from outside.

Requirement N (*Scalability*)

(N1) How well does the scheme scale? How big can an electoral constituency be?

Since ballots are counted by hand, the scheme scales with the number of available poll workers. The voting process is done mostly sequentially, there is only a very limited number of voting booths in one polling station that can only be used one at a time. Therefore, only a limited number of voters can cast a ballot in one voting station in one day.

A.3. Conclusion of the Analysis

(S1) Which of the requirements listed above are fully met under the underlying assumptions? *Requirements A, B, C, D, E, G, H, I, L and M are met.*

(S2) Which requirements are only partly met, and in which way? *Requirement F is met except for forced-abstention and pattern voting attacks, and the inconspicuous marking of ballots. Requirement J and K are met but a missing ballot can only be proven with witnesses and the error cannot be corrected after the voting phase. Requirement K is not fully met if the voting process takes place in several polling stations: everyone can only be physically present at one of them.*

(S3) Which requirements could be met with minor improvements of the scheme? What are the suggested improvements? *A problem of the paper election is its lack of redundancy: lost ballots cannot be recovered. This problem could be solved by additionally scanning the ballots.*

(S4) Which requirements are not met? *None*

(S5) For which requirements does it depend on a concrete implementation or application if they are met? *None*

(S6) Are there any concluding remarks? *From a security and verifiability point of view, the paper election seems to meet high standards. But even though it is often used as an ideal, it is not secure against some coercion attacks. More robustness could be achieved by introducing a backup mechanism of the votes. It is funny that while for electronicly stored ballots, paper audit trails (VVPATs) are suggested, the same is advisable vice versa.*

B. Analysis of the Student Parliament Election

This chapter contains the full analysis of the student parliament election held in January 2008 at the University of Karlsruhe, now Karlsruhe Institute of Technology, with the voting scheme Bingo Voting. Election details are described in Section 4.3. The analysis is done according to the evaluation criteria of our taxonomy. Voters could choose between either using Bingo Voting or traditionally casting a paper ballot. This taxonomy concentrates on the electronic voting part. Revoting was not possible, an electronicly cast vote could not be overwritten by a paper ballot.

B.1. Preliminaries about the Voting Scheme

Election type

(Q1) Is the election scheme ...

- ... paper based, scanner based or computer based?
- ... meant for presential or internet elections?

The 2008 election was a presential election, held in several polling stations. Students could vote electronically with the Bingo Voting scheme in one of the polling stations, where also an ordinary paper election was offered. Details on the election rules are explained in Section 4.3.

(Q2) For which kind of elections is the scheme designed?

This is not specified by the scheme. We estimate the election of the student parliament as an election with low coercion and manipulation risk.

(Q3) Does the election scheme allow any special election types? (E. g. vote-splitting, write-in candidates, vote delegation etc.) *The scheme as used in the 2008 election supported vote-splitting and cumulation. A newer implementation also supports write-in candidates.*

Preliminaries and Assumptions

(Q4) On what cryptographic assumptions is the scheme based?

Verifiability relies on the discrete logarithm problem.

(Q5) If the scheme relies on trusted parties, which parties need to be trusted?

In Bingo Voting, for verifiability, the parameters g and h for Pedersen commitments must not be created by the voting machine but a trusted authority. For privacy, the voting machine, and therefore, its administrators, needed to be trusted. However, in the student parliament election, we implemented the trusted voting machine the following way: upon start of the election, The voting machine was not connected to the internet or any other network. No keyboard was available in the voting booth, the voters could not exit the vote casting program and access the underlying system. For administrating purposes, the administrators only had access to the voting machine under surveillance of at least one member of the voting authority, and the other way around.

(Q6) Which components have to be trusted? Does the scheme rely on trusted hardware?

The scheme relies on a trusted random number generator.

(Q7) Are there other assumptions?

The presence of a voting booth, and a sufficiently small probability of collisions between random numbers. For privacy, the distribution of the dummy random numbers has to be indistinguishable from the distribution of the numbers generated by the TRNG. The random numbers created by the TRNG must not contain side-channels.

What makes this scheme secure?

(Q8) What is the main attribute that leads to the security of the analyzed voting scheme?

Bingo Voting relies on a trusted random number generator (TRNG) that is hard to manipulate. On the voter's receipt, the real vote is hidden between dummy votes to protect voter privacy while giving the voter a proof that her ballot has been processed correctly. The voting booth asures voter privacy while filling out the ballot on the voting machine.

B.2. How the Requirements are met

The student parlaiment election was done without any critical incidents, all proofs of correctness were checked and turned out to be correct, no voter complained about a missing receipt and no person complained about incorrect proofs. There were very few smaller incidents with the chip card when pulled out by the voters too early, for details see Section 4.3, the errors could always be corrected. But for completeness, when answering the questions, incidents that *could have happened* are described.

Requirement A (*Eligibility*)

(A1) How does the system identify whether persons presenting themselves to vote are eligible?

Students presented their student card to the voting authority, and then got a random chip card, not connected to their ID, with which they could start the voting process at the voting machine. The voting authority encoded on the chip card for which poll the voter is eligible.

Requirement B (*Equality*)

(B1) How does the system ensure that each eligible voter can cast at most one vote?

Voters were ticked off in a voter roll by poll workers. This part of the election could be observed by everyone. The information if the voter has already cast a vote in a poll was recorded on her login chip card. Each login card could be used only once for each poll, afterwards it was not accepted by the voting machine. The number of votes cast could be checked against the voter roll, so an error would have been detectable.

(B2) How is it made sure that each eligible voter has the possibility to cast her vote?

Since it was a presential election, forced abstention attacks were not prevented. The ticking off in the voter's roll was observed, and there were always two poll workers per table who observed each other, so there was a high probability to get caught when ticking off a voter who was no there.

Requirement C (*Layout neutrality*)

(C1) How are choices presented to the voter?

For a description of the implementation see Section 4.3. Each voter saw the same UI, the order in which candidates were displayed was specified by a XML file generated by the voting authority and was the same for each voter. The

scheme was tested by members of the voting authority before the election phase, but this part was not publicly verifiable. Since the chip cards for login did not encode the identity of the voter, the voting machine could not depend the UI on the voter's ID. But the chip card encoded for example if the voter was eligible for the poll of woman's representative, i. e. if the voter was a woman.

