Diploma Thesis

A Coercion-Resistant Cryptographic Voting Protocol - Evaluation and Prototype Implementation

Darmstadt University of Technology
Department of Computer Science
Cryptography and Computeralgebra
Prof. Dr. Johannes A. Buchmann

Stefan G. Weber

July 2006

Advisor: Roberto Samarone dos Santos Araújo

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, Juli 2006 Stefan G. Weber

Acknowledgements

First of all, I want to thank Prof. Dr. Johannes A. Buchmann and Roberto Samarone dos Santos Araújo, who made this thesis possible, supervised and guided me in the process of its creation.
I also would like to thank Vangelis Karatsiolis and Dr. Katja Schmidt-Samoa for additional support, and Melanie Volkamer for some helpful discussions.
Special thanks go to my family who supported me throughout my studies in every possible way. And nevertheless I want to thank all friends and especially Christina for always giving me new motivation for the next steps of my work.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Voting technology and procedures evolved throughout the history of mankind - from clay balls put in clay pots in ancient Greek, over paper ballots in the arising democratic societies, to the use of electronic devices in modern polling stations [Jon03].
Nowadays, with the Internet being a part of everyday's life, voting technology is imminent to take the next step towards *e-voting*, or more specially *remote electronic voting over the Internet*. This form of electronic voting allows a voter to cast his vote from his personal computer or any other available computer, e.g. in an Internet cafe. Remote electronic voting has the potential to allure great groups of voters that abstain from regular elections as it offers the advantages of convenience, as one can vote without leaving the home, and it does not require geographical proximity on the election day. Especially citizens temporarily living abroad could greatly benefit from this. Moreover computers speed up the computation of the election results and can help to reduce the costs.
However, the use of information technology to conduct elections bears severe risks. As voters should be able to use arbitrary and potentially insecure computers with Internet access, they are in risk of malware and untrusted client software to manipulate their votes as well as denial of service attacks and hackers to threaten the communication infrastructure [Rub02].
Even though fraud, manipulation and errors have appeared in traditional elections as well[1], the use of networking computers allows to commit *automated* fraud [CTL05] on a very large scale. This means that a small number of attackers can influence electronic elections more severely than they could in the traditional case. As well, even small software bugs in the voting system can largely tamper the election results. And furthermore, the voters themselves turn out to be a point of attack for influencing the election. So the important issue is to design *secure* and *reliable election systems* that tackle these shortcomings.

Cryptography is the key technology to secure electronic data flows. It offers advanced cryptographic techniques that can be combined to design *cryptographic voting protocols*. For over two decades cryptographic research has been done on this topic since in 1981 Chaum [Cha81] proposed the first such

[1] See e.g. [Smi05a] for a survey on this issue.

cryptographic election protocol. His work initiated a vast amount of research on several approaches to realize secure solutions for electronic elections. The efficiency and practical applicability of all these approaches has experienced a strong increase in the last years. In the meantime several companies develop commercial solutions and offer technologies for e-voting.
It has shown that a wide range of non-parliamentary elections greatly benefit from employing voting technology, e.g. elections for workers' councils and for student parliaments have seen a growing use of e-voting systems[2]. As well the elections for the most valuable player and best goal keeper in the soccer world championship 2002 were conducted over the Internet[3].
But still it is highly questionable whether electronic voting systems have already reached a level of security and reliability to conduct large scale general elections over the Internet. At least, all those non-political elections smooth the way for such further developments, offering scenarios to test the new technologies in real world applications. This contributes to develop more robust and secure voting technology that might be used to conduct general elections in future.
As pioneer work, Estonia has already practiced legally binding general elections with the possibility to vote online [Com05, BT06]. While further countries are planning to introduce e-voting throughout the next years, the USA recently stopped their plans for security reasons [JRSW04, Mas04].

An important issue is to design and evaluate new voting schemes for future application. In this context this thesis deals with concepts and protocols for remote electronic elections, realizing a high level of security for the voters, that might be suitable for general elections.
To illustrate the meaning of security in the context of voting schemes, we define the following security requirements in the next section.

1.2 Requirements of Voting Schemes

Electronic elections are security-critical applications of information technology. To be called *secure* an election scheme has to fulfil a number of requirements[4] [SP06, Smi05b, SK95]:

- *Correctness:* The protocol includes all (last-cast) votes of authorized voters correctly into the election result.

[2] See [VV06] for a list of recent elections of that kind in Germany.
[3] See http://www.iris.or.kr/evoting/evoting.htm for more details. The election was based on Fujioka et al.'s protocol [FOO92].
[4] There are several different specifications of requirements of a voting system, with different emphases. Consult alternatively e.g. [Gri02, Saf01, HMR04].

- *Universal Verifiability:* Anyone can verify that the protocol correctly processed and tallied all valid votes.
- *Individual or Voter Verifiability:* Each voter can verify that his own (last-cast) vote is correctly included in the final result.
- *Robustness:* The scheme can tolerate a certain amount of cheating participants or authorities.
- *Privacy:* The votes must be kept secret and not be linked to a voter.

Beside these security requirements, a voting system that shall be used in practice has to provide [SP06, BLS+03]:

- *Usability:* The system must be easy to use and offer an understandable user interface. It must offer assistance for handicapped persons.
- *Efficiency / Scalability:* An efficient system has to be scalable concerning computation, storage and communication work factors, to allow for large scale elections.
- *Practicality:* The system may not rely on assumptions and requirements that cannot be implemented in large scale scenarios.

From a legal point of view, elections are supposed to be *free*, *universal*, *secret* and *equal* [Com02, Com04, VH04]. In this thesis we are in particular interested in mechanisms to realize free and secret elections:

- *Free:* The voter is able to vote freely, without being forced or coerced to vote in a particular manner.
- *Secret:* No one may be able to see how a voter voted. No vote may be linked to a voter. It also implies that the voter *has* to keep his vote secret, he may not be able to show his vote validly to anybody else.

The secrecy property is partly covered by the security requirement of privacy. A stronger form of privacy in voting schemes is defined by Benaloh [BT94] in the notion of

- *Receipt Freeness:* A voter may not be able to create a *receipt*, i.e. any information that can be used to convince an attacker or a coercer that he voted in a specific manner.

Receipt-free voting schemes are designed to tackle the threats of *vote buying* and *voter coercion*. Both threats are prevented by the fact that a voter cannot show how he actually voted. Thus the voter has no argument to be paid by a vote buyer, and a coercer cannot force a voter to submit a specific vote for the

same reason. These threats are not introduced by by e-voting[5], but potentially broadened.
Receipt-free voting schemes have the drawback that they do not protect voters against all kinds of coercive attacks. Voters can still be forced not to participate in an election or to let somebody else vote on their behalf. To overcome these serious threats, Juels, Catalano and Jakobsson [JCJ05] define the stronger notion of

- *Coercion Resistance:* "We allow the adversary to demand of coerced voters that they vote in a particular manner, abstain from voting, or even disclose their secret keys. We define a scheme to be coercion-resistant if it is infeasible for the adversary to determine whether a coerced voter complies with the demands."

1.3 Coercion-Resistant Electronic Elections

The first coercion-resistant scheme was described by Ari Juels, Dario Catalano and Markus Jakobsson in "Coercion-Resistant Electronic Elections" [JCJ05][6]. Beside the stronger guaranteed security than in previous receipt-free schemes, Juels et al.'s protocol relies on less physical assumptions. Unfortunately their scheme suffers from a serious drawback: it's runtime is *quadratic* in the number of voters, and thus it is only practical for small elections.
Warren D. Smith [Smi05b] proposed a way to speed up the Juels et al. scheme. Smith mentioned new weaknesses in the scheme and described a new coercion-resistant protocol that includes countermeasures against these and exhibits a *linear* runtime. But his protocol employs *secure multi-party computation* techniques, which make it inefficient and unpractical for large elections, includes errors and departs from the concepts mentioned by Juels et al. in some ways. Smith sketched as well a solution to overcome this efficiency drawback, he proposed to replace the secure multi-party computation steps by more efficient algorithms, but his descriptions and explanations were imprecise in the key parts.

In this thesis, we evaluate Smith' ideas and develop and implement an election

[5] Recent real world examples of vote-buying are the case of the general elections in Thailand in 1996, in which one third of the households had been offered vote-buying [PTCB00], rumors on an Olympic vote-buying scandal, in which IOC members were offered money to vote for certain cities to host the next Olympic Games [BBC98], and the website www.vote-auction.net, which offered a forum to sell votes. As well, many cases of coercion are suspected in homes for the aged in postal votings.

[6] A previous version of Juels et al.'s scheme, [JCJ02], was published before in the Cryptology ePrint Archive, http://eprint.iacr.org/. Thus, some references on their work concern this preliminary version.

scheme with a linear runtime that is *coercion-resistant*, *correct*, *verifiable* and yet *practical*.

1.4 Related Work

In this section we briefly sketch and give an overview on related work.

Cryptographic Voting Protocols

The theory of voting schemes can roughly be divided in mixnet based schemes, blind signature based schemes and schemes based on homomorphic encryption. Hirt [Hir01] offers a brief introduction to these approaches. An extensive list and comparison of voting schemes is given by Sampigethaya and Poovendran [SP06][7]. A broader discussion of voting schemes is given by the EU Cybervote[8] Report [Cyb02].

Receipt-Free Schemes

Benaloh and Tuinstra [BT94] instigated the work on voting schemes that protect voters against coercion by introducing the notion of receipt freeness. Several subsequently published schemes, e.g. those of Benaloh and Tuinstra [BT94], Okamoto [Oka96], Lee and Kim [LK00] and Magkos et al. [MBC01], were later shown not to possess the postulated receipt freeness property. Recent receipt-free schemes suffer from the drawback that they rely on unpractical physical assumptions. For example Hirt and Sako's approach [HS00], the scheme of Baudron et al. [BFP+01] and Moran and Naor's protocol [MN06] assume physical secure (*untappable)* channels, whereas the existence of trusted hardware devices is assumed by Lee and Kim [LK02], Lee et al. [LBD+03] and Aditya et al. [ALBD04a].

Coercion-Resistant Schemes

Coercion resistance as an extension to receipt freeness was introduced by Juels, Catalano and Jakobsson [JCJ05]. Their authorisation mechanism, the use of hidden credentials, was adapted by Acquisti [Acq04], combining it with homomorphic encryption, and by Clarkson and Myers [CM05], combining it with decryption mixes. As Smith [Smi05b], Schweisgut [Sch06] tried to speed up the

[7] We recommend this survey, even though it contains errors in the description and analysis of Juels et al.'s scheme.

[8] The EU Cybervote project, www.eucybervote.org, is a research and development programme funded by the European Commission. It's aim is to develop and demonstrate a high-secure system for remote electronic voting. More on the realized system can be found in [VLB+02].

Juels et al. scheme and combined it with trusted hardware devices. But his solution did only improve the efficiency of one of two crucial protocol steps, so its runtime remained quadratic. Kutylowski and Zagorski [KZ06] presented a different approach to deal with coercion in voting protocols, based on voting cards.

1.5 Scope and Roadmap of this Thesis

This thesis documents the work and results of a diploma thesis project dealing with coercion-resistant cryptographic voting protocols. It focuses on cryptographic topics on an abstract protocol level, but presents a prototype of a voting protocol as well and sketches some implementation aspects.

In chapter 1 we introduce the reader to our thesis, motivate the need for coercion-resistant voting protocols and describe related work. Chapter 2 introduces the necessary background, describing advanced cryptographic concepts. Building on this, chapter 3 presents two variants of a voting protocol that applies these techniques to achieve the coercion resistance property. An evaluation follows in chapter 4. Chapter 5 presents variants and extensions. In chapter 6 we describe our implementation of the coercion-resistant protocol. In chapter 7 we summarize and conclude on our work and point out further work. Finally, appendix A clarifies how to understand Smith' approach.

2 Cryptographic Primitives

In this chapter we describe advanced cryptographic concepts and primitives in the context of electronic voting protocols. These primitives constitute the building blocks of our coercion-resistant voting protocol to be presented in chapter 3, respectively voting protocols in general.
We assume the reader is familiar with the basics of cryptography and modular arithmetics in particular. An introduction to cryptography with focus on the mathematical details is given by Buchmann [Buc04]. For a broader description of modern cryptographic concepts we refer to [Mao03]. Ferguson and Schneier [FS03] introduce the reader to cryptography as an engineering discipline.

We begin our descriptions with the notion of cryptographic protocols.

2.1 Cryptographic Protocols

A *protocol* describes how several participants (*parties*) can accomplish a common task, e.g. how to securely conduct elections over the Internet.
In a protocol each party has to do computations and communicate with the other participants. The computations and the series of communication steps are exactly described. A party that does not follow the protocol can be detected, if the protocol includes verification mechanisms.
A protocol that uses cryptographic methods is called a *cryptographic protocol*. Cryptographic protocols are designed to satisfy various security requirements additionally to achieving a specific task. As mentioned by Lee [Lee01] such requirements can be necessary if the communication takes place in insecure networks as the Internet or if the involved parties mistrust each other.
For a general discussion of cryptographic protocols see e.g. Jakobsson [Jak06].

2.2 The ElGamal Cryptosystem

The *ElGamal Public Key Cryptosystem* was proposed by Taher ElGamal [ElG85], as an extension to the Diffie-Hellman-Key-Exchange [DH76]. Its security is based on the hardness of solving the Diffie-Hellman problem and computing discrete logarithms [Buc04].

We describe the ElGamal cryptosystem for subgroups G_q of order q of Z_p^*,

where p and q are large primes and $p = 2q + 1$ holds[1], as specified by Damgard [Dam04]. Moreover, ElGamal works for any family of groups for which the discrete logarithm problem is considered intractable.

2.2.1 System Parameters and Keys

System Parameters: The primes p, q and a primitive element g of G_q specify the finite cyclic group G_q of order q the cryptographic operations take place in. For our purpose p, q, g are common system parameters, since the same underlying group will be used for the whole voting process described in the next chapter. In other settings they can be part of the public key.

Private Key: The private key s is a random value in Z_q.

Public Key: The public key is $h = g^s \bmod p$. It is published so anyone can encrypt messages under this public key.

2.2.2 Encryption and Decryption

The plaintext space for this variant of ElGamal is G_q while the ciphertext space is $G_q \times G_q$.

Encryption: To encrypt a message $m \in G_q$ under the public key h, a random value $r \in Z_q$ is chosen. The tuple $(x, y) = (g^r \bmod p, h^r m \bmod p)$ is the ciphertext.
The ciphertext depends on both the plaintext message and the random value. ElGamal is a non-deterministic encryption scheme, one plaintext has different ciphertext representations.

Decryption: The plaintext m can be recovered using the private key s as $m = y/x^s \bmod p$.

This variant of ElGamal suffers from the drawback that the plaintext space is G_q. To encrypt arbitrary bit strings a subgroup encoding of the messages is required. It is described next.

2.2.3 Subgroup Encoding

For the ElGamal variant in Z_p^* the messages can simply be encoded as numbers in $\{1, .., p-1\}$. For the subgroup variant this is more difficult. The subgroup

[1] Those are so called *safe primes*. See Ferguson and Schneier [FS03] for more details and motivation on their use.

G_q consists of only some numbers of Z_p^*. But we can use Z_q as plaintext space which is more convenient for practical use as we can map those numbers to G_q bijectively.
Two possible mappings are described next:

- A message $m \in Z_q$ is mapped to g^m, since $G_q = \{g^0, g^1, g^2, ..., g^{q-1}\}$ for a primitive element g generating the subgroup. This encoding, proposed by Cramer et al. [CGS97], allows for homomorphic encryption and combination of ciphertexts[2].
  The drawback of this encoding is that the decoding of m involves the computation of a discrete logarithm, which is only practical for small values of m.

- We can map between Z_q and G_q more efficiently [Dam04]:

$$f(x) = \begin{cases} x+1 & \text{if } (x+1)^q = 1 \bmod p \\ p-x-1 & \text{else} \end{cases}$$

  To map back from G_q to Z_q compute

$$f(g) = \begin{cases} g-1 & \text{if } g \leq q \\ p-g-1 & \text{else} \end{cases}$$

Smith [Smi05b] proposed to used the first mapping, but we will employ the second choice, as it allows for a more efficient decoding.

Therefore when ElGamal is employed over G_q the following steps have to be taken: To encrypt a message $m \in Z_q$, it has to be mapped to G_q first and can then be encrypted as described in section 2.2.2. Decrypting yields again the message subgroup encoded in G_q. It has to be mapped back to Z_q to recover the initial message m.

2.2.4 Properties

The ElGamal cryptosystem as described above has the following properties that qualify it to be the right cryptosystem for our purposes:

– *non-deterministic encryption and semantical security:* A non-deterministic encryption scheme is semantically secure [GM84] if

[2]Homomorphic encryption enables computations on encrypted data, such that multiplication of ciphertexts yields their sum. This can be used to compute the tally in an election without decrypting single votes. See section 5.4 for more discussions.

no partial information about the plaintext is leaking from the ciphertext. So an adversary cannot determine whether two ciphertexts encrypted under the same public key represent encryptions of the same plaintext. Thus the encrypted voters' choices cannot be deduced. ElGamal is semantically secure in G_q (under the assumption that the Decisional Diffie Hellman problem is hard there), but not semantically secure in Z_p^*. For more discussion see e.g. [TY98, FS03, Bon98].

- *re-encryption without private key possible:* This allows to anonymize messages and to design re-encryption mixnets - see section 2.6.
- *threshold version can be constructed:* This allows for a distributed trust model and a robust implementation - see section 2.5.

2.3 Zero Knowledge Proofs

Zero knowledge proof protocols, introduced by Goldwasser, Micali and Rackoff [GMR85, GMR89], are two-party protocols between a *prover* and a *verifier*. They allow the prover to convince the verifier that he knows a secret, but without giving any information about the secret - after the execution of the protocol the verifier has gained *zero knowledge* concerning the secret.
A zero knowledge proof (ZKP) is comparable to a challenge-response protocol. Figure 2.1 illustrates the concept of such interactive proofs. The prover convinces the verifier about the statement X, e.g. "I know the secret X.", in a series of challenges and responses. The verifier finally accepts the proof if the prover validly responded to a sufficient number of challenges, which is possible if the prover indeed knows the secret.
So this kind of proofs offers a probabilistic security. The cheating probability, i.e. the probability that the prover correctly guesses all the challenges, decreases by the number and length of challenges the prover has to respond to.

For a formal introduction to zero knowledge proofs see Goldreich [Gol02]. We omit these details and a wide range of definitions here. Instead we confine ourself to the basic definition that a zero knowledge proof gives nothing beyond the validity of the proved assertion. We assume that only negligible information about the secret is leaking in the execution of a zero knowledge proof protocol.

2.3.1 Non-Interactive Zero Knowledge Proofs (NIZKPs)

The aim of zero knowledge proofs in voting protocols is to assure and to document the correctness of certain protocol steps without leaking the involved

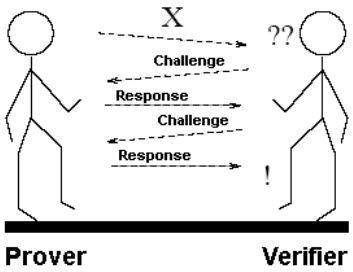

Figure 2.1: An interactive Proof Protocol. Source: modified from [Gol02]

secrets. The correctness should be publicly verified, preferably by multiple parties, but interactive proofs only convince the single verifier involved in this two-party protocol.
A way to overcome this problem is to make proofs non-interactive, and thus public verifiable. This can be done by applying the Fiat-Shamir heuristic [FS86][3] to the proof protocols.
In this solution, the challenges are created by the prover himself by means of a standard hash function, e.g. SHA-1. So interaction between prover and verifier is not required anymore, which makes this approach efficient. The input to the hash function consists of the prover's initial message and common system parameters. A published proof consists of all messages including the self-created challenge. The role of the verifier can now be played by any party that checks the published proofs.

In the following we present zero knowledge proof protocols restricted to a constant number of rounds, three-move protocols, due to Cramer [Cra96] known as Σ-protocols.
In these, first, the prover sends an initial commitment message, then the verifier sends a challenge. Last the prover sends his response. Upon verification of this the verifier decides whether to accept the proof or not. To make the protocol non-interactive the challenge is computed by a hash function. For NIZKPs, the verification includes an additional step, that we do not mention in the following: it has to be checked that the challenge was created correctly, by reproducing it.

All presented proofs work in same setting as our ElGamal variant, i.e. in the subgroup G_q of Z_p^* and p, q, g are common system parameters and h is a public encryption key.

[3] Bellare and Rogaway formalized this concept as the *random oracle model* [BR93]. Even though schemes may be secure in the random oracle model but insecure for any implementation of the hash function [CGH98], a proof of security in this model is a strong indicator for practical security of a scheme.

2.3.2 Proving Knowledge of Plaintext

Given an ElGamal encryption $(x, y) = (g^k, h^k m)$ the following protocol due to Schnorr [Sch91] allows to prove the knowledge of the random value k used in the encryption. This implies the knowledge of the message m, as g and h are public.

1. Prover P chooses at random $w \in Z_q$, computes $A = g^w \bmod p$ and sends A to the verifier.
2. Verifier V chooses at random a challenge $c \in Z_q$ and sends it to P.
3. P computes the response $r = w + ck \bmod q$ and sends it to V.
4. V checks that $g^r \stackrel{?}{=} Ax^c \bmod p$ and accepts the proof in this case.

As $g^r = g^{w+ck}$ and $Ax^c = g^w g^{k^c} = g^w g^{kc} = g^{w+kc}$ the check succeeds. To construct a valid response the knowledge of k is required, or to guess the correct value in Z_q.

The proof can be made non-interactive by computing the challenge value using a hash function $c = H(A, x, g) \bmod q$. Then (A, c, r) is the non-interactive proof of knowledge.

Based on this basic proof protocol, more complex zero knowledge protocols can be designed. The general techniques for combining zero knowledge proofs are discussed by Cramer et al. [CDS94].
A conjunction of proofs of knowledge is described in the next section, followed by a disjunctive proof.

2.3.3 Proving Equality of Discrete Logarithms

The following protocol due to Chaum and Pedersen [CP93] is a proof for the relation $\log_g x = \log_h y$, for two values $x = g^\alpha$ and $y = h^\alpha$. The prover demonstrates his knowledge of α and the proof assures that both x and y have the same discrete logarithm α in the bases g and h.

