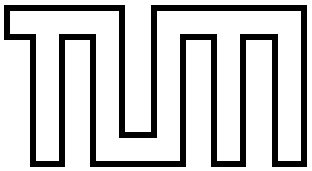


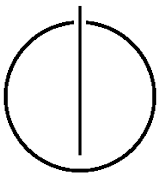
DEPARTMENT OF INFORMATICS

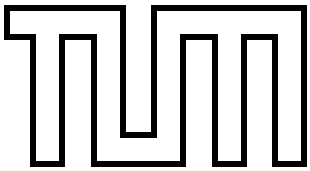
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

# Cryptographically Secure, Distributed Electronic Voting

Florian Dold






DEPARTMENT OF INFORMATICS

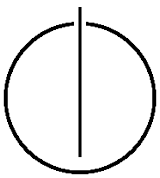
TECHNISCHE UNIVERSITÄT MÜNCHEN


Bachelor's Thesis in Informatics

# Cryptographically Secure, Distributed Electronic Voting

# Kryptographisch sicheres Wählen in verteilten Systemen


| | |
|---|---|
| Author: | Florian Dold |
| Supervisor: | Christian Grothoff, PhD (UCLA) |
| Advisor: | Christian Grothoff, PhD (UCLA) |
| Date: | October 15, 2014 |

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, August 24, 2014 Florian Dold

# Acknowledgments

I wish to thank Christian Grothoff for providing invaluable feedback and guidance throughout my work on this thesis as well as on other projects.

I also thank the GNUnet team, especially Sree Harsha Totakura, Bart Polot and Matthias Wachs, for their persistent support regarding bugs and questions.

# Abstract

Elections are a vital tool for decision-making in democratic societies. The past decade has witnessed a handful of attempts to apply modern technology to the election process in order to make it faster and more cost-effective.

Most of the practical efforts in this area have focused on replacing traditional voting booths with electronic terminals, but did not attempt to apply cryptographic techniques able to guarantee critical properties of elections such as secrecy of ballot and verifiability.

While such techniques were extensively researched in the past 30 years, practical implementation of cryptographically secure remote electronic voting schemes are not readily available. All existing implementation we are aware of either exhibit critical security flaws, are proprietary black-box systems or require additional physical assumptions such as a preparatory key ceremony executed by the election officials. The latter makes such systems unusable for purely digital communities.

This thesis describes the design and implementation of an electronic voting system in GNUnet, a framework for secure and decentralized networking. We provide a short survey of voting schemes and existing implementations.

The voting scheme we implemented makes use of threshold cryptography, a technique which requires agreement among a large subset of the election officials to execute certain cryptographic operations. Since such protocols have applications outside of electronic voting, we describe their design and implementation in GNUnet separately.

# Contents

# 1 Introduction

During the last decade, electronic voting has received a lot of negative attention. Strong criticism was directed at a very primitive notion of electronic voting, namely direct-record electronic voting, which aims to replace traditional, paper-based large-scale democratic elections by computerized voting terminals. Not only is the idea of trusting such a black-box voting machine in itself questionable; actual implementations of these systems were shown to exhibit critical security flaws [KSRW04, Oos10].

A more radical deviation from the concept of paper-based elections is *remote electronic voting*, which has been an active research area in the academic community at least since Chaum published his seminal paper about mix-nets [Cha81] in 1981. In these voting schemes, the voters prepare and submit a ballot on their own computing device. The use of cryptography makes it possible to guarantee ballot secrecy, to provide voters and external auditors a means to verify the correctness of the final tally, as well as other useful properties.

This, however, assumes that voters have basic technological literacy and control over their computing devices. With the recent revelations about the capabilities and activities of security agencies such as the NSA and GCHQ, the latter can not be taken for granted, which poses a problem for political elections.

Nevertheless, electronic voting is a tool that can be useful for fast and cost-effective democratic decisions outside the scope of large-scale political elections. The German Pirate Party, for example, uses an electronic voting software called LiquidFeedback[1] for internal decision-making. A distinguishing feature of LiquidFeedback is that voters can delegate their right to vote on certain topics to other voters. The system, however, does not guarantee secrecy of ballot: The vote of every member becomes publicly visible after voting concludes. To prevent fraud, voters are expected to check if their vote occurs correctly in the final list. Cryptographically secure voting schemes might be an attractive addition to such a system, enabling polls where secrecy of ballot is guaranteed and fraud is detected more easily. As these elections are low-stake, the risk of attacks on the voters' computing devices is less significant.

The same reasoning applies to decentralized social networks (such as the social messaging service being implemented in GNUnet [Tot13b]), where cryptographic voting protocols could be used to for decision-making and opinion polls in social groups, with a reduced chance of manipulation and surveillance when compared to black-box polls.

Unfortunately complete implementations of remote electronic voting schemes are not readily available, especially not in a form that is easy to integrate into existing applications.

Many cryptographic voting schemes build upon distributed public key operations (such as key generation and cooperative decryption). Especially distributed key generation is costly to execute and challenging to implement correctly, and thus often neglected in ex-

[1] `http://liquidfeedback.org/`

isting prototypes. Instead, some existing systems rely on trusted third parties or physical assumptions such as key generation ceremonies, where the organizers of the election are required to meet in person. This is often impractical for digital communities, as a trusted third party is generally unavailable and in-person meetings are too expensive.

This thesis describes the design and implementation of a cryptographically secure remote electronic voting system in GNUnet[2], a framework for secure peer-to-peer networking. Chapter 2 provides some background on the cryptographic primitives and protocols required in the following chapters. Chapter 3 describes the design, implementation and performance of the distributed key operations service in GNUnet. Chapter 4 discusses common cryptographic voting schemes and their properties, as well as existing implementations. We then discuss the design and implementation as well as performance characteristics of electronic voting in GNUnet.

[2] `https://gnunet.org`

# 2 Prerequisites

This chapter establishes some cryptographic primitives and protocols that are utilized by both the distributed key operations and electronic voting protocols. For a more in-depth introduction to cryptography, see for example the book by Katz and Lindell [KL08].

Note that parts of this chapter were already published in a seminar paper by this author [Dol13].

## 2.1 Public-Key Cryptography

In contrast to symmetric cryptography, where the same key is used for encryption and decryption of a message, public-key cryptography uses two keys: A *public key* to encrypt messages, and a corresponding *private key* to decrypt them.

### 2.1.1 Groups

Many modern public-key cryptosystems are based on problems assumed to be hard in finite groups. Let $g$ be a generator of the group $\mathcal{G} = \langle g \rangle$, and let $q$ be the order of $\mathcal{G}$.

**Hardness Assumptions** The first common hardness assumption concerns the discrete logarithm problem (DLP), which consists of solving the equation

$$g^x = h \tag{2.1}$$

for $x \in \mathbb{Z}_q$, written $x = \log_g(h)$. While exponentiation is easy to compute (for example with $O(\log n)$ group operations by repeated doubling), the DLP is believed to be intractable in most large groups, and modular exponentiation is conjectured to be a one-way function.[1]

The computational Diffie-Hellman assumption (CDH) [Bon98] is stronger[2] than the DLP, and states that $g^{ab}$ is hard to compute from only $g^a$ and $g^b$ for arbitrary $a, b \in \mathbb{Z}_q$.

The decisional Diffie-Hellman assumption (DDH) [Bon98] is stronger than CDH, and states that for uniformly random $a, b, c \in \mathbb{Z}_q$, the distributions of $(g^a, g^b, g^{ab})$ and $(g^a, g^b, g^c)$ are indistinguishable and thus $g^{ab}$ is pseudo-random even if $g^a$ and $g^b$ are known.

**Commonly used Group** The DDH is assumed to hold in a number of groups [Bon98]. We use the multiplicative subgroup $\mathcal{G}_q \subset \mathbb{Z}_p^*$ of large prime order $q$, where $p = 2q + 1$ is a safe prime.[3] This group is also known as the group of quadratic residues modulo a safe

[1] However, the DLP is still easy to solve for some groups. For example, when the order of $\mathcal{G}$ is smooth (only has small prime factors) the Pohling-Hellman algorithm [PH78] can be used to solve the DLP.

[2] In the sense that it is a stricter requirement which holds in fewer groups.

[3] This is a special case of a Schnorr group where $p = rq + 1$.

prime. Note that $\mathbb{Z}_p^*$ is the group of integers under multiplication modulo $p$ with order $p-1=2q$.

In practice, concrete values for the parameters $q, p$ can be found by generating a random prime $q$ of the desired size (depending on the required security, typically between $1024$ and $4096$ bits) and repeating the process until $2q+1$ is also prime. A generator $g$ of $\mathcal{G}_q$ can be computed as follows [MVO96, Section 4.6]:

1. Repeatedly choose an $\alpha \in \mathbb{Z}_p^*$ at random, until it satisfies $\alpha^q \neq 1$ and $\alpha^2 \neq 1$, that is, the order of $\alpha$ is neither $q$ nor $2$ nor $1$. Then $\alpha$ is a generator of $\mathbb{Z}_p^*$.

   *Proof:* By Lagrange's Theorem, $\mathbb{Z}_p^*$ has exactly two proper non-trivial subgroups of order $q$ and $2$, respectively. As $\alpha$ is neither of order $q$, $2$ nor $1$, it can only be a generator of the full group $\mathbb{Z}_p^*$.

2. Compute $g = \alpha^k$, where $k := (p-1)/q = 2$. Then $g$ is a generator of $\mathcal{G}_q$.

   *Proof:* Let $ord(\cdot)$ be the order of a group element. As $k$ divides $ord(\alpha)$, it follows from a standard result of group theory [Jud94, Proposition 4.5] that $ord(\alpha^k) = ord(\alpha)/k = q$.