(C2) How is it made sure that no option is favored more than specified by election rules?

See the answer to question C1.

(C3) How is it made sure that choices are presented fairly to the voters?

See the answer to question C1.

Requirement D (*Revisable ballot marking*)

(D1) How does the voter mark her choice?

By clicking plus and minus buttons next to the name of a candidate. The UI showed the voter how many votes she had left to cast.

(D2) How can the voter verify that the ballot she just created contains a faithful representation of her preferences? Under what assumptions?

After the voter marked her choice, the voting machine presented her ballot to the voter where she could check her vote. A proof of correct recording of the ballot was given only after casting (see next requirement).

(D3) How can the voter revise her ballot?

As mentioned above, the UI displayed the ballot to the voter before casting. The voter could click the back button and change her choices. This could be done an arbitrary number of times.

Requirement E (*Irreversible ballot casting*)

(E1) How does the voter cast her ballot?

After double checking her ballot, the voter clicked cast. After that, dummy random numbers for each not voted candidate were taken from the list of dummy votes, the TRNG generated and displayed a fresh random number and the receipt was printed and stored electronically on the voting machine.

(E2) What defines when a ballot is cast?

The transfer of the dummy votes and the fresh number to the voter's receipt.

(E3) What makes it irreversible? Under what assumptions is it irreversible?

The dummy votes that are taken away from the candidates are printed on a receipt that is later published, so they cannot be given back and opened as unused or given to another voter.

(E4) What defines the ballot box?

The unused dummy votes together with the stored receipts and the commitments to the used dummy votes.

Requirement F (*Privacy and incoercibility*)

(F1) How is individual privacy assured? Under which assumptions?

Privacy is assured under the assumption that the voting machine does not leak the used dummy random numbers or the unveil information of their commitments, otherwise they can be distinguished from the fresh numbers which would break privacy.

(F2) How is individual privacy assured in the future? And under which assumptions?

Pedersen commitments are unconditionally hiding, so as long as the TRNG and the voting machine can be trusted, and the distribution of dummy and fresh random numbers is indistinguishable, Bingo Voting offers unconditional privacy. Pedersen commitments can be trapdoor-opened to arbitrary values, so in princible it is possible to lie about their contents if one knows the discrete logarithm $log_g(h)$.

(F3) How is receipt-freeness assured? And under which assumptions?

See the answer to question F4.

(F4) How is coercion-freeness assured? And under which assumptions?

The voter has no influence on the form of any published data. Each receipt contains the same candidate names and random numbers from the same or at least an indistinguishable distribution. The voter can only be coerced to abstain from voting.

F5 Who learns the vote or is able to calculate it?

The voting machine and whoever is able to unveil the dummy vote commitments can learn the voter's choice.

F6 Who gets critical information? What information is this and in which way is it critical?

The plaintext dummy votes and unveil information to their commitments is critical information. It was stored on the voting machine.

Requirement G (*Secrecy of intermediate results*)

(G1) How does the system guarantee that no information about the contents of the ballot box leaks before the tallying of the votes has completed? Under which assumptions?

This fact was not publicly verifiable and relied on trust in the voting machine.

How the trustworthiness of the voting machine was implemented, see the answer to Question (Q4).

Requirement H (*Inviolatability of the ballot box*)

(H1) How does the system make sure that no ballots are modified? Under which assumptions?

The voter could check that her receipt appeared on the bulletin board correctly, and a proof of the correct processing of the voter's choice was given according to this receipt. Under the assumption that there was no collision between a fresh random number and a dummy random number for another candidate, no ballots could be changed. For possible attacks at the occurrence of collisions, see [Hen12].

(H2) How does the system make sure that no ballots are removed? Under which assumptions?

The receipts are published and the voter can check them.

((H3)) How does the system make sure that no ballots are added?

The number of voters who have cast votes was visible in the voter roll. Poll workers who maintained the voter roll could be observed by each other and the voters. Everyone could check that the number of voters according to the voter roll matched the number of published receipts as well as the number of unused dummy votes.

Requirement I (*Tally integrity*)

(I1) How is the tallying of the votes implemented?

For details see the description of the Bingo Voting scheme in Section 4.2. The tally matches the number of unused dummy votes of each candidate.

(I2) How is it made sure that only the ballots correctly cast are counted?

It was publicly observable that only eligible voters had access to the voting machine. The number of voters according to the voter roll had to match the number of receipt. Under the assumption that the poll workers maintaining the voter roll were honest or sufficiently observed, and enough voters checked their published receipts, the proofs of correctness proved that only correctly cast receipts are counted.

(I3) If not correctly cast ballots are counted, who sees it?

The voter could see if her receipt was correctly published and correctly processed in the tally. There was one proof of correctness per receipt, which could be checked by everyone.

(I4) Who can prove it and how?

If the proofs of correctness had been wrong, everyone could have seen and

proven this. If an incorrect receipt had been used in the tally, there are two possibilities how this could have happened: either a receipt not cast by an eligible voter was added, then there had been one receipt too much. This could have been proven with the help of the voter roll. Or a receipt of an eligible voter was manipulated, then this voter could have proven this issue with her signed receipt, assuming the signatur was valid.

(I5) What can be done in this case to correct the error?

In the case of an incorrect receipt and resulting incorrect prove of correctness, the receipt could have been substituted by the voter's correct receipt and the tally be redone. In case of an incorrect receipt and correct prove of correctness, a wrong dummy random number would have been marked as unused, this error could not have been corrected. If a receipt had been added, the error could only have been corrected by having all voters show their receipts.

Requirement J (*Individual verifiability*)

(J1) How can a voter convince herself that her vote is included in the tally? Under which assumptions?

The voter could follow her receipt from when it was printed to its proof of correctness. She could check "her" proof and every other. In the voting booth, she had the chance to check that the fresh random number is in the right place.

(J2) What is the underlying principle used?

Publishing on the public bulletin board, trust in the TRNG, negligible probability of collisions of large random numbers, binding property of commitments (DLog hardness).

(J3) If the above check fails, how can the voter prove that her vote is not included in the tally?