1. Prover P chooses at random $w \in Z_q$, computes $a = g^w \bmod p$, $b = g^w \bmod p$ and sends a, b to the verifier.
2. Verifier V chooses at random a challenge $c \in Z_q$ and sends it to P.
3. P computes the response $r = w + c\alpha \bmod q$ and sends it to V.

4. V checks that $g^r \stackrel{?}{=} ax^c \bmod p$ and $h^r \stackrel{?}{=} by^c \bmod p$ and accepts the proof in this case.

In the non-interactive version the challenge value is computed as $c = H(h, y, a, b) \bmod q$ using a hash function. Then (a, b, c, r) is the non-interactive proof.
A further discussion of this proof protocol is given in [CGS97]. It will be applied to construct a distributed decryption protocol in section 2.5.4.

2.3.4 Proving Validity

Based on an OR-combination of L proofs of knowledge, the following protocol due to Lee [Lee01] proves that an ElGamal ciphertext $(x, y) = (g^k, h^k m_i)$ encrypts a message m_i, which belongs to the set $M = (m_1, m_2, ..., m_L)$ of L allowed plaintexts, without exposing m_i. It is a proof for the relation

$$\log_g x = \log_h(y/m_1) \vee ... \vee \log_g x = \log_h(y/m_L)$$

We describe the non-interactive variant:

Prover P:

1. P chooses a random number $w \in Z_q$ and computes $a_i = g^w \bmod p, b_i = h^w \bmod p$.
2. For $j = 1, ..., i-1, i+1, ..., L$, P chooses at random $r_j, d_j \in Z_q$, and computes $a_j = g^{r_j} x^{d_j} \bmod p$ and $b_j = h^{r_j}(y/m_j)^{d_j} \bmod p$.
3. P computes $c = H(a_1, b_1, ..., a_L, b_L) \bmod q$, where $H()$ is a hash function.
4. P computes $d_i = c - \sum_{j \neq i} d_j \bmod q$ and $r_i = w - kd_i \bmod q$.
5. P publishes $(A, B, D, R) = (a_1, b_1, d_1, r_1, ..., a_L, b_L, d_L, r_L)$.

Verifier V:

1. V checks $d_1 + ... + d_L \stackrel{?}{=} H(g^{r_1} h^{d_1}, h_1^r (y/m_1)^{d_1}, ..., g^{r_L} h^{d_L}, h_L^r (y/m_L)^{d_L})$. This checks implicitly

   (1) $c \stackrel{?}{=} \sum_i d_i \bmod q$

   (2) $a_i \stackrel{?}{=} g^{r_i} x^{d_i} \bmod p$

$$(3)\ b_i \stackrel{?}{=} h^{r_i}(y/m_i) \bmod p$$

(A, B, D, R) is the non-interactive validity proof for an ElGamal encryption $(x, y) = (g^k, h^k m_i)$. The prover proves knowledge of the random k in the encryption of one valid plaintext among L possible plaintexts, which implies the validity of the encryption.
This proof is later applied to guarantee the correctness of an encrypted vote, i.e. that it encrypts a valid candidate, without revealing which.
This proof allows a flexible encoding of the message m_i. Therefore we prefer it over Smith' proposal to use *range proofs* [Bou00, Lip01] as validity proofs. These protocols prove that an exponent belongs to an interval, thus messages need to be encoded in the exponent.

2.3.5 Indecomposable Conjunction of NIZKPs

In the voting protocol, voters have to include a validity proof and a proof of knowledge to their ballot. To prevent an adversary from copying one of these proofs to construct a new ballot, proofs are *indecomposable anded*.
The indecomposability, as proposed by Smith [Smi05b], is achieved by including common bits from the ballot into the input of the hash function that computes the challenges. So without the context of the original ballot, one cannot validly reproduce this challenge, thus verification will fail.

Further Reading

Recent work on zero knowledge proofs can be found in [Lee01, Gro04]. See also Mao [Mao03] and Stinson [Sti95] for general discussions.

2.4 Secret Sharing

Adi Shamir [Sha79] introduced *secret sharing* in 1979. It is a technique to distribute information among several parties.

To share a secret $s \in Z_p$ among n parties a random polynomial $f(x)$ over Z_p of degree $t-1$ is chosen, such that $f(0) = s$:

$$f(x) = s + \sum_{i=1}^{t-1} a_i x^i$$

Each of the n parties receives a point $(x_i, f(x_i))$ on this polynomial, and $(0, f(0))$ is not among these points.

If any t out of n parties cooperate they can reconstruct the entire polynomial $f(x)$ using Lagrange interpolation and thus they can produce $f(0) = s$.
If only $t - 1$ parties cooperate they cannot reconstruct the polynomial and each constant part of the polynomial, the secret $s = f(0)$, is equally likely. In general, no party can derive information about the secret from his share.
The technique described above is called a (t, n) secret sharing. t is the so called threshold, it specifies the minimum number of parties that need to cooperate to reconstruct the secret[4]. For a more detailed discussion and mathematical details see [Buc04].
The drawback of basic secret sharing is that the dealer who distributes the shares has to be trusted. *Verifiable secret sharing*[5] is a stronger variant which overcomes this drawback and also guarantees the integrity of the shares in the distribution and reconstruction phases.

In the context of voting one application of secret sharing is to construct threshold cryptosystems. The next section presents this concept and especially a threshold ElGamal cryptosystem.

2.5 Threshold ElGamal Cryptosystem

2.5.1 The Goal of Threshold Cryptosystems

Threshold cryptography is the concept to distribute the operations of a cryptosystem among a group of authorities[6]. This approach guarantees robustness, as it removes the single point of failure in *centralized* cryptosystems and avoids abuse by a single authority [Des97]. The required trust in the cryptographic service is distributed among the group of authorities.

In a (t, n) threshold cryptosystem the private key is (t, n) secret shared among the authorities, while *one* public key is published. Any group of at least t authorities can jointly decrypt messages encrypted under this public key using a distributed decryption protocol[7].
An adversary thus needs to compromise at least t of the authorities to decrypt

[4] We are aware of different notations, e.g. Gennaro et al. [GJKR99, GJKR03], Jarecki [Jar01] and literature concerning the theory of *secure multi-party computation* specify the threshold to be at $t + 1$.

[5] Verifiable Secret Sharing and its applications have widely been studied throughout the last years. The interested reader is referred to [Ped92, Gen96, Feh03, Nik05] for more discussions.

[6] In real world organizations, e.g. banks, individuals are often not allowed to take security-critical actions alone. Instead a group of people is responsible for these actions. Threshold cryptography is the cryptographic analogon to this. Straub [Str04] calls this a *four-eye* oder *double verification principle*.

[7] Threshold signature schemes are existent, too, but are beyond the interest of this thesis.

messages or to mount a denial of service attack against the system's cryptographic service. A *minority* of compromised authorities can be tolerated[8] [Jar01].

2.5.2 Threshold ElGamal Cryptosystem

We now present the threshold ElGamal cryptosystem due to Cramer et al. [CGS97] that will be used in the election protocol with multiple authorities. Again, the setting is a subgroup G_q of Z_p^*.

The threshold ElGamal cryptosystem consists of

1. a *distributed key generation protocol* to jointly generate a key pair
2. a *distributed decryption protocol* to decrypt a ciphertext jointly without reconstructing the private key.

2.5.3 Distributed Key Generation Protocol

Before the actual key generation starts, the authorities jointly generate the system parameters p, q, g. This can be done by one authority generating the parameters and the other authorities verifying their correctness, or by using Bach's algorithm for publicly choosing such a triplet [BS96], to cooperatively generate these.

Subsequently, in the main *distributed key generation protocol*, the n authorities collectively generate and deal out shares corresponding to a verifiable (t, n) secret sharing of a random value, the private key $x \in Z_q$. It is generated cooperatively such that no single authority knows its value[9].

The following description shows a simplified variant[10] of Pedersen's [Ped91, Ped92, GJKR03] well-known protocol for this task:

1. Each authority P_i chooses $x_i \in Z_q$ at random.
2. Each P_i chooses at random a polynomial $f_i(x)$ over Z_q of degree $t-1$ such that $f_i(0) = x_i = f_{i0}$:

$$f_i(x) = f_{i0} + f_{i1}x + f_{i2}x^2 + ... + f_{i,t-1}x^{t-1}$$

[8] The system can tolerate $t-1$ authorities with malfunctions, but still requires t correct working authorities. Thus, the total number of authorities n has to be at least $n \geq t-1+t = 2t-1$, so $t-1$ is the minority.

[9] Smith [Smi05b] calls this an *immaculately conceived shared secret*.

[10] See Jarecki [Jar01] for a discussion of the simplifications. We skip an initial commitment stage and do not use digital signatures.

3. Each P_i broadcasts $F_{ij} = g^{f_{ij}} \bmod p$ for $j = 0, ..., t-1$.

4. Each P_i sends secretly $s_{ij} = f_i(j) \bmod q$ to P_j for $j = 1, ..., n$.

5. Each P_i verifies the shares received from the other authorities by checking for $j = 1, ..., n$:

$$g^{s_{ji}} \stackrel{?}{=} \prod_{l=0}^{t-1} F_{jl}^{i^l} \bmod p$$

   If the check fails, P_i broadcasts a *complaint* against the corresponding authority P_j.

6. If more than $t-1$ authorities complain against P_j, that authority is disqualified.

7. Each P_i sets his share s_i of the private key x as the sum of all received shares:

$$s_i = \sum_{j=1}^{n} s_{ji} \bmod q$$

   This is a (t, n) secret sharing of the private key x.

8. The Public Key h is computed (by every P_i) as:

$$h = \prod_{i=1}^{n} F_{i0} = \prod_{i=1}^{n} g^{x_i} \bmod p$$

9. For each share s_i of x let ρ_i denote the commitment to s_i, $\rho_i = g^{s_i} \bmod p$. Each P_j can compute ρ_i as:

$$\rho_i = \prod_{j=1}^{n} g^{s_{ji}} = \prod_{j=1}^{n} (h_j \prod_{l=1}^{t-1} F_{jl}^{i^l}) \bmod p$$

Authorities violating the protocol are detected and disqualified. After the execution of the protocol the values ρ_i are published, so that the authorities are committed to their shares of the private key [CGS97]. These commitments will be used to control the authorities' correct behaviour in the *distributed decryption protocol*.

Security Concerns on Pedersen's Protocol

The protocol has been analyzed by Gennaro, Jarecki, Krawczyk and Rabin. Even so it does *not* guarantee a uniform distribution of the generated private key [GJKR99], it is *secure enough in practice* in this respect [GJKR03].
An alternative protocol guaranteeing a uniform distribution is presented in [GJKR99][11].

2.5.4 Distributed Decryption Protocol

Whilst the encryption is retained unchanged from the ElGamal cryptosystem as described in section 2.2.2, the decryption operation is distributed among the authorities as well.

The authorities jointly decrypt a ciphertext $(x, y) = (g^r, h^r m)$, with $m \in G_q$, without explicitly reconstructing the private key using the following *distributed decryption protocol* [CGS97]:

1. Each authority computes its partial decryption $w_j = x^{s_j}$, using its share s_j of the private key, and broadcasts it together with a non-interactive zero knowledge proof that

   $$\log_g \rho_j = \log_x w_j$$

   using a proof of equality of discrete logs as specified in section 2.3.3. This assures that the authority indeed utilized its correct share to produce the partial decryption.

2. For any subset Λ of t authorities with valid zero knowledge proofs the plaintext can be reconstructed using discrete Lagrange interpolation

   $$m = \frac{y}{\prod_{j \in \Lambda} w_j^{\lambda_{j,\Lambda}}} \bmod p$$

   where

   $$\lambda_{j,\Lambda} = \prod_{l \in \Lambda \setminus \{j\}} \frac{l}{l - j} \bmod q$$

   are the appropriate Lagrange coefficients.

[11] The EU Cybervote protype [VLB+02] implements this more complex protocol. Since the key generation is executed in a setup phase before the processing of ballots, the extra costs of this solution can be tolerated.

No invalid partial decryption is included in the reconstruction of the plaintext, and the decryption is successful if at least t authorities follow the protocol. The private key stays secret as it is not reconstructed.
A subgroup decoding is applied after the decryption is reconstructed, to represent the message in Z_q.

2.5.5 The Need for a Reliable Combiner

The distributed decryption protocol requires a further party that *combines* t partial decryptions to produce the complete decryption. This combiner needs to be reliable [Des97] to produce valid plaintexts. *Any* of the authorities can fulfil this task.

This illustrates the general model for computations in a distributed setting: operations involving a secret shared piece of data, e.g. the private key, are distributed among all authorities in the sense that each authority produces partial operation results. All other operations can be done by a *single* authority, and the result is verifiable, as it is deterministic and can be redone, like the combination of all partial results.

Further Reading

For more discussions on threshold cryptography, we refer the interested reader to the work of Jarecki [Jar01], Pedersen [Ped92] and Desmedt et al. [DF89, Des97].

2.6 Mixnets

A *mixnet* is a distributed computation and communication protocol for achieving privacy in electronic communications. Mixnets provide a mechanism to establish anonymous channels and they are therefore a central tool for a certain type of voting protocols[12]. They were originally introduced by Chaum [Cha81].

A mixnet consists of a set of *mix severs*, the *mixes*. Each mix processes a batch of incoming messages in a way that the outgoing messages are unlinkable to the incoming ones, and forwards the whole batch to the next server in the mixnet. Basically there are two types of mixnets, *decryption mixnets* and *re-encryption mixnets*. In Chaum's decryption mixnet construction, incoming messages are anonymized by removing the messages' encryption layers and permuting the order of each batch of messages.
The re-encryption property of certain non-deterministic encryption schemes,

[12]This basic approach will be shortly presented in section 3.2.

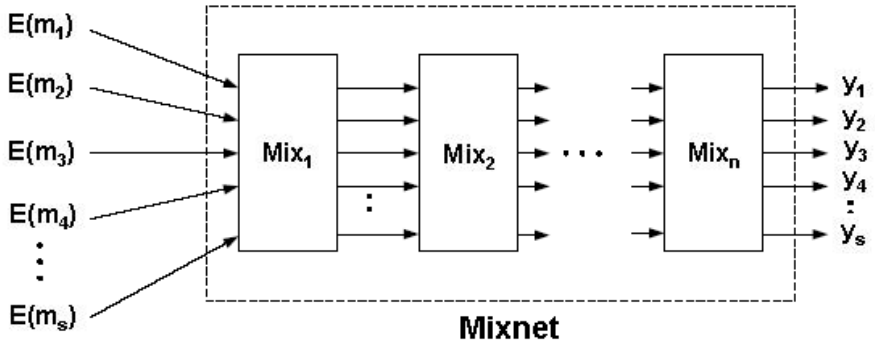

Figure 2.2: Re-Encryption Mixnet. Source: modified from [SP06]

e.g. the ElGamal cryptosystem and its variants, allows to design re-encryption mixnets. In these, originally proposed by Park et al. [PIK93], incoming messages are re-encrypted in each mixing stage, which is possible *without knowledge of a private key*, and shuffled by a random permutation.
Figure 2.2 illustrates this processing and forwarding of messages in a re-encryption mixnet. Here $E(m_i)$ denotes an encrypted message. The output consists of re-encrypted messages Y_i in random order.

2.6.1 An ElGamal Re-Encryption Mixnet

We describe the model of ElGamal re-encryption mixnets as specified by Boneh and Golle [BG02] adapted to our context. The authorities running the mix servers also operate a threshold ElGamal cryptosystem. Its system parameters p, q, g and the public key h are required for processing the inputs.

The operation of a re-encryption mixnet consists of the following steps:

1. *Setup Phase:* The authorities responsible for running the mix servers jointly generate a key pair and appropriate system parameters of the threshold ElGamal cryptosystem, as described in section 2.5.2.

2. *Submission of Inputs:* A batch of ElGamal encrypted inputs $(g^r, h^r m_i)$ is submitted to the mixnet.

3. *Mixing Phase:* Mix server M_i receives the the batch of ciphertexts output by the previous mix server M_{i-1} (or the initial input respectively). M_i re-encrypts each ciphertext $c_i = (g^r, h^r m_i)$ by selecting a random value $r^{'} \in Z_q$ and computing $c_i^{'} = (g^r * g^{r^{'}}, h^r * h^{r^{'}} m_i) =$

$(g^{r+r'}, h^{r+r'} m_i)$. Then it permutes the batch of all ciphertexts randomly and passes it to the next mix server M_{i+1}.

4. *Decryption Phase:* After the last mix server has processed the ciphertexts, the authorities jointly decrypt each message in the final output.

In the next chapter, the operation of a mixnet is embedded in the voting protocol, and thus confined to the submission of inputs and the mixing phase. Setup and decryption phases are executed in the voting protocol itself.

2.6.2 Verifiable Mixnets

A *verifiable* mixnet extends the mixing phase. A mix server additionally has to prove in zero knowledge that he re-encrypted and shuffled the inputs correctly. This prevents cheating mixes from replacing and altering processed messages. There are several approaches to realize verifiable mixnets, the main difficulty is to construct *efficient* proof techniques. We omit these details in this thesis. See e.g. [BG02] for a short comparison of recent approaches.

In the next chapter we assume the availability of an efficient verifiable reencryption mixnet.

Further Reading

For further reading on mixnets we refer the interested reader to the recent work of Serjantov [Ser04] and Wikström [Wik05] and references therein.

2.7 Bulletin Board

The *bulletin board model*, introduced by Benaloh [CF85, BY86, Ben87], specifies the communication model for the election process, enabling broadcast communication and universal verifiability.
A bulletin board is a public readable communication channel with memory. Writing to a bulletin board is append-only, data already written cannot be altered or deleted anymore and its content can be read by anyone.
The communication channels in all described protocols are implemented by means of a bulletin board. It can be used to realize a broadcast channel and private channels, using additional public key encryption.

A bulletin board can have a designated section for each participant in a protocol, implemented with digital signatures. It can also be used to implement an anonymous broadcast channel without designated sections.
Bulletin boards further document the steps of all protocols and store the broadcast non-interactive zero knowledge proofs of correct work e.g. in the distributed decryption protocol (see section 2.5.4), so they are accessible to any external verifier.
A fault-tolerant implementation of the bulletin board service is assumed in the next chapter. Practically bulletin boards can be implemented as replicated webservers with a database backend.

Further Reading

A further discussion of the bulletin board model is given by Schoenmakers [Sch00].

2.8 Anonymous Credential System

The voting protocol to be presented in the next chapter relies on an anonymous authorisation and authentication mechanism. To achieve this it employs *anonymous credentials*, comparable to the approach of Camenisch et al. [CL01].
Each user, i.e. any voter, of the anonymous credential system obtains his credential in a secure way from the issuing organization, i.e. the election authority. He can use it to authorize his ballots to be included in the final tally.
Each credential is a unique secret value $\sigma \in G_q$, where G_q is the subgroup specified by the discrete logarithm setting, as described in section 2.2. The value σ is only known to the voter and the issuer. The issuing organization maintains a reference list of non-deterministic encrypted credentials, that can be used to *blindly* check the validity of employed credentials, if necessary.
A basic application of credentials it to delegate them to other parties, to allow those to fulfil the associated task. In the special scenario of elections this is not a desired property. Only a voter shall be able to vote for himself.
So this special kind of anonymous credential cannot be delegated to someone else in the sense that the receiver is unable to verify if the credential is valid, as he cannot distinguish between a valid and fake credential. The aim of this mechanism is to protect the users against coercion.

Detailed discussions of the use of the credentials, storage problems and conditions are given in sections 4.6 and 4.7.

Further Reading

For more general discussions on credential systems, see Brands et al. [Bra02, BL02] and Ferguson and Schneier [FS03].

3 A Coercion-Resistant Voting Protocol

In this chapter we apply the building blocks described in chapter 2 to construct efficient coercion-resistant cryptographic protocols for secure remote electronic elections.
First, we introduce our modelling of electronic elections and sketch a mixnet based voting scheme and the inefficient Juels et al. scheme [JCJ05] as well. Building on this, Smith' [Smi05b] ideas allow to construct our efficient coercion-resistant protocol, which we describe in two variants.

3.1 Modelling Elections

This section presents the abstract modelling of the election process due to Juels et al. [JCJ05], Smith [Smi05b], and Schoenmakers [Sch00]. The modelling specifies the participants in a voting scheme and their roles. Note that this is a *general* modelling of mixnet based protocols.

3.1.1 Involved Parties

The following entities or respectively sets of entities participate in the coercion-resistant voting protocol:

- *Candidates:* Compete in the election. The candidates can be real persons, political parties, or more generally a set of possible choices, e.g. $\{\text{Yes}, \text{No}\}$ for a public referendum.
- *Voters:* Denoted by V_i. Cast encrypted votes expressing their intented choice of candidate. In the following we consider 1-out-of-k elections, i.e. one candidate among a set of k possible candidates may be chosen by a voter.
- *Election Authorities:* Denoted by EA_i, responsible for conducting the election. The role of the EA's can be further subdivided into

    * *Registrars*, responsible for registering voters and issuing keying material to them.
    * *Talliers*, responsible for processing the ballots and computing the final tally, i.e. the election result.
    * *Mixers*, that anonymize the cast ballots, thus unlinking the ballots from voters. All mixers together run a verifiable re-encryption mixnet. Mixers prove in zero knowledge that they do their work correctly. Compare with section 2.6.
    * *Officers*, operating the bulletin board servers, see section 2.7 for details.

  – *Adversaries and Coercers:* Threaten the election process by compromising the *EAs* and forcing the voters to vote in a specific manner, abstain from voting or disclose their secret keys respectively credentials.

  – *Verifiers (Scrutineers/Observers):* Responsible for auditing the whole election process and verifying the correctness of the election process and the election result.

A participant in the election is not restricted to play a single role, as stated by Schoenmakers [Sch00]. For example, each voter is also able to play the role of the verifier, or may furthermore be among the candidates. This depends on the legal constraints of the certain election.

This is an abstract model of the election process. In an implementation several server or client computers perform the tasks, i.e. the cryptographic operations and data processing. But real persons (or groups of persons) are responsible for deploying, administrating and running these computers or casting their ballots from client machines.