Other commonly used groups are obtained from elliptic curves. A bijection between elements of an elliptic curve group and $\mathbb{Z}_q$ is usually hard to construct (but possible [BHKL13]), making it challenging to encode messages as group elements. Consequently some common cryptosystems (such as Elgamal [ElG85] and Paillier [Pai99]) are not easily usable with elliptic curves.

### 2.1.2 Diffie-Hellman Key Exchange

The Diffie-Hellman key exchange [DH76] is used to establish a shared secret between two parties (let us call them Alice and Bob), where Alice has a secret key $a \in_R \mathbb{Z}_q$ and Bob has secret key $b \in_R \mathbb{Z}_q$. The corresponding public keys $A = g^a$ and $B = g^b$ are known to both Alice and Bob.[4]

A shared secret $s \in \mathcal{G}$ only known to Alice and Bob can be established in the following way: Bob computes $s = A^b = (g^a)^b = g^{ab}$ and Alice computes $s = B^a = (g^b)^a = g^{ab}$.

### 2.1.3 The Elgamal Scheme

The Elgamal cryptosystem [ElG85] is obtained by a straightforward application of the Diffie-Hellman key exchange. The main purpose of Elgamal is to encrypt messages with a public key $h = g^x$ such that only the owner of the private key $x \in_R \mathbb{Z}_q$ can decrypt them.

A message $\bar{m}$ is then encrypted in the following way: First, $\bar{m}$ is encoded as a group element $m \in \mathcal{G}$. Then the sender creates a random ephemeral private key $\alpha \in_R \mathbb{Z}_q$. The encryption of $\bar{m}$ is then the pair $(c_1, c_2) = (g^\alpha, sm)$ where $s = h^\alpha$. Note that $c_1$ can be interpreted as the ephemeral public key corresponding to $\alpha$.

The receiver can than use his private key $x$ to compute $s = c_1^x = g^{\alpha x} = h^\alpha$. By multiplying $c_2$ with $s^{-1}$ (the modular inverse of $s$ in $Z_q$), the encoded message $m$ is obtained and consequently $\bar{m}$ can be decoded.

[4] These public keys should be exchanged or verified over a secure channel; otherwise the Diffie-Hellman key exchange is susceptible to a man-in-the-middle attack.

Elgamal is only semantically secure (in the sense that an attacker cannot even obtain partial information about the plaintext corresponding to a ciphertext) if the decisional Diffie-Hellman assumption holds in the underlying group $\mathcal{G}$.

**Homomorphic Properties** Let $E(\cdot)$ and $D(\cdot)$ be Elgamal encryption and decryption respectively. The following properties hold:

$$D(E(x) \cdot E(y)) = x \cdot y \tag{2.2}$$

$$D(E(x^a)) = D(E(x)^a) \quad \text{for all } a \in \mathbb{Z}_q \tag{2.3}$$

This can be easily derived from the equations for encryption and decryption.

While this kind of ciphertext malleability is usually not desired (and can be prevented by padding a message with random data), it is useful in some cryptographic protocols, as computations can be done on ciphertexts without revealing the plaintext. Additive homomorphic encryption allows addition, while multiplicative homomorphic encryption allows multiplication. Fully homomorphic encryption allows both addition and multiplication to be performed over encrypted data [Gen09].

## 2.2 Secret Sharing

Secret sharing is an important building block for cryptosystems with multiple parties, where single parties are generally not trusted.

### 2.2.1 The Scenario

In a secret sharing protocol, a *dealer* holds a *secret* that he wishes to share[5] with a group of $\ell$ players, in such a way that at least $k$ players must cooperate to recover the secret. He does so by dividing the secret $s$ into $\ell$ *shares* $D_1, \ldots, D_\ell$, such that at least $k$ different shares are necessary to restore $s$, and a group of players smaller $k$ is not able to gain any information about the secret. The shares are then sent over a private channel to the intended recipients. Note that secret sharing with a dealer does not suffice for establishing a distributed key without a trusted third party, as in this scheme the dealer must know the secret.

Three common approaches exist for secret sharing [BKS09]. The most well known scheme is *Shamir's secret sharing*, a scheme based on polynomial interpolation over finite fields. Other constructions include Blakley's geometric approach using the intersections of hyperplanes [Bla99], and the Asmuth-Bloom construction [AB83] that relies on the Chinese remainder theorem. In this thesis we will focus on Shamir's scheme.

### 2.2.2 Shamir's Secret Sharing Scheme

Shamir's secret sharing [Sha79] utilizes the fact that in order to fully determine a polynomial of degree $k-1$, at least $k$ different data points have to be known. In order to make the

[5] In contrast to colloquial usage of the term "sharing", we do not interpret sharing as replication.

memory requirements of the scheme feasible and ensure information-theoretic security[6], computations are done in a finite field of appropriate size to fit the secret.

The dealer wishing to share the secret $s$ chooses a polynomial $f$ of degree $k-1$ with random coefficients $a_1, \ldots, a_{k-1}$ and $a_0 = s$:

$$f(X) = \sum_{0 \leq i < k} a_i X^i \tag{2.4}$$

Consequently, $f(0) = s$ and at least $k$ data points are required to fully determine $f$. The dealer can now evaluate the polynomial at $\ell \geq k$ distinct points to obtain the shares $D_i = f(i)$ for $i \in \{1, \ldots, \ell\}$, where the share $D_i$ is intended for the $i$-th player.

In order to restore the secret, at least $k$ of these shares must be known in order to interpolate the polynomial and obtain the secret by evaluating $f(x)$ at $x = 0$.

A polynomial can be interpolated using Lagrange's formula. The secret $s$ is restored with

$$s = \sum_{j \in \Lambda} D_j \lambda_{j,\Lambda} \tag{2.5}$$

where the Lagrange coefficients are

$$\lambda_{j,\Lambda} := \prod_{\substack{l \in \Lambda \\ l \neq j}} \frac{l}{l-k}. \tag{2.6}$$

and $\Lambda$ is the set of indices for the available shares and the arithmetic operations are done over $\mathbb{Z}_q$.

### 2.2.3 Verifiable Secret Sharing

The above scheme does not preclude the dealer from sending inconsistent shares to the other players. The dealer could, for instance, choose the coefficients such that subset $A$ of the players would reconstruct a different secret than subset $B$.

A technique due to Feldman [Fel87] uses a *masking* and *binding* commitment in order to ensure that the dealt shares are consistent, that is, each sufficiently large subset of shares will yield the same secret upon reconstruction.

The dealer commits to each $a_i$ for $0 \leq i \leq k-1$ by publishing the value $A_i := g^{a_i}$. This allows the consistency of commitments to be verified, as each receiver of a share $D_i$ can check that

$$g^{D_i} = \prod_{0 \leq j < k} A_j^{i^j}. \tag{2.7}$$

The previous equation follows from raising both sides of the formula for evaluating the polynomial $f$ over $g$, namely

$$D_i = f(i) = \sum_{0 \leq j < k} a_j i^j. \tag{2.8}$$

No useful information is leaked by the commitments if in the underlying group the decisional Diffie-Hellman assumption holds.

[6] In non-finite fields, partial information about one share can be computed from only a subset of the shares

## 2.3 Zero-Knowledge Proofs

In a zero-knowledge proof, a *prover* wants to convince a *verifier* that a given statement is true, without revealing any information that the *verifier* did not have before [GO94].

All zero-knowledge proofs given here are interactive, three-round protocols consisting of a *commitment*, *challenge* and *response* round.

### 2.3.1 Knowledge of Discrete Logarithms

A well-known zero-knowledge proof, known as Schnorr's protocol, convinces the verifier that the prover knows the discrete logarithm $x$ of a value $h = g^x$ known to the verifier [Sch90]. Many other zero-knowledge proofs presented here will build upon this zero-knowledge proof.

**Start** The verifier knows $h = g^x$ and the prover additionally knows $x$.

**Commitment** The prover chooses $k \in_R \mathbb{Z}_q$ and sends $t := h^k$ to the verifier.

**Challenge** The verifier computes and sends to the prover the challenge $c \in_R \mathbb{Z}_q$.

**Response** The prover computes $r \in \mathbb{Z}_q$ as $r := k + xc$, using her knowledge of $x$. She then sends $r$ to the verifier.

**Verification** The verifier checks whether $h^r \stackrel{?}{=} th^c$.

As the protocol requires interaction, it would be cumbersome for the prover to convince multiple verifiers that she knows a discrete logarithm. However, the Fiat-Shamir trick [FS87, GK03] can be used to transform the proof into a non-interactive zero-knowledge proof (NIZK). Instead of requiring the verifier to generate a challenge each time, the challenge is derived off-line by hashing the previous protocol transcript — the prover computes the challenge as $c = H(h, z, k, t)$. Note that this transformation can only be proven secure under the *Random Oracle Model*, where $H$ is replaced by a truly random function.

### 2.3.2 Knowledge of Discrete Logarithm Equality

A similar protocol [CP93] can be used to prove the *equality* of two logarithms to different bases, i.e. that

$$\alpha = \log_g x = \log_h y$$

where $x, y \in \mathbb{Z}_q$ and $g, h \in \mathbb{Z}_p^*$ are known, but $\alpha \in \mathbb{Z}_q$ is only known to the prover.

**Start** The verifier knows $x$, $y$, $g$, $h$ and the prover additionally knows $\alpha$.

**Commitment** The prover chooses $w \in_R \mathbb{Z}_q$ and sends $(a, b) := (g^w, h^w)$ to the verifier.

**Challenge** The verifier computes and sends to the prover the challenge $c \in_R \mathbb{Z}_q$.

**Response** The prover computes $r \in \mathbb{Z}_q$ as $r := w + \alpha c$, using her knowledge of $\alpha$. She then sends $r$ to the verifier.