A disadvantage was that this check could only be done after casting. If the voter had seen in the voting booth that her fresh random number was not in the right place, she could have complained but it had been hard to prove. If her receipt had not appeared online, she could have proven this with her signed receipt.

(J4) How and by whom can this error be corrected?

This question is partly answered with the answer to I5. If the voter had seen the wrong random number next to her candidate, it had been hard to correct the error without breaking privacy. With the help of the voting authority together with the administrators, the used dummy random numbers could have been given back to the candidates and the voter's receipt deleted, but these dummy votes could not have been used on another receipt because the voter had already seen them.

Requirement K (*Auditability and public verifiability*)

(K1) How can an interested party be convinced that the result published by the BTA is correct?

For details of the proof of correctness please refer to Section 4.2. The tally had to be in accordance with the number of unused dummy votes of each candidate. For each receipt, the commitments to the used dummy votes and a fresh commitment to the fresh random number were mixed and opened. Everyone could check that each commitment published in the pre-voting phase was either opened or used in a correctness proof of exactly one receipt.

(K2) What is the underlying principle used?

A verifiable shuffle.

(K3) If the result is not correct, how can this be proven?

Everyone could check the proves of correctness. So if the result had not been correct, everyone would have seen it.

(K4) If the result is not correct, is it traceable who/what caused it? Who can prove this?

This question was partly answered in question J3. If there had been discrepancies between the voting machine and the voter roll, it had been hard to prove the source, but since the voter roll was observed at all time it seems more trustworthy than the voting machine.

(K5) If the result is not correct, how can the error be corrected?

If the error had come from an incorrect receipt, see the answers to questions I5 and J4. Otherwise it strongly depends on the error. In the Bingo Voting scheme, the tally can be redone as long as dummy votes and unveil information are not deleted from the voting machine.

Requirement L (*Robustness*)

(L1) How is invalid or malformed voter data treated?

For malformed receipts, see he answer to J4. If wrong eligibility information had been encoded on the voter's chip card, this could be corrected by the voting authority.

(L2) What happens when a voter aborts the voting process?

The voter could abort the voting process at anytime, but had to wait a few seconds for the voting machine to record on the chip card that the voter was still eligible in the poll. If the chip card was pulled out too early, it was recorded on the chip card that the voter had started the ballot marking process in a poll, but not if she had cast a vote in it. Therefore, with a card in this state the voter could not have logged in again, she had lost her possibility to vote in this poll.

This had actually happened a few times, but the error could always be corrected by immediately checking how many votes have been cast. After checking this information on the voting machine against the voter roll, the chipcard of the voter could be unlocked for this poll by the voting authority.

Requirement M (*Availability*)

(M1) How does the system handle the unintentional breakdown of its components?

The voting machine is a single point of failure since it contains all the votes cast. While it adds to security that it is not connected to any network, a good backup mechanism would be advisable.

(M2) How can the system made to gracefully deal with denial of service attacks?

Since the voting machine is offline, it cannot be attacked from outside. If it stops working, it can be substituted by another one, assuming it has an appropriate backup mechanism.

Requirement N (*Scalability*)

(N1) How well does the scheme scale? How big can an electoral constituency be?

The bottle neck in the student parliament election was the slow write-access to the chip cards, which led to the voting process taking several minutes. This could have been compensated by using several voting machines. In Bingo Voting, the dummy random numbers can be distributed to arbitrary many voting machines. For less probable collisions of fresh random numbers with dummy votes, one can treat each voting machine as a distinct election or use longer random numbers. The impact of the number of voters handled in one election on the suggested length of the used random numbers and their probability of collision is not very critical: as Henrich showed in [Hen12], with 45 bit random numbers, i. e. about 12 alphanumeric letters, and an election with 714 million voters, which was the number of voters in the Indian general election of 2009, the expected number of overall collisions occuring in an election is less than one.

B.3. Conclusion of the Analysis

In a system setup like it was applied in the student parliament election, Requirements would be met as follows:

(S1) Which of the requirements listed above are fully met under the underlying assumptions? *Requirement A, B, C, D, E and H.*

(S2) Which requirements are only partly met, and in which way? *Requirements I, J and K are partly met: a manipulation can always be proven, the source of the error is not traceable in some cases, and an error cannot always be corrected. Requirement C and G are met, but the fact is verifiable only by auditors who test the system in advance, not publicly. Requirement F is met except for the possibility of forced abstention attacks. Requirement L was is partly met, errors cannot always be corrected.*

(S3) Which requirements could be met with minor improvements of the scheme? What are the suggested improvements? *Requirement M could be met with a good redundancy mechanism so no votes are lost. This would compete with privacy, though, since more components would get critical information. Requirement N could be met if a faster card reader was used, preferably one that pulls in the card completely so less states need to be documented on the card. The size of published data is also a matter, for a discussion on their reduction please refer to the dissertation of Christian Henrich [Hen12].*

(S4) Which requirements are not met? *There are no requirements that are impossible to achieve with the scheme. All requirements can be met with a suitable adaption of the scheme.*

(S5) For which requirements does it depend on a concrete implementation or application if they are met? *None*

(S6) Are there any concluding remarks? *As a concluding remark one could say that The assumption of a trusted voting machine for privacy is very strong since it is in the hands of the voting authority. It is hard to make the implementation publicly verifiable, one would have to publish the implementation (this was actually done in the student parliament election), and then prove that the published code is the same as the code running on the voting machine (this was only done by the voting authority).*

C. Analysis of our Revoting Scheme

This section contains an analysis of the revoting scheme introduced in Section 5.3.

C.1. General Information about the Voting Scheme

Election type

(Q1) Is the election scheme ...

- ... paper based, scanner based or computer based?
- ... meant for presential or internet elections?

The election scheme is an electronic internet election scheme.

(Q2) For which kind of elections is the scheme designed? (E.g. governmental elections, non-political elections, etc.)

It is designed as a proof of concept and not yet meant for large-scale elections. It has not been designed for efficiency.

(Q3) Does the election scheme allow any special election types? (E.g. vote-splitting, write-in candidates, vote delegation etc.)

In its current version it is designed for one out of n choices only. It can be adapted to allow vote delegation.