3.1.2 Phases of the Voting Process

An election is divided into distinct phases, that make up the whole process. The main phases are:

  – *Setup:* Cryptographic keys and system parameters for the election are generated.

  – *Registration:* Eligible voters are registered, thus added to a list of voters and therefore authorized to participate in the election. Voters receive secret keying material in a secure manner in this phase.

- *Voting:* The voters cast their ballots.
- *Tallying:* The responsible authorities process the ballots according to the voting protocol to compute the tally, i.e. the result of the election.
- *Verifying:* Verifiers audit the whole electoral process. While the other phases are consecutive, the verification phase proceeds parallel to the election and does not end with the publishing of the election result.

3.1.3 Communication Channels

The communication channels in a voting protocol and subprotocols respectively are provided by a set of bulleting boards $\{BB_1, .., BB_s\}$, as described in section 2.7. Additionally, public hashtables are required in the coercion-resistant protocol to be presented. Those can be stored and maintained on bulletin board servers as well.

The Juels et al. scheme makes additional assumptions on the registration and voting phases, which are required for our coercion-resistant scheme as well:

- For the *registration phase*, we assume that voters communicate over an *untappable channel* [SK95] with the registrars.
  An untappable channel is a *physical* secure channel allowing for unobservable communication. It *cannot* be implemented with cryptographic techniques.
- In the *voting phase*, we assume that voters cast their ballots over anonymous channels.
  An anonymous channel is "a channel such that an attacker cannot determine whether or not a given voter cast a ballot" [JCJ05].

3.2 A Mixnet based Election Protocol

To motivate the key ideas behind the coercion-resistant protocol, we sketch a simple mixnet based voting protocol and point out its inability to protect voters against certain forms of coercion.
In mixnet based schemes voters submit their encrypted and digitally signed ballots to a bulletin board. Then the signatures are checked to authenticate the voter, according to a pre-registered list of authorized voters. Afterwards the encrypted ballots are run through a mixnet to anonymize them. The output of

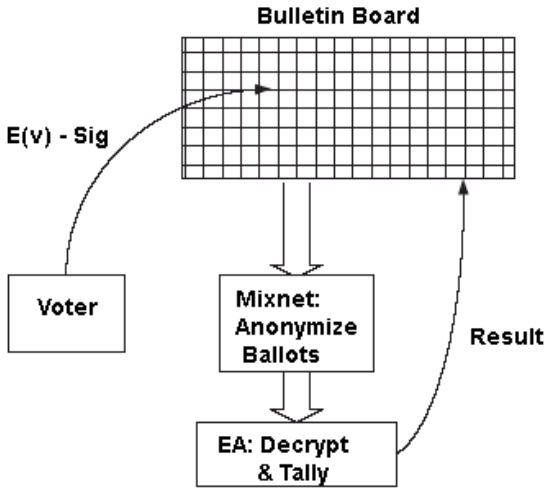

Figure 3.1: Voting Protocol based on Mixnets. Source: modified from [SP06]

the mixnet is decrypted[1] by the election authority afterwards and then tallied. Figure 3.1 illustrates this basic concept. $E(v)$ denotes the encrypted vote, Sig denotes the attached signature.
Protocols of this kind are e.g. [Cha81, PIK93, FS01, BG02, JJR02, GZB+02].

Though the election is private and an adversary cannot tell which vote was cast by which voter, he can verify *if* a voter has actually cast a ballot, by the signature and the according public signature keys. Thus, an adversary can force a voter to abstain from voting, threatening him with some kind of punishment, as he is able to see if the voter complies with his demands. A direct authentication exposes the voters to such coercive attacks.
Receipt-free voting schemes based on homomorphic encryption, e.g. [HS00, BFP+01, LK02], suffer from the same drawback as well. Even though voters cannot prove to anyone *how* they voted a coercer can deduce whether or not they voted.

3.3 Juels et al.'s Scheme

As described above a different authentication mechanism is required to design a coercion-resistant voting scheme. We present Juels et al.'s new idea and sketch their proposed scheme afterwards.

[1] If a decryption mixnet [Cha81] is used, the decryption step is performed by the mixnet itself.

3.3.1 Idea behind the Coercion-Resistant Scheme

Juels et al.'s key idea to design a coercion-resistant voting protocol is that the identity of each voter must remain hidden during the whole execution of the protocol. This enables a voter to deceive a coercer about his behaviour and participation in the election. A coerced voter is able to pretend to behave as requested, leaving the coercer in doubt about deviating behaviour, giving him no reason to punish the voter.
To achieve this Juels et al. proposed an indirect authentication and authorization mechanism. A voter does not authenticate directly at the time he casts his ballot, e.g. by attaching a signature. Instead he casts his vote together with a *non-deterministic encrypted credential*, i.e. an encrypted secret value known only to the voter and the election authorities. These credentials can be checked blindly, i.e. without revealing the voters identity, against a list of registered voters when votes are tallied, thus authorizing the ballots of registered voters.
A voter can give away fake credentials to an adversary that tries to impersonate him. The adversary is unable to verify the correctness of the received credential.
If a voter is forced to cast his vote in a specific way he can also use a fake credential. The coercer cannot distinguish between valid ballots and those cast just to satisfy the adversary's demands and which will be discarded in the vote processing.

3.3.2 Sketch of Juels et al.'s Scheme

Combining the above idea with advanced cryptographic techniques such as *non-interactive zero-knowledge proofs*, *verifiable mixnets* and *plaintext equality tests* Juels, Catalano and Jakobsson [JCJ05] give a high-level description of their coercion-resistant election scheme. As the indirect authentication mechanism allows voters to submit multiple ballots with the same valid credential, the protocol assures that only one ballot is considered for tallying. We describe the phases of the protocol next:

In the *registration phase* voters receive their credential through an untappable channel. A list of registered voters and their non-deterministic encrypted credentials is set up on a bulletin board. Figure 3.2 illustrates the registration phase. Here σ denotes the credential and $E(\sigma)$ its encryption.

In the *voting phase* voters submit their ballots over an anonymous channel to a bulletin board. Each ballot contains non-deterministic encryptions of their vote and their credential together with a non-interactive zero knowledge validity proof.

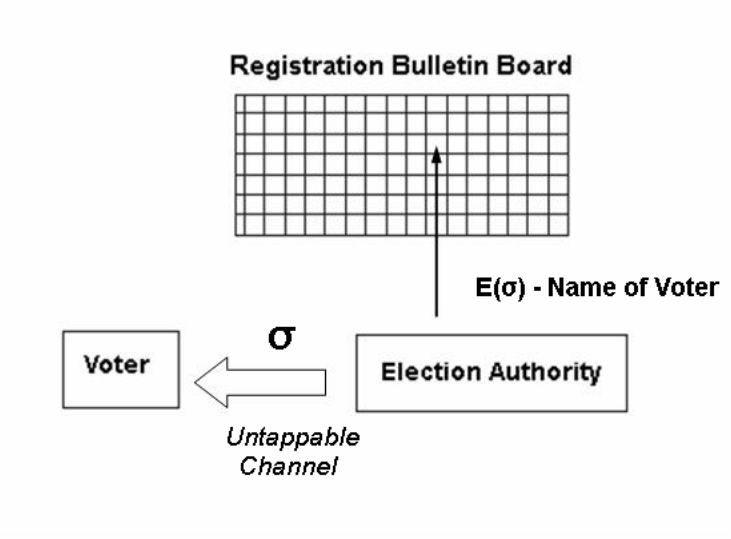

Figure 3.2: Juels et al. Voting Protocol - Registration Phase

In the tallying phase those ballots are processed by

1. excluding ballots with invalid proofs,
2. eliminating ballots containing duplicate credentials via pairwise blind comparisons of encrypted credentials,
3. mixing the ballots as well as the encrypted credentials from the registration list and checking the ballots' credentials via pairwise blind comparisons against this mixed reference list, thus authorizing valid votes for each credential pair found,
4. and finally decrypting and tallying all votes .

Figure 3.3 illustrates the voting and tallying phases. Here $E(v)$ denotes the encrypted vote, $E(\sigma)$ the encrypted credential and ZKP the validity proof.

3.3.3 Checking Credentials blindly

In order to account only valid votes, the protocol realizes the elimination of duplicate ballots, i.e. ballots with same credentials, and authorization of valid ballots via *plaintext equality tests* [JJ00, MSJ02].
A plaintext equality test *(PET)* is a protocol that tests whether two ElGamal

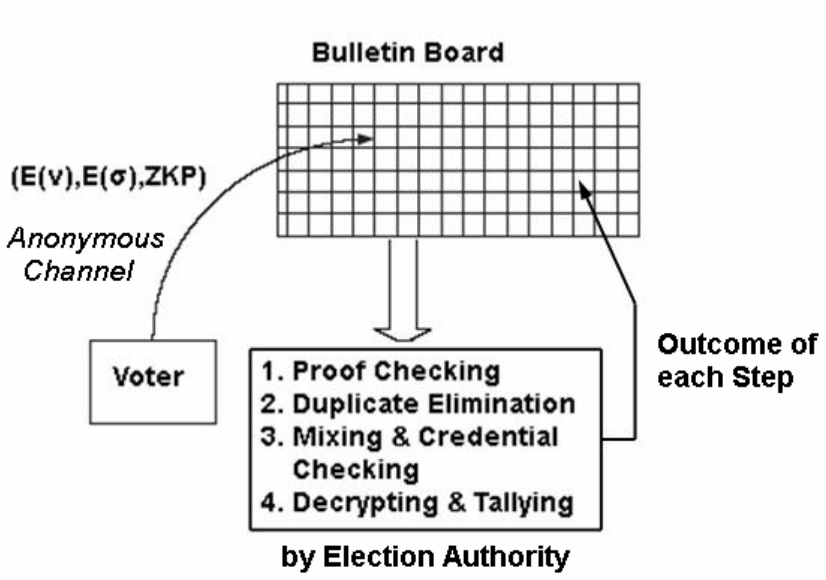

Figure 3.3: Juels et al. Voting Protocol - Voting and Tallying Phases

ciphertexts represent the same plaintext without revealing it[2].
It exploits the following fact:

Let $(x_1, y_1) = (g^{r_1}, h^{r_1} m_1)$ and $(x_2, y_2) = (g^{r_2}, h^{r_2} m_2)$ be two ElGamal ciphertexts with underlying plaintext m_1 and m_2. If m_1 and m_2 are equal, then $(x_3, y_3) = (x_1/x_2, y_1/y_2) = (g^{r_1 - r_2}, h^{r_1 - r_2} m_1/m_2) = (g^{r_3}, h^{r_3} m_3)$ is an encryption of 1, since $m_1/m_2 = m_3 = 1$.

In the *PET* protocol, the ciphertext (x_3, y_3) is first blinded by raising each component to a random exponent $z \in Z_q$, $(x_4, y_4) = (x_3^z, y_3^z) = (g^{r_3 z}, h^{r_3 z} m_3^z)$, and then decrypted to the blinded value, m_3^z.
The decryption yields $m_3^z = 1^z = 1$ if the plaintexts are equal and a random integer $(m_1/m_2)^z$ otherwise, and so only negligible information about the plaintexts is leaking.

The plaintext equality tests realize the *pairwise* blind comparisons of the encrypted credentials in Juels et al.'s scheme. As every ciphertext has to be tested against every other ciphertext, these pairwise checks result in an inefficient scheme, with a work factor quadratic in the number of votes.

[2]The Juels et al. scheme originally employs a *modified* ElGamal cryptosystem. *PET*s can be adapted to other cryptosystems based on the discrete logarithm problem as well.

3.4 Speedup by Hashtable Lookup

In order to improve Juels et al.'s proposal and to overcome the quadratic work factor, Smith proposes to replace both *PET* stages by lookups in a hashtable. We describe Smith' idea next.

A hashtable is a data structure which maps *keys* to *values*. Equivalent values are supposed to have equivalent keys, different values are supposed to have different keys.

For each ballot, in the *duplicate elimination step*, the whole ballot is considered as the value, a *blind deterministic fingerprint* of its associated credential is considered as the key. The ballots are put into a public hashtable consecutively. If a collision is found, i.e. two equal keys, a pair of duplicate ballots is detected. Upon a predetermined policy one of them is eliminated, while the other is stored in the hashtable. Juels et al. propose to use the order of postings on the bulletin board to distinguish between ballots and to select the newer one[3]. Smith proposes to attach timestamps to the ballots for this task.
Figure 3.4 illustrates the duplicate elimination process via hashtable lookup, where $h_k(\sigma_i)$ denotes the fingerprint of a ballot's credential.

In the case of authorisation, i.e. the *credential checking step*, the hashtable is initially filled with the fingerprints of all pre-registered credentials, with no associated values[4]. Then all remaining ballots are put consecutively into the hashtable. Here a collision detects an authorized ballot, because it originates from a valid pre-registered credential.
Figure 3.5 illustrates the ballot authorisation process via hashtable lookup, where $h_j(\sigma_i)$ denotes a fingerprint of a credential.

To enable the hashtable lookup, blind deterministic fingerprints of each non-deterministic encryption $(g^r, h^r\sigma_i)$ of a credential σ_i need to be computed, without violating the privacy of the voters by leaking information about the credentials.

[3] Juels et al. propose this to distinguish between ballots in their *PET* duplicate elimination stage.

[4] It depends on the actual implementation of the hashtable data structure, if the value field *may* be empty. Otherwise, the associated value has no function.

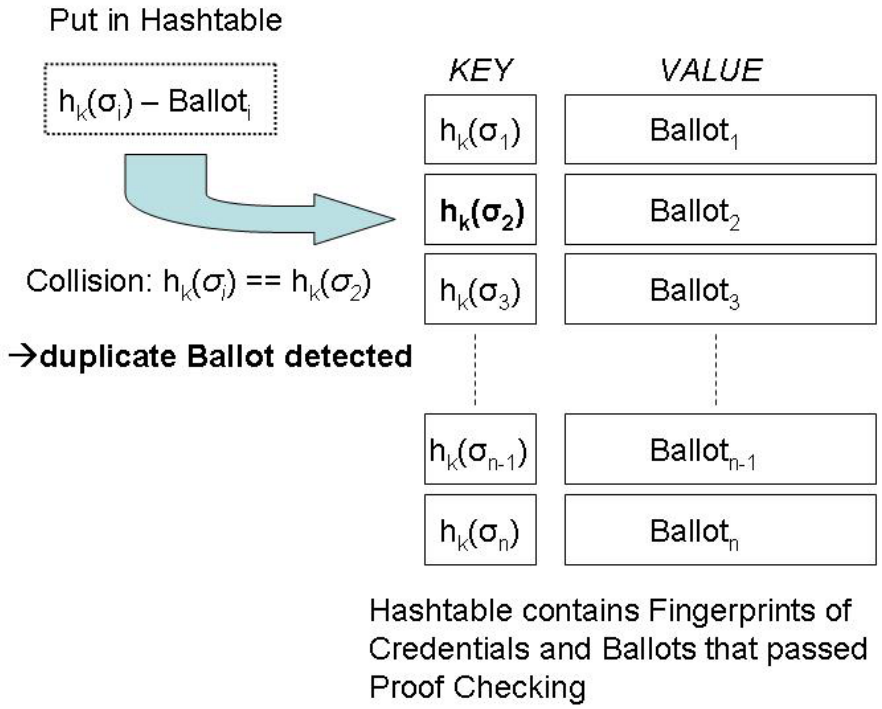

Figure 3.4: Hashtable Lookup - Duplicate Elimination

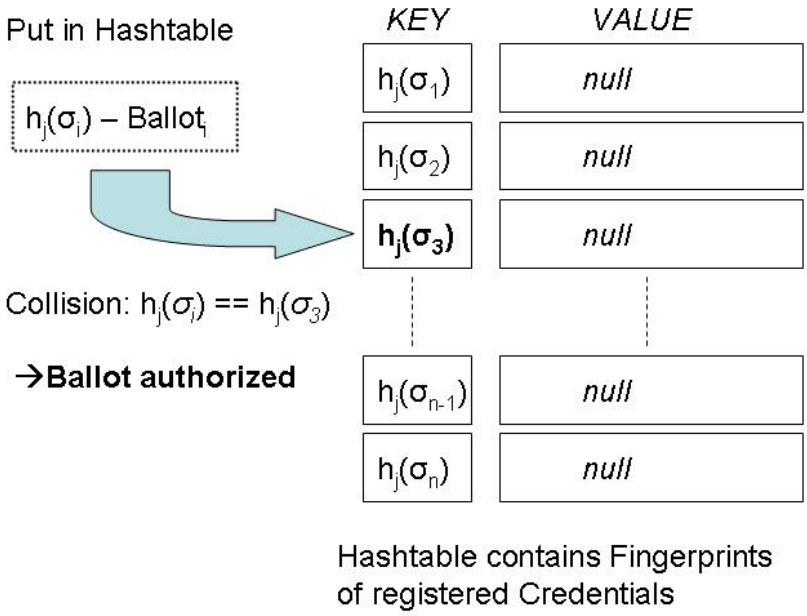

Figure 3.5: Hashtable Lookup - Ballot Authorisation

3.4.1 Computation of Fingerprints

We propose to realize the computation of deterministic fingerprints by the following protocol[5], similar to a plaintext equality test:

1. Each component of $(g^r, h^r\sigma_i)$ is blinded to a fix exponent $z \in Z_q$: $((g^r)^z, (h^r\sigma_i)^z) = (g^{rz}, h^{rz}\sigma_i^z)$.
2. The blinded ciphertext is decrypted to the blinded plaintext σ_i^z.

We denote $h_z(\sigma_i) = \sigma_i^z$, the involved value z is called a *hashkey*. This blinded credential value serves as deterministic fingerprint. Its size depends on the chosen ElGamal parameters.
Smith proposes to use only a subset of σ_i^z's bits as fingerprint. Alternatively a freely chosen deterministic hash function, e.g. SHA-1 or MD-5, can be employed to reduce the size of the fingerprint[6].

A coercer cannot re-do the computation of the blind fingerprints, as he is not in possession of the secret hashkey. Thus the privacy of each voter is protected.

The fingerprints $h_z(\sigma_i)$ enable the hashtable lookup, which replaces both *PET* stages in the original scheme. The runtimes of these hashtable lookups are $O(V)$ and $O(N + V)$ respectively, where V is the number of ballots and N the number of voters. This is a significant speedup compared to the quadratic runtime of the original mechanisms based on pairwise checks, allowing to design an *efficient* coercion-resistant voting protocol.

[5] Compared to Smith' similar proposal the advantage of our solution is that it can easily be extended to a distributed variant. Compare with section 3.6.2.

[6] Again, this depends on the actual implementation of the hashtable data structure. Compare with section 6.3.2.

3.5 The Protocol with a Single Election Authority

Next we present the single election authority variant of the coercion-resistant voting protocol using hashtable lookups to eliminate duplicates and to detect valid ballots.
It is based on Juels et al's scheme and on Smith' "New cryptographic election protocol with best known theoretical properties" [Smi05b]. See appendix A on how our protocol originates from Smith' descriptions.
In difference to Juels et al.'s proposal, the proposed protocol includes an additional mixing step after the verification of zero knowledge proofs. Thus, the initial order of postings on the bulletin board cannot be used to distinguish between later or earlier cast ballots. The ballots include encrypted timestamps instead that are decrypted in the duplicate elimination stage and compared in the case a duplicate ballot is detected. All timestamps are discarded after the duplicate elimination step. This additional mixing step was introduced by Smith to prevent two further attacks, namely the *1009 attack* and the *time-stamping attack*. We discuss these attacks later in section 4.1.5 and section 4.1.7.

The following protocol employs a *single* election authority controlling the whole voting process. For simplicity, we assume that this single authority has all responsibilities of the registrar, the tallier, the officer and the mixer, as specified in section 3.1.1. The verifiable mixnet consists of a single mixserver. The employed cryptosystem is the semantical secure ElGamal cryptosystem as presented in section 2.2.

3.5.1 Protocol Description

1. Setup

First the election authority EA generates the ElGamal system parameters p, q, g, its ElGamal key pair (SK_{EA}, PK_{EA}), and two secret hashkeys, k and j. Then the EA publishes its public key PK_{EA}, together with the system parameters.

2. Registration

Each eligible voter V_i is registered by the election authority, playing the role of the registrar. The EA generates a unique credential $\sigma_i \in_R G_q$ and transmits it to voter V_i over an untappable channel. The EA publishes a list of all registered voters and their encrypted credentials $C_i = E_{PK_{EA}}(\sigma_{R_i})$ on bulletin board BB_1.

Here we denote credentials on the registration list as σ_{R_i}, while the matching credential transmitted to the voter is denoted σ_i.

3. Candidate Slate Publication

The EA publishes an integrity protected list of possible choices in the election, i.e. the competing candidates, and their unique identifiers, and announces the beginning of the voting phase and its designated deadline.

4. Voting

Voters V_i cast their ballots for their chosen candidates by submitting tuples (B_i, P_i, T_i, C_i) to bulletin board BB_2 over an anonymous channel.
B_i is the encrypted vote, P_i is a non-interactive zero knowledge proof. It consists of a validity proof indecomposable[7] anded with a proof of knowledge of the credential. (See section 2.3 for details.) T_i is an encrypted timestamp, C_i is an encryption of the voter's credential σ_i. All encryptions are done under PK_{EA}.
Voters can submit ballots repeatedly, until the deadline of the voting phase is reached.

5. Tallying

To compute the tally the EA performs the following steps to sort out invalid, duplicate and unauthorized votes:

5.1 Verifying proofs The EA verifies the NIZKPs associated with the ballots. Any ballot with a valid proof is posted on BB_3, the P_i datafield of that ballot is discarded.
Ballots with invalid proofs will not be processed furthermore.

5.2 Mixing The ballots on BB_3 are anonymized and re-encrypted by means of a verifiable mixnet. The EA is responsible for running this mixnet. The mixnet permutes and re-encrypts the B_i, T_i and C_i datafields of each ballot. The resulting list of anonymized ballots is posted on BB_4.

[7] Indecomposability is achieved by including bits from the timestamp into the creation of the challenges of both NIZKPs.

