Secret-Ballot Electronic Voting Procedures Over the Internet

Michael James Korman

B.S.E., University of Connecticut, 2004

A Thesis
Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science
at the
University of Connecticut

2007

APPROVAL PAGE

Master of Science Thesis

Secret-Ballot Electronic Voting Procedures Over the Internet

Presented by

Michael James Korman, B.S.E.

Major Advisor ______________________
Aggelos Kiayias

Associate Advisor ______________________
Alexander Russell

Associate Advisor ______________________
Zhijie Shi

University of Connecticut
2007

Acknowledgements

I would like to thank the following people: my advisor, Aggelos Kiayias, for his guidance and support; David Walluck, for his work on the Adder system; Alex Russell and Jerry Shi, for serving on my thesis committee; Hong-Sheng Zhou, for his suggestions and corrections; and Stephanie, for her endless encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Electronic voting is one of the most controversial subjects in information technology. Debates on the topic cross all areas of discourse: politics, usability, public policy, computer and physical security, cryptography, and mathematics. Before embarking on our study of how cryptography can be used to address this problem, it is necessary to review the nature of the problem itself. We pose the question: what is wrong with current voting technology?

1.1 Motivation

Voting is the central element of a democracy. Elections give the citizens of a country the ability to voice their opinions, choose representatives, and ensure that the government remains in the hands of the people.

Political elections are not the only place for voting. Smaller-scale elections are even more common. For example, many organizations hold elections to select officials, boards of directors, etc. If one was to evaluate the global need for correct and worthwhile voting systems, smaller organizations would probably rank higher than governments.

However, just because a government or organization allows voting does not make it a democracy. Voting can be useless if the election outcome does not accurately represent the combined desires of the voters. What properties must a voting scheme possess to be considered "useful?" At the very least, it must allow each voter to have his opinion counted in the election outcome. Ideally, the election outcome should represent the solution most favorable to the majority. How this is defined is a matter of debate, and several mathematical results have been posed that attempt to rigorously define this concept.

Furthermore, the voting scheme must be in some sense *robust*. That is, individual participants should not be able to corrupt or compromise the election. Malicious behavior can take many forms, including inaccurately influencing the election outcome, disenfranchising voters, compromising voter privacy, and disrupting the entire election procedure.

1.2 Classical Technology

Current non-electronic voting technology comes in many forms. One of the simplest forms of voting is the paper ballot. With this method, each voter is given a ballot, printed on paper, and is asked to mark his choices. The ballots are then collected, shuffled, and counted by hand. Ballots can be submitted by mail, or filled out at a polling place. Another common method of voting is with a mechanical lever-based machine. Voters arrive at the polling place, and enter a private booth containing a machine. Corresponding to each candidate is a lever. Voters pull the lever of their choice, and a mechanical counter is updated to reflect the newly cast ballot.

Non-electronic voting methods, in some form or another, have existed for centuries. As such, they are well understood, and widely trusted. Paper-based methods, in particular, are easily comprehensible by the general public. Another advantage of paper-based methods is the fact that they are *auditable*. That is, it is a simple matter to count the paper ballots by hand. Furthermore, assuming that no ballots are lost, a recount is also easy to perform.

The public first became widely suspicious of paper-based voting after the 2000 United States Presidential election. In Florida, the difference between the two primary candidates was so close that the state mandated a recount. The count was deemed valid, but numerous concerns were raised following the election. Regarding the voting technology, the so-called "butterfly ballots" used in Palm Beach County were seen as overly confusing, and may have led to many miscast votes. Additionally, some punch-card ballots were not punched properly, and thus read incorrectly by the counting machine. Following this election, there was a strong push towards electronic voting systems which were easier to use, and supposedly less prone to error.

1.3 Failures of Electronic Voting

Most electronic voting systems used in practice are known as *direct recording electronic* (DRE) systems. That is, they are digital devices that contain internal counters representing the running total of votes. A voter fills out a ballot by either using a touch-screen interface, or completes a card by filling in ovals with a pencil. The voting machine accepts the ballot and adds the vote to the internal counter.