Preliminaries and Assumptions

(Q4) On what cryptographic assumptions is the scheme based?

The ElGamal encryption depends on the DDH-assumption. The GS-proofs depend on the CRS-Model and the SXDH-assumption. The signature scheme is based on the ADH-SDH assumption and the AWF-CDH assumption, the definitions of these assumptions can be found in [AFG+10].

(Q5) If the scheme relies on trusted parties, which parties need to be trusted?

All servers can be distributed, there is no single trusted party.

(Q6) Which components have to be trusted? Does the scheme rely on trusted hardware?

None.

(Q7) Are there other assumptions? (E.g. the existence of a voting booth, at least one voting process done in private etc.)

Existing PKI, at least one voting process can be done in private.

What makes this scheme secure?

(Q8) What is the main attribute that leads to the security of the analyzed voting scheme?

The voter can vote and revote without revealing her identity. Ballots which are counted cannot be linked to certain cast ballots.

C.2. Analysis of the Requirements

Requirement A (*Eligibility*)

(A1) How does the system identify whether persons presenting themselves to vote are eligible?

By checking the GS-proof of knowledge of a signature corresponding to a verification key which has a certificate.

Requirement B (*Equality*)

(B1) How does the system ensure that each eligible voter can cast at most one ballot?

The sorting algorithm and the opening of identities after the sorting ensure that only one vote per voter is counted.

(B2) How is it made sure that each eligible voter has the possibility to cast her vote?

In fact, since identities are opened in the end, a voter can be coerced not to cast a vote. This problem can be solved as discussed in Section 5.3.

Requirement C (*Layout neutrality*)

(C1) How are choices presented to the voter?

The voter creates and encrypts her ballot by herself.

(C2) How is it made sure that no option is favored more than specified by election rules?

See (C1).

(C3) How is it made sure that choices are presented fairly to the voters?

See (C1).

Requirement D (*Revisable ballot marking*)

(D1) How does the voter mark her choice?

See (C1).

(D2) How can the voter verify that the ballot she just created contains a faithful representation of her preferences? Under what assumptions?

See (C1).

(D3) How is it made sure that the voter can revise her ballot?

The voter does not have to cast her ballot. She can create as many encryptions as she wishes.

Requirement E (*Irreversible ballot casting*)

(E1) How does the voter cast her ballot?

Via a vote caster who publishes it on a public bulletin board.

(E2) What defines when a ballot is cast?

The publishing on the bulletin board

(E3) What makes it irreversible? Under what assumptions is it irreversible?

The publishing on the bulletin board. It is not changeable by third parties, and the voter cannot take back a ballot, but she can overwrite it with a revote.

(E4) What defines the ballot box?

The bulletin board

Requirement F (*Privacy and incoercibility*)

(F1) How is individual privacy assured? Under which assumptions?

Ballots are encrypted with the ElGamal encryption, so privacy depends on the DDH assumption.

(F2) How is individual privacy assured in the future? And under which assumptions? (Is it computational? Unconditional? Why?)

Privacy is not everlasting.

(F3) How is receipt-freeness assured? And under which assumptions?

Receipt-freeness is assured depending on the security of the ElGamal encryption.

(F4) How is coercion-resistance assured? And under which assumptions? (I. e. how is it ensured that a ballot/voter's choice cannot be marked in a way that shows to the adversary that the voter has been successfully coerced?)

Coercion-Resistance depends on the sorting algorithm. If it does not leak information about revoting behavior, the scheme is coercion-resistant.

(F5) Who learns the vote or is able to calculate it?

The voter-PC (who does not know if the ballot is overwritten from another PC). The tally servers can jointly decrypt ballots.

(F6) Who gets critical information? What information is this and in which way is it critical?

Critical information is never processed in plaintext. The sorting servers could jointly decrypt public keys and timestamps, and with this link ballots to each other, or learn the number of revotes. But they can not decrypt the voter's choice. The tally servers could jointly decrypt single ballots, but not the identities of the voters who have cast them, or any other item than the vote itself which could link a counted ballot to a cast ballot, in particular not the voter's identity or a timestamp. The casting server learns the timestamp but cannot recognize it later in the counted ballots.

Requirement G (*Secrecy of intermediate results*)

(G1) How does the voting scheme guarantee that no information about the contents of the ballot box leaks before the tallying procedure is completed? Under which assumptions?

Ballots are encrypted until they are counted. Decryption can only be done jointly by the tally servers.

Requirement H (*Inviolatability of the ballot box*)

(H1) How does the system make sure that no ballots are modified? Under which assumptions?

Ballots are digitally signed. Afterwards, each step in the processing of ballots is publicly verifiable.

(H2) How does the system make sure that no ballots are removed? Under which assumptions?

See (H1).

(H3) How does the system make sure that no ballots are added?

See (H1), and the ballots contain a proof for a certificate of the signer. To add a ballot, the adversary would have to forge a signature. Overwriting ballots with replays are not possible because of the timestamps, which is signed (replays would be sorted out).

Requirement I (*Tally integrity*)

(I1) How is the tallying of the votes implemented?

Old votes are sorted out without decrypting critical information. The tally is then done mix-based with standard procedures.

(I2) How is it made sure that only the ballots correctly cast are counted?

Everyone can check the proofs of knowledge of a signature and a corresponding certificate.

(I3) If not correctly cast ballots are counted, who sees it?

Everyone sees it.

(I4) If not correctly cast ballots are counted, who can prove it and how?

It is visible on the public bulletin board.

(I5) If not correctly cast ballots are counted, what can be done to correct the error?

They will not be included in the sorting process in the first place. Everyone can check this. If, however, an invalidly cast ballot is included, the tally can be repeated without the invalidly cast ballot.

Requirement J (*Individual verifiability*)

(J1) How can a voter convince herself that her ballot is included in the tally? Under which assumptions?

It is published on the bulletin board and enters the sorting process which is done in public. From there, the correctness of all steps is proven.

(J2) What is the underlying principle used? Physical? Statistical/Probabilistic? Trust in the authorities?

Verifiable shuffles

(J3) If the above check fails, how can the voter prove that her ballot is not included in the tally?