5.3 Elimination of duplicates For each ballot on BB_4, the EA computes a deterministic fingerprint $h_k(\sigma_i)$ (according to section 3.4.1) on the encrypted credential C_i, using hashkey k, and decrypts the timestamp T_i. The EA posts these values together with the associated ballots on BB_5.
To detect duplicate ballots, i.e. ballots with the same fingerprint $h_k(\sigma_i)$, the EA performs a hashtable lookup. If a collision is found, i.e. two identical credential fingerprints, only the last-posted ballot according to the decrypted timestamp is stored in the hashtable.
This process retains at most one ballot per credential, only the last-cast ballots stored in the hashtable will be processed further.
The EA posts the final list of ballots with unique credentials on BB_6, after removing the timestamps T_i from each ballot.

5.4 Mixing The EA applies the verifiable mixnet to the list of ballots with unique credentials on BB_6. The mixnet anonymizes the remaining B_i and C_i datafields of each ballot. The output of the mixnet, i.e. a list of permuted and re-encrypted ballots, is posted on BB_7.
The EA runs the encrypted credentials C_i from the list of registered voters on BB_1 through a different verifiable mixnet. The output, unlinkable to BB_1, is posted on BB_8.

5.5 Authorizing votes The EA computes deterministic fingerprints for each encrypted credential associated with a ballot on BB_7, using hashkey j. The fingerprints $h_j(\sigma_i)$ are posted together with the associated ballots on BB_9.
For each re-encrypted credential on BB_8, the EA computes a deterministic fingerprint $h_j(\sigma_{R_i})$, using hashkey j, and posts it on BB_{10}. Bulletin board BB_{10} contains the fingerprints of registered, and thus authorized, voters' credentials.
To authorize the ballots, i.e. to sort out ballots with invalid credentials, the EA puts all fingerprints from BB_{10} into a hashtable.
For each ballot on BB_9, the EA performs a hashtable lookup on the associated fingerprint $h_j(\sigma_i)$. If a collision is found, the ballot is posted on BB_{11} as authorized ballot.

5.6 Decrypting votes For each authorized ballot on BB_{11}, the EA decrypts the B_i datafield, i.e. the encrypted vote, and posts it on BB_{12}.

5.7 Tallying The EA computes the final election result from BB_{12} and publishes the final tally, i.e. a list of candidates together with the number of votes each received, on BB_{13}.

3.5.2 Security and Trust Issues

In the protocol described in the last section the single election authority has to be trusted completely. It is alone responsible for all protocol steps. As it possesses the private key the authority is able to decrypt the sensitive data.
Benaloh and Yung [BY86] argue that such *centralized* election schemes, i.e. those employing a single election authority, do not offer a degree of privacy that is sufficient for practical elections, because votes can be read by the authority. The EA has to be trusted not to violate the protocol.

As further drawback a single trusted authority generates a single point of attack. If an attacker successfully compromises the EA, the privacy of all voters is threatened. In this case the attacker can also try to manipulate the election result.

3.5.3 Verifiability

As described, the scheme employing a single authority is not universally verifiable, but it can be extended to use zero knowledge proof techniques to achieve verifiability of the private computation steps. These proofs, then, show the correct use of the private key (and hashkeys as well), but depend on commitment values generated by the election authority itself, e.g. the public key commits the authority to its private key.
In this case verifiability helps to detect manipulation of the election, but it does not prevent the authority from comprising the privacy of the voters by decrypting sensitive data. Still, the EA must be trustworthy.

A detailed evaluation of the coercion-resistant protocol is given in chapter 4. Although this evaluation treats the multiple authority scheme to be presented next, the line of argumentation holds analogously for the single election scheme in the most parts.

3.6 The Protocol with Multiple Election Authorities

The discussion in the last section shows that a single election authority is a severe drawback in an election scheme. This problem can be addressed by distributing its tasks and responsibilities among a set of authorities. This approach results in a more complex scheme, to be presented next.

We extend the coercion-resistant protocol to multiple election authorities. This can be done in the following ways:

- multiple authorities, each one is responsible for specific steps alone, e.g. there is one registrar, one mixer, one tallier and one officer. We do not discuss this possibility further.
- multiple authorities, all together are responsible for each step. We call this a threshold version, if for any step involving private data t out of n authorities have to participate for a correct operation.
- multiple groups of authorities, each group is responsible for specific steps, e.g. the registration, the mixing, the tallying. This extends the above possibility, but will not be considered further, either.

3.6.1 Protocol - Threshold Version

In this model n election authorities are responsible for the election, of which t authorities need to work correctly. Together they operate the (t, n) threshold ElGamal cryptosystem, as specified in section 2.5, and the verifiable mixnet. Only the operations using a private key are distributed among all authorities and need a reliable combiner, but every authority can play this role. The other deterministic thus verifiable operations are all done by one authority.

3.6.2 Distributed Computation of Fingerprints

One operation that is distributed among the authorities is the computation of the fingerprints of the credentials. The election authorities jointly compute these deterministic fingerprints of each encrypted credential, to enable the hashtable lookups.
Analog to the single authority case, hashkeys are involved. Those are secret shared private keys created in the setup phase via a distributed key generation protocol, as described in section 2.5. Each authority is publicly committed to his share z_j of each hashkey z by a public value $\rho_{z_j} = g^{z_j}$.

Smith does not specify how to *jointly* compute the blinded credentials σ_i^z, but he proposes a solution based on *multiplication of shared secrets*. As he points out [Smi05a] that a main approach for multiplication of shared secrets, that of Gennaro et al. [GRR98], is flawed, and he considers other approaches to be inefficient and "grotesque", we present an approach that is more practical.

As a subprotocol for the computation of fingerprints, we specify a *distributed blinding protocol*, analog to the distributed decryption protocol described in section 2.5.4.

To jointly blind a value $x \in G_q$, the authorities have to execute the following steps:

1. Each authority computes its partial blinding $b_j = x^{z_j}$, using its share z_j of the involved hashkey, and broadcasts it together with a non-interactive zero knowledge proof that

   $$\log_g \rho_{z_j} = \log_x b_j$$

   using a proof of equality of discrete logs as specified in section 2.3.3. This assures that the authority indeed utilized its correct share to produce the partial blinding.

2. For any subset Λ of t authorities with valid zero-knowledge proofs the blinded value x^z can be reconstructed using discrete Lagrange interpolation

   $$x^z = \prod_{j \in \Lambda} b_j^{\lambda_{j,\Lambda}} \bmod p$$

   where

   $$\lambda_{j,\Lambda} = \prod_{l \in \Lambda \setminus \{j\}} \frac{l}{l-j} \bmod q$$

   are the appropriate Lagrange coefficients.

Based on this, we propose the following protocol for the task of jointly producing deterministic fingerprints, analog to the single authority case:

1. To each component of $(g^r, h^r \sigma_i)$ the distributed blinding protocol is applied, blinding it to a fix secret shared exponent $z \in Z_q$: $((g^r)^z, (h^r \sigma_i)^z) = (g^{rz}, h^{rz} \sigma_i^z)$.

2. The blinded ciphertext is jointly decrypted to the blinded plaintext σ_i^z using the distributed decryption protocol as presented in section 2.5.4.

Again $h_z(\sigma_i) = \sigma_i^z$ denotes a fingerprint, produced with a hashkey z, and further deterministic hash functions can be applied to it, to reduce its size.

3.6.3 Protocol Description

We now describe the voting protocol with multiple authorities. It extends the single authority case.

1. Setup

First, the election authorities jointly agree on the ElGamal system parameters p, q, g and jointly generate an ElGamal key pair (SK_{EA}, PK_{EA}) and two secret hashkeys $k,\ j$ via the distributed key generation protocol, according to section 2.5.3. By this the secret keys are (t, n) secret shared among the authorities, which are committed to their shares, as commitment values produced in the protocol runs are published.
Then the EAs publish the public key PK_{EA}, together with the system parameters.

2. Registration

In this phase each eligible voter V_i is registered by the election authorities. The EAs generate a unique credential $\sigma_i \in_R G_q$ and transmit it to voter V_i over an untappable channel. The EAs publish a list of all registered voters and their encrypted credentials $C_{R_i} = E_{PK_{EA}}(\sigma_{R_i})$ on bulletin board BB_1.
Here we denote credentials on the registration list as σ_{R_i}, while the matching credential transmitted to the voter is denoted σ_i.

3. Candidate Slate Publication

The election authorities publish an integrity protected list of possible choices, i.e. the competing candidates and their unique identifiers, and announce the beginning of the voting phase and its designated deadline.

4. Voting

Voters V_i cast their ballots for their chosen candidates by submitting tuples (B_i, P_i, T_i, C_i) to bulletin board BB_2 over an anonymous channel.
B_i is the encrypted vote, P_i is a non-interactive zero-knowledge proof. It consists of a validity proof indecomposable[8] anded with a proof of knowledge of the credential. (See section 2.3 for details.) T_i is an encrypted timestamp, C_i is an encryption of the voter's credential σ_i. All encryptions are done under

[8] Indecomposability is achieved by including bits from the timestamp into the creation of the challenges of both NIZKPs.

PK_{EA}.
Voters can submit ballots repeatedly, until the deadline of the voting phase is reached.

5. Tallying

To tally the votes posted on BB_2, the election authorities perform the following steps to sort out invalid, duplicate and unauthorized votes:

5.1 Verifying proofs The election authorities verify the NIZKPs associated with the ballots. Any ballot with a valid proof is posted on BB_3, the P_i datafield of that ballot is discarded.
Ballots with invalid proofs will not be processed furthermore.

5.2 Mixing The ballots on BB_3 are anonymized and re-encrypted by means of a verifiable mixnet. The election authorities are responsible for running this mixnet. The mixnet permutes and re-encrypts the B_i, T_i and C_i datafields of each ballot. The resulting list of anonymized ballots is posted on BB_4.

5.3 Elimination of duplicates For each ballot on BB_4, the election authorities jointly compute a deterministic fingerprint $h_k(\sigma_i)$ (according to section 3.6.2) on the encrypted credential C_i, using hashkey k, and jointly decrypt the timestamp T_i. These values are posted on BB_5 together with the associated ballots.
To detect duplicate ballots, i.e. ballots with the same fingerprint $h_k(\sigma_i)$, the election authorities perform a hashtable lookup. If a collision is found, i.e. two identical credential fingerprints, only the last-posted ballot according to the decrypted timestamp is stored in the hashtable.
This process retains at most one ballot per credential, only the last-cast ballots stored in the hashtable will be processed further.
The final list of ballots with unique credentials is posted on BB_6, after removing the timestamps T_i from each ballot.

5.4 Mixing The election authorities apply a verifiable mixnet to the list of ballots with unique credentials on BB_6. The mixnet anonymizes the remaining B_i and C_i datafields of each ballot. The output of the mixnet, i.e. a list of permuted and re-encrypted ballots, is posted on BB_7.
The election authorities run the encrypted credentials C_{R_i} from the list of registered voters on BB_1 through a different verifiable mixnet. The output, unlinkable to BB_1, is posted on BB_8.

5.5 Authorizing votes The election authorities jointly compute deterministic fingerprints for each encrypted credential associated with a ballot on BB_7, using hashkey j. The fingerprints $h_j(\sigma_i)$ are posted together with the associated ballots on BB_9.
For each re-encrypted credential on BB_8, the authorities jointly compute a deterministic fingerprint $h_j(\sigma_{R_i})$, using hashkey j, and post it on BB_{10}. Bulletin board BB_{10} contains the fingerprints of the registered, and thus authorized, voters' credentials.
To authorize the ballots, i.e. to sort out ballots with invalid credentials, the election authorities put all credential fingerprints taken from BB_{10} into a hashtable. For each ballot on BB_9, a hashtable lookup on the associated fingerprint $h_j(\sigma_i)$ is performed. If a collision is found, the ballot is posted on BB_{11} as authorized ballot.

5.6 Decrypting votes For each authorized ballot on BB_{11}, the election authorities jointly decrypt the B_i datafield, i.e. the encrypted vote, and post it on BB_{12}.

5.7 Tallying The election authorities jointly compute the final election result from BB_{12} and publish the final tally, i.e. a list of candidates together with the number of votes each received, on BB_{13}.

6. Verification

Verifiers audit the whole election process. They control whether the steps involving no private keys or secret data have been done correctly as any step is documented on a bulletin board and they can re-do any of these computations. For the steps involving private data they verify the correctness of broadcast zero knowledge proofs.

4 Evaluation

The list of flawed approaches to build receipt-free schemes, presented in section 1.4, shows that a proposed protocol has to be thoroughly evaluated to detect its drawbacks and shortcomings.
We evaluate our multiple authority scheme next, contrasting it to Smith' [Smi05b] protocol in some aspects. The line of argumentation holds analogously for the single election scheme in the most parts. First, we analyse the phases of the coercion-resistant voting protocol. Then, we describe which requirements the protocol satisfies and mention further points of interest.

4.1 Analysis of the Protocol

We begin with an analysis of each phase of the multiple authority scheme.

4.1.1 Setup Phase

The distributed key generation protocol as described in section 2.5.3 is universally verifiable [GJKR03], so cheating authorities can be detected. The protocol can tolerate up to $t-1$ dishonest or compromised authorities, but it is reasonable to exclude dishonest, compromised or malfunctioning authorities from the following phases and execute the key generation protocol again, with appropriate substituted or checked authorities respectively servers.

4.1.2 Registration Phase

Comparable to the case of certificate enrollment in a public key infrastructure as described by Straub [Str04], the security of the credentials depends on the registration process. So the correctness and security properties of the whole voting protocol depend on a trustworthy registration.
Each voters' credential must be unique to allow for a correct authorisation and kept secret to prevent coercion against the voter. Credentials are received through an *untappable channel*, as specified in section 3.1.3, thus the absence of an adversary in the whole registration phase is assumed. Practically this can be

realized e.g. through postal mail, comparable to traditional general elections. If an adversary can break the registration phase, the whole security of the election protocol is annulled, as he can issue credentials incorrectly, e.g. issue double credentials, manipulate the registration list or exclude voters completely from the election by not issuing credentials to them. Therefore the registration phase is a critical part of the protocol, and as described by Alvarez [Alv05], it is a critical part of any election.

4.1.3 Candidate Slate Publication

An adversary who can manipulate the candidate slate is able to exclude candidates from the election. Such manipulation is not likely to be successful as it is easy to detect by any party that holds a copy of the candidate slate.

4.1.4 Voting Phase

It is assumed that a voter submits his ballots via an *anonymous channel*, i.e. a channel such that a coercer cannot determine whether a voter actually voted or not. Thus, the assumption is made that a coercer cannot continuously observe the voter. Otherwise the coercer is able to mount a forced-abstention attack against a voter.
The anonymous channel assumption is accomplished by the fact that a voter can use an arbitrary computer[1] connected to the Internet to submit his ballots, and the coercer is unable to observe this.
Here the anonymous channel is not meant to be realized by a mixnet. Juels et al.'s scheme [JCJ05] uses the order of ballots on the bulletin board to distinguish between later and earlier cast ballots. A mixnet that shuffles the order of ballots would negate this possibility.

In order to prevent the number of invalid votes to become large and so to prevent denial of service attacks, Smith proposes that voting should take a certain period of time, e.g. voters should only be able to cast one ballot per minute. This has to be considered on an implementation level.

The ballots include *encrypted* timestamps. Smith' protocol wrongly proposes to use plaintext timestamps. This is not reasonable, as the following mixing stage cannot anonymize the ballots in this case.

[1] We discuss constraints to this in section 4.7.

4.1.5 Tallying Phase: Proof Checking

The non-interactive zero knowledge proofs associated with the ballots have two functions. First, the *validity proof* assures that no invalid candidate choices will be found in the final tally. Without this mechanism a voter can be forced to submit an observable invalid choice, which deprives him of the possibility to vote freely. Second, the *proof of knowledge* assures that it is not possible to validly copy an encrypted credential from the bulletin board, re-encrypt and include it in a new ballot, as the proof becomes invalid then. This is comparable to the concept of non-malleable ElGamal encryption [TY98]. Additionally, proofs of knowledge can be included in the ballots for any encrypted datafield.
Both proofs are indecomposable anded, i.e. it is impossible to reuse only one of them to construct a new ballot. Only a complete and unchanged ballot can be copied and validly injected to the protocol, but that does not affect the result, as it will be detected as duplicate and eliminated.

The proof checking does *not* detect if a credential is valid. This detail was mistaken by Acquisti [Acq04] and Smith as well. In a discussion of Juels et al.'s scheme Acquisti argues that a coercer can force a voter to cast a large number x of ballots containing the same *valid* credential. In the duplicate elimination stage the coercer can observe if $x - 1$ ballots containing the same credential are discarded. Acquisti argues that now a coercer can identify the remaining ballot to contain a valid credential, because he erroneously mistakes the proofs checked before to be proofs of correctness of the credentials. In that case only ballots containing valid credentials could have passed the proof checking and his attack could succeed.
Smith mentions the same weakness in Juels et al.'s scheme, and describes it as the *1009 attack*. He draws the same erroneous conclusions, even though he gives no convincing explanations on the attack. A coercer has no reason to believe that the remaining one ballot has a valid credential, when he notices the elimination of 1008 ballots. Smith includes an additional initial mixing stage to his protocol to prevent the 1009 attack.

4.1.6 Tallying Phase: Initial Mixing

This mixing stage is introduced by Smith to counter the 1009 attack. But the protocol is unaffected by this attack, even without this additional mixing, as explained above.
Thus it can be skipped, to design a more efficient protocol. If this mixing stage is skipped, the protocol can even abandon timestamps, because then the order of posted ballots on the bulletin board can be used to distinguish between ballots.

4.1.7 Tallying Phase: Duplicate Elimination

The correctness of the duplicate elimination mechanism via hashtable lookup depends on the collision resistance of the employed fingerprints. Different credential values having the same fingerprints would seriously divert the protocol mechanisms by eliminating arbitrary votes.
For each unique credential value $\sigma \in G_q$ the blinded credential $\sigma^z \in G_q$ for a fix $z \in Z_q$ is itself the fingerprint or respectively the basis for the fingerprint.
The modular exponentiation is injective for appropriate choices of $z \in Z_q$. Using $z = 0$ the mapping is clearly not injective, for $z = 1$ the credential plaintext is leaking. For $z \in \{2, .., q-1\}$ each $\sigma^z \in G_q$ is unique and therefore a *collision-free* fingerprint[2], and the credential remains secret. Thus the duplicate elimination is correct, if the complete σ^z is used as fingerprint.
If an additional hash function is used to reduce the size of the fingerprint the correctness of the duplicate elimination depends on the practical collision resistance of the hash function.
In the setup phase, after the execution of the distributed key generation protocol, inappropriate values for a hashkey z can be detected, as the created public key[3] is $h = g^0 = 1$ or $h = g^1 = g$ in this case, so the protocol can be re-run.

After the duplicate elimination timestamps have to be removed from the ballots, to avoid the *timestamping attack*, mentioned by Smith. If a ballot contains a timestamp after it is authorized, it can be identified by remembering the exact time it was cast and thus be sold[4]. So timestamps are discarded before the validity of credentials is determined.

4.1.8 Tallying Phase: 2nd Mixing

The mixing of the registration list allows for an anonymous authorisation. The resulting list of encrypted credential is unlinkable to the registration list, so the sender of ballots with valid credentials cannot be deduced in the next step.

The protocol has to include *one* mixing of cast ballots before the authorisation step, so that it cannot be deduced which cast ballots will be included in the final tally.

[2]Collision freeness in Z_q follows from the theorem: Let $g \in G_q$, $g \neq 1$, and x, y integers: $g^x = g^y \Leftrightarrow x = y \bmod order(g)$. As $order(g) = q$ in this case, there are no collisions for $x, y \in Z_q$. See Buchmann [Buc04] for further mathematical details.
[3]The public key is created in the key generation protocol, but has no function beyond this.
[4]Is is unclear, why Smith mentions the timestamping attack as weakness of Juels et al.'s scheme. It does *not* employ timestamps.

4.1.9 Tallying Phase: Credential Checking

Analog to duplicate elimination the correctness of the authorisation mechanism depends on the collision resistance of the employed fingerprints. As argued above a *collision-free* implementation is possible.
This step sorts out ballots containing invalid credentials (with respects to the collision-resistance of the fingerprints), at most one ballot per each pre-registered credential will be considered in the final tally.

As the hashtable lookup is performed on blinded credentials and an adversary cannot re-do the blinding, it is not perceivable (together with the former mixing) which ballots are authorized.

4.1.10 Tallying Phase: Tallying

The joint decryption protocol can tolerate up to $t-1$ corrupt authorities. Violation of the decryption protocol can be detected as zero knowledge proofs are broadcast. And the final election result can be reproduced by any verifier.

Basing on this analysis, we evaluate in the next sections which security requirements the protocol satisfies and mention further important aspects.

4.2 Correctness

The analysis of the phases shows that the protocol is correct with respects to collision-free fingerprints. The protocol assures that only the last-cast authorized vote of a pre-registered voter is counted in the final tally.
Even thought we expect no incorrect processed ballots if a practical collision-resistant hash function, e.g. SHA-1, is used to decrease the size of the fingerprints[5], we stress that Smith' original proposal is not correct, as stated by him, in the sense that collisions can occur.

4.3 Robustness

Due to the use of threshold cryptography (see section 2.5.1) the protocol can tolerate a minority of cheating, dishonest or compromised parties.
The line of argumentation for robustness holds analogously to the trust analysis in the following section.

[5]Compare to section 6.3.2.

4.4 Trust Requirements

In this section we list up the trust requirements, i.e. which participants of the election protocol need to trustworthy. Each listed trust requirement can be a risk, that has to be dealt with, as Ferguson and Schneier [FS03] emphasise, when a secure implementation and operation of a complete system shall be accomplished.

We identified the following trust requirements in the coercion-resistant voting protocol:

- The *election authorities* respectively the *talliers* have to be trusted not to do further steps beyond those specified in the protocol, e.g. to decrypt votes and credentials. At least t out of n authorities are required to execute steps involving shared keys, a minority of dishonest or compromised authorities can be tolerated. In comparison to the protocol with a single election authority the trust is distributed among the set of all authorities.

- The *registrars* have to be trusted to enforce a correct registration phase.

- The *officers* have to be trusted to operate the bulletin boards correctly. A reliable communication subsystem is crucial for the whole voting protocol and the audit process.