The Diebold AccuVote-TS touch-screen DRE system is one of the most heavily criticized [KSRW04] non-Internet-based electronic voting systems used in practice. Problems pointed out include incorrect use of cryptography, poor code quality, and possibility of smartcard forgery, among many others. Despite Diebold's rebuttal [Sys03], the system is mistrusted by a number of experts. Diebold also manufactures an optical scan system, the AccuVote-OS, which itself has been subject to numerous attacks [Hur05], [KMR+06].

In 2004, the United States Department of Defense canceled the Secure Electronic Registration and Voting Experiment project (known as SERVE), an Internet-based PC voting system. This project, a $22 million contract, was hoped to be used to count votes in the 2004 general elections. However, numerous technological flaws were revealed by Rubin, et al. [JRSW04]. The stated flaws include: use of closed, proprietary software, and thus vulnerabilities to insider attacks; lack of a verifiable audit trail; use of the Internet, opening the door to numerous security problems such as denial-of-service attacks, spoofing, viruses, etc. The report concluded that there are no acceptable alternatives to SERVE; all suffer from the same fundamental vulnerabilities.

As a result of these failures, the public has become increasingly wary of electronic voting systems. Indeed, their fears are not without merit, as secure electronic voting may never be possible over a network such as the Internet. Security experts often view voting as the "holy grail" of security. Noted security professional Bruce Schneier has said, "a secure Internet voting system is theoretically possible, but it would be the first secure networked application ever created in the history of computers." [Sch00]

1.4 The Need for Strong Cryptography

If secure voting is ever to succeed on the Internet or any other insecure network, it will only happen through the use of cryptography. Strong cryptographic mechanisms can provide solutions to many of the inherent problems in existing electronic voting systems, such as integrity and privacy.

Cryptography cannot possibly solve all problems associated with voting. Some of these problems are fundamental to the nature of computer networks. Cryptography cannot make up for network deficiencies, and such as susceptibility to denial-of-service attacks. Furthermore, there are several social problems that are completely unrelated to the specific technology used. For example, the problem of vote buying and coercion will exist with any system where voters either vote unsupervised, or are permitted to vote on their own machines.

What problems can cryptography solve? While designing a cryptographic scheme for voting, we consider the following design goals.

1. *Transparency*. All of the data on any networked servers should be accessible to the public. This includes the encrypted votes, public encryption keys, and final tallies. Public servers should not store secrets.

2. *Universal Verifiability*. Any election result obtained by the system should be verifiable by any third party. By inspecting the election transcript, it should be possible to perform a complete audit of any procedure.

3. *Privacy*. All voters in an election should be confident that their individual choices will remain hidden. Only the outcome is made available to the public.

4. *Distributed Trust*. Each procedure is supervised by multiple *trustees*, and the final sum cannot be revealed without the cooperation of a given number of trustees. Any attempt to undermine the procedure will require the corruption of a large number of trustees. Trustees and voters may overlap arbitrarily. Thus, it is possible for the voters themselves to ensure trustworthiness, or have an active role in it.

In this thesis, we will investigate how cryptography can be used to achieve all of these goals.

1.5 Outline

The remainder of this thesis is organized as follows. In Chapter 2, we will provide a concise review of the mathematics background necessary to understand the cryptography we use. In Chapter 3, we explain the fundamentals of cryptography, focusing on the aspects needed for electronic voting. Chapter 4 gives an analysis of several encryption schemes that are particularly well-suited to voting. In Chapter 5, we introduce proofs of knowledge, a powerful technique for proving knowledge of information without revealing it. In Chapter 6, we describe techniques for distribution of trust, a necessary component of any system to be deployed over the Internet. In Chapter 7, we describe how all of these techniques can be combined to build robust voting systems. In Chapter 8, we give examples of related work in electronic voting. In Chapter 9, we present an implementation of the ideas given in this thesis. Finally, in Chapter 10, we give a brief summary and discussion, as well as suggestions for future work in the area.

1.6 Prerequisites

The reader is assumed to have a basic working knowledge of computer science and mathematics, such as that obtained by an undergraduate degree in either field. However, no specific background knowledge beyond high school mathematics, rudimentary algorithms, and proof techniques is required. In particular, no knowledge of cryptography, complexity theory, or abstract algebra is assumed. Where there is uncertainty, definitions are provided for the reader's convenience. However, our presentation of the basic fundamentals is given in a rather terse manner. The inexperienced reader will find it helpful to supplement this manuscript with additional references, which will be given as needed.