**Verification** The verifier checks whether $h^r \stackrel{?}{=} by^c$ and $g^r \stackrel{?}{=} ax^c$.

Note that this protocol is only honest-verifier zero knowledge [CP93], meaning that a dishonest verifier could choose the challenge in such a way that information about $\alpha$ is revealed. Therefore applying the Fiat-Shamir trick to the protocol is essential, as now the verifier cannot request a specially crafted challenge anymore.

As an example application, this proof can show that an Elgamal pair $(x, y)$ encrypts the message $m = g^0 = 1$, without the verifier being able to run the decryption protocol directly (as the verifier does not know $\alpha$).

This obviously is not immediately useful. However, a construction such as [CDS94] can be employed to obtain a *disjunctive* zero knowledge proof for discrete logarithm equality. That is, a statement of the following form (with $g, h \in \mathbb{Z}_p^*$ and $x, m_i, y_i \in \mathbb{Z}_q$ for $1 \leq i \leq n$).

$$\alpha = \log_h x = \log_g(y_1 m_1) \quad \vee \quad \ldots \quad \vee \quad \alpha = \log_h x = \log_g(y_n m_n)$$

can be proven in zero knowledge. This now has a useful application: It can convince the verifier that a ciphertext is an encryption of one of $n$ alternatives from $M = \{m_1, \ldots, m_n\}$. For an intuitive explanation of this technique, as well as other discrete logarithm zero knowledge proofs, see [CS97].

Let $v$ be the index of the message $m_v$ that is actually encrypted in $(x, y)$. The basic idea is to *simulate* $n - 1$ proofs for the values that are not encrypted in $(x, y)$ by selecting a commit/challenge/response triple $((a_i, b_i), d_i, r_i)$ for $i \neq v$ which passes the verification. Finding such a triple is trivial if the prover can select the challenge freely. For the remaining disjunction, the challenge is derived from the actual challenge $c$ as $d_v = c - \sum_{i \neq v} d_i$. Due to this construction, exactly one of the $n$ proofs cannot be simulated, as exactly one challenge for a sub-proof cannot be freely selected.

**Start** The verifier knows $M = \{m_1, \ldots, m_n\}$, $x = g^\alpha$, $y = h^\alpha m_v$, $g$, $h$ and the prover additionally knows $\alpha$ as well as $v \in \{1, \ldots, n\}$.

**Commitment** The prover chooses $w \in_R \mathbb{Z}_q$ and sets $(a_v, b_v) := (g^w, h^w)$. For $i \neq v$, the prover simulates the sub-proofs $((a_i, b_i), d_i, r_i)$ with $r_i \in_R \mathbb{Z}_q$, $d_i \in_R \mathbb{Z}_q$ and $a_i := g_i^r x^{d_i}$, $b_i := h^r (y m_i^{-1})^{d_i}$. All pairs $(a_i, b_i)$ for $1 \leq i \leq n$ are sent to the verifier.

**Challenge** The verifier computes and sends to the prover the challenge $c \in_R \mathbb{Z}_q$.

**Response** The prover computes the challenge for the remaining sub-proof as $d_v := c - \sum_{i \neq v} d_i$. She then computes $r_v := w - \alpha d_v$ and sends $(r_i, d_i)$ for $1 \leq i \leq n$ to the verifier.

**Verification** The verifier checks that $c \stackrel{?}{=} \sum d_i$ and $(a_i, b_i) \stackrel{?}{=} (g^{r_i} x^{d_i}, h^{r_i} (y m_i)^{d_i})$.

Note that again the Fiat-Shamir trick should be applied, as the proof is only honest-verifier zero knowledge.

### 2.3.3 Fair Paillier Encryption

The Paillier cryptosystem [Pai99] has the same basic use as Elgamal, but exhibits different homomorphic properties, which are useful for certain zero knowledge proofs. In contrast to Elgamal, Paillier only works in very specific groups.

In particular, Paillier allows for a *proof of fair encryption* [FS01] in zero knowledge. This proof convinces a third party who knows $g^x$ that the logarithm $x \in \mathbb{Z}_q$ can be restored from a ciphertext that encrypts $x$ by the intended recipient (using his private Paillier key). Note that this is slightly different from a proof of *correct* encryption, as it only guarantees that $x$ can be *restored* with relatively high computational effort of the recipient, but not that $x$ has been encrypted correctly. A more direct proof of correct encryption is not yet known to exist.

#### The Paillier Cryptosystem

We now describe the simplified Paillier scheme as introduced by Katz et al. [KL08]. Let $N = pq$, where $p, q$ are primes of the same length. Then the following holds:

1. $\gcd(N, \phi(N)) = 1$.
2. $(1+N)^a = (1+aN) \bmod (N^2)$ for $0 \leq a \leq N$.
3. $\mathbb{Z}_N \times \mathbb{Z}_N^*$ is isomorphic to $\mathbb{Z}_{N^2}^*$, and

$$f : \mathbb{Z}_N \times \mathbb{Z}_N^* \to \mathbb{Z}_{N^2}^*$$
$$f(a,b) = (1+N)^a \cdot b^N \bmod N^2$$

   a bijection [KL08]. It is conjectured that $f$ is a one way function.

We call $y \in \mathbb{Z}_{N^2}^*$ an $N^{\text{th}}$ residue modulo $N^2$ if there is an $x$ such that $y = x^N \bmod N^2$. Note that is is the case for all $f(0,b) \in \mathbb{Z}_{N^2}^*$ with $b \in \mathbb{Z}_N^*$ [KL08].

The security of Paillier can be reduced to the hardness of distinguishing $f(0,r)$ from $f(r',r)$ for random $r' \in \mathbb{Z}_N$ and $r \in \mathbb{Z}_N^*$.

**Keys.** The public key is $(G, N) = (N+1, N)$, and the private key is $\phi(N) = (p-1)(q-1)$, where $\phi$ is Euler's totient function.

**Encryption.** A message $m \in \mathbb{Z}_n$ is encrypted as

$$c = f(m,1) \cdot f(0,r) = f(m,r) \tag{2.9}$$

for a random $r \in \mathbb{Z}_N^*$; this implies $\gcd(r, N) = 1$ and thus according to Euler's theorem $r^{\phi(N)} = 1 \mod N$.

**Decryption.** Let

$$\hat{c} := c^{\phi(N)} \bmod N^2 \tag{2.10}$$

and note that

$$\begin{aligned}\hat{c} &= f^{\phi(N)}(m, r) \\ &= f(m \cdot \phi(N) \bmod N, r^{\phi(N)} \bmod N) \\ &= f(m \cdot \phi(N) \bmod N, 1) \\ &= (1+N)^{m \cdot \phi(N) \bmod N} \\ &= (1 + (m \cdot \phi(N) \bmod N) \cdot N) \bmod N^2,\end{aligned} \tag{2.11}$$

and thus the message can be restored with

$$m = \frac{\hat{c} - 1}{N} \cdot \phi(N)^{-1} \bmod N, \tag{2.12}$$

where the fraction denotes integer division.

### Proof of Fair Encryption

We now give the proof of fair encryption [FS01, PS00], reformulated in the simplified Paillier scheme.

Let $\langle g \rangle = \mathcal{G}_{q'} \subset \mathbb{Z}_{p'}^*$ be the group of quadratic residues modulo a safe prime $p'$ of order $q'$, and $y = g^x \bmod p'$ with $x \in \mathbb{Z}_{q'}$. Additionally let $H$ be a hash function which produces a value in $[0, B)$. Fix $A$ such that $A \geq B \cdot q' + k'$ holds for a security parameter $k'$.

The proof of fair encryption convinces a verifier that, using the Paillier private key $\phi(N)$, it is possible to find values $\sigma < A$ and $\tau < B$ such that $\sigma = \tau x \bmod q'$ and $G^\sigma Y^{-\tau} = f(0, \kappa)$ for some $\kappa$. These two equalities will later be used in the reconstruction of $x$.

The proof consists of the following steps executed by the verifier and prover, and fails with probability smaller than $1 - 2^{-k'}$ for an honest prover [FS01].

**Start** The verifier knows the encryption $Y = f(x, u)$, the group element $y = g^x$ and the Paillier public key $(G, N)$. The prover additionally knows $x$.

**Commitment** The prover chooses random $r \in_R [0, A)$ and random $s \in \mathbb{Z}_N^*$. He then sends $t := (g^r \bmod p, G^r s^N \bmod N^2)$ to the verifier.

**Challenge** The verifier sends to the prover the randomly selected challenge $e \in_R \mathbb{Z}_q$.

**Response** The prover computes $z := r + ex$ and $w := su^e \bmod N$ and sends $(z, w)$ to the verifier.

**Verification** The verifier checks whether $z < A$, $g^r \stackrel{?}{=} g^z y^{-e} \bmod p$ and $G^r s^N = G^z w^N Y^{-e} \bmod N^2$.

The protocol is made non-interactive with the Fiat-Shamir trick by using $e = H(g, N, y, Y, g^r \bmod p, G^r s^N \bmod N^2)$. Due to the failure probability for an honest prover, the generation of the non-interactive proof must be repeated with different values for $t$ until $z < A$.

**Reconstruction.** Let $\tilde{x}$ be the decryption of $Y$. If $g^{\tilde{x}} = y \bmod p'$, then $x = \tilde{x}$ and no reconstruction is necessary. Otherwise, $x$ can be reconstructed from $N$ and $\tilde{x}$ if the proof of fair encryption was valid (see [PS00] for an algorithm to compute $Y$ that make reconstruction necessary).

From the proof of fair encryption it follows that

$$\begin{aligned} f(0,\kappa) &= G^{\sigma} Y^{-\tau} \\ &= G^{\sigma}(G^{x}u^{N})^{-\tau} \\ &= f(\sigma,1)\cdot f(x,u)^{-\tau} \\ &= f(\sigma,1)\cdot f(-\tau x, u^{-\tau}) \\ &= f(\sigma-\tau x, u^{-\tau}) \end{aligned} \tag{2.13}$$

and thus $\sigma - \tau\tilde{x} = 0 \bmod N$.