- The *mixnets* guarantee privacy if at least one of the mixers is honest, thus at least one mixer has to be trusted. Violations of the verifiable mixnet protocol will be detected, so mixers cannot replace or alter processed ballots.

- The *voter* does not have to be trusted. The protocol assorts invalid ballots, so the voter cannot manipulate the election result.

- In an implementation the *voter client* has to be trusted to encode manually entered votes and credentials correctly.
  If a timestamp has to be included in the ballot, a trusted *timestamping service* is required.

- The *verifiers* have to be trusted to correctly verify the election process. As a large number of independent parties can take over this role the required trust is minimal.

4.5 Verifiability

The protocol is *universal verifiable* in the bulletin board model, defined in section 2.7. Each protocol step of the *tallying phase* can be verified by either checking the broadcast zero knowledge proofs stored on bulletin boards in case secret inputs are involved, e.g. private keys or secret permutations in the mixing, or by re-performing the deterministic steps. In the setup phase the distributed key generation is also universal verifiable, since verification data is published.

Through universal verifiability the correctness of a voting scheme is transparent to external verifiers. For voting systems to be broadly accepted, Jones [Jon04a, Jon04b] demands that such audit mechanisms have to be sufficiently simple so that large groups of verifiers can actually perform the verification.
For the proposed protocol, this simplicity requirement is not achieved. A deep understanding of advanced cryptographic techniques is required to comprehend the verifiability. Thus, large groups of voters will not be able to comprehend the verification mechanisms.

Verifiability minimizes the amount of trust required. Despite the offered universal verifiability, there is still an "audit gap" [DLD03] concerning the client software, performing the cryptographic operations for the voter.
It can encode and submit altered candidate choices or credentials, and the voter will not notice this. Such a malfunction of the client can be caused by programming errors, malware like trojans and viruses or by malicious intent of the developer of the software. In section 5.1 we propose a voter verification mechanism to lower the required trust in the client.

4.6 Dealing with Coercion

In this section we analyse the mechanisms of the protocol to protect voters against coercion.
The coercion resistance property as defined by Juels et al., see section 1.2, depends on the uncertainty in the coercer's view of the voters' participation in an election. Coercion resistance is satisfied, if the coercer cannot verify the voters' true behaviour. In the case that a coercer knows how all but one voters will vote, coercion against the remaining voter is unavoidable.
The coercer's inability to verify the voters' behaviour mainly bases on the fact that he is not able to distinguish between fake and valid credentials. Furthermore, the coercer cannot decrypt the registration list and is absent in the registration phase. It is assumed that a coercer can only corrupt up to $t-1$ election authorities, otherwise he could decrypt the list of registered credentials.

Coercion resistance implies receipt freeness, which implies privacy [DKR06]. Unlike in the single authority variant, privacy is achieved due to the distribution of power between the election authorities. Only a majority of at least t authorities can decrypt sensitive data and thus violate the privacy of the voters.

4.6.1 Receipt Freeness

As specified in section 1.2, receipt freeness means that a voter cannot create a *receipt*, i.e. any information that can be used to convince a coercer that he voted in a specific manner. In the coercion-resistant scheme the voter cannot construct a receipt because he cannot prove that his ballot contains a valid credential.
The approach of Juels et al. as adopted in our coercion-resistant protocol, i.e. the use of encrypted credentials, is different to other receipt-free protocols, e.g. [HS00, BFP$^+$01, LK02]. These protocols assure that the voter is not in possession of the random value used in the encryption of the vote, as it is generated by a trusted hardware device or communicated through untappable channels from authorities to voter. In our scheme the random value is known to the voter and he can visibly re-encrypt his ballot, but this information does not indicate that the ballot contains a valid credential. Thus a voter cannot sell his vote[6] and cannot be forced to vote in a certain away.

4.6.2 Coercion Resistance

Coercion resistance extends receipt freeness upon resistance against further possible attacks. In this attack model, the coercer is able to interact with the voter after the registration phase, as stated by Delaune et al. [DKR06], but cannot continuously observe the voter. This isolation of the voter from the coercer is modelled by the anonymous channel.

Coercion resistance prevents the following additional attacks:

- *randomization attack:* The coercer forces the voter to submit a random string as his vote. In the coercion-resistant protocol, such a vote will never be decrypted, as it will be discarded in the proof checking stage, as it does not represent a valid candidate choice.

- *forced abstention attack:* In this attack, the voter is forced not to vote. As the voter can submit his vote via an anonymous channel

[6] The case of cellphone vote-buying [Sha05] illustrates the meaning of a receipt in real world elections. This attack, in which voters record their voting procedure by the internal camera of their cellphone, is prevented by the protocol as well.

and authentication data is hidden during the whole run of the protocol, a coercer cannot deduce a voter participating in the election and so he cannot mount this attack.

– *simulation attack:* In this attack, the coercer forces the voter to give away his credential to vote on his behalf. In the protocol, the voter can give away a fake credential to a coercer trying to impersonate him. The coercer cannot verify if this credential is indeed valid. A protocol run does not allow him to see if his cast ballot is included in the final tally, as mixing stages unlink the authorisation from incoming ballots and hashtable lookups are only performed on blinded credential values, where the hashkey is unknown to the coercer.

To further dissociate the coercer's view of the voters' behavior, Juels et al. propose that the election authorities inject additional invalid ballots to the system in order to conceal the actual number of invalid ballots.

Coercion resistance excludes voting methods that allow to make a ballot unique, e.g. write-in votes or ranked voting methods, in which the low ordered rankings create a covert channel [CM05], as unique votes can serve as receipts.
The applicability of a certain voting method depends on he availability of zero knowledge validity proofs of votes, since it has to be assured that votes encrypt a valid candidate. A wide range of validity proofs is described by Groth [Gro05].

4.6.3 Shoulder Surfing

Shoulder surfing describes the situation in which a coercer *can be* physically present at the moment the voter is casting his ballot. He can "look over the voter's shoulder" and see on the monitor the voter's choice of candidate. In this way, a shoulder surfer can coerce the voter to cast a particular vote.
This kind of coercive attack is not considered in the attack model of coercion resistance, which assumes remote coercion, i.e. voters are able to cast their ballots anonymously. Nevertheless, the protocol offers countermeasures against this form of coercion. A countermeasure is the fact that the voter may simply vote for a requested candidate or a random choice, if he does not want to disclose his true intent, using a fake credential. It is important that the shoulder surfer cannot perceive that a fake credential was used. Later, the voter can vote without the shoulder surfer present. Then he can submit a ballot containing his truly intended choice of candidate together with his valid credential.

The discussions above show that a voter should always use an invalid credential under coercion. If he e.g. gives away a valid credential to a coercer, he can still try to vote at the end of the voting phase to cancel the coercer's previous cast ballot, but in this case he can never be sure if the coercer did not cast a ballot

after him. This is not the correct usage of the mechanisms to protect voters against coercion.

4.7 Reliable Storage of Credentials

The protocol's countermeasures against coercion, as discussed in the last section, rely on the possibility to use and give away fake credentials to deceive the coercer. Neither Juels et al. nor Smith mention the point of how to store and represent the credential on the voter's side. In this section this important issue is discussed.
Each credential is a unique element $\sigma \in G_q$. For realistic security parameters, e.g. 1024 bit security for ElGamal encryption, each credential is a *very long* bit string. Upon receiving it in the registration phase, the voter has to store it for later use. The storage media must support to give away and to release fake credentials. The possibilities to reliably store credentials and their drawbacks are listed up next[7]:

- *in voter's brain:* Voters are unable to remember a credential of appropriate length reliably.
- *printed on paper:* Voters have difficulties to enter the credential manually, without typing errors[8].
- *on voter's computer:* Voting is restricted to one computer. To realize voting without observation, an arbitrary computer must be able to be used. Others, e.g. persons, trojans and viruses, have to be prevented from accessing it.
- *on smart cards:* Smart cards are expensive and can be stolen or forced to be given away. So multiple smarts cards must be available for each voter, otherwise a forced abstention attack is possible.
  A further drawback is that smart card readers are not available on arbitrary computers.
- *on hardware tokens, e.g. USB sticks:* A voter can have multiple cheap USB sticks. USB slots are available on any arbitrary computer.
  This can be combined with a *2-factor-authentication*, i.e. additionally a PIN has to be entered to access the credential.

[7] A general discussion of how to store secrets reliably is given by Ferguson and Schneier [FS03].
[8] A recent remote voting pilot project [Scy03] showed that voters had difficulties in entering their credential manually, even though the credential consisted of only 16 characters.

– *on a web server:* Credentials are stored on a web server under pseudonyms and protected with passwords or passphrases. Each voter can possess multiple pseudonyms and access the service from any computer connected to the Internet. Additional trust in the service of the web server is required.

It is not immediately clear which is the best solution to store credentials. The use of simple hardware tokens or the use of a web server seem to be the most appropriate.

If a single smart card per voter is used, which is a further realistic scenario, a revocation mechanism is required, to allow for a quick revocation of given away cards. In this case further questions arise, of how to get a new smart card and how to communicate in this process. Analog to registration, an untappable channel between a voter and the registrars is required.

4.7.1 Usability vs. Protection against Shoulder Surfing

If protection against shoulder surfing is considered, a usability problem concerning the release of invalid credentials arises. In this case, the coercer or shoulder surfer actually sees the voter casting his ballot. But the coercer or shoulder surfer may not be able to perceive that an invalid credential is submitted.
This can either be achieved by releasing a random fake credential on input of invalid authentication data to the webserver or hardware token, or by accessing a dedicated additional but invalid credential.
In the first case, the voter never receives a feedback that he correctly entered his PIN or password. This limits the usability, as voters can never be sure of having entered the correct code. In the second case, voters have to remember at least two authentication data sets.

4.8 Practicality

The assumptions made in the voting scheme can be practically realized. The *anonymous channel* assumption is satisfied by the use of arbitrary computers, *untappable registration phases* are common in general elections, but realized as offline procedures, e.g. they rely on postal mail. The storage of credentials can be accomplished with common solutions, as discussed in section 4.7.

The system design is complex, due to the use of threshold cryptography and multiple election authorities. In chapter 6 we show that the cryptographic algorithms can indeed be implemented.

4.9 Efficiency

This section sums up efficiency considerations. In the following, all runtimes are assumed for a fixed number of authorities. For the mixnet we assume a linear runtime[9].

Smith states a work factor of $O(V + N)$ for his protocol, to process V votes sent by N voters. The same holds for the coercion-resistant protocol described in the section 3.6. The hashtable lookup mechanism allows to design a protocol that overcomes the quadratic runtime of Juels et al.'s approach. The amount of storage and verification of non-interactive zero knowledge proofs, broadcast by the election authorities in each distributed computation step, reduces to $O(V+N)$ as well. The credential checking step is the most complex step in the coercion-resistant protocol. It consists of putting in a hashtable N registered credentials, followed by a lookup on at most V unique votes.
Smith' $O(N + V)$ statement can be simplified. If we assume that $N \in O(V)$, the work factor of the protocol can be expressed as $O(V)$. Thus, the runtime of the coercion-resistant protocol is linear in the number of votes[10].

The use of mixnets in voting protocols is generally a little efficient approach. Its disadvantages are listed up by Smith [Smi05a]. We discuss efficiency improvements to our coercion-resistant protocol in section 5.3 and describe how homomorphic encryption mechanisms can be applied to it in section 5.4.

4.10 Not Considered

We shortly list up further points of interest that are not considered in our work and in the evaluation.

4.10.1 Applicable Mixnets

We did not specify a certain verifiable mixnet to be used in the coercion-resistant protocol. Juels et al. propose to use those of Furukaw and Sako [FS01] and Neff [Nef01]. Worth evaluating seems also the *randomized partial checking* approach of Jakobsson, Juels and Rivest [JJR02]. Its security in the context of a voting protocol is analyzed in [GKK03]. Smith [Smi05b] specifies a conceptual simple mixnet for his scheme. We prefer the above mentioned approaches, which are better evaluated.
Optimistic mixnets [GZB$^+$02, Wik02, Wik03, Wik05, AI03] are not applicable

[9] See e.g. [BG02] for a comparison of the runtimes of some verifiable mixnets.
[10] See section 6.3.1 for actual runtimes drawn from a prototype implementation.

directly because they employ a double encryption, which contradicts with the computation on encrypted data.

4.10.2 Internet Voting Security Concerns

We did not discuss in detail general threats against voting systems. If a complete voting system is evaluated on an implementation level instead of on the abstract protocol level, various further vulnerabilities have to be handled, e.g. viruses, trojans, attacks on the communication infrastructure like denial of service attacks and domain name service attacks. For discussions see e.g. [Rub02, Sch04, JRSW04, ALBD04b, KSW05, Cyb02, GA03, Mas04, RSBA05, CTL05]. Countermeasures against these threats can be both technical and procedural. For example, one possibility to overcome denial of service attacks is to have voting phases last a sufficient period of time to allow to recover the voting system in case of an attack.

5 Extensions, Improvements and Variants

In this chapter we propose several extensions, improvements and variants to the coercion-resistant protocol presented in section 3.6. They can be applied to the single authority variant described in section 3.5 as well.
First, we propose a mechanism to realize voter verifiability. Then we describe how to avoid the multiple cast possibility in the coercion-resistant scheme, followed by several ideas to improve its efficiency. As well, we describe how to apply the concept of homomorphic encryption to our scheme. Building on this, we sketch a variant that does not disclose the final tally of all votes, but only if the tally belongs to a set of predefined values, e.g. if it exceeds a certain threshold.

5.1 Extension to Voter Verifiability

In this section we propose a mechanism to realize *individual* or *voter verifiability*[1]. By this, each voter is enabled to verify that his last-cast valid ballot is included in the final tally. This allows the voter to control if the client correctly encoded his vote. However, the difficulty is to design this mechanism without violating coercion resistance and receipt freeness properties, i.e. it may not be used to sell a vote or expose the voter to coercive attacks.
After the election has finished, each voter's encrypted credential can serve as a voter-verifiable receipt. In order to verify that his vote was correctly counted, the voter can send his encrypted credential over a two-way channel to the election authorities. The election authorities process it by computing its fingerprint and can therefore link it to a ballot by a hashtable lookup, on the hashtable used in protocol step 5.5. This hashtable has to store the authorized ballots, as extension to the basic protocol. The authorities then jointly decrypt that ballot's B_i datafield and send it to the voter. He can now verify which vote associated with his submitted credential was included in the final tally.
As this mechanism may not produce a receipt that is convincing to a coercer, each voter must be able to deceive the coercer about his voter-verifiable receipt,

[1] Juels et al. [JCJ05] proposed such an extension to their scheme as further work.

analog to the case of credentials when casting a ballot. Therefore, a voter must be able to construct a fake receipt that nevertheless receives a valid response from the authorities. To realize this the authorities have to offer a service that issues fake receipts for any possible requested vote. If a voter is coerced to give away his receipt showing that he voted for a specific candidate, he can give such a fake receipt to the coercer.
In the case a coercer wants to verify his cast ballot (which contained a demanded thus invalid credential), the election authorities send a response that does not disclose that the ballot was invalid. If the hashtable lookup on the authorized votes does not produce a collision, the authorities perform a hashtable lookup on the hashtable used in protocol step 5.3, that stores the unique but not yet authorized ballots. They respond the verification request with the decrypted vote. So the coercer receives his last-cast vote for the specific credential and has not reason to believe that the voter gave him an invalid credential.
Again trust in the voter's client software is required to encode verification requests correctly. As the aim of this voter verification service is to overcome the trust in the client, it has to be available employing *trusted* devices[2], e.g. in special verification boothes supervised by the election authorities. These provide the two-way communication channel as well.

Further concepts for voter verifiability are e.g. Chaum's mechanism [Cha04] based on visual cryptography, and Dall'Olio and Markowitch's scheme [DM04] based on designated verifier signatures.

5.2 Variants without Multiple Cast Possibility

The protocol proposed in section 3.6 allows the voter to submit more than one valid ballot, and therefore to change his mind and vote for a different candidate. Only the last-cast ballot, according to a timestamp or the order of postings on the bulletin board, is included in the final result.
This multiple cast possibility is *not* a countermeasure against coercion and shoulder surfing. Both can be avoided, as explained in section 4.6, by using invalid credentials. Due to the indirect authentication mechanism, there is no way to keep a voter from voting multiple times. The consequences of multiple cast approaches are discussed in [VG06].
We disencourage the approach to allow multiple casts. It is neither common in democratic elections to be able to change a once submitted choice nor common in cryptographic voting protocols, and it does not lower the security of the protocol if this possibility is prevented.

[2]Solutions could also be based on *Trusted Computing* concepts, see e.g. www.trustedcomputinggroup.org. A discussion of this is beyond the scope of this thesis.

We propose two protocol variants that exclude the multiple cast possibility:

1. The protocol can simply be modified to select the ballots with the oldest timestamp in the duplicate elimination stage. Repeatedly cast valid ballots will simply be discarded then.

2. A more rigorous approach, also mentioned by Smith [Smi05b], is to eliminate *all* duplicate ballots on detection in the duplicate elimination stage. If m equal fingerprints are detected, then all m ballots are discarded, not just $m - 1$. Thus, this variant does not need timestamps to distinguish between the ballots, so a trusted time-stamping service is not required.
   Voters still have the possibility to invalidate their choice by cas-ting a second ballot containing a valid credential, but they cannot change their vote once the first valid ballot is submitted.

The second variant is vulnerable to a ballot copying attack. If a ballot submitted to the bulletin board is copied and sent again, this will nullify both ballots. An adversary can mount this attack to completely nullify the whole election.
It is possible to counter this attack by comparing all ballots on the bulletin board after the voting phase has ended. If identical ballots are found, all but one are discarded, as they can only originate from such a copy attack. It does not matter which ballot is kept, as they have identical content. The comparison can be realized again by a hashtable lookup.
An adversary cannot undermine this security mechanism by constructing a related and thus not identical ballot based on a re-encrypted credential copied from the bulletin board, as he cannot produce a valid proof of knowledge for the re-encrypted credential.

5.3 Efficiency Improvements

In this section we propose further ideas to speedup the scheme:

- The processing of registered credentials (including the mixing) in the tallying phase can be relocated to a pre-processing stage after the registration has ended and all credentials are available. In this pre-processing stage a hashtable is initially filled with blind fingerprints of registered credentials. In this way the actual runtime of the tallying phase can be reduced.

- If the scheme is implemented and actually uses timestamps, the protocol step 5.3 can be changed, so that timestamps are only decrypted in case of a collision. So the decryption of all timestamps can by avoided.

– The protocol can be simplified by using only *one* hashtable lookup to realize both the duplicate elimination and authorisation step at once. Only one secret shared hashkey is required then. We sketch this approach:
In a pre-processing stage the hashtable is initially filled with blind fingerprints of registered credentials. Then voters submit their ballots, including encrypted timestamps. After proof checking and mixing, fingerprints with associated ballots are put into the hashtable consecutively. A collision detects a candidate for an authorized ballot, which is then stored in the hashtable. If more than one candidate ballot per fingerprint is detected, timestamps are compared to determine the last-cast among them.
A drawback of this approach is that timestamps may leak information about authorized ballots. To overcome this, rough granular timestamps have to be employed, or a timestamp-free variant without multiple cast possibility can be designed. Alternatively a blind comparison of timestamps can be devised, e.g. based on variants of protocols solving Yao's millionaire problem[3] [Yao82].

In the next section, we describe a further efficiency inprovement to our scheme, based on homomorphic encryption.

5.4 Integrating Homomorphic Encryption

This section describes how to apply homomorphic encryption to our scheme in order to make it more efficient. First, we describe how this mechanism is employed to design efficient voting protocols. Then, we show how to integrate it into our coercion-resistant scheme. Finally, we describe how this allows to construct a minimal disclosure variant of the protocol.

5.4.1 Homomorphic Encryption Based Voting Protocols

An efficient approach to design election protocols is based on homomorphic encryption. These exploit that for certain non-deterministic encryption schemes the following holds[4], for some operations $\otimes, \oplus$:

$$E(m_1) \otimes E(m_2) = E(m_1 \oplus m_2)$$

[3] In this classical problem, two millionaires want to know who is richer, but without revealing their actual wealth to the other. This can be solved with *secure multi-party computation* techniques. More on this topic can be found in appendix A

[4] Next $E(m)$ denotes the encryption of a message m.

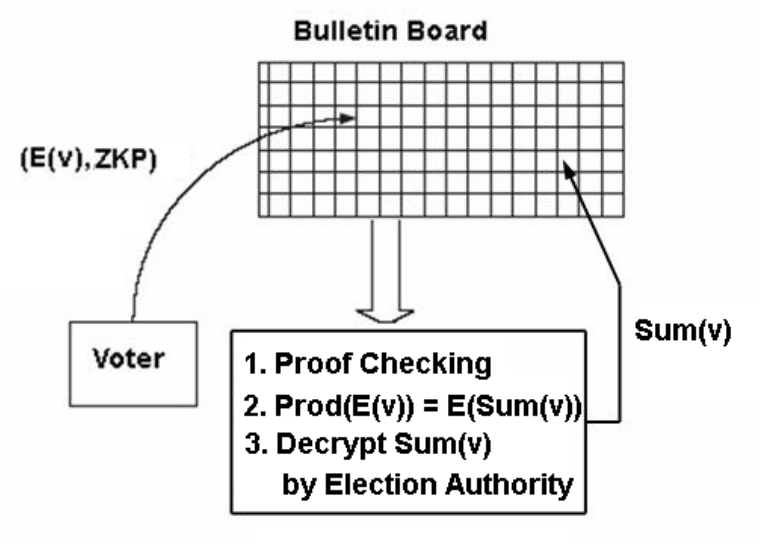

Figure 5.1: Homomorphic Voting Protocol. Source: modified from [SP06]

This allows to compute the tally of an election, without decrypting single votes, if $\otimes$ and $\oplus$ are actually multiplication and addition.

Figure 5.1 illustrates the basic concept of homomorphic election schemes. In a simple homorphic based protocol, each voter submits his encrypted vote (denoted $E(v)$) together with a non-interactive zero knowledge validity proof (denoted ZKP), assuring the vote is a correctly encoded choice of a candidate, to his designated section on a bulletin board. The encrypted votes with valid proofs are then multiplied (denoted $Prod(E(v)) = E(Sum(v))$), to produce the encryption of the result, which is then decrypted (denoted $Sum(v)$) and posted on a bulletin board.