The voter has no receipt. She cannot prove it if her ballot is not published

on the bulletin board. But she sees this before the end of the voting phase and can cast another ballot. If it is published, the it is visible to anyone whether the ballot is included.

(J4) How and by whom can this error be corrected?

Not specified

Requirement K (*Auditability and public verifiability*)

(K1) How can an interested party be convinced that the result published by the BTA is correct?

The voting scheme is end-to-end verifiable: the voter can see that her ballot is published and enters the sorting process, and that from there, all ballots are processed correctly.

(K2) What is the underlying principle used?

Verifiable shuffles and homomorphic encryption.

(K3) If the result is not correct, who can prove it and how?

Everyone can see this on the bulletin board.

(K4) If the result is not correct, can the error be corrected? If so, by whom and how?

The tally can be repeated.

(K5) Is it traceable who/what has caused the error?

The corresponding server(s) will not be able to create a proof of correctness, so it is traceable. These proofs are created on several occasions. In the shuffle procedures, each shuffling server provides its own proof. In joint decryption procedures, each server has to provide an individual proof of using the correct secret key as well. In the masking of the identity differences, each server has to reveal its randomness if the difference is the neutral element, so there, too, the error is traceable.

Requirement L (*Robustness*)

(L1) How is invalid or malformed voter data treated?

Malformed voter data would be a ballot with an invalid signature. This ballot would be ignored and not be processed in the tally procedure. Ballots with invalid content are treated as invalid votes and not counted. However, a proof of validity could be included in the casting process to prevent this.

(L2) What happens when a voter aborts the voting process?

The voter can abort the voting process before casting at any time without any consequences. Once her ballot is signed and published however, it is considered as cast and is included in the sorting procedure.

Requirement M (*Availability*)

(M1) How does the system handle the unintentional breakdown of its components?

See (M2).

(M2) How can the system made to gracefully deal with denial of service attacks?

All used procedures like shuffling, decrypting and sorting can be setup as k-out of-n threshold schemes. The bulletin board can be mirrored for redundancy.

Requirement N (*Scalability*)

(N1) How well does the scheme scale? How big can an electoral constituency be?

The ballots, or parts of them, have to be shuffled on several occasions. The sorting process consists of several shuffle procedures and creates a huge amount of published data.

C.3. Conclusion of the Analysis

(S1) Which of the requirements listed above are fully met under the underlying assumptions?

Requirements A, C, D, E, G, H, I, K, L and M are fully met.

(S2) Which requirements are only partly met, and in which way?

Requirement F depends on the information leakage of the sorting algorithm. Privacy does not hold unconditionally. Requirement J is only partly met because a voter cannot prove that her ballot has not been published.

(S3) Which requirements could be met with minor improvements of the scheme? What are the suggested improvements?

Requirement B can be met if identities are not opened in the end, as described in the discussion of Section 5.3.

(S4) Which requirements are not met?

None.

(S5) For which requirements does it depend on a concrete implementation or application if they are met?

Requirement N is only met for sufficiently small elections. For large-scale elections, a more efficient sorting procedure should be used.

(S6) Are there any concluding remarks?

Our revoting scheme is a working proof of concept for fully verifiable receipt-free elections with revoting. Coercion-resistance depends on the leakage of information of the sorting algorithm, which depends on the number of voters

who revote. The voting scheme is coercion-resistant for ballots with one out of n choices if a sufficient number of voters revote. However, the scheme is not yet suitable for large-scale elections because of its lack of efficiency.

D. Analysis of Prêt à Voter

This analysis refers to the original scheme Prêt à Voter as introduced in [CRS05].

D.1. General Information about the voting scheme

Election type

(Q1) Is the election scheme ...

- ... paper based, scanner based or computer based?
- ... meant for presential or internet elections?

The scheme is paper and optical reader based, and meant for presential elections.

(Q2) For which kind of elections is the scheme designed?

This is not specified.

(Q3) Does the election scheme allow any special election types? (E. g. vote-splitting, write-in candidates, vote delegation etc.)

It allows vote-splitting and could be adapted to allow cumulative voting, though this would make the scheme vulnerable to pattern voting attacks, which is th case with any voting scheme which reveals plaintext balots.

Preliminaries and Assumptions

(Q4) On what cryptographic assumptions is the scheme based?

Encryption of the permutation acts as binding commitment and cannot be trapdoor-opened to another permutation.

(Q5) If the scheme relies on trusted parties, which parties need to be trusted?

Trusted ballot creators/printers (they see the permutation of each ballot)

(Q6) Which components have to be trusted? Does the scheme rely on trusted hardware?

None

(Q7) Are there other assumptions?

The existence of a voting booth

What makes this scheme secure?

(Q8) What is the main attribute that leads to the security of the analyzed voting scheme?

Each ballot has another candidate order which is invisible, but present as encryption on the voter's receipt. The voter can check if her receipt is published, i. e. her vote is included in the tally. There is also a proof that all ballots were counted correctly by proving the correct application of all permutations.

D.2. Analysis of the Requirements

Requirement A (*Eligibility*)

(A1) How does the system identify whether persons presenting themselves to vote are eligible?

Assumed as given.

Requirement B (*Equality*)

(B1) How does the system ensure that each eligible voter can cast at most one ballot?

Assumed as given.

(B2) How is it made sure that each eligible voter has the possibility to cast her vote?

Assumed as given.

Requirement C (*Layout neutrality*)

(C1) How are choices presented to the voter?

On a ballot with two parts: candidate/choice list in random order on the left side, space to put marks on the right side.

(C2) How is it made sure that no option is favored more than specified by election rules?

Ballot audits can make sure that the permutations are distributed randomly and no candidate appears on top more often than others.

(C3) How is it made sure that choices are presented fairly to the voters?

See C2.

Requirement D (*Revisable ballot marking*)

(D1) How does the voter mark her choice?

By putting a mark on the right side next to her choice, on a paper ballot.

(D2) How can the voter verify that the ballot she just created contains a faithful representation of her preferences? Under what assumptions?

She made a mark next to her candidate. Her receipt contains an encryption to

her ballot's permutation. Assuming the encryption acts as a binding commitment and cannot be opened to another permutation, the proof of correctness after the tally shows that for each ballot, the right permutation was used to decipher the ballot. The voter can also use another ballot for auditing before she votes.