Finding $\sigma$ and $\tau$ can thus be reduced to the shortest vector problem for the lattice with base $((N,0)^T, (\tilde{x},1)^T)$. From $\tau$ and $\sigma$, the discrete logarithm $\log_g y = x$ can then be restored as $x = \sigma\tau^{-1} \bmod q'$.

In practice, for finding $\sigma$ and $\tau$ the Gauss algorithm[Coh93, Algorithm 1.3.14] can be used, which finishes in $O(\log N)$ iterations.

#### Gauss Algorithm for Discrete Logarithm Reconstruction

Let $\vec{a} = (N,0)^T$ and $\vec{b} = (\tilde{x},1)^T$ be two linearly independent vectors in an Euclidean vector space. The following algorithm (sometimes called the Gauss algorithm [Coh93, Algorithm 1.3.14]) finds the shortest[7] non-zero vector $\vec{z} = (\sigma,\tau)^T$ such that $c_1\vec{a} + c_2\vec{b} = \vec{z}$ for some $c_1, c_2 \in \mathbb{Z}$:

1. Set $A := \|\vec{a}\|$, $B := \|\vec{b}\|$. If $A < B$, exchange $A$ with $B$ and $\vec{a}$ with $\vec{b}$.
2. Set $n := a_1b_1 + a_2b_2$, $r := \lfloor n/B + 1/2 \rfloor$ and $T := A - 2rn + r^2B$.
3. If $T \geq B$ then output $\vec{z} := \vec{b}$ and terminate. Otherwise, set $\vec{t} := \vec{a} - r\vec{b}$, $\vec{a} := \vec{b}$, $\vec{b} := \vec{t}$, $A := B$, $B := T$ and go to step 2.

[7] For some given norm. For our purposes, we use the norm $\|\vec{z}\| = z_1^2 + z_2^2$

# 3 Distributed Key Generation and Cooperative Decryption

In traditional asymmetric cryptostems, a single party maintains ownership of a secret key used for signing and/or encryption. This is unacceptable in distributed computations, where there is commonly no single party that is trusted to be both honest and sufficiently immune to compromise.

Threshold cryptography is an approach to solve this problem. It replaces the trust in a single party with trust in the assumption that no large subset of the participants will be compromised by the same party or form a collusion. We only consider a $(k, \ell)$ threshold scheme, where at least $k$ of $\ell$ parties must cooperate in order to restore the secret key. The parties $P_1, \ldots, P_\ell$ of the group are known at the start of the protocol; the group cannot change during the protocol.

While threshold cryptosystems exist for a variety of cryptographic operations, we only focus on the operations immediately relevant for the implementation of electronic voting: Distributed key generation (DKG) and cooperative decryption. The former is necessary as no single party can be trusted to "forget" a secret key after generating and distributing it. The latter protocol enables the decryption of a message which has been encrypted with the group's public key, without allowing recovery the secret key, as this would again expose the secret key to a single party.

This chapter first explains the principle behind the threshold cryptography required for some electronic voting schemes. After giving the basic scheme of Pedersen [Ped91], we consider two advanced protocols due to Gennaro [GJKR99] and Fouque [FS01] respectively, which aim to fix problems with Pedersen's scheme. We proceed to discuss our implementation of Fouque's scheme in GNUnet and give some performance figures.

Some of the material in this chapter has been adapted from a seminar paper written by this author [Dol13].

## 3.1 Background

Elgamal [ElG85] is used as the underlying cryptosystem for the threshold schemes we discuss here (see section 2.1.3). A basic building block for Pedersen's and Fouque's distributed key generation schemes is Feldman's Verifiable Secret Sharing scheme [Fel87]. The cooperative decryption relies on a zero-knowledge proof for the equality of logarithms (see section 2.3.1).

Fouques scheme additionally requires Pailliers cryptosystem [Pai99] and a zero-knowledge proof of fair encryption; both are described in section 2.3.3.

## 3.2 Pedersen's Distributed Key Generation

The basic idea of Pedersen's DKG protocol [Ped91] is that each player $P_i$ deals a pre-secret $x_i$ using the Feldman Verifiable Secret Sharing (VSS) scheme [Fel87]. The resulting shares of all pre-secrets are then re-combined to yield a threshold-shared secret that is not known to any of the participants. The values published as a by-product of the Feldman VSS can be used to compute the public key $h = g^x = \prod g^{x_i}$ corresponding to the secret-shared private key $x = \sum x_i$.

### 3.2.1 Basic Protocol

Let $f$ be a polynomial of degree $k-1$ where $x = f(0)$ is the private key that will be jointly generated by and threshold-shared among the group, in a matter such that $P_i$ learns $f(i)$ for $i \in \{1, \ldots, \ell\}$.

First, each player $P_i$ generates a secret pre-share $x_i \in \mathbb{Z}_q$, and deals it with Feldman's VSS protocol (see section 2.2.3) This results in $\ell$ polynomials $\phi_1, \ldots, \phi_\ell$, where $\phi_i$ represents the threshold-shared pre-secret of player $P_i$.

Player $P_i$ now knows

- his own pre-secret $x_i = \phi_i(0)$ and
- one share $\phi_j(i)$ of the pre-secret $x_j$ for each player $P_j$ with $j \in \{1, \ldots, \ell\}$.

A player $P_i$ can now recombine his shares of the pre-secret to a share $f(i)$ of the public key $x = f(0)$, as

$$f(i) = \sum_{1 \le j \le \ell} \phi_j(i). \tag{3.1}$$

From that it follows that

$$x = f(0) = \sum_{1 \le j \le \ell} \phi_j(0) = \sum_{1 \le j \le \ell} x_j. \tag{3.2}$$

As a by-product of the Feldman VSS protocol, each player has computed and published $g^{\phi_i(0)} = g^{x_i}$. These values are utilized to compute the public key $h = g^x$ as

$$h = g^x = g^{\sum_{1 \le i \le \ell} x_i} = \prod_{1 \le i \le \ell} g^{x_i}. \tag{3.3}$$

### 3.2.2 Handling Complaints

If a player receives no shares or a share that does not pass verification, he has to file a complaint against the faulty player. However, it is also possible for complaints to be filed fraudulently against an honest player. A possible strategy for handling complaints is the following [FS01]:

- If at least $k$ complaints are filed against player $P_i$, he must be faulty and is disqualified, as we can assume that no threshold of players files a fraudulent complaint.

- A player with less complaints reveals all shares he sent to a complaining player. If the shares do not match the commitment, the player is disqualified. Otherwise, less than $k$ shares of the player's pre-secret have been revealed, and the scheme is still secure.

Note that this can only work when filing and resolving complaints is done over a reliable broadcast channel, as peers much reach (Byzantine) consensus on the complaints and the corresponding answers.

#### Problems with Pedersen's Protocol

In the asynchronous or bounded synchronous communication model, Pedersen's protocol is susceptible to an attack by a *rushing adversary*. A rushing adversary is able to wait for and observe values sent to him before contributing his own values.

A coalition of $t$ malicious peers can influence the distribution of the public key in the following way: The malicious peers wait until all other peers revealed their values they previously committed to. The malicious coalition can then choose between $2^t$ different public keys by deciding for each of the malicious $t$ players whether they will reveal his values or drop out of the protocol.

A modified version of the attack [GJKR99], which requires $t > 1$, also works in the synchronous model (which precludes rushing adversaries). The malicious players each distribute $k - 1$ incorrect shares to the honest players, which will not yet disqualify them. The malicious coalition can then decide for each malicious peer whether an additional complaint will be filed against him, which will disqualify him. In this version of the attack, it is not clear who played incorrectly, as complaints could also be fraudulent.

It is not fully clear to what extent the rushing attack is purely theoretical. While the distribution of keys is indeed biased (causing problems with some security proofs), the attacker can only select between a number of public keys that is negligible small compared to the total number of public keys. Gennaro et al. have shown [GJKR03] that some protocols are still provably secure when Pedersen's key generation protocol is used.

## 3.3 Gennaro's Protocol

Gennaro's protocol [GJKR99] is a modification on Pedersen's protocol that repairs the possible bias on the public key distribution introduced by malicious players.

The key insight of Gennaro's scheme is to mask the group's public key until all shares have been exchanged. The protocol introduces a second round, where all players that completed the first round without being disqualified reveal information that allows all players to compute the group's public keys. The attack where malicious players do not contribute their share in the second round is addressed by a reconstruction algorithm run by the honest peers.

Gennaro's protocol is secure against rushing adversaries, and a malicious collusion of peers cannot use the handling of complaints to bias the distribution of the public key [GJKR03].

## 3.4 Fouque's Protocol

Fouque's protocol [FS01] is another variation of Pedersen's protocol, where communication over private channels is replaced by reliable broadcast and fair encryption. This approach has the advantage that it is easy to verify that all parties played correctly, making the handling of complaints unnecessary.

### 3.4.1 The Protocol

Each player $P_i$ generates random coefficients for a polynomial

$$f_i(X) = \sum_{0 \le j < k} a_{i,j} X^k \in \mathbb{Z}_q[X] \tag{3.4}$$

and a Paillier key pair $(SK_i, PK_i)$.

**First Round.** Each player $P_i$ for $1 \le i \le \ell$ broadcasts his Paillier public key $PK_i$ as well as a commitment $g^{f_i(0)}$ to his pre-share $f_i(0)$. Note that the original description omits the commitment, which is necessary to guarantee security against a rushing adversary without a incoercible third party (see section 3.4.2).