Voting protocols based on homomorphic encryption are e.g. [CGS97, CFSY96], receipt-free variants are e.g. [HS00, BFP$^+$01]. For more general details on homomorphic cryptosystems see Rappe [Rap04].

The homomorphic approach using multiple election authorities requires only *one* distributed computation in the tally phase, the joint decryption of the product of the encryptions. In comparison to our coercion-resistant scheme, the storage and verification amounts are much lower: only one NIZKP per authority has to be verified.

5.4.2 Variant with Homomorphic Tally

The homomorphic calculation of the result as sketched in the last section can be integrated into our scheme in order to replace the individual decryption of each authorized ballot at the end of the tally phase, respectively a homomorphic election protocol can be made coercion-resistant. This section describes this approach.

To enable homomorphic calculation of the election result in the coercion-resistant scheme, the following encoding of messages respective candidates [KY04] is applied, again describing a 1-out-of-c election.
In this variant the set of candidates is encoded as $\{1, M, M^2, ..., M^{c-1}\}$, where c is the number of candidates, M an integer with $M > n$, and n is the number of voters, and all M^{c_i}'s are $\in Z_q$. A voter ElGamal encrypts his choice of candidate l_i as $E(g^{M^{l_i-1}})$, where $l_i \in \{1, ..., c\}$, and $E(m)$ denotes ElGamal encryption of a message $m \in G_q$ under the public key of the EAs. Note that $g^{M^{l_i-1}} \in G_q$ is the subgroup encoding of $M^{l_i-1} \in Z_q$.
The protocol proceeds as described before. Voters submit their ballots including credentials, timestamps, both encrypted as before, and zero knowledge proofs, over an anonymous channel to a bulletin board without designated sections. The validity proof of the basic protocol is applicable, as it is relative to the encoding of votes.
After the authorisation step, i.e. the credential checking, instead of decrypting each single vote, the votes are multiplied, which yields an encryption of g^T, with $T = \sum M^{l_i}$, because $g^x * g^y = g^{x+y} \bmod p$, for $x, y \in Z_q,\ g \in G_q$.
After the joint decryption the value of T can be determined, e.g. using Shanks' baby-step giant-step algorithm.[5] If $k_0, .., k_{c-1}$ are the numbers of votes received by each of the c candidates, it holds that $T = k_0 + k_1 M + ... + k_{c-1} M^{c-1}$, and $k_0, .., k_{c-1} < M$. If T is written as an integer in the base M the result for each candidate can be obtained as the digits of this number.

The drawback of this approach is the need to compute a discrete logarithm, which is a hard task in general. See [CGS97] for a discussion of this. Kiayias and Yung [KY02] note that the searching task in Shanks' algorithm can be partitioned among multiple election authorities.
In order to make this variant more efficient for large values of T, an implementation can be based on the Paillier cryptosystem [Pai99][6], which is homomorphic with respects to addition without the need to compute a discrete logarithm. Threshold versions are described in [FPS00, DJ01, DJN03]. Alternatively, multiplicative homomorphic concepts, as described by Peng et al. [PAB+04], can be explored.

[5] See Buchmann [Buc04] for details on algorithms to compute discrete logarithms.
[6] Under the proviso that an appropriate blinding mechanism can be devised in the Paillier setting as well.

As described above, the tallying step in the basic protocol can be replaced with a homomorphic calculation of the election result. The ballot processing via mixnets, zero knowledge validity proofs, encrypted credentials and respectively the hashtable lookup is an universal building block to achieve coercion resistance. It can be combined with the efficient concept of homomorphic calculation of the election results.

5.4.3 Minimal Disclosure Variant

The homomorphic variant presented in the last section can be extended to construct a coercion-resistant voting scheme, in which the final result only discloses if the tally exceeds a certain threshold or not, e.g. if the majority of voters agreed on one choice. We denote this our *minimal disclosure variant* of the protocol. This approach assures additional privacy, desirable e.g. for Yes/No elections involving small groups of voters. In such small scale elections, a coercer might deduce a voters' choice despite coercion resistance mechanisms, if he knows how other voters will vote.
The minimal disclosure variant can be realized by blindly testing if the final homomorphic combined tally belongs to a set of possible results. We propose to realize this blind testing analogously to the authorisation mechanism of the coercion-resistant protocol. Here the possible results take over the role of preregistered credentials. After their blind fingerprints are put into a hashtable, one hashtable lookup on the fingerprint of the result of the election detects if the result actually belongs to the result set.

The described approach improves on Hevia and Kiwi's "Jury Voting Protocol"[7] [HK04], which realizes a comparable minimal disclosure concept based on a mechanism similar to plaintext equality tests.

Beside the application in the voting protocol, the described mechanism is a general concept to anonymously test the membership in a set. As a direct application, the function of zero knowledge validity proofs could be replaced by this mechanism. It seems worth to evaluate further possible applications of it. See [HK04] for further discussions.

[7] As an application, Hevia and Kiwi describe the case of votings in trials by jury. The approach can be employed to protect the privacy of the jurors.

6 Prototype

One goal of this thesis is to develop a *prototype* of our coercion-resistant election scheme as a proof of concept. To implement a certain algorithm or protocol is a good way to find out the obstacles of theoretical concepts and to develop a deeper understanding of it.
In the context of this diploma thesis developing a prototype means to simplify the voting system in some aspects to keep the whole project feasible. We focus on demonstrating the feasibility of the cryptographic algorithms and set aside distributed system and networking issues. The prototype does not distribute the cryptographic services on several servers.

Full implementations to be used in real world remote electronic elections require a more complex design. For implementation aspects of cryptographic protocols see Ferguson and Schneier [FS03] and consult Varghese [Var05] for issues concerning the efficiency of networking and distributed systems.

6.1 Design Decisions

In this section we describe important decisions that were made in the design of the prototype.

6.1.1 Cryptography in Java

The prototype is realized in Java, which offers a framework to implement cryptographic software, the Java Cryptography Architecture (JCA) and the Java Cryptography Extension (JCE). The implementation partly builds upon the Flexiprovider[1], developed by the Cryptography and Computeralgebra group of Professor Buchmann at Darmstadt University of Technology.

6.1.2 Implementing Protocols

A major design decision is how to implement cryptographic protocols. In the context of this diploma thesis an implementation as a distributed system, run-

[1] www.flexiprovider.de

ning the cryptographic services on dedicated servers, cannot be realized. Instead we roughly orientate the protocol implementation on sample programs presented in the JCE Reference Guide [Sun].
In the case of distributed key generation, decryption and blinding, a protocol is modelled as a program offering a runtime environment, in which the participants of the protocol, modelled as instances of certain object types, i.e. `ElectionAuthority`s, consecutively execute their operations and exchange messages. The communication channels in the protocols are modelled by data structures representing bulletin boards. The communication is simplified, especially we do not realize a public key infrastructure to secure the channels.
The zero knowledge proof protocols are implemented in their non-interactive variants. The voting protocol implements the non-distributed operations and instantiates the above mentioned subprotocols to realize the distributed computations.

6.1.3 Client/Server Communication

The submission of ballots over the Internet in this prototype is implemented via `Socket`s, `ServerSocket`s and `ObjectInputStream`s, `ObjectOutputStream`s. The networking is orientated on examples given by Ullenboom [Ull06].

6.2 Implementation

Next we describe our developed implementation of the voting protocol with multiple election authorities, as specified in section 3.6.3. It realizes a 1-out-of-k election, where one candidate can be chosen by the voters among a set of k candidates.

6.2.1 Package Overview

The implementation consists of the packages:

- `de.evoting.elgamalsubgroup`
- `de.evoting.bulletinboard`
- `de.evoting.elgamalthreshold`
- `de.evoting.mixnet`
- `de.evoting.protocol`

The implementation of the cryptographic algorithms is based on the `java.math.BigInteger` library, which implements modular arithmetics in Z_p^*, G_q and Z_q, supporting integers of arbitrary length.

6.2.2 Package `de.evoting.elgamalsubgroup`

This package provides an implementation of the ElGamal cryptosystem over subgroups G_q of Z_p^*, as described in section 2.2.
It re-uses and modifies parts of the `de.flexiprovider.core.elgamal` package. It consists of the classes `ElGamalGq` and `ElGamalGqKeyPairGenerator`.
The `ElGamalGq` cipher encrypts and decrypts plaintexts (represented as single `BigInteger`) and ciphertexts (`BigInteger[]`) and implements the mapping between Z_q and G_q. Its design is orientated on the provider architecture, but simplified. It can be initialized with keys generated by the `ElGamalGqKeyPairGenerator`.
The `ElGamalGqKeyPairGenerator` generates `KeyPair`s containing `ElGamalPublicKey`s and `ElGamalPrivateKey`s, which are reused. The keys contain the system parameters p, q, g as well. The algorithm to generate the primitive element $g \in G_q$ is due to Damgard [Dam04].
The main purpose of the `ElGamalGqKeyPairGenerator` is to generate system parameters to specify the setting. The implementation of the threshold ElGamal cryptosystem is described in section 6.2.5.

6.2.3 Package `de.evoting.bulletinboard`

This package contains several data structures that model the role of bulletin boards in the protocols.
The `DecBBoard` is used in the distributed decryption protocol (class `DistDecryption` in package `de.evoting.elgamalthreshold`) and the distributed blinding protocol (`DistBlinding`), it stores partial decryptions and blindings and the associated NIZKPs.
The `DkgBBoard` implements the broadcast channel in the distributed key generation protocol (`DistKeyGen`), it is designed to handle internal data of three runs of this protocol.
`VoteBBoard`s store cast and further processed votes in the voting protocol (class `VotingProtocol` in package `de.evoting.protocol`). Furthermore, a `RegBBoard` stores the names of registered voters and their encrypted credentials, and a `ResultBBoard` holds the decrypted votes in the tallying phase of an election.

6.2.4 Package `de.evoting.mixnet`

This package provides the functionality of a re-encryption mixnet. It contains the classes `ReEncMix` and `SimpleMixNet`.
The `ReEncMix` class implements a single re-encryption mix. It processes a batch of ballots by re-encrypting the included ElGamal ciphertexts, i.e. it alters the random value used in the encryption, and permutes the order of the ballots. The random permutation is created using the algorithm of Moses and Oakford [MO63], as described by Skiena [Ski97].
The `SimpleMixNet` class represents a complete re-encryption mixnet. In this implementation, it simply wraps *one* `ReEncMix`, instead of containing several mixing stages.
This mixnet implements the mixing steps in the voting protocol, which is supported by three operation modes (`run()`, `runNoTime()`, `runOnlyCred()`), according to the different data fields that need to be anonymized in the specific steps.

6.2.5 Package `de.evoting.elgamalthreshold`

This package contains implementations of the protocols of the threshold ElGamal cryptosystem, as well as the distributed blinding protocol and the responsible authorities. It contains the classes:

- `Parameters`
- `ElectionAuthority`
- `DistKeyGen`
- `DKGParamSet`
- `PolynomialZq`
- `DistDecryption`
- `LaGrange`
- `ProofOfSameExp`
- `DecAndZKP`
- `DistBlinding`
- `BlindAndZKP`

The `Parameters` class capsules system parameters p, q, g and threshold parameters n, t. It is accessed by the protocols in this package. The actual values for n, t are specified in the constructor of the `VotingProtocol` (in package `de.evoting.protocol`), here $t - 1$ must be a *minority* of authorities.

The `ElectionAuthority` class models an election authority. An array `ElectionAuthority[]` represents the set of n authorities. In this prototype there is only this one kind of authority, it participates in the `DistKeyGen`, `DistDecryption` and `DistBlinding` protocols, the non-interactive zero knowledge `ProofOfSameExp` protocol and the `VotingProtocol` as well.
This class implements the several computational steps as specified in the protocols, cheating authorities are not implemented. An authority stores data generated and exchanged in three initial runs of the distributed key generation protocol in its `DKGParamSet[]` field.

The `DistKeyGen` class implements the complex distributed key generation protocol, as described in section 2.5.3. Broadcast channels are modelled by a `DkgBBoard`, secret channels by directly passing the messages between the authorities.
A helper class is `PolynomialZq`, it implements a polynomial over Z_q and polynomial evaluation via Horner's scheme[2].

The `DistDecryption` class implements the distributed decryption protocol, as specified in section 2.5.4. Its communication channel is modelled by a `DecBBoard`, see section 6.2.3.
In this protocol the authorities have to construct NIZKPs to show that they produced their partial decryption correctly, as specified in section 2.5.4. The `ProofOfSameExp` class is the container for this proof data, all computations are done in the `ElectionAuthority.constructProof(..)` method. The challenge is computed by the `ElectionAuthority.randomOracleHashSha1(..)` method, which uses an instance of `de.flexiprovider.core.md.SHA1` to compute the hash value. The `DecAndZKP` class is used in the protocol to wrap data, it stores a partial decryption together with the associated NIZKP.
To support the reconstruction of plaintexts, the `LaGrange` class implements computation of Lagrange coefficients and selection of an appropriate group Λ of t authorities.

The `DistBlinding` class implements the distributed blinding protocol, as described in section 3.6.2. Analog to the `DistDecryption` class its communication channel is given by a `DecBBoard`, and the `BlindAndZKP` class stores two `ProofOfSameExp`s (generated via the `ElectionAuthority.constructBlindProof(..)` method) together with the partial blinding of a ciphertext.

[2]The Horner's scheme is an algorithm for the efficient evaluation of polynomials, named after William George Horner.

6.2.6 Package `de.evoting.protocol`

It contains the classes:

- `CandidateSlate`
- `ProofOfKnowledge`
- `ProofOfValidity`
- `Ballot`
- `Voter`
- `VotingProtocol`
- `VotingProtocolServer`
- `VotingApplet`
- `VotingProtocolClient`

The `CandidateSlate` class represents the list of candidates and implements methods to count the number of votes each candidate received (`countForCandidate(..)`) and to sort the list by this number (`organizeResult()`).
The list of candidates contains one abstention choice, so voters have the possibility to cast a vote that is not counted for any candidate.

The `ProofOfValidity` and `ProofOfKnowledge` classes are containers for the NIZKPs, specified in sections 2.3.4 and 2.3.2. They store the messages of the proof protocols and are both constructed by a `Voter` instance.

A `Ballot` represents a ballot cast. It stores `BigInteger[]` ciphertexts of credentials, the vote and a timestamp as well as a `ProofOfValidity` of the vote and a `ProofOfKnowledge` of the plaintext of the credential. Additionally, it temporarily stores the fingerprints of the credential.

The `Voter` class models a voter in the election. It implements the construction of NIZKPs (`constructProofCred(..)`, `constructValidityProof(..)`), for which it has to store the random value used in the encryption of the vote and the credential. The proofs are made non-interactive analog to the case of the `ProofOfSameExp`.
The `Voter` constructs timestamps (`dateBigInt()`) represented as `BigInteger` by extracting digits of a `java.sql.Timestamp`. If the voting system is tested with a small key strength, it has to be assured that the timestamp can be encoded as one number in Z_q, by only using a part of its digits.

The `Voter` casts ballots (`castBallot(..)`) in which it includes encrypted ciphertexts of his credential, the vote and the timestamp, and the NIZKPs.

The `VotingProtocol` class implements the voting protocol as described in section 3.6.3. It generates a number of ballots by itself and processes them.
The `VotingProtocolServer` class extends this to allow casting of ballots over the Internet via a client, the `VotingApplet`, or alternatively by using the `VotingProtocolClient` class.
We describe the implementation of theses classes detailed in the next sections.

6.2.7 Class `VotingProtocol`

The protocol is executed by creating an instance of it (parameters t, n and the strength of the encryption, i.e. the bit-length of p, are specified in the constructor) and calling the `run()` method. This executes consecutively:

1. Setup Phase (`setup()`)
2. Registration Phase (`registrationPhase()`)
3. Candidate-Slate Publication (`candSlateGeneration()`)
4. Voting Phase (`votingPhase()`)
5. Processing of Votes/Tallying Phase (`voteProcessingPhase()`)

1. Setup Phase

In the setup phase, system parameters are generated by a `ElGamalGqKeyPairGenerator`, n `ElectionAuthority`s are created and the `DistKeyGen` protocol is executed three times to generate the shared keys (one private key and hashkeys j and k) and the public key of the authorities.
Alternatively to generating the system parameters in the `setup()` method, the methods `setup512bit()` and `setup1024bit()` can be used, which store predefined values for p and g of the appropriate strength.

2. Registration Phase

In the registration phase a number of voters is registered. One can assign arbitrary credentials, i.e. `BigInteger` values $\in Z_q$, to a voter, but 0 should not be used.
The list of registered voters is set up on a `RegBBoard`, which contains the voters' names and their credentials encrypted under the public key of the authorities.
A sample content of the `RegBBoard`, for a small group of voters and 32 bit encryption strength, is shown in figure 6.1.

```
-----------------------------------------------
List of Registered Voters:  (Name - Credential)
-----------------------------------------------
[0] :  Voter0 # C: 903160764-2098141834
[1] :  Voter1 # C: 1575967524-391675472
[2] :  Voter2 # C: 1716796201-2531341978
[3] :  Voter3 # C: 482332971-289070446
[4] :  Voter4 # C: 1636148528-2490219542
[5] :  Voter5 # C: 913583566-1098468149
[6] :  Voter6 # C: 1454255711-957846773
[7] :  Voter7 # C: 1153827204-1051988407
[8] :  Voter8 # C: 1870910790-2017197506
[9] :  Voter9 # C: 183716678-536953198
```

Figure 6.1: List of Registered Voters

3. Candidate-Slate Publication

A list of candidates is generated and published. The names of the candidates can be specified in the constructor of the `CandidateSlate`.
The candidate list is then displayed. A sample slate is shown in figure 6.2.

```
----------------------------
Candidate Slate for Election
----------------------------
[0]  :  Abstention
[1]  :  Candidate 1
[2]  :  Candidate 2
[3]  :  Candidate 3
[4]  :  Candidate 4
[5]  :  Candidate 5
[6]  :  Candidate 6
[7]  :  Candidate 7
[8]  :  Candidate 8
[9]  :  Candidate 9
[10] :  Candidate 10
```

Figure 6.2: Candidate Slate

The numbers associated to the candidates are their unique identifiers, they specify the possible votes.

4. Voting Phase

In this phase a number of ballots is generated by the system itself. To do this, a `Voter` instance must first be initialized with a credential value (`authenticate(..)`) and a valid choice of a candidate (`inputChoice(..)`). Then the ballot is constructed (`castBallot(..)`) and added to the bulletin board for incoming ballots (`addBallot(..)`).

5. Processing of Votes/Tallying Phase

In this phase the following steps are executed:

- 5.1. Validation of NIZKPs (`validateZKP()`)
- 5.2. Anonymization (`mixVotes()`)
- 5.3. Duplicate Elimination
  - 5.3.1. Computation of Fingerprints of Credentials and Decryption of Timestamps (`hashVotesK()`)
  - 5.3.2. Hashtable Lookup (`eliminateDuplicates()`)
- 5.4. Authorisation
  - 5.4.1. Mixing of unique Ballots (`mixVotesBB6()`)
  - 5.4.2. Mixing of Credentials on Registration List (`mixCredentials()`)
  - 5.4.3. Computation of Fingerprints of Credentials of Ballots (`hashVotesJ()`) and of Credentials of Registration List (`hashCredentialsJ()`)
  - 5.4.4. Hashtable Lookup (`authorizeVotes()`)
- 5.5. Decryption of Votes (`decryptVotes()`)
- 5.6. Tallying (`tally()`, `organizeResult()`)

5.1. Validation of NIZKP

For each `Ballot` both NIZKPs are verified, and ballots with valid proofs are added to the next bulletin board.

5.2. Anonymization

A `SimpleMixNet` is instantiated, initialized and used to anonymize all cast ballots. It re-encrypts the ciphertext of each ballot's vote, credential and timestamp, and shuffles the list of ballots.

5.3. Duplicate Elimination

First, fingerprints of the credentials are computed (`hashVotesK()`), by applying to each ballot the `DistBlinding` protocol (using the first shared hashkey), followed by the `DistDecryption` protocol. Timestamps are decrypted as well.
Second, a hashtable lookup is performed to eliminate ballots containing duplicate fingerprints (`eliminateDuplicates()`). Each `Ballot` is put in a `java.util.Hashtable`, using its fingerprint as `key`. As Java internally applies a `hashCode()` method to the `key` value, this implementation of the hashtable lookup is *not collision-free*.
For each ballot put in the hashtable the previous value of the specified key is returned, or `null` if it did not have one. If it is not `null`, i.e. a collision occurs, timestamps of both ballots are compared and the newer ballot is kept for further processing.
After the hashtable lookup is finished, the timestamps on the unique ballots are discarded (`deleteTimestamp()`).

5.4. Authorisation

Both the list of unique ballots (`mixVotesBB6()`) and the list of registered credentials (`mixCredentials()`) are anonymized by a `SimpleMixNet`. The credentials are wrapped as `Ballot`s for convenient processing.
For both lists fingerprints are computed (`hashVotesJ()`, `hashCredentialsJ()`) (using the second shared hashkey), and a hashtable lookup is performed (`authorizeVotes()`). The hashtable is initially filled with fingerprints of registered credentials, then the ballots are consecutively put in the hashtable, and each collision detects an authorized ballot.

5.5. Decryption of Votes

For each authorized ballot, the vote is decrypted (`decryptVotes()`) via the `DistDecryption` protocol. These plaintexts are then published on the `ResultBBoard`.