Chapter 2

Mathematical Preliminaries

In this chapter, we introduce the basic mathematics needed to understand cryptography. We focus only on the definitions and results necessary for the development of this thesis.

2.1 Basic Definitions

We use the symbols $\mathbb{N}, \mathbb{R}, \mathbb{Q}$ and $\mathbb{Z}$ to refer to the sets of naturals ($\{1, 2, 3, \ldots\}$), reals, rationals, and integers, respectively. If $a, n \in \mathbb{Z}$, then $a \bmod n$ is equal to the remainder of $a \div n$. The symbol $\mathbb{Z}_n$ refers to the "integers modulo n," that is,

$$\mathbb{Z}_n \stackrel{\text{def}}{=} \{x \mid 0 \leq x \leq n-1\}.$$

For $n \in \mathbb{N}$, the notation $[n]$ refers to the set $\{1, \ldots, n\}$.

Definition 2.1.1. Let X and Y be sets, and let $f : X \longrightarrow Y$ be a function mapping X into Y. We say f is *injective* if $f(x) = f(y)$ implies that $x = y$. We say f is *surjective* if for all $y \in Y$, there exists an $x \in X$ such that $f(x) = y$. We say f is *bijective* if f is both injective and surjective.

Definition 2.1.2. A number $p \in \mathbb{N} \setminus \{1\}$ is called *prime* if its only factors are 1 and itself. We say p is an *odd prime* if p is prime and $p \neq 2$.

Definition 2.1.3. The *greatest common divisor* of $a, b \in \mathbb{Z}$ is the largest integer g such that $g \mid a$ and $g \mid b$. In this case, we write $g = \gcd(a, b)$. Integers a and b are called *relatively prime* if $\gcd(a, b) = 1$, and we write $a \perp b$.

Definition 2.1.4. The *least common multiple* of $a, b \in \mathbb{Z}$ is the least integer m such that $m = ax = by$, for some $x, y \in \mathbb{Z}$. In this case, we write $m = \operatorname{lcm}(a, b)$.

Definition 2.1.5. Let $x \in \mathbb{Z}$. Then, the *length* of x, denoted $\operatorname{len} x$, is the length of the binary representation of x. That is, $\operatorname{len} x \overset{\text{def}}{=} \lfloor \lg x \rfloor + 1$.

2.2 Computational Complexity

As we will eventually be discussing the nature of the computational resources required by various protocols and algorithms, it is necessary to review a basic framework for measuring computational complexity.

Definition 2.2.1 (Asymptotic Notation)**.** Let $f : \mathbb{N} \longrightarrow \mathbb{R}$ and $g : \mathbb{N} \longrightarrow \mathbb{R}$ be functions. We define the following sets.

1. We say $f \in \mathrm{O}(g)$ ("*big omicron*") if there exists an $N \geq 0$ such that for all $n \geq N$, $f(n) \leq cg(n)$, for some $c \in \mathbb{R}^+$.

2. We say $f \in \Omega(g)$ ("*big omega*") if there exists an $N \geq 0$ such that for all $n \geq N$, $f(n) \geq cg(n)$, for some $c \in \mathbb{R}^+$.

3. We say $f \in \Theta(g)$ ("*big theta*") if there exists an $N \geq 0$ such that for all $n \geq N$, $c_1 f(n) \leq f(n) \leq cg(n)$, for some $c_1, c_2 \in \mathbb{R}^+$.

4. We say $f \in o(g)$ ("*little omicron*") if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.$$

Throughout this thesis, algorithms are assumed to be modeled by Turing machines, either deterministic or probabilistic. The reader is referred to [Sip97] for a precise definition of Turing machine.

Definition 2.2.2. A *probabilistic Turing machine* is a Turing machine with access to an additional *random tape*. The random tape contains an infinite sequence of 0s and 1s, and is provided to the Turing machine as input. A probabilistic Turing machine that is restricted to operating in polynomial time in its input is called a *probabilistic polynomial time Turing machine*, abbreviated *PPT*.

Definition 2.2.3. Let L be a language, and let $\mathscr{O}$ be a Turing machine deciding L. Then, we say that $\mathscr{O}$ is an *oracle* for L. If $\mathscr{M}$ is a Turing machine, we use the notation $\mathscr{M}^{\mathscr{O}}$ to refer to the Turing machine $\mathscr{M}$ with the added capability of querying the oracle $\mathscr{O}$ on inputs of its choosing.