**Second Round.** Each player $P_i$ for $1 \le i \le \ell$ broadcasts the following values

- $A_{i,j} = g^{a_{i,j}}$ for $0 \le j \le k$
- $y_{i,j} = g^{f_i(j)}$ for $1 \le j \le \ell$
- $Y_{i,j} = \mathsf{Enc}_{PK_j}(f_i(j)) = G_j^{f_i(j)} u_{i,j}^{N_i} \mod N_j^2$ for $1 \le j \le \ell$ and random $u_{i,j} \in \mathbb{Z}_N^*$.
- A zero-knowledge proof $(e_{i,j}, w_{i,j}, z_{i,j})$ that proves that $f_i(j)$ can be restored from $Y_{i,j}$ by $P_j$ for $1 \le j \le \ell$. See section 2.3.3 for the description of the zero-knowledge proof.

**Verification.** All players must check all zero-knowledge proofs (even the ones that do not prove anything about the values encrypted with their own public key) and verify that

$$\prod_{k=0}^{t} A_{i,j}^{j^k} = y_{i,j}. \tag{3.5}$$

for all $1 \le i \le \ell$ and $1 \le j \le \ell$.

All players that pass the verification are in the set $QUAL$, which is the same for all honest players.

**Shares and Public Key.** The share of $P_j$ for the private key $x = \sum_{i \in QUAL} f_i(0)$ is computed as $s_j := \sum_{i \in QUAL} f_i(j)$.

The public key $h = g^x$ is computed as $h = \prod_{i \in QUAL} A_{i,0}$.

### 3.4.2 Defense against Rushing Adversaries

The rushing attack against Pedersen's protocol also applies to Fouque. To defend against it, Fouque suggests employing an *incoercible player*. The incoercible player is essentially a trusted third party that supplies a uniformly distributed random value that can neither be influenced nor predicted by any of the players. The incoercible player will provide an additional share *after* all other players provided their values in round two.

While in practice an incoercible player may be hard to attain, it is possible to simulate one with a collective coin flipping protocol [BOL90], executed as a third round.

## 3.5 Cooperative Decryption

After executing the distributed key generation protocol, a sufficiently large set of players could cooperate to restore the secret key in order to use it for a signing or decryption operation. This, however, is not desirable, as then some parties would know the full key, and could use it for further operations without any cooperation from the rest of the players.

Fortunately, it is possible to combine Lagrange interpolation with Elgamal decryption to obtain an algorithm which allows the group of players to decrypt a value without recovering the secret key $x$ [CGS97].

From the equation for Elgamal decryption

$$m = c_2 \cdot s^{-1}, \tag{3.6}$$

we obtain

$$m = c_2 / \prod_{j \in \Lambda} w_j^{\lambda_{j,\Lambda}}, \tag{3.7}$$

where $w_j = c_1^{s_j}$ is a *partial decryption* that is computed by each participating player and can only be used to decrypt one specific message.

## 3.6 Robust Decryption with Zero-Knowledge Proofs

While the generation of the key is verifiable, a malicious player $\tilde{P}_i$ could still disrupt the decryption protocol by contributing a wrong value for $w_i$. To prevent this, each player must construct a non-interactive zero-knowledge proof for the fact that $w_i$ has been computed correctly, i.e. that

$$s_i = \log_g g^{s_i} = \log_{c_1} w_i. \tag{3.8}$$

In other words, the $P_i$ has to prove that that $w_i$ has actually been computed with the player's share.

For this, the Chaum-Pedersen protocol can be used, which builds on the Schnorr-protocol [Sch90] (see also section 2.3):

**Start** The prover knows her share $s_j$ and the verifier knows the ciphertext $(c_1, c_2)$

**Commitment** The prover sends $(g^\beta, c_1^\beta)$ to the verifier, with $\beta \in_R \mathbb{Z}_q$.

**Challenge** The verifier sends to the prover the challenge $\gamma \in_R \mathbb{Z}_q$.

**Response** The prover computes $r := \beta + s_i\gamma$, using her knowledge of $s_i$. She then sends $r$ to the verifier.

**Verification** The verifier checks the following equations:

$$g^r \stackrel{?}{=} g^\beta (g^{s_i})^\gamma \tag{3.9}$$

$$c_1^r \stackrel{?}{=} c_1^\beta w_i^\gamma \tag{3.10}$$

Note that this protocol is only zero-knowledge under the assumption that the verifier is honest, which requires either the Fiat-Shamir trick to be applied, or access to a shared source of randomness.

## 3.7 Implementation

We implemented Fouque's protocol and robust cooperative decryption for GNUnet in form of the DKO (distributed key operations) service. Instead of using reliable broadcast, values are communicated with the CONSENSUS service, which implements Byzantine agreement on sets. Its implementation is described in a separate report [Dol14].

In our implementation, the size of the generated Elgamal public key is fixed to 1024 bits. We use Paillier with 2048-bit keys.

### 3.7.1 Key Generation

The generation of a new threshold key is initiated by all peers calling `GNUNET_DKO_create_session` with the same value for

- the list of other peers in the group,
- the threshold value $k$,
- a unique session identifier (which allows the same group of peers to have multiple sessions and thus generate multiple threshold-shared keys simultaneously),
- the time when key generation should start[1] and end.

Peers that are in the group but do not participate are considered faulty.

When the key generation concludes, every peer receives a share and the group's public key, which can be used to encrypt messages.

[1] This value is passed to the CONSENSUS service. If the start time is in the future, the key generation will be delayed to the start time. If the start time is in the past, the local peer will skip some consensus rounds, and the consensus might still succeed.

### 3.7.2 Decryption

Decryption is initiated by calling `GNUNET_DKO_decrypt` and providing

- the value to encrypt,
- the peer's share of the group secret and
- the time when decryption should start[2] and end.

The decryption does not have a dedicated session identifier, as the decryption session is already identified by the value that is to be decrypted. If a peer requests decryption and less than $k$ other peers request decryption of the same value in the same interval, the decryption will time out and fail.

### 3.7.3 Limitations of the Implementation

The current implementation does not simulate the incoercible third party. This reduces the security of the protocol to the security of Pedersen's protocol. We argue, however, that it is still favorable to implement this version of the protocol, as the communication between peers (especially for the treatment of complaints) is highly simplified.

Gennaro et al. [GJKR03] have shown that even Pedersen's protocol is secure for some applications. It is an open question if and to what degree an attacker could exploit the rushing attack to compromise the voting system discussed in chapter 4.

### 3.7.4 Performance Evaluation

The total number of bytes broadcast during the distributed key generation (including the EdDSA signatures on the consensus set elements) is $528\ell + 120k\ell + 1796\ell^2$, where $\ell$ is the number of peers and $k$ is the threshold. The total size the of zero-knowledge proofs, which grows quadratically with $\ell$, limits the maximum number of peers to $\ell \leq 11$, as a single consensus set element may not exceed 64 Kilobytes in the current implementation of the CONSENSUS service. For a single decryption, only $872\ell$ bytes are broadcast.

We simulated the execution of the key generation protocol using GNUnet's testbed infrastructure [Tot13a]. We measured the CPU user time (shown in table 3.1 taken by *gnunet-service-dko* with the `time` command commonly available on GNU/Linux systems.

Note that the measurements only include cryptographic operations executed in the DKO service, and do not take the resources required by other GNUnet components into account. We do not give measurements for the execution time of the full protocol including communication, as the underlying communication primitive (CONSENSUS) is not implemented in the fully robust (i.e. Byzantine fault tolerant) version. Furthermore the execution time is highly dependent on the performance of the underlying transport protocol as well as the CADET service[3], whose implementation is still evolving.

The large standard deviation of measured user time can be explained by the large variance of the number of context switches (obtained with `time -f '%c'`). While the context switches itself are not accounted for in the user time, a higher number of cache misses

[2] See footnote 1.

[3] GNUnet's end-to-end transport protocol, comparable to a TCP/IP for P2P networks [PG14].

| group size $\ell$ | user time for key generation | CPU user time for key generation |
|---|---|---|
| 3 | $1\,s \pm 0.2\,s$ | $1.1\,s \pm 0.2\,s$ |
| 5 | $1.25\,s \pm 0.36\,s$ | $1.36\,s \pm 0.40\,s$ |
| 10 | $1.92\,, s \pm 0.31\,s$ | $2.0\,s \pm 0.36$ |

Table 3.1: Performance of distributed key generation measured with TESTBED on a 4-core Intel i5 3.3 GHz processor. The standard deviation of the measurements is shown in parentheses. For the threshold, $k = \lceil 2\ell/3 \rceil$ was chosen. Measurements were repeated and averaged over 10 executions.

due to context switching could be responsible for the higher execution times. In our experiments, the number of context switches for processes in the same TESTBED execution differed by a factor of up to two. Note that number of context switches differs too severly to be explainable by just the difference in execution time.

# 4 Electronic Voting

A myriad of cryptographically secure voting schemes have been proposed in the literature, beginning with Chaum's mix-net voting [Cha81] in the 1980s.

In this chapter, we will first discuss some electronic voting schemes and implementations we consider relevant and notable. As we are mainly concerned with remote electronic voting in digital communities interacting in a peer-to-peer network, paper-based schemes such as Prêt à Voter [RBH+09], PunchScan [PH06], Scratch & Vote [Adi06] or Scantegrity [CCC+08] will not be further discussed.

We then describe and evaluate the performance of our implementation of the scheme by Cramer et al. in GNUnet.

## 4.1 Properties of Voting Schemes

To evaluate which of the proposed voting systems is most appropriate for this kind of use, we consider the following desirable properties (for a more detailed discussion of security properties see e.g. [Cet08]):

**Secrecy of Ballot** It is infeasible to determine which option a voter has voted for.

**Correctness** Votes are counted according to the rules of the election. In particular, it is impossible for a voter to submit more votes than designated, or to tamper with the result by contributing an invalid vote.