(D3) How is it made sure that the voter can revise her ballot?

Before the right-hand side is thrown into the urn, she can just get another ballot.

Requirement E (*Irreversible ballot casting*)

(E1) How does the voter cast her ballot?

After the right-hand side is marked, it is scanned by an optical reader, the scanned data is transferred to the tellers and the scanned right-hand side given to the voter as a receipt. The left-hand side is shredded.

(E2) What defines when a ballot is cast?

Scanning of the right-hand side and transferring it to the tellers.

(E3) What makes it irreversible? Under what assumptions is it irreversible?

The right-hand side being scanned and transferred to the tellers. The receipt appears on a public bulletin board and cannot be taken back.

(E4) What defines the ballot box?

The device where the scanned right-hand sides are stored.

Requirement F (*Privacy and incoercibility*)

(F1) How is individual privacy assured? Under which assumptions?

Privacy depends on the security of the encryption scheme used to encode the permutation, and on the mechanism to calculate the encryption from the germs the tellers choose, and/or on the ballot creation, since the complete ballots also show the permutation.

(F2) How is individual privacy assured in the future? And under which assumptions? (Is it computational? Unconditional? Why?)

It is not everlasting. It depends on whatever mathematical problem the used encryption scheme is based on.

(F3) How is receipt-freeness assured? And under which assumptions?

Measures have to be taken that the voter either really shreds the left-hand side of her ballot or that she can take out dummy left-hand sides with corresponding receipts. (with the second approach, how can the real receipt be distinguished from the dummy ones and act as a proof of manipulation? Commitments to the dummy receipts, maybe).

(F4) How is coercion-resistance assured? And under which assumptions?

Not given, the voter can for example be forced to mark the choice on top whatever the permutation is.

(F5) Who learns the vote or is able to calculate it?

Each authority who can see the complete ballots. It is not specified how ballots are issued to the voter and who except the voter sees them.

(F6) Who gets critical information? What information is this and in which way is it critical?

See F5; Critical information is each ballot's permutation, which is visible in the complete plaintext ballots.

Requirement G (*Secrecy of intermediate results*)

(G1) How does the voting scheme guarantee that no information about the contents of the ballot box leaks before the tallying procedure is completed? Under which assumptions?

The optical reader and the device were its scanned data is stored have to be trusted, or
the authority who knows the permutation can be trusted and the tellers only start tallying after the election phase.
Careful: tellers are used as oracles for auditing, this oracle access possibility must not be abusable for decrypting some real votes.

Requirement H (*Inviolatability of the ballot box*)

(H1) How does the system make sure that no ballots are modified? Under which assumptions?

The receipts—they equal the cast ballots—are published on a public bulletin board, so the voter can see if her vote appears there unmodified.

(H2) How does the system make sure that no ballots are removed? Under which assumptions?

Right hand sides of the ballots have to show up on the bulletin board. Removal of receipts of voters who do not check their receipts is not detected. The original paper suggests additional paper audit trails, this would at least make removals detectable by auditors.
During the tallying process, the correct mixing is publicly proven and everyone sees that the number of the processed right hand sides stays the same.

(H3) How does the system make sure that no ballots are added?

This is not specified.

Requirement I (*Tally integrity*)

(I1) How is the tallying of the votes implemented?

After the voting phase, the ballots are processed as follows: the permutation on each ballot as well as its encryption was calculated by several tellers in an onion kind of way. During the tallying process, the onion is removed level by level by each teller (two steps per teller): each teller removes his encryptions and applies his parts of the ballot permutation for each ballot, then shuffles the modified ballots. In the end, the permutation is decrypted, the vote transformed to a readable ballot and all votes are mixed, so the links to the voters are broken. The shuffling is later checked with Remote Partial Checking (RPC).

(I2) How is it made sure that only the ballots correctly cast are counted?

This is not directly specified by the scheme, but some suggestions are made by the paper. Together with the electoral role one can check the number of cast ballots. It is easy to cast an invalid ballot, though, if the scanner does not check validity, which is not specified by the original paper.

(I3) If not correctly cast ballots are counted, who sees it?

The ballots are not signed by the voter, so each voter can only check her own ballot. Ballot stuffing might be detected.

(I4) If not correctly cast ballots are counted, who can prove it and how?

If the voter's ballot does not appear, she can prove the fraud by showing her receipt, which is stamped and digitally signed by the voting authority. Ballot stuffing might be detected but its origin cannot be proven.

(I5) If not correctly cast ballots are counted, what can be done to correct the error?

We do not see if the ballot was just not counted or substituted by another one. So a ballot for the voter can be added but we do not know if therefore a ballot needs to be removed or which one, except if the substitution happens to have the same encrypted permutation on it.

Requirement J (*Individual verifiability*)

(J1) How can a voter convince herself that her ballot is included in the tally? Under which assumptions?

She finds her receipt on the bulletin board and checks the RPC proofs of correctness. Assumption: encryption acts as binding commitment

(J2) What is the underlying principle used? Physical? Statistical/Probabilistic? Trust in the authorities?

Public bulletin board, RPC, "binding encryption".

(J3) If the above check fails, how can the voter prove that her ballot is not included in the tally?

See I4, she can show her receipt which is stamped and digitally signed. Ballot auditing should prevent invalidly signed and stamped ballots.

(J4) How and by whom can this error be corrected?

The voting authorities can let the voter cast a vote; see I5.

Requirement K (*Auditability and public verifiability*)

(K1) How can an interested party be convinced that the result published by the BTA is correct?

Public proofs of correctness (RPC of the tellers decryption and mixes)

(K2) What is the underlying principle used?

RPC/Verifiable shuffling, auditing (also statistical)

(K3) If the result is not correct, who can prove it and how?

Everyone can see that the proofs of correctness fail for the (public) input data.

(K4) If the result is not correct, can the error be corrected? If so, by whom and how?

Correction is difficult since the manipulation could either have happened during ballot creation or during tallying. The latter is more probable, though, since there was auditing before the election phase. If the teller is able to decrypt correctly (in the paper it is seen as a device) or its key pair is stored somewhere else, the error can be corrected.