5.6. Tallying

The number of votes for each candidate is counted (`tally()`) and the election result is sorted by the number of votes each candidate received (`organizeResult()`).
The result of the election is then displayed. A sample result shows figure 6.3.

```
----------------------------
Result of Election
----------------------------

[0]  :  Candidate 2 - Votes received:  891
[1]  :  Candidate 7 - Votes received:  718
[2]  :  Candidate 3 - Votes received:  712
[3]  :  Candidate 5 - Votes received:  637
[4]  :  Candidate 4 - Votes received:  609
[5]  :  Candidate 6 - Votes received:  544
[6]  :  Abstention  - Votes received:  365
[7]  :  Candidate 9 - Votes received:  228
[8]  :  Candidate 8 - Votes received:  197
[9]  :  Candidate 1 - Votes received:  194
[10] :  Candidate 10 - Votes received:  90

TOTAL NUMBER of votes counted:  5185
```

Figure 6.3: Election Result

6.2.8 Class `VotingProtocolServer`

This class extends the `VotingProtocol` by additional functionality to allow the casting of ballots over the Internet.
This is implemented in the method `startElectionServer()`, which is called in the `votingPhaseServer()` method, replacing the former `votingPhase()` method. The `VotingProtocolServer` accepts incoming connections from a client, until a timeout (specified by `setSoTimeout()`) occurs.
Communication between the server and a client is implemented via `Sockets`, `ServerSockets` and `ObjectInputStreams`, `ObjectOutputStreams`.
The `handleCommunication(..)` method specifies the communication protocol:

1. On accepting an incoming connection from a client, the server sends the candidate slate, represented by its length, to the client.

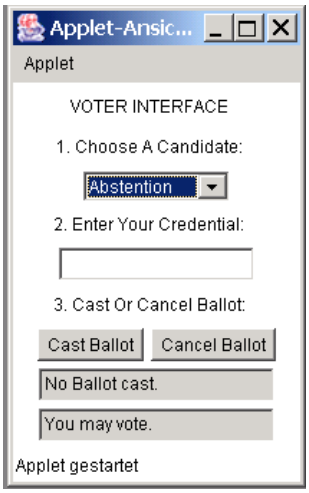

Figure 6.4: Applet Voter Interface - Initialized

2. Next the server sends the election authorities' public key to the client.

3. The client responds by sending a ballot, encrypted under the received public key and encoding one of the candidates specified by the slate.

In this implementation the server is designed to accept only one incoming connection from a client at a time. The server is not multi-threaded.

6.2.9 Class `VotingApplet`

The `VotingApplet` class provides a graphical user interface for the voters, realized as applet. It implements the communication protocol, as given above, between the server and the client.
The voter interface is shown in Figures 6.4, 6.5, 6.6 and 6.7. After initialisation (Figure 6.4), users can select a candidate (Figure 6.5), enter their credentials[3] (Figure 6.6) and cast their ballots or cancel the input. After one ballot is cast, this instance of the applet becomes inactive (Figure 6.7) and cannot be used to vote again.

[3]Even though we disencourage a manual input of credentials (compare section 4.7), different solutions are beyond the scope of our prototype. For its use we propose to register credential values that are simple to enter.

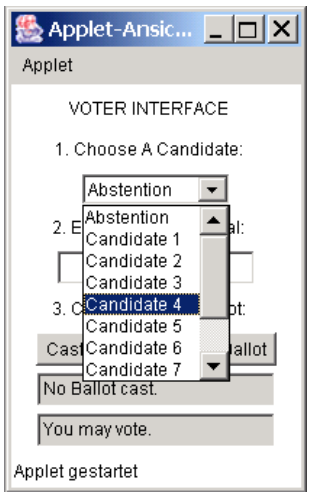

Figure 6.5: Applet Voter Interface - Candidate Selection

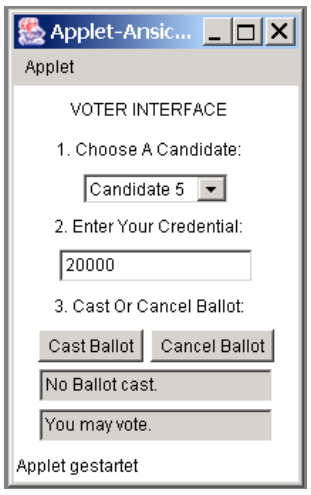

Figure 6.6: Applet Voter Interface - Credential Input

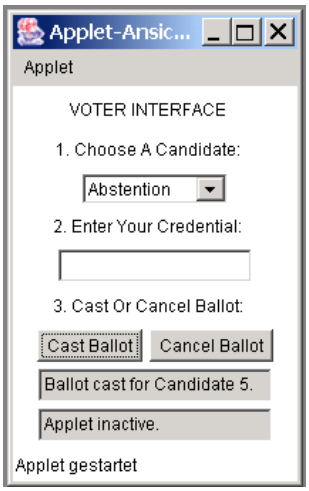

Figure 6.7: Applet Voter Interface - Inactive

6.2.10 Class `VotingProtocolClient`

Alternatively to the applet, a client with a simple text user interface is implemented by the `VotingProtocolClient` class. A run of this client is shown in figure 6.8.

6.3 Evaluation

In this section we briefly sketch some further results that were drawn from the prototype implementation.

6.3.1 Performance

We evaluated the performance of the `VotingProtocol` on an AMD Athlon XP 2400+ with 512 MB RAM.
Table 6.1 lists up the runtime of the `voteProcessingPhase()` for several parameters. Here *Strength* indicates the bit length of the prime p, n is the total number of authorities and t is the threshold.

As expected, the runtime is linear in the number of votes. The results show the influence of the strength of the encryption and the number of authorities on the performance as well.

A further time critical step in the protocol is the generation of the system parameters in the setup phase. A probabilistic algorithm, partly adopted from

```
-------------------------------------
VOTING CLIENT WITH TEXT USER INTERFACE
-------------------------------------

CLIENT: STARTING RUN
CLIENT: RECEIVED CANDIDATE SLATE
CLIENT: RECEIVED PUBKEY
---------------------------
Candidate Slate for Election
---------------------------
[0] :  Abstention
[1] :  Candidate 1
[2] :  Candidate 2
[3] :  Candidate 3
[4] :  Candidate 4
[5] :  Candidate 5
[6] :  Candidate 6
[7] :  Candidate 7
[8] :  Candidate 8
[9] :  Candidate 9
[10] :  Candidate 10
-----------------------------------------------------------------
-----------------------------------------------------------------
Please enter a valid candidate number:  (Push ENTER afterwards.)
2
-----------------------------------------------------------------
Please enter your credential:  (Push ENTER afterwards.)
200000
-----------------------------------------------------------------
CLIENT: SENT BALLOT FOR CANDIDATE 2
CLIENT RUN COMPLETED.
```

Figure 6.8: Text Voter Interface

Strength in bit	n	t	**Ballots**		**Runtime in seconds**
			cast	**valid**	
256	5	3	20	10	7
256	5	3	200	100	68
256	5	3	2000	1000	653
256	5	3	10000	5000	3360
256	9	5	20	10	12
256	9	5	200	100	112
256	9	5	2000	1000	1124
256	13	7	20	10	16
256	13	7	200	100	155
256	13	7	2000	1000	1485
512	5	3	20	10	40
512	5	3	200	100	397
512	5	3	2000	1000	3838
512	9	5	20	10	67
512	9	5	200	100	675
512	9	5	2000	1000	6547
512	13	7	20	10	95
512	13	7	200	100	952
1024	5	3	20	10	272
1024	5	3	200	100	2677
1024	9	5	20	10	448
1024	9	5	200	100	4410
1024	13	7	20	10	633
1024	13	7	200	100	6235

Table 6.1: Performance of Vote Processing

the Flexiprovider, implements this task. The runtimes of this algorithm, included in the `ElGamalGqKeyPairGenerator()`, lie in a wide range. In sample test runs, for 256 bit strength, the runtime was in between 1 and 20 seconds, for 512 bit strength, it was in the range of 10 seconds to 15 minutes. For 1024 bit strength, the runtime was in the range of 20 seconds to 35 minutes.
To skip the individual generation of system parameters, predefined values can be used (`setup512bit()`, `setup1024bit()`).

We stress that our implementation is not performance optimized. A guide to eliminate performance problems in Java applications is given by Shirazi [Shi03].

6.3.2 Practical Collision Resistance of Hashtable Lookup

As the `java.util.Hashtable` data structure internally applies a hash function on the `key` object to realize an efficient storage mechanism, this implementation is not collision free, as theoretical possible. Its collision resistance depends on the collision resistance of the `hashCode()` method. This method computes the hash as `int`, thus the number of possible hash values is limited.

An extension to a collision-free implementation is possible: in case of a collision in the hashtable lookup, the actual fingerprints have to be extracted from both ballots and compared.

Nevertheless, none of the test runs of the system revealed a collision that affected the correctness of the results of those test elections.

7 Conclusions and Future Work

In this thesis a voting protocol for remote electronic elections over the Internet was presented, prototypically implemented and evaluated. The scheme fulfils the strong security requirement of coercion resistance, is verifiable, correct, robust and practical. The drawbacks of Juels et al.'s initial approach [JCJ05], the quadratic runtime, storage and verification work factors, could be eliminated.

7.1 On Smith' Work

Based on Smith' imprecise description [Smi05b], we were able to describe and develop a coercion-resistant voting protocol with a runtime linear in the number of votes. We reformulated the distributed computation of fingerprints, that allows to realize the hashtable lookup mechanism.
Several of Smith' ideas were identified as inappropriate. Especially his new attacks on Juels et al.'s scheme do not work as claimed.

7.2 On the Protocol Mechanisms

The protocol's ballot processing via encrypted credentials, non-interactive zero knowledge proofs, mixnets and hashtable lookups has been demonstrated as an efficient refinement of Juels et al.'s basic approach. A correct implementation is possible, otherwise the correctness depends on the collision resistance of additionally employed hash functions.

An argument that the use of hidden credentials is worth to be extended further is given by the fact that several people adapted this concept [Acq04, CM05, Smi05b, Sch06]. An integration of this concept into a homomorphic scheme has been demonstrated in this thesis as well.
Compared to other approaches to achieve a similar degree of security, less physical assumptions are required: an anonymous channel is required for voting, an untappable channel is only required for the registration. Other receipt-free schemes assume the existence of an untappable channel for voting, or the existence of tamper-resistant hardware devices instead.
Comparisons to existing solutions for online voting show that the presented

concept is not only of academic interest, but quite practical.
The Estonian case [BT06] demonstrates a related concept to avoid the possibility of vote-buying in electronic voting: voters had the possibility to vote again in polling stations, while only the second vote was counted then. The use of hidden credentials is an refinement of this idea, allowing the voters to cast undistinguishable fake votes additional to the valid ones.
As an example of a commercial product, the *Polyas system* [Mic05, VK06], frequently used for non-political elections in Germany, employs a comparable mechanism. In its registration phase, a voter receives an anonymous random token, comparable to a credential, that is used to authenticate a ballot later. In difference to our scheme, the token is not stored externally. Our scheme refines on this by allowing to give away invalid tokens to counter coercive attacks.

On the other hand, the credential infrastructure system has been identified as a possible weakness in the scheme. The registration phase has to be trustworthy, as the whole protocol depends on correctly distributed credentials. As well the storage media is a crucial component. It has to support the release of fake credentials, must be transferable and usable on arbitrary computers, without limiting the usability.

Indeed, our protocol has higher communication and computational costs than previous receipt-free homomorphic encryption based schemes. The heavy use of threshold cryptography results in a complex system architecture in case of a large scale implementation. The proposed concepts are complex to implement, and moreover complex to administrate and to operate. But they allow for a robust implementation and a distributed trust model. Several refinements to improve the efficiency of the scheme were proposed as well.

Even though it seems that our coercion-resistant protocol is not applicable to general elections in countries with mandatory voting, e.g. in Australia, as nobody can prove that he actually voted, this is indeed possible. We presented an extension to voter verifiability, and even though the voter-verified receipts are unusable to convince a coercer, they are convincing to the election authorities.

Moreover, the scheme demonstrates a general mechanism to realize a blind membership test, which might have further applications as well. It could e.g. replace plaintext equality tests in several applications. In this context an interesting question is whether different efficient blind membership testing schemes exist, that can the basis for a coercion-resistant protocol.

7.3 On Coercion Resistance

The property of coercion resistance guarantees a high level of security, but has some practical limitations, as it only considers remote coercion. But the pro-

posed protocol can also protect against the practical threat of shoulder surfing as well, if the storage media supports imperceptible release of invalid credentials.
If the coercer indeed knows how all-but-one voters will vote, coercion against the remaining one voter is unavoidable with the basic scheme. Our proposed minimal disclosure concept can overcome this drawback, especially in the case of small scale elections.
Also if an attacker can continuously observe a voter, coercion resistance fails. In such a case a mechanism like Damgard et al.'s [DJ02] proposal to vote on a permuted candidate list might be appropriate, but requires additional trust assumptions and is complicated to use.

In a practical use, the question arises whether voters would accept and make use of the provided mechanisms to achieve coercion resistance. We do not believe that voters would accept a multiple cast mechanism that enables them to change their vote. To achieve coercion resistance they must be able to vote again, too. The difference is that a vote cast under coercion contains an invalid credential and the candidates choice does not express the voter's true intent. So a voter is still encouraged to vote again validly.

And even more it is questionable, if especially those groups of persons that seem to be affected by coercion in elections, e.g. people living in homes for the aged, are able to properly use the offered mechanisms, as they are often limited in their ability to use information technology. So the aim is to design a system for remote electronic voting that meets their usability requirements. Straub and Baier [SB04] outline this accurately :

> "Users must be able to make use of the offered security features in practice, otherwise the best security application will fail."

Coercion and vote-buying are real threats, that have to be dealt with in any election. The presented approach seems to be one of the candidates to realize free and secret general elections, as postulated by electoral laws.

7.4 Implementation of a Prototype

On the practical side, we implemented a prototype of our coercion-resistant voting protocol and demonstrated the feasibility of the proposed cryptographic algorithms. The prototype shows that an implementation with a runtime linear in the number of votes can be realized.
Our system does not distribute the cryptographic services on several dedicated servers, but it can be the basis to develop a distributed variant. This can greatly improve its performance since the computations can be parallelized.

Even though this causes additional communication costs, we expect them to make up only a small ratio of the total costs, if the system is employed with realistic key strength and a large number of votes.
Beside this, our implementation provides a basic library for building and developing further voting protocols. It offers implementations of basic primitives voting protocols are built upon, like zero knowledge proofs, a semantical secure ElGamal threshold cryptosystem and re-encryption mixnets. For example these primitives can be used to refine Klink's prototype [Kli06], which was recently developed at the Cryptography and Computeralgebra group as well, but excluded the issues of zero knowledge proofs and threshold cryptography.

7.5 On the Security of E-Voting Systems

Even though a protocol, we believe to be secure to the best of our knowledge, was presented and prototypically implemented, this is far from having a voting system that is usable in a real world scenario.
Ferguson and Schneier [FS03] express this dilemma accurately:

> "Almost all protocol descriptions are done at a very high level. Most of the details are not described. This allows you to focus on the core functionality of the protocol, but it creates a great danger. Without careful specifications of all the actions that each participant should take, it is extremely difficult to create a safe implementation of the protocol."

And beyond the difficulty to provide a secure and reliable implementation of a voting system, its deployment, administration, the involved persons and more details have to be considered to secure the *whole process* of an electronic election.

The example of e-commerce shows that information technology has reached a level of reliability and security, that it is trusted to secure transactions worth millions of euros.
But the important difference between e-commerce and e-voting is that financial transactions are not private and can be re-done or cancelled in case of failures [Sch01]. In elections this is not possible. Thus, verifiability and audit mechanism that do not violate the privacy of the voters are even more of great importance for the realisation of secure electronic elections. For our scheme, we proposed such an extension, providing voter verifiability.

7.6 Future Work

Our work can largely be extended, some points of future work have already been mentioned above. On the theoretical side, there are a lot of further points to explore:

- Define a certain verifiable mixnet, that fits best in the protocol.
- Design a robust registration phase, e.g. based on mechanisms similar to the distributed key generation.
- Evaluate if batch verification techniques [APB+04, Adi05] are applicable to improve efficiency.
- Extend the applicable voting methods, e.g. based on validity proofs described by Groth [Gro05].
- A write-in vote mechanism can possibly be designed based on an extended candidate slate defined by the *passive suffrage*. So any possible valid write-in candidate is already present on the candidate slate. To reduce the enormous complexity of the validity proofs in this case, a blind membership test on the candidate choice can be performed instead.
- The primitives offer a lot of possibilities for refinements. The cryptosystem can be sped up, e.g. with techniques presented by Stam [Sta03], exchanged by an elliptic curve variant, or can be replaced by a Paillier implementation.
  The distributed key generation can be replaced with alternative protocols [GJKR99, CS04] as well.
- The concepts of threshold cryptography and secret sharing allow for further refinements as well. More complex access structures, i.e. the set of election authorities that need to cooperate, can be realized to model hierarchies or asymmetric roles. Proactive secret sharing techniques can be applied to refresh shares of secret keys if attackers managed to compromise certain authorities.

On the practical side, beside the implementation of the points mentioned above, the prototype can be extended to a real distributed application. Better storage, logging and audit mechanism can be included and a collision-free implementation can be realized as well. Our proposed variants can be implemented and evaluated, to determine for which scenarios and parameters which is the most efficient one. Finally, the usability and user-acceptance of the voting system can be evaluated in test elections.

Moreover, the voting protocols of Acquisti [Acq04], Kiayias and Yung [KY04], Clarkson and Myers [CM05] and Kutylowski and Zagorski [KZ06] should be evaluated to see, whether they propose concepts that are applicable to extend our scheme.

Interesting is furthermore the question, whether a coercion-resistant e-bidding protocol can be build upon our concepts, along the lines of Burmester et al. [BMC04]. As well it seems worth to evaluate possible applications of the blind membership testing protocol part. Hevia and Kiwi [HK04] suggest applications in *private information retrieval* for their related approach.

Beside all this, further evaluation of the coercion-resistant protocol is necessary. Protocols have always been broken and improved again. We are interested in which shortcomings can be found in our work.

Figure 7.1: Error! Source: Electronic Election 2004 by Mark Fiore

A Understanding Smith' Approach

Smith [Smi05b] describes in detail his voting protocol that is based on *secure multi-party computation*. He states that this solution, even though it is secure and its runtime is linear in the number of votes, is yet inefficient for a practical use. He then proposes a speedup to the point of practicality, by replacing the secure multi-party computation steps. But Smith' explanations are imprecise and hard to understand.

In this appendix, we add the crucial detail that Smith omits in order to clarify his approach. So we contribute to understand his partly confusing publication "New cryptographic election protocol with best-known theoretical properties". These insights led to the protocol described in chapter 3.

A.1 Secure Multi-Party Computation

To understand Smith' approach a background on the theory of secure multi-party computation is necessary. Our descriptions largely follow Hirt [Hir01].

Secure multi-party computation deals with the problem of n parties that want to jointly compute an agreed function of their inputs in a secure way. They want to guarantee the correctness of the output as well as the privacy of each parties' inputs, even if some participants cheat.

Secure multi-party computation is a general solution to design secure protocols that do not rely on trusted third parties (TTP). In secure multi-party computation the functionality of such a TTP is distributed among all parties. Every piece of data, that should be processed by a TTP, is instead secret shared among the parties. The function to be computed is specified by a circuit of logical gates. In order to evaluate the function, the parties perform the logical operations defined by the circuit, by using protocols that realize the basic boolean operations on secret shared data in a threshold setting. This approach is robust, as a minority of dishonest parties can be tolerated, and secure, as the input bits remain secret shared thus secret in the protocol. But it is highly inefficient for a practical use.
The main difference to threshold cryptography, as described in section 2.5, is that the distributed operations are performed on the boolean gate level and on secret shared data.

More details on secure multi-party computation can be found in [Can96, Hir01, Jar01, Feh03] and especially references therein.
The *plaintext equality test* protocol, described in section 3.3.3, is a key tool for the *mix and match* approach [JJ00] to secure multi-party computation.

A.2 Smith' Protocol based on Secure Multi-Party Computation

Smith' protocol, given in section 8 of [Smi05b], realizes a secure multi-party approach. It is structured analogously to our protocol in section 3.6, but each distributed computation step, i.e. the computation of fingerprints in the duplicate elimination and credential checking steps, and the final decryption, is carried out on secret shared data. Thus, these steps are preceded by runs of secret sharing protocols, in which the ballot data is shared among the election authorities. The computation steps are then carried out on these lists of shared ballot data.

The clear drawback of this approach, besides further concerns how to practically realize the sharing of such a huge amount of ballot data, is its inefficiency. Even though its runtime is $O(V + N)$ as well, where V is the number of votes and N the number of voters, the constants hidden in the O-notation are unacceptable large for secure multi-party computation, caused by the computation on boolean gate level. Smith discusses this drawback more intensively.

A.3 Replacing the Secure Multi-Party Computation

In section 11 of [Smi05b], Smith proposes to eliminate the inefficient secure multi-party computation steps and replace them with faster special purpose algorithms.
It is crucial to understand that replacing the secure multi-party computation means both eliminating the secret sharing steps and replacing the main computation step. Smith does not point this out clearly. It is only implicitly understandable, if one considers secure multi-party computation protocols to contain inherent secret sharing steps.

Bibliography

[Acq04] Alessandro Acquisti. Receipt-free homomorphic elections and write-in ballots. Cryptology ePrint Archive, Report 2004/105, 2004. http://eprint.iacr.org/.

[Adi05] Riza Aditya. *Secure Electronic Voting with Flexible Ballot Structure*. PhD thesis, Queensland University of Technology, November 2005.

[AI03] Masayuki Abe and Hideki Imai. Flaws in some robust optimistic mix-nets. In *Advances in Cryptology - ACISP 03*, 2003.

[ALBD04a] Riza Aditya, Byoungcheon Lee, Colin Boyd, and Ed Dawson. An efficient mixnet-based voting scheme providing receipt-freeness. In *International Conference on Trust and Privacy in Digital Business (TrustBus 2004)*, volume 3184 of *Lecture Notes in Computer Science*, pages 152–161. Springer-Verlag, September 2004.

[ALBD04b] Riza Aditya, Byoungcheon Lee, Colin Boyd, and Ed Dawson. Implementation issues in secure e-voting schemes. In *APIEMS 2004*, pages 36.6.1–36.6.14. QUT Publications, December 2004.

[Alv05] Michael R. Alvarez. Voter registration: Past, present and future, June 2005. Written Testimony Prepared for the Commission on Federal Election Reform.