**Individual Verifiability** Every voter is able to convince himself that his vote was counted in the final result.

**Universal Verifiability** Any third-party auditor is able to verify that the votes were tallied correctly.

**Coercion-resistance** It is infeasible for an adversary to verify that a voter complied to the adversary's demands, for example that he voted for a certain candidate or abstained from voting. A weaker version of coercion-resistance is *receipt-freeness*, where the voter is not able to construct a proof that shows to a third party which option he voted for.

**Fairness** It is not possible to leak a partial result of the election's outcome, as the partial result could influence the decision of subsequent voters.

**Flexibility** The voting scheme should support commonly used electoral systems. At the least, the voting scheme should allow to select from a list of candidate, allowing cumulative voting.

**Robustness** The election scheme tolerates that a certain threshold fraction of the election officials may behave unexpectedly in arbitrary ways, for instance by being unavailable, corrupted or under attack of an adversary.

## 4.2 General Setting

A widely used setting for describing voting schemes was introduced by Benaloh [Ben87]. The main entities in this model are the voters $V_1, \ldots, V_n$, the voting authorities (sometimes called trustees) $A_1, \ldots, A_\ell$ and a *bulletin board*.

The bulletin board conceptually is a stateful, append-only public channel: Messages may be *posted* to it by (authorized) voters and authorities, but not removed. The content of the bulletin board is publicly accessible to facilitate election audits by third parties. In practice, the bulletin board is often implemented by the authorities themselves. See for example the survey by Peters [Pet05] or our implementation in section 4.6.1.

Note that some schemes rely on additional physical assumptions, such the secret one-way communication channels from Hirt and Sako [HS00].

An election generally proceeds in the following phases:

1. *Initialization.* The parameters of the voting scheme are initialized, by an election supervisor and/or by cooperation of the authorities.

2. *Ballot Preparation.* The vote preparation phase is optional, and allows voters to compute and post values to the bulletin board which will speed up the next phase, without deciding on their vote yet.

3. *Ballot Casting.* The voters post a message to the bulletin board that represents their vote.

4. *Tallying.* Authorities cooperate to count all votes. The result, including proofs allowing verification, are posted to the bulletin board.

## 4.3 Voting Scheme of Cramer et al.

The voting scheme of Cramer et al. [CGS97] (henceforth called CDS for the authors' initials) is particularly attractive as it is efficient and satisfies all properties discussed in section 4.1 except for coercion resistance. Let $\kappa$ be a security parameter. To cast their vote, voters only need to post a message of size $O(\kappa)$ to the bulletin board. The computation time to generate the message is dominated by $O(\kappa)$ modular exponentiations.

**Initialization** In the initialization step of the scheme, the authorities $A_1, \ldots, A_\ell$ jointly generate a $(k, \ell)$ threshold Elgamal key pair (see chapter 2 for the full key generation protocol). Note that the decisional Diffie-Hellman assumption (see section 2.1.1) must hold in the group used for Elgamal. In our work, we will use the order $q$ group $\mathcal{G}_q \subset \mathbb{Z}_p^*$ of quadratic residues modulo a safe prime $p$.

Let $x \in \mathbb{Z}_q$ be the private key of the group (which is shared among the authorities, but not stored in memory anywhere) and $h = g^x$ the corresponding public key. A message $m$

which is encrypted with random nonce $\alpha \in \mathbb{Z}_q$ as $(g^\alpha, h^\alpha \cdot m)$ can then only be decrypted by at least $k$ cooperating authorities, and no information can be derived about $m$ by less than $k$ authorities. For an election with $K$ choices, the election additionally must agree on $K$ independent generators $G_1, \ldots, G_K$ of $\mathcal{G}_q$.

**Vote Casting** Each voter encrypts his vote for choice $v \in \{1, \ldots, K\}$ as

$$(x, y) = (g^\alpha, h^\alpha \cdot G_v).$$

where $\alpha \in_R \mathbb{Z}_q$ is a nonce. In order to prove that the encryption was computed correctly, the voter generates a zero-knowledge proof for

$$\log_g x_i = \log_h(y/G_1) \quad \vee \quad \ldots \quad \vee \quad \log_g x_i = \log_h(y/G_\eta)$$

This proof can be obtained by applying the construction from [CDS94] to the Chaum-Pedersen non-interactive zero-knowledge proof for equality of discrete logarithms (see section 2.3.1). Let $(x_i, y_i)$ denote the encrypted voted of voter $V_i$.

**Tallying** After the vote submission phase, authorities compute

$$(X, Y) = \left( \prod_{i=1}^{n} x_i, \prod_{i=1}^{n} y_i \right)$$

and use the cooperative decryption protocol to obtain the encoded tally

$$W = G_1^{T_1} \cdot \ldots \cdot G_\eta^{T_K}$$

where $T_i$ is the number of votes for choice $i$ respectively.

The tally $(T_1, \ldots, T_K)$ can be obtained via brute-force for smaller elections by trying all non-negative valuations for $T_1, \ldots, T_K$ that satisfy $\sum_{i=1}^{K} T_i = n$. For larger elections, more sophisticated algorithms such as baby-step giant-step [Sut07] may be employed to improve the $O(n^K)$ runtime of the naive algorithm to $O\left(n^{\frac{K-1}{2}}\right)$ [CGS97].

## 4.4 Other Cryptographic Voting Schemes

### 4.4.1 Voting with Mix-nets

In mix-nets [Cha81, Nef03], messages are routed through a sequence of mutually distrusting servers $S_1, \ldots, S_\ell$ in order to hide the correspondence between message accepted by $S_1$ and messages output by $S_\ell$. In particular, messages are transformed and their order is shuffled when being passed from one server to the next. The origin of a message remains hidden as long as at least one server operates correctly.

An electronic voting system can be built with a mix-net by requiring all voters to post their encrypted vote together with their identity to the bulletin board. After the voting phase has ended, all encrypted votes are sent through the mix-net, which anonymizes the votes.

Not all types of mix-nets are appropriate for electronic voting. In *decryption mix-nets*, first described by Chaum [Cha81], each server $S_i$ has an asymmetric key pair $(PK_i, SK_i)$, where the $PK_i$ is the public key and $SK_i$ is the private key. Let $\mathsf{Enc}_i(m)$ denote the encryption of message $m$ with $PK_i$. A message $m$ given to $S_i$ is encrypted as

$$E_{1,\dots,n} = \mathsf{Enc}_1(\mathsf{Enc}_2(\dots \mathsf{Enc}_n(m)\dots)),$$

and each server in the chain peels off one layer of encryption by decrypting the received message with its private key. The exit server $S_\ell$ will output $m$. While this type of mix-net is useful for implementing an anonymous communication channel [DMS04], it is not appropriate for electronic voting, as the mixing is not reliable: Servers can drop and forge messages[1].

*Re-encryption mix-nets*[Nef03], on the other hand, are more appropriate for electronic voting. In this variation, a message $m$ is encrypted with a threshold public key (see chapter 3) before sending it to $S_1$. Instead of decrypting it, the servers *re-encrypt* the message before passing it on to the next server. Re-encryption is the process of transforming a ciphertext into a different ciphertext without changing the corresponding plaintext, making it hard to link the two ciphertexts. To achieve verifiability, the servers additionally publish a proof which demonstrates that the shuffling and re-encryption was performed correctly.

The set of peers that established the threshold public key must then cooperate to decrypt the votes that are output by $S_n$. It is easy to verify that the result of the election is correct by checking all proofs of shuffling and re-encryption as well as the proof of correct of decryption.

Mix-net-based electronic voting schemes provide incoercibility, a property that currently no other construction can provide. The main disadvantage of the approach, aside from its complexity, is that failure of just one of the mix servers can disrupt the whole election process. While this can be detected and cannot lead to a wrong election result, it may delay an election. Due to the large number of zero-knowledge proofs, mix-net based voting systems tend to be inefficient and hard to implement, though some proposals exist to simplify checking and creating proofs by making them probabilistic [Ben06, JJR02].

A more recent scheme based on mix-nets [CKLM13] claims to be more efficient, having linear runtime in the number of voters, mix servers and cooperative decryption authorities.

### 4.4.2 Voting with Blind Signatures

Blind signatures schemes [Cha83] are a well-studied variation of digital signature schemes, where the party that requests a signature applies a *blinding factor* to the message that is sought to be signed before sending it to the signer. The signer then responds to the blinded message with a blinded signature, and the requester is able to produce a valid signature by removing the blinding from the blinded signature. In addition to the usual properties of digital signatures, the following must hold for blind signatures:

- The signer is not able to learn the content of the message being signed, and
- the signer cannot link an unblinded message to a corresponding blinded message.

[1] To provide secrecy of ballot, votes must not signed.

It is possible to construct a simple electronic voting scheme with blind signatures [Cha83, FOO93]. In this scheme, the voters must authenticate themselves with the voting administration in order to obtain a blind signature over their vote. The voters send their signed vote to a counting authority over an anonymous channel. After the election has ended, the counting authority makes the list of submitted signed votes publicly available.

While this scheme is cheap, simple and elegant, it lacks a number of desirable properties:

**Fairness.** The counting authority could publish a preliminary result of the election, possibly influencing the decision of the remaining voters. The variation by Fujioka [FOO93] prevents this by having voters request a signature over a commitment to the vote, instead of the vote itself. Only after the vote submission period has ended, voters anonymously reveal their votes. This only partially solves the problem, as voters can still decide not to reveal their vote (and thus effectively abstain from voting), after a preliminary result was leaked in the reveal phase.

**Verifiability.** While voters can verify that their vote was counted for the final tally, it is possible for the voting administration to create an arbitrary number of valid votes, simply by signing them. It is only possible to detect this if the number of counterfeit votes exceeds the number of abstaining voters, which is typically quite large.