(K5) Is it traceable who/what has caused the error?

The RPC proof shows which teller made the manipulation. So if it was a teller, this can be proven.

Requirement L (*Robustness*)

(L1) How is invalid or malformed voter data treated?

Not specified

(L2) What happens when a voter aborts the voting process?

Either the vote is scanned or not. If it is scanned, it will be counted (except if there is an error handling strategy, this is not specified by the scheme.) If it is not scanned, nothing happens. The voter can prove to the authority that she has not scanned her ballot by showing the ballot and letting the authorities check the recordings.

Requirement M (*Availability*)

(M1) How does the system handle the unintentional breakdown of its components?

The onion mixing and tallying process is not very robust, each teller can deny service. But the process can be substituted by more robust mixing techniques, which has been done in newer versions of the scheme.

(M2) How can the system made to gracefully deal with denial of service attacks?

See M1.

Requirement N (*Scalability*)

(N1) How well does the scheme scale? How big can an electoral constituency be?

he scheme seems to scale very well. If the number of ballots per scanner is high enough, several scanners can be used to record ballots. The tallying process involves several tellers, so this could be a bottleneck, but there can also be several groups of tellers assigned to several ballots, if the number of ballots per group is big enough (e.g. one group of tellers for each polling station). But, the bigger the number of voters in a polling stations, the more important is a more robust teller setup.

D.3. Conclusion of the Analysis

(S1) Which of the requirements listed above are fully met under the underlying assumptions?

Requirements D, K and N are met, Requirement C is met if audits check the random distribution if the permutations. Requirements E and H are met with an additional paper audit trail, as suggested in the paper. Requirement G is met under the assumption of a trusted authority. Requirement J is met if the encryption scheme acts as a binding commitment. Requirements A, B and L are up to implementation.

(S2) Which requirements are only partly met, and in which way?

Requirement I is met apart from the ballot stuffing attack. In Requirement F, privacy and receipt-freeness are given, but coercion-resistance is not met.

(S3) Which requirements could be met with minor improvements of the scheme? What are the suggested improvements?

Requirement M can be met with a more robust technique than the onion mixing, as suggested in the literature.

(S4) Which requirements are not met?

None

(S5) For which requirements does it depend on a concrete implementation or application if they are met?

It is not specified how Requirement A and B are met.

(S6) Are there any concluding remarks?

A big advantage of the scheme is that no component learns the voter's choice, under the assumption of a honest ballot creation authority. The analysis of this scheme showed us that if the candidates are presented in a random order, one has to be careful that the permutation is really chosen at random to provide a ballot layout which is fair towards the candidates.

E. Analysis of Scantegrity II

Scantegrity II is a paper-based cryptographic votingscheme. The voter marks her choice on a paper ballot with confirmation codes printed in invisible ink. She marks her choice with a decoder pen and scans the ballot, the confirmation code is not recorded by the scanner.

A trusted authority has a table which records the link between each confirmation code and the corresponding choice. Entries belonging to ballots not used in the election period are reveald for audit. Entries belonging to codes/ballots used in the election are reveald RPC-like.

E.1. General Information about the Voting Scheme

Election type

The first two questions are about the type of the election scheme:

(Q1) Is the election scheme ...

- ... paper based, scanner based or computer based?
- ... meant for presential or internet elections?

Scantegrity II is a paper-based voting scheme designed for presential elections.

(Q2) For which kind of elections is the scheme designed?

It has been used in municipal elections.

(Q3) Does the election scheme allow any special election types?

A version of Scantegrity II provides write-in candidates.

Preliminaries and Assumptions

(Q4) On what cryptographic assumptions is the scheme based?

This depends on the assumptions on which the used commitment schemes are based.

(Q5) If the scheme relies on trusted parties, which parties need to be trusted?

The authorities who generate the tables and the ballots need to be trusted.Auditors are trusted as a group to create good randomness.

(Q6) Which components have to be trusted? Does the scheme rely on trusted hardware?

Invisible ink,Scanner memory

(Q7) Are there other assumptions?

The existence of a voting booth

What makes this scheme secure?

(Q8) What is the main attribute that leads to the security of the analyzed voting scheme?

Invisible ink, and confirmation codes which the voter can keep in mind without noting them.

E.2. Analysis of the Requirements

Requirement A (*Eligibility*)

(A1) How does the system identify whether persons presenting themselves to vote are eligible?

This is assumed as given.

Requirement B (*Equality*)

(B1) How does the system ensure that each eligible voter can cast at most one ballot?

Assumed as given

(B2) How is it made sure that each eligible voter has the possibility to cast her vote?

Assumed as given

Requirement C (*Layout neutrality*)

(C1) How are choices presented to the voter?

Standard paper ballot

(C2) How is it made sure that no option is favored more than specified by election rules?

Auditors audit half of the ballots, and each voter can have ballots for auditing.

(C3) How is it made sure that choices are presented fairly to the voters?

Auditing

Requirement D (*Revisable ballot marking*)

(D1) How does the voter mark her choice?

As in paper elections, but with a special decoder pen which reveals invisible ink.

(D2) How can the voter verify that the ballot she just created contains a faithful representation of her preferences? Under what assumptions?

She gets two ballots: one for audit, one for casting. Interpretation of codes are partly revealed in tables.

(D3) How is it made sure that the voter can revise her ballot?

She can go to pollworker, her ballot is marked as spoiled, she gets a new one. This has to happen before the ballot is scanned.

Requirement E (*Irreversible ballot casting*)

(E1) How does the voter cast her ballot?

Her ballot is scanned.

(E2) What defines when a ballot is cast?

The scanning.

(E3) What makes it irreversible? Under what assumptions is it irreversible?

The scanning and recording of the ballot. The voter's recording of her confirmation code makes it provably irreversible.

(E4) What defines the ballot box?

Scanned ballots, recorded in the scanner's memory.

Requirement F (*Privacy and incoercibility*)

(F1) How is individual privacy assured? Under which assumptions?

Assumption: Ballots are not revealed, the voter is not observered while scanning, used commitments in the tables are hiding. It is not stated in the paper what happens with the ballots after voting, however.