[APB+04] Riza Aditya, Kun Peng, Colin Boyd, Ed Dawson, and Byoungcheon Lee. Batch verification for equality of discrete logarithms and threshold decryptions. In *ACNS 2004*, volume 3089 of *Lecture Notes in Computer Science*, pages 494–508. Springer-Verlag, June 2004.

[BBC98] BBC. Olympic 'vote buying' scandal, December 1998.

[Ben87] Josh Cohen Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, Yale University, Department of Computer Science, September 1987.

[BFP+01] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *PODC: 20th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 274–283, 2001.

[BG02] Dan Boneh and Philippe Golle. Almost entirely correct mixing with applications to voting. In *SIGSAC: 9th ACM Conference on Computer and Communications Security*. ACM SIGSAC, 2002.

[BL02] Stefan Brands and Frédéric Légaré. Digital identity management based on digital credentials. In *GI Jahrestagung*, pages 120–126, 2002.

[BLS+03] Benjamin B. Bederson, Bongshin Lee, Robert M. Sherman, Paul S. Herrnson, and Richard G. Niemi. Electronic voting system usability issues. In *Proceedings of ACM CHI 2003 Conference on Human Factors in Computing Systems*. ACM Press, 2003.

[BMC04] Mike Burmester, Emmanouil Magkos, and Vassilios Chrissikopoulos. Uncoercible e-bidding games. *Electronic Commerce Research*, 4(1-2):113–125, 2004.

[Bon98] Dan Boneh. The decision diffie-hellman problem. In *ANTS-III*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63. Springer-Verlag, 1998.

[Bou00] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *Advances in Cryptology - EUROCRYPT 00*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444. Springer-Verlag, 2000.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security*, pages 62–73, November 1993.

[Bra02] Stefan Brands. A technical overview of digital credentials, 2002.

[BS96] Eric Bach and Jeffrey Shallit. *Algorithmic Number Theory, Volume I: Efficient Algorithms*. MIT Press, 1996.

[BT94] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *ACM Symposium on Theory of Computing (STOC)*, pages 544–553. ACM Press, May 1994.

[BT06] Fabian Breuer and Alexander H. Trechsel. E-voting in the 2005 local elections in estonia. Technical report, Council Of Europe, 2006.

[Buc04] Johannes A. Buchmann. *Introduction to Cryptography*. Springer-Verlag, second edition, 2004.

[BY86] Josh Cohen Benaloh and Moti Yung. Distributing the power of a government to enhance the privacy of voters. In *Fifth Annual ACM Symposium on Principles of Distributed Computing*, pages 52–62, August 1986.

[Can96] Ran Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute of Science, Israel, March 1996.

[CDS94] Ronald Cramer, Ivan Damgard, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology - CRYPTO 94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer-Verlag, 1994.

[CF85] Josh D. Cohen and Michael J. Fischer. A robust and verifiable cryptographically secure election scheme (extended abstract). In *26th Annual Symposium on Foundations of Computer Science (FOCS 85)*, pages 372–382, October 1985.

[CFSY96] Ronald Cramer, Matt Franklin, Berry Schoenmakers, and Moti Yung. Multi-authority secret-ballot elections with linear work. In *Advances in Cryptology - EUROCRYPT 96*, volume 1070 of *Lecture Notes in Computer Science*, pages 72–83. Springer-Verlag, 1996.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. In *30th ACM Symposium on Theory of Computing (STOC 98)*, pages 209–218, May 1998.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology - EUROCRYPT 97*, 1997.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[Cha04] David Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy*, 2(1), 2004.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology - EUROCRYPT 01*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–117. Springer-Verlag, 2001.

[CM05] Michael R. Clarkson and Andrew C. Myers. Coercion-resistant remote voting using decryption mixes. In *Workshop on Frontiers in Electronic Elections (FEE 2005)*, Milan, Italy, September 2005.

[Com02] European Commission For Democracy Through Law (Venice Commision). Code of good practice in electoral matters, October 2002.

[Com04] European Commission For Democracy Through Law (Venice Commision). On the compatibility of remote voting and electronic voting with the standards of the council of europe, March 2004.

[Com05] The National Election Committee. E-voting system - overview. Technical report, The National Election Committee, Estonia, 2005.

[CP93] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *Advances in Cryptology - CRYPTO 92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer-Verlag, 1993.

[Cra96] Ronald Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis, CWI and University of Amsterdam, the Netherlands, November 1996.

[CS04] John Canny and Stephen Sorkin. Practical large-scale distributed key generation. In *Advances in Cryptology - EUROCRYPT 04*, volume 3027 of *Lecture Notes in Computer Science*, pages 295–310. Springer-Verlag, 2004.

[CTL05] Richard Celeste, Dick Thornburgh, and Herbert Lin, editors. *Asking the Right Questions About Electronic Voting*. National Academies Press, 2005.

[Cyb02] CyberVote. Report on review of cryptographic protocols and security techniques for electronic voting. Technical report, EUCyberVote.org, January 2002.

[Dam04] Ivan Damgard. Discrete log based cryptosystems, 2004. Introductory Course in Cryptography, Course Material 2005, Date: November 1, 2004.

[Des97] Yvo Desmedt. Some recent research aspects of threshold cryptography. In *1st International Workshop on Information Security(ISW'97)*, volume 1396 of *Lecture Notes in Computer Science*, pages 158–173. Springer-Verlag, 1997.

[DF89] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *Advances in Cryptology - CRYPTO 89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer-Verlag, 1989.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[DJ01] Ivan Damgard and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *4th International Workshop on Practice and Theory in Public Key Cryptography, (PKC 2001)*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136. Springer-Verlag, 2001.

[DJ02] Ivan Damgard and Mads Jurik. Client/server tradeoffs for online elections. In *Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, (PKC 2002)*, volume 2274 of *Lecture Notes in Computer Science*, pages 125–140. Springer-Verlag, 2002.

[DJN03] Ivan Damgard, Mads Jurik, and Jesper Buus Nielsen. A generalization of Paillier's public-key system with applications to electronic voting, 2003.

[DKR06] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *IEEE Computer Security Foundations Workshop (CSFW'06)*, Venice, Italy, July 2006. IEEE Computer Society Press.

[DLD03] Barbara Simons David L. Dill, Bruce Schneier. Voting and technology: who gets to count your vote? *Communications of the ACM*, 46(8):29–31, 2003.

[DM04] Emmanuel Dall'Olio and Olivier Markowitch. Voting with designated verifier signature-like protocol. In *ICWI*, pages 295–301, 2004.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[Feh03] Serge Fehr. *Secure Multi-Player Protocols: Fundamentals, Generality, and Efficiency*. PhD thesis, BRICS, Department of Computer Science, University of Aarhus, Denmark, August 2003.

[FOO92] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *Advances in Cryptology - ASIACRYPT 92*, volume 718 of *Lecture Notes in Computer Science*, pages 244–251. Springer-Verlag, 1992.

[FPS00] Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. In *Financial Cryptography*, volume 1962 of *Lecture Notes in Computer Science*, pages 90–104. Springer-Verlag, 2000.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO 86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, August 1986.

[FS01] Jun Furukawa and Kazue Sako. An efficient scheme for proving a shuffle. In *Advances in Cryptology - CRYPTO 01*, volume 2139 of *Lecture Notes in Computer Science*, pages 368–387. Springer-Verlag, 2001.

[FS03] Niels Ferguson and Bruce Schneier. *Practical Cryptography*. John Wiley and Sons, 2003.

[GA03] R. Green and J. Adler. Election verification technology - threat analysis. Technical report, VoteHere, Inc., 2003.

[Gen96] Rosario Gennaro. *Theory and Practice of Verifiable Secret Sharing*. PhD thesis, Massachusetts Institute of Technology, May 1996.

[GJKR99] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Advances in Cryptology - EUROCRYPT 99*, volume 1592 of *Lecture Notes in Computer Science*, pages 295–310. Springer-Verlag, 1999.

[GJKR03] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure applications of pedersen's distributed key generation protocol. In *CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 373–390. Springer-Verlag, 2003.

[GKK03] Marcin Gomulkiewicz, Marek Klonowski, and Miroslaw Kutylowski. Rapid mixing and security of Chaum's visual electronic voting. In *ESORICS 2003*, volume 2808 of *Lecture Notes in Computer Science*, pages 132–145. Springer-Verlag, 2003.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *ACM Symposium on Theory of Computing (STOC '85)*, pages 291–304. ACM Press, 1985.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[Gol02] Oded Goldreich. Zero-knowledge twenty years after its invention, 2002.

[Gri02] Dimitris A. Gritzalis. Principles and requirements for a secure e-voting system. *Computers Security*, 21, No. 6:539–556, 2002.

[Gro04] Jens Groth. *Honest Verifier Zero-knowledge Arguments Applied*. PhD thesis, BRICS, Department of Computer Science, University of Aarhus, Denmark, October 2004.

[Gro05] Jens Groth. Non-interactive zero-knowledge arguments for voting. In *ACNS 2005*, volume 3531 of *Lecture Notes in Computer Science*, pages 467–482. Springer-Verlag, 2005.

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified vss and fasttrack multiparty computations with applications to threshold cryptography. In *17th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 101–111, 1998.

[GZB+02] Philippe Golle, Sheng Zhong, Dan Boneh, Markus Jakobsson, and Ari Juels. Optimistic mixing for exit-polls. In Y. Zheng, editor, *Advances in Cryptology - ASIACRYPT 02*, volume 2501 of *Lecture Notes in Computer Science*. Springer-Verlag, September 2002.

[Hir01] Martin Hirt. *Multi-Party Computation: Efficient Protocols, General Adversaries, and Voting*. PhD thesis, ETH Zurich, September 2001.

[HK04] Alejandro Hevia and Marcos Kiwi. Electronic jury voting protocols. *Theoretical Computer Science*, 321(1):73–94, 2004.

[HMR04] Volker Hartmann, Nils Meißner, and Dieter Richter. Online voting systems for non-parliamentary elections: Catalogue of requirements. Technical report, Physikalisch-Technische Bundesanstalt Braunschweig und Berlin, 2004.

[HS00] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *Advances in Cryptology - EUROCRYPT 00*, volume 1807 of *Lecture Notes in Computer Science*, pages 539–556. Springer-Verlag, 2000.

[Jak06] Markus Jakobsson. Cryptographic protocols. In Hossein Bidgoli, editor, *Handbook of Information Security*. John Wiley and Sons, 2006.

[Jar01] Stanislaw Jarecki. *Efficient Threshold Cryptosystems*. PhD thesis, Massachusetts Institute of Technology, June 2001.

[JCJ02] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. Cryptology ePrint Archive, Report 2002/165, 2002. http://eprint.iacr.org/.

[JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *ACM Workshop On Privacy In The Electronic Society 2005 (WPES'05)*, pages 61–70, November 2005.

[JJ00] Markus Jakobsson and Ari Juels. Mix and match: Secure function evaluation via ciphertexts (extended abstract). In *ASIACRYPT 00*, volume 1976 of *Lecture Notes in Computer Science*, pages 162–177. Springer-Verlag, 2000.

[JJR02] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *Proceedings of the 11th USENIX Security Symposium*, pages 339–353. USENIX Association, 2002.

[Jon03] Douglas Jones. A brief illustrated history of voting, 2003. http://www.cs.uiowa.edu/ jones/voting/pictures/.

[Jon04a] Douglas W. Jones. Auditing elections. *Communications of the ACM*, 47(10):46–50, 2004.

[Jon04b] Douglas W. Jones. Minimizing the trusted base, December 2004. Presentation for A Framework for Understanding Electronic Voting.

[JRSW04] David Jefferson, Aviel D. Rubin, Barbara Simons, and David Wagner. A security analysis of the secure electronic registration and voting experiment (serve), January 2004.

[Kli06] Alexander Klink. Cryptographic voting protocols - a prototype design and implementation for university elections at TU Darmstadt. Master's thesis, TU Darmstadt, Germany, March 2006.

[KSW05] Chris Karlof, Naveen Sastry, and David Wagner. Cryptographic voting protocols: A systems perspective. In *Fourteenth USENIX Security Symposium (USENIX Security 2005)*, August 2005.

[KY02] Aggelos Kiayias and Moti Yung. Self-tallying elections and perfect ballot secrecy. In *Public Key Cryptography - PKC 2002*, volume 2274 of *Lecture Notes in Computer Science*, pages 141–158. Springer-Verlag, 2002.

[KY04] Aggelos Kiayias and Moti Yung. The vector-ballot e-voting approach. In *Financial Cryptography*, volume 3110 of *Lecture Notes in Computer Science*, pages 72–89. Springer-Verlag, 2004.

[KZ06] Miroslaw Kutylowski and Filip Zagorski. Coercion-free internet voting with receipts. In *Workshop on Electronic Voting and e-Government in the UK*, Edinburgh, UK, February 2006.

[LBD+03] Byoungcheon Lee, Colin Boyd, Ed Dawson, Kwangjo Kim, Jeongmo Yang, and Seungjae Yoo. Providing receipt-freeness in mixnet-based voting protocols. In *ICISC*, volume 2971 of *Lecture Notes in Computer Science*, pages 245–258. Springer-Verlag, 2003.

[Lee01] Byoungcheon Lee. *Zero-knowledge Proofs, Digital Signature Variants, and Their Applications*. PhD thesis, School of Engineering, Information and Communications University, Daejeon, Korea, December 2001.

[Lip01] Helger Lipmaa. Statistical zero-knowledge proofs from diophantine equations. Cryptology ePrint Archive, Report 2001/086, 2001. http://eprint.iacr.org/.

[LK00] Byoungcheon Lee and Kwangjo Kim. Receipt-free electronic voting through collaboration of voter and honest verifier. In *Workshop on Information Security and Cryptology*, 2000.

[LK02] Byoungcheon Lee and Kwangjo Kim. Receipt-free electronic voting scheme with a tamper-resistant randomizer. In *ICISC: International Conference on Information Security and Cryptology*, volume 2587 of *Lecture Notes in Computer Science*, pages 389–406. Springer-Verlag, 2002.

[Mao03] Wenbo Mao. *Modern Cryptography: Theory and Practice*. Prentice Hall PTR, 2003.

[Mas04] Stephen Mason. Is there a future for internet voting? *Computer Fraud Security*, 2004 issue 3:6–13, March 2004.

[MBC01] Emmanouil Magkos, Mike Burmester, and Vassilios Chrissikopoulos. Receipt-freeness in large-scale elections without untappable channels. In *I3E 2001*, volume 202 of *IFIP Conference Proceedings*, pages 683–694. Kluwer, 2001.

[Mic05] Micromata. Online-Wahlen für Verbände und Vereine. Technical report, 2005.

[MN06] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In *CRYPTO 06*, 2006. To Appear.

[MO63] L.E. Moses and R.V. Oakford. *Tables of random permutations*. Stanford University Press, 1963.

[MSJ02] Philip D. MacKenzie, Thomas Shrimpton, and Markus Jakobsson. Threshold password-authenticated key exchange (extended abstract). In *Advances in Cryptology - CRYPTO 02*, volume 2442 of *Lecture Notes in Computer Science*, pages 385–400. Springer-Verlag, 2002.

[Nef01] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *SIGSAC: 8th ACM Conference on Computer and Communications Security*. ACM SIGSAC, 2001.

[Nik05] Ventzislav Stefanov Nikov. *Verifiable Secret Sharing and Applications*. PhD thesis, Technische Universiteit Eindhoven, the Netherlands, September 2005.

[Oka96] Tatsuaki Okamoto. An electronic voting scheme. In *IFIP World Conference on IT Tools*, pages 21–30, 1996.

[PAB+04] Kun Peng, Riza Aditya, Colin Boyd, Ed Dawson, and Byoungcheon Lee. Multiplicative homomorphic e-voting. In *INDOCRYPT 2004*, volume 3348 of *Lecture Notes in Computer Science*, pages 61–72. Springer-Verlag, 2004.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT 99*, volume 1599 of *Lecture Notes in Computer Science*, pages 223–238. Springer-Verlag, 1999.

[Ped91] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In *Advances in Cryptology - EUROCRYPT 91*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526. Springer-Verlag, April 1991.

[Ped92] Torben Pryds Pedersen. *Distributed Provers and Verifiable Secret Sharing Based on the Discrete Logarithm Problem*. PhD thesis, University of Aarhus, Department of Computer Science, Denmark, March 1992.

[PIK93] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *Advances in Cryptology - EUROCRYPT 93*, volume 765 of *Lecture Notes in Computer Science*, pages 248–259. Springer-Verlag, 1993.

[PTCB00] Pasuk Phongpaichit, Nualnoi Treerat, Yongyuth Chaiyapong, and Chris Baker. Corruption in the public sector in thailand: Perception and experience of households, 2000. Political Economy Center (Bangkok: Chulalongkorn University).

[Rap04] Dörte K. Rappe. *Homomorphic Cryptosystems and their Applications*. PhD thesis, University of Dortmund, Germany, August 2004.

[RSBA05] John Ross, Paul Spencer, John Borras, and Farah Ahmed. EML v4.0 process and data requirements - committee specifications. Technical report, OASIS, eGovernment Unit, Cabinet Office, UK, December 2005.

[Rub02] Aviel D. Rubin. Security considerations for remote electronic voting. *Communications of the ACM*, 45(12):39–44, 2002.

[Saf01] Safevote. Voting system requirements, 2001. http://www.thebell.net/papers/vote-req.pdf.

[SB04] Tobias Straub and Harald Baier. A framework for evaluating the usability and the utility of pki-enabled applications. In *EuroPKI*, volume 3093 of *Lecture Notes in Computer Science*, pages 112–125. Springer-Verlag, 2004.

[Sch91] Claus Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[Sch00] Berry Schoenmakers. Fully auditable electronic secret-ballot elections. *Xootic Magazine*, Juli 2000.

[Sch01] Bruce Schneier. Internet voting vs. large-value e-commerce. Crypto-Gram Newsletter, February 2001.

[Sch04] Guido Schryen. How security problems can compromise remote internet voting systems. In *Electronic Voting in Europe*, pages 121–131, 2004.

[Sch06] Jörn Schweisgut. Coercion-resistant electronic elections with observer. In *2nd International Workshop on Electronic Voting 2006 (2-4 Aug 2006), Bregenz*, 2006. To Appear.

[Scy03] Scytl. Elections to the parliament of catalonia 2003 - report on the remote electronic voting pilot. Technical report, Scytl Online World Security, Barcelona, Spain, December 2003.

[Ser04] Andrei Serjantov. *On the Anonymity of Anonymity Systems*. PhD thesis, Computer Laboratory, University of Cambridge, UK, March 2004.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Sha05] Michael I. Shamos. Cellphone vote-buying, October 2005. NIST Workshop Threats to Voting Systems.

[Shi03] Jack Shirazi. *Java Performance Tuning*. O'Reilly, 2003.

[SK95] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In *Advances in Cryptology - EUROCRYPT 95*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403. Springer-Verlag, 1995.

[Ski97] Steven S. Skiena. *The Algorithm Design Manual: Hypertext Edition*. Springer-Verlag, 1997.

[Smi05a] Warren D. Smith. Cryptography meets voting, September 2005.

[Smi05b] Warren D. Smith. New cryptographic voting scheme with best-known theoretical properties. In *Workshop on Frontiers in Electronic Elections (FEE 2005)*, Milan, Italy, September 2005.

[SP06] Krishna Sampigethaya and Radha Poovendran. A framework and taxonomy for comparison of electronic voting schemes. *Computers Security*, 25, issue 2:137–153, March 2006.

[Sta03] Martijn Stam. *Speeding up Subgroup Cryptosystems*. PhD thesis, Technische Universiteit Eindhoven, the Netherlands, June 2003.

[Sti95] Douglas R. Stinson. *Cryptography: Theory and Practice*. CRC Press, first edition, 1995.

[Str04] Tobias Straub. A method for strengthening certificate enrollment. In *WartaCrypt 2004*, 2004.

[Sun] Sun. Java cryptography extension (JCE) reference guide for the java 2 sdk, standard edition, v 1.4. http://java.sun.com/j2se/1.4.2/docs/guide/security/.

[TY98] Yiannis Tsiounis and Moti Yung. On the security of ElGamal based encryption. In *First International Workshop on Practice and Theory in Public Key Cryptography (PKC '98)*, volume 1431 of *Lecture Notes in Computer Science*, pages 117–134. Springer-Verlag, 1998.

[Ull06] Christian Ullenboom. *Java ist auch eine Insel*. Galileo Computing, fifth edition, 2006.

[Var05] George Varghese. *Network Algorithmics : An Interdisciplinary Approach to Designing Fast Networked Devices*. Morgan Kaufman Publishers, 2005.

[VG06] Melanie Volkamer and Rüdiger Grimm. Multiple casts in online voting - analyzing chances. In *2nd International Workshop on Electronic Voting 2006 (2-4 Aug 2006), Bregenz*, 2006. To Appear.

[VH04] Melanie Volkamer and Dieter Hutter. From legal principles to an internet voting system. In *Electronic Voting in Europe*, pages 111–120, 2004.

[VK06] Melanie Volkamer and Robert Krimmer. Secrecy forever? analysis of anonymity in internet-based voting protocols. In *First International Conference on Availability,Reliability and Security (ARES'06)*, pages 340–347. IEEE Computer Society, 2006.

[VLB+02] J. Väre, S. Lakshmeshwar, S. Brunessaux, D. De Cock, D. Parkinson, and S. Cavallar. Report on voting and communication protocols, pc and mobile phone clients, local and global servers. Technical report, EUCyberVote.org, November 2002.

[VV06] Melanie Volkamer and Roland Vogt. Überblick über die Online-Wahlen in Deutschland, July 2006. http://www.dfki.de/fuse/.

[Wik02] Douglas Wikström. How to break, fix and optimize "optimistic mix for exit-polls". Technical report, Swedish Institute of Computer Science, 2002.

[Wik03] Douglas Wikström. Four practical attacks for "optimistic mixing for exit-polls". Technical report, Swedish Institute of Computer Science, February 2003.

[Wik05] Douglas Wikström. *On the security of mix-nets and hierarchical group signatures*. PhD thesis, KTH, Numerical Analysis and Computer Science, Stockholm, Sweden, 2005.

[Yao82] A. C. Yao. Protocols for secure computations (extended abstract). In *23th Annual Symposium on Foundations of Computer Science (FOCS 82)*, pages 160–164. IEEE Computer Society Press, November 1982.