**Secrecy of Ballot.** If a voters want to prove that either a blind signature was not issued correctly or their vote was not counted, they have to reveal their vote when filing a complaint.

**Robustness.** The voting administration, which blindly signs votes, is a single point of failure.

### 4.4.3 Voting with Publicly Verifiable Secret Sharing

An elegant but relatively unknown electronic voting scheme is based on Schoenmakers' publicly verifiable secret sharing scheme (PVSS) [Sch99]. A secret sharing scheme is publicly verifiable if not just the share-holders but any third party can verify the validity of a sharing. An advantage of Schoenmakers' scheme is that it does not require the authorities to generate a threshold public key. This makes the scheme especially suitable for smaller elections, where the overhead of the distributed key generation dominates.

Unfortunately, the scheme described by Schoenmakers does only allow binary elections (i.e. with two choices). It might be fruitful to investigate whether the constructions in [CDS94] could be applied to yield a multi-choice election protocol.

Schoenmakers' protocol does not provide coercion resistance.

In a certain sense, Schonmakers' scheme is dual to the CDS protocol. While in CDS, the authorities threshold-share the decryption key and voters encrypt ballots, in Schoenmakers' scheme the voters threshold-share their vote with all authorities by posting the encrypted shares on the bulletin board. This makes the size of one ballot linear in the number of authorities $\ell$. As the secret sharing scheme is publicly verifiable, the correctness of a casted ballot can be verified by third parties.

In the tallying phase, the authorities decrypt their respective shares from the bulletin board and homomorphically add them. This results in every authority having a share of the final tally. If at least $k$ authorities (where $k$ is the threshold parameter) post a partial decryption of the final tally on the bulletin board, the final tally can be restored in plaintext.

## 4.5 Existing Implementations

Despite the enormous amount of research on electronic voting protocols, practical implementations are scarce. Some of the claimed implementation such as the CyberVote [ER06] or Sensus [CC96] either have no source code available or are unmaintained and proprietary prototypes.

To out knowledge, the ADDER voting system [KKW06] is the first implementation of a voting protocol based on homomorphic encryption that uses a distributed key generation protocol. Although the authors do not explicitly mention the origin of the construction they use for homomorphic tallying, it appears that a variation of the scheme of Cramer et al. is employed. ADDER uses a single MySQL database to implement the bulletin board. Authorities do not sign messages written to the bulletin board, but must authenticate with their email address and a password to a "guard server" before writing to the bulletin board. This renders the "guard server" a single point of failure.

Helios 1.0 [Adi08] is an implementation of mix-net voting. The system does not implement threshold decryption, but requires a trusted third party to hold the decryption key.

In Helios 2.0 [ADMP+09], the mix-net approach was replaced by a voting protocol with homomorphic encryption in the style of the CDS scheme. Helios 2.0 does not implement distributed key generation, but relied on physical assumptions (the removal of all network devices and writable storage devices except a "trusted" USB thumb drive in a key ceremony). Helios 2.0 as been successfully used for the real-world election of a university president. The client implementation, however, was shown to exhibit security flaws [ED10], which were subsequently fixed in Helios 3.0. To date, the bulletin board in Helios is implemented by a single server. The implementation of Helios is free and open source software[2].

Civitas [CCM07] is an implementation of mix-net voting with focus on coercion freeness and verifiability. Civitas claims to implement a robust and scalable bulletin board by having multiple "ballot box servers" collect votes. At the end of the voting period, a trusted election supervisor signs the content of each of the ballot boxes, before they are sent through the mix net. This ensures that votes submitted to at least one honest ballot box are counted, but only if the election supervisor is honest. While there is a technical report about some implementation details of Civitas [DCC08], to date there is no publicly available implementation.

[2] `https://github.com/benadida/helios-server`, `https://vote.heliosvoting.org/`

## 4.6 Implementation of Electronic Voting in GNUnet

We now describe our implementation of the CDS voting scheme in GNUnet. The source code is available in the `org.gnunet.voting` package of GNUnet-Java, licensed under the GPLv3+.

### 4.6.1 Bulletin Board

As a bulletin board, we use GNUnet's Consensus service, which is fully described in an separate technical report [Dol14]. Deviating from other common implementations of bulletin boards (see Peters [Pet05] for a survey of other bulletin board implementations), we do not implement a replicated state machine. Instead, voters must post their ballots to at least one correct authority (similar to Civitas). The authorities then jointly execute a Byzantine fault tolerant multi-peer set reconciliation protocol on their respective set of ballots received from voters. This stands in contrast to state-machine replication, where latency due to Byzantine agreement is incurred on every submission of a ballot. We believe that our approach is necessary in a peer-to-peer system such as GNUnet, where high latency is expected.

Other agreement protocols such as Paxos [Lam01, Lam06] favor consistency over progress, which makes less suitable in environments where it is expected that a number of authorities will temporarily fail to be available. We argue that our approach offers more simplicity, at least compared to the notoriously complex Paxos protocol.

### 4.6.2 Roles

Our system has three roles:

**Election Supervisor** The election supervisor is identified by an ECDSA public key[3]. By signing a description of the election (see below), the Election Supervisor attests that he will accept the result of the election. The Election Supervisor is also responsible for selecting the Election Authorities. No other trust is placed on the Supervisor

**Election Authority** The election authority is identified by an EdDSA public key[4]. An Authority is responsible for collecting and verifying voters' ballots, tallying the result after the voting period, as well as providing audit data to third parties.

**Voter** A voter is identified by his ECDSA public key.

### 4.6.3 Voter Authentication

The election supervisor includes a signature on the list of eligible voters in the ballot description, and sends the list to all authorities on the election registration.

[3] In GNUnet, each user pseudonym (called an ego) corresponds to an ECDSA key.
[4] Each election authority corresponds to a GNUnet peer

### 4.6.4 Ballot Issuing

The election supervisor initiates an election by writing the election data in a ballot file (implemented as a GNUnet configuration file), signing it and registering it with the authorities that should be responsible for the election. If an authority agrees to participate in the election, it add its signature to the ballot form, and sends it back to the supervisor.

The initial ballot form contains the following information:

- The topic of the election.
- Available choices (e.g. candidates)
- List of Election Authorities used in the election
- The threshold $k$.
- Signed hash of the list of eligible voters
- A time interval for each of the election phases.
- The list of generators used for multi-choice elections.
- A signature on all items above.

The election phases are described by the following timestamps:

**KEYGEN_START** Start of the distributed key generation

**KEYGEN_END** Deadline for the distributed key generation

**VOTE_START** Start of the vote submission period, where authorities accept ballots from voters.

**VOTE_CLOSE** End of the vote submission period. Once the vote submission period is over, authorities immediately run the set reconciliation protocol on all received ballots.

**CONCLUDE** Deadline for the set reconciliation protocol and start of the cooperative decryption protocol for the product of encrypted votes.

**TALLY** Deadline for the cooperative decryption and start of the brute-force decryption of the final tally.

**QUERY** Time when voters are allowed to ask for the final tally.

**END** Time when the authorities are allowed to delete all information about the election.

The election is only successful if at least $k$ honest peers finish the respective protocols for the deadlines `KEYGEN_START`, `CONCLUDE` and `TALLY`.

| Choices $K$ | Generation | Verification |
|---|---|---|
| 2 | 35 ms | 23ms |
| 10 | 131ms | 115ms |
| 100 | 1.17s | 1.14s |
| 1000 | 11.7s | 11.5s |

Table 4.1: Time required to generate and verify a ballot with our implementation, measured on an Intel i5 3.3 GHz CPU with OpenJDK7. Measurements are averaged over 20 repetitions.

### 4.6.5 The Authority Service

An authority is started by running the `voting-authority` service on the respective GNUnet peer. The service can be configured to only participate in elections requested by specified supervisors. By default, registration requests from all supervisors are accepted.

Authorities communicate with the election authority and voters through GNUnet's CADET service [PG14].

### 4.6.6 Voting

After a voter receives a copy of the ballot file, he can amend it with his own encrypted vote, including a signature and a zero-knowledge proof of the vote's correctness. During the vote submission interval of the election, the voter (or any other party that has the ballot file) sends the amended ballot file to at least $k$ authorities.

### 4.6.7 User Interface

While our voting implementation is mainly intended to be used as a library by other applications, we also provide a command line interface in the form of the `gnunet-voting-ballot` application.

It serves four purposes:

- Signing and registering a ballot with the election authorities as a supervisor
- Encoding a vote as a voter
- Sending a complete ballot file (with the vote) to the Election Authorities
- Querying and verifying the result of an election.

## 4.7 Performance Evaluation

For $K$ available choice and using 1024-bit Elgamal, the size of a vote is $480 + K \cdot 512$ bytes, including the voter's identity, an ECDSA signature, the encrypted choice and the proof of validity.

Even with our unoptimized, pure Java implementation (i.e. not using any cryptographic libraries other than `java.math.BigInteger`) the time to generate a vote (see table 4.1)

is reasonable small and scales linearly with the number of available choices (due to the size of the disjunctive proof of validity).

The tally authorities employ a naive brute-force algorithm to reconstruct the tally by trying all $\binom{n+1}{K-1}$ possibilities, making the current implementation of the authorities infeasible for large-scale elections, as one iteration of our brute-force algorithm takes approximately $40\,\mu s$ on our test machine (Intel i5 3.3 GHz CPU).

# 5 Conclusion and Future Work

We described the design and implementation of the CDS voting scheme [CGS97] in GNUnet, including a distributed key generation protocol. We showed that the cryptographic operations have acceptable execution time on modern computers for a moderate number of authorities and voters.