(F2) How is individual privacy assured in the future? And under which assumptions? (Is it computational? Unconditional? Why?)

Privacy only holds as long as ballots are not revealed and commitments used in the tables are hiding. (The ballots contain an ID which the voter has on her receipt). Unconditional/computational privacy depends on the hiding property of the used commitment scheme.

(F3) How is receipt-freeness assured? And under which assumptions?

Assumption: The scanner's memory is not read by the adversary, commitments are hiding, we have enough voters (commitments are revealed RPC-like.

(F4) How is coercion-resistance assured? And under which assumptions?

Whoever gets hold of marked paper ballots can coerce. They contain a ballot ID which the voter has on her receipt. Apart from that, as long as the adversary does not know the connection between receipt-codes and candidates, he will not be able to coerce to voter that a certain receipt-code has to appear in the published tables.

(F5) Who learns the vote or is able to calculate it?

The scanner records the ballot allong with its ballot ID; Whoever sees the ballot learns vote and ballot ID, their processing after scanning is not specified.

(F6) Who gets critical information? What information is this and in which way is it critical?

The tables are the most critical information. They have to be contained in trusted hardware or created and processed by a trusted authority.

Requirement G (*Secrecy of intermediate results*)

(G1) How does the voting scheme guarantee that no information about the contents of the ballot box leaks before the tallying procedure is completed? Under which assumptions?

The scanner memory has to be trusted, so no one can access it before the election period is over.

Requirement H (*Inviolatability of the ballot box*)

(H1) How does the system make sure that no ballots are modified? Under which assumptions?

The voter records her confirmation code. The tables which show the connection between code and choice are opened RPC-like. Each voter can check that the right receipt-codes are opened and the flags are in the right places.

(H2) How does the system make sure that no ballots are removed? Under which assumptions?

The voter recognizes if her receipt code does not appear on the public bulletin board.

(H3) How does the system make sure that no ballots are added?

This is not specified.

Requirement I (*Tally integrity*)

(I1) How is the tallying of the votes implemented?

Scanner-memory is read.

(I2) How is it made sure that only the ballots correctly cast are counted?

Has to be assumed as given.

(I3) If not correctly cast ballots are counted, who sees it?

No one. Authentication has to happen before election, ballot stuffing is only prevented through joint observation.

(I4) If not correctly cast ballots are counted, who can prove it and how?

-

(I5) If not correctly cast ballots are counted, what can be done to correct the error?

-

Requirement J (*Individual verifiability*)

(J1) How can a voter convince herself that her ballot is included in the tally? Under which assumptions?

She checks the reveiled confirmation code in the line of the table belonging to her ballot ID. Used commitment scheme has to be binding.

(J2) What is the underlying principle used? Physical? Statistical/Probabilistic? Trust in the authorities?

See above.

(J3) If the above check fails, how can the voter prove that her ballot is not included in the tally?

She knows her receipt-code, and the probability that she guesses an unrevealed receipt-code which is on a certain ballot is low. Voter privacy is revealed, though.

(J4) How and by whom can this error be corrected?

It can be corrected by the authorities who have the tables.

Requirement K (*Auditability and public verifiability*)

(K1) How can an interested party be convinced that the result published by the BTA is correct?

Everyone can check the revealed parts of the tables (either opened for audit or RPC-like for real votes). Soundness of this proof depends on the binding property of the used commitment scheme. If the voting authority knows which confirmation codes are not checked, it can publish the wrong code and count the corresponding choice.

(K2) What is the underlying principle used? Physical? Mechanical? Electrical? Electronic? Statistical/Probabilistic? Trust in the authorities?

Audits, RPC, commitments

(K3) If the result is not correct, who can prove it and how?

If anyone sees that the tables are wrong, everyone does. If a voter does not find her confirmation code, see J3.

(K4) If the result is not correct, can the error be corrected? If so, by whom and how?

If the audit before the election phase fails, everyone sees immediately that the tables are wrong. The voter can prove a fraud with her confirmation code. The authority can correct the error.

(K5) Is it traceable who/what has caused the error?

If the voter's confirmation code is actually on her ballot (which can only be proven by opening corresponding commitments in the tables), the error was probably done by the authority holding the tables (unless the voter guessed well).

Requirement L (*Robustness*)

(L1) How is invalid or malformed voter data treated?

Not specified by the scheme

(L2) What happens when a voter aborts the voting process?

Either she scans her ballot, or she doesn't. If she doesn't, she has not voted. If she does, her ballot is counted. It is unspecified what happens if she aborts, if she can come back later and obtain a new ballot for casting.

Requirement M (*Availability*)

(M1) How does the system handle the unintentional breakdown of its components?

The only thing that could brake down are the scanners and the tables. Table-backups compete with privacy. If the tables are lost, verifiability becomes impossible, but the tally can still be computed. Backup of the scanner-memory also competes with privacy. (If this is lost, all votes are lost, too).

(M2) How can the system made to gracefully deal with Denial of Service attacks?

There is no internet connection needed while voting. If the scanners do not work, they can be replaced, but their data (cast ballots) might be lost.

Requirement N (*Scalability*)

(N1) How well does the scheme scale? How big can an electoral constituency be?

The scheme scales very well. More than one scanner can be used as long as on each scanner, enough ballots are recorded.

E.3. Conclusion of the Analysis

(S1) Which of the requirements listed above are fully met under the underlying assumptions?

Requirements C,D, E, F, G, H, K and N are met.

(S2) Which requirements are only partly met, and in which way?

-

(S3) Which requirements could be met with minor improvements of the scheme? What are the suggested improvements?

-

(S4) Which requirements are not met?

None.

(S5) For which requirements does it depend on a concrete implementation or application if they are met?

Assumptions A,B,I,J,L and M depend on environment and implementation.

(S6) Are there any concluding remarks?

Scantegrity II has a lot of trust assumptions to hardware and authorities (tables, scanner-memory, handling of marked and scanned ballots). The most critical part is the processing of the tables. Error correcting is easy in some cases and difficult in others. Ballot Stuffing might be a problem. If the voting authority knows that a certain voter will not check her confirmation code, he can count this voter's ballot as he wishes. However, the code is so short that the adversary can never be sure the voter doesn't remember it, even without her receipt.