The fully Byzantine fault tolerant implementation of the bulletin board, as well as an investigation of the applicability of the PVSS voting scheme [Sch99] to elections with multiple choices remains future work.

Although the the mathematics behind the voting algorithm we implemented are not easily understood by the average citizen, we believe that the availability of free software implementations of voting systems is vital to allow the gradual transition to more secure tools for democratic decision-making in online communities.

Surprisingly, even groups of knowledgeable professionals still advocate the use of black-box voting system. The German *Gesellschaft für Informatik* [1], for example, approves the use of Polyas [OKN+12, RJ07, MR+10], which uses centralized and proprietary components, and only provides partial verifiability with a proposed modification [OKN+12].

The reason why these kind of voting systems are popular can probably be attributed to their ease of deployment and use. We hope to address this issue by integrating our voting implementation in the social networking components proposed for GNUnet [Tot13b]

[1] German Association for Informatics

# Bibliography


[AB83] Charles Asmuth and John Bloom. A modular approach to key safeguarding. *IEEE Transactions on Information Theory*, 30(2):208–210, 1983.

[Adi06] Ben Adida. *Advances in cryptographic voting systems*. PhD thesis, Massachusetts Institute of Technology, 2006.

[Adi08] Ben Adida. Helios: Web-based open-audit voting. In *USENIX Security Symposium*, volume 17, pages 335–348, 2008.

[ADMP+09] Ben Adida, Olivier De Marneffe, Olivier Pereira, Jean-Jacques Quisquater, et al. Electing a university president using open-audit voting: Analysis of real-world use of helios. *EVT/WOTE*, 9, 2009.

[Ben87] Josh Benaloh. *Verifiable secret-ballot elections*. PhD thesis, PhD thesis, Yale University, 1987.

[Ben06] Josh Benaloh. Simple verifiable elections. In *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, pages 5–5. USENIX Association, 2006.

[BHKL13] Daniel J Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: Elliptic-curve points indistinguishable from uniform random strings. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 967–980. ACM, 2013.

[BKS09] Ilker Nadi Bozkurt, Kamer Kaya, and Ali Aydın Selçuk. Practical threshold signatures with linear secret sharing schemes. In *Progress in Cryptology–AFRICACRYPT 2009*, pages 167–178. Springer, 2009.

[Bla99] George Robert Blakley. Safeguarding cryptographic keys. In *Managing Requirements Knowledge, International Workshop on*, pages 313–313. IEEE Computer Society, 1899.

[BOL90] Michael Ben-Or and Nathan Linial. Collective coin flipping. *Randomness and Computation*, 5:91–115, 1990.

[Bon98] Dan Boneh. The decision diffie-hellman problem. In Joe Buhler, editor, *Algorithmic Number Theory*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63. Springer Berlin / Heidelberg, 1998. 10.1007/BFb0054851.

[CC96] Lorrie Faith Cranor and Ron K Cytron. *Design and implementation of a practical security-conscious electronic polling system*. Washington University, Department of Computer Science, 1996.

[CCC+08] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L Rivest, Peter YA Ryan, Emily Shen, and Alan T Sherman. Scantegrity ii: end-to-end verifiability for optical scan election systems using invisible ink confirmation codes. *EVT*, 8:1–13, 2008.

[CCM07] Michael R Clarkson, Stephen Chong, and Andrew C Myers. Civitas: A secure voting system. Technical report, Cornell University, 2007.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology–CRYPTO'94*, pages 174–187. Springer, 1994.

[Cet08] Orhan Cetinkaya. Analysis of security requirements for cryptographic voting protocols. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, pages 1451–1456. IEEE, 2008.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Proceedings of the 16th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'97, pages 103–118, Berlin, Heidelberg, 1997. Springer-Verlag.

[Cha81] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[Cha83] David Chaum. Blind signatures for untraceable payments. In *Advances in cryptology*, pages 199–203. Springer, 1983.

[CKLM13] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Verifiable elections that scale for free. In *Public-Key Cryptography–PKC 2013*, pages 479–496. Springer, 2013.

[Coh93] Henri Cohen. *A course in computational algebraic number theory*, volume 138. Springer, 1993.

[CP93] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *Advances in Cryptology–CRYPTO'92*, pages 89–105. Springer, 1993.

[CS97] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. 1997.

[DCC08] Adam M Davis, Dmitri Chmelev, and Michael R Clarkson. Civitas: Implementation of a threshold cryptosystem. Technical report, Cornell University, 2008.

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

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[Dol13] Florian Dold. Threshold cryptography. unpublished seminar paper, 2013.

[Dol14] Florian Dold. The gnunet consensus service. unpublished technical report, 2014.

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

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology*, pages 10–18. Springer, 1985.

[ER06] Khaled Elleithy and Ihab Rimawi. Design, analysis and implementation of a cyber vote system. In *Advances in Computer, Information, and Systems Sciences, and Engineering*, pages 219–226. Springer, 2006.

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Foundations of Computer Science, 1987., 28th Annual Symposium on*, pages 427–438. IEEE, 1987.

[FOO93] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *Advances in Cryptology–AUSCRYPT'92*, pages 244–251. Springer, 1993.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology–CRYPTO'86*, pages 186–194. Springer, 1987.

[FS01] Pierre-Alain Fouque and Jacques Stern. One round threshold discrete-log key generation without private channels. In *Public Key Cryptography*, pages 300–316. Springer, 2001.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

[GJKR99] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Advances in Cryptology–EUROCRYPT'99*, pages 295–310. Springer, 1999.

[GJKR03] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure applications of Pedersen's distributed key generation protocol. In *Topics in Cryptology–CT-RSA 2003*, pages 373–390. Springer, 2003.

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in) security of the fiat-shamir paradigm. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 102–113. IEEE, 2003.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, 1994.

[HS00] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *Advances in Cryptology–EUROCRYPT 2000*, pages 539–556. Springer, 2000.

[JJR02] Markus Jakobsson, Ari Juels, and Ronald L Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *USENIX security symposium*, pages 339–353. San Francisco, USA, 2002.

[Jud94] T. W. Judson. *Abstract Algebra: Theory and Applications*. PWS Pub. Co., 1994.

[KKW06] Aggelos Kiayias, Michael Korman, and David Walluck. An internet voting system supporting user privacy. In *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*, pages 165–174. IEEE, 2006.

[KL08] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC Press, 2008.

[KSRW04] Tadayoshi Kohno, Adam Stubblefield, Aviel D Rubin, and Dan S Wallach. Analysis of an electronic voting system. In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pages 27–40. IEEE, 2004.

[Lam01] Leslie Lamport. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[Lam06] Leslie Lamport. Fast paxos. *Distributed Computing*, 19(2):79–103, 2006.

[MR+10] Niels Menke, Kai Reinhard, et al. Compliance of polyas with the common criteria protection profile-a 2010 outlook on certified remote electronic voting. In *Electronic Voting*, pages 109–118, 2010.

[MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

[Nef03] C Andrew Neff. Verifiable mixing (shuffling) of elgamal pairs. *VHTi Technical Document, http://www. votehere. net/vhti/documentation/egshuf. pdf, VoteHere, Inc*, 2003.

[OKN+12] M Maina Olembo, Anna Kahlert, Stephan Neumann, Melanie Volkamer, and Projektgruppe verfassungsverträgliche Technikgestaltung. Partial verifiability in polyas for the gi elections. In *Electronic Voting*, pages 95–109, 2012.

[Oos10] Anne-Marie Oostveen. Outsourcing democracy: Losing control of e-voting in the netherlands. *Policy & Internet*, 2(4):201–220, 2010.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in cryptology–EUROCRYPT'99*, pages 223–238. Springer, 1999.

[Ped91] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *Proceedings of the 10th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'91, pages 522–526, Berlin, Heidelberg, 1991. Springer-Verlag.

[Pet05] RA Peters. A secure bulletin board. Master's thesis, Technische Universiteit Eindhoven, 2005.

[PG14] Bartlomiej Polot and Christian Grothoff. Cadet: Confidential ad-hoc decentralized end-to-end transport. In *Med-Hoc-Net 2014*, 2014.

[PH78] Stephen Pohlig and Martin Hellman. An improved algorithm for computing logarithms over $GF(2^n)$. *Information Theory, IEEE Transactions on*, 24(1):106–110, 1978.

[PH06] Stefan Popoveniuc and Ben Hosp. An introduction to punchscan. In *IAVoSS Workshop On Trustworthy Elections (WOTE 2006)*, pages 28–30. Robinson College United Kingdom, 2006.

[PS00] Guillaume Poupard and Jacques Stern. Fair encryption of rsa keys. In *Advances in Cryptology–EUROCRYPT 2000*, pages 172–189. Springer, 2000.

[RBH+09] Peter YA Ryan, David Bismark, James A Heather, Steve A Schneider, and Zhe Xia. The prêt à voter verifiable election system. *IEEE transactions on information forensics and security*, 4(4):662–673, 2009.

[RJ07] Kai Reinhard and Wolfgang Jung. Compliance of polyas with the bsi protection profile–basic requirements for remote electronic voting systems. In *E-Voting and Identity*, pages 62–75. Springer, 2007.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology–Crypto'89 Proceedings*, pages 239–252. Springer, 1990.

[Sch99] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Advances in Cryptology–CRYPTO'99*, pages 148–164. Springer, 1999.

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

[Sut07] Andrew V Sutherland. *Order computations in generic groups*. PhD thesis, Massachusetts Institute of Technology, 2007.

[Tot13a] Sree Harsha Totakura. Large scale distributed evaluation of peer-to-peer protocols. Masters, Technische Universitaet Muenchen, Garching bei Muenchen, 06/2013 2013.

[Tot13b] Gabor X Toth. Design of a social messaging system using stateful multicast. *Master's, University of Amsterdam, Amsterdam*, 2013.