Definition 2.2.4. Define $f : \mathbb{N} \longrightarrow \mathbb{N}$. Then, TIME($f$) is the set of all problems that can be solved in time O(f). Furthermore,

$$\mathrm{P} \overset{\text{def}}{=} \bigcup_{k=1}^{\infty} \mathrm{TIME}(n^k).$$

The class P is the set of all problems that can be solved in *polynomial time*. This class corresponds quite naturally to the set of problems that are "tractable."

Definition 2.2.5. Define $f : \mathbb{N} \longrightarrow \mathbb{N}$. Then, NTIME($f$) is the set of all problems that can be solved in time O(f) by a non-deterministic Turing machine. Furthermore,

$$\mathrm{NP} \overset{\text{def}}{=} \bigcup_{k=1}^{\infty} \mathrm{NTIME}(n^k).$$

The class NP is the set of all problems that can be solved in *non-deterministic polynomial time*. Since a non-deterministic Turing machine is capable of performing numerous simultaneous deterministic computations, and is only charged for the one that actually finds the answer, we can also think about NP as the set of problems that have solutions that are verifiable in polynomial time. It is thus obvious that P $\subseteq$ NP, as computing a solution is certainly no easier than verifying one. One of the greatest unsolved problems in complexity theory is whether or not P equals NP. The current popular opinion is that these two class differ, with P being a strict subset of NP.

Definition 2.2.6. Let L_1 and L_2 be languages over some alphabet Σ. Then, we say that L_1 is *polynomial-time reducible* to L_2 if there exists a polynomial time computable function $f : \Sigma^* \longrightarrow \Sigma^*$ such that $w \in L_1$ if and only if $f(w) \in L_2$, and we write $L_1 \leq_{\mathrm{P}} L_2$. If $L_1 \leq_{\mathrm{P}} L_2$ and $L_2 \leq_{\mathrm{P}} L_1$, then we say L_1 and L_2 are *polynomial-time equivalent*, and write $L_1 =_{\mathrm{P}} L_2$.

Now, suppose P_1 and P_2 are function problems. We say that P_1 is polynomial time reducible to P_2 if there exist two polynomial time computable functions $f_1, f_2 : \Sigma^* \longrightarrow \Sigma^*$ such that if x is an instance of P_1, then $f_1(x)$ is an instance of P_2, and if y is a correct output of $f_1(x)$, then $f_2(y)$ is a correct output of x.

Definition 2.2.7. Let L be a language. We say L is NP-*hard* if $M \leq_{\mathrm{P}} L$, for all $M \in$ NP. We say L is NP-*complete* if L is NP-hard and $L \in$ NP.

Theorem 2.2.8. *If L is* NP-*complete, and $L \in$* P, *then* P = NP.

Theorem 2.2.8 says that the NP-complete problems are somehow the "hardest" problems in NP. Thus, if one of them is shown to be in P, all of NP must be in P.

2.3 Probability

We give here an axiomatic viewpoint of probability.

Definition 2.3.1. Let Ω be a set, and let $\mathscr{F}$ be a collection of subsets of Ω. We call $(\Omega, \mathscr{F})$ a *σ-field* if the following hold.

1. $\emptyset \in \mathscr{F}$.
2. If $E \in \mathscr{F}$, then $\Omega \setminus E \in \mathscr{F}$.
3. If $E_1, E_2, \ldots$ is a countable collection of subsets of $\mathscr{F}$, then $E_1 \cup E_2 \cup \cdots \in \mathscr{F}$.

Definition 2.3.2. Let $(\Omega, \mathscr{F})$ be a σ-field. A function $\Pr\colon \mathscr{F} \longrightarrow \mathbb{R}^+$ is called a *probability measure* if the following conditions hold.

1. For all $E \in \mathscr{F}$, it holds that $0 \leq \Pr[E] \leq 1$.
2. $\Pr[\Omega] = 1$.
3. If $E_1, E_2, \ldots \in \mathscr{F}$ are disjoint, then $\Pr[\bigcup_{i=1}^{\infty} E_i] = \sum_{i=1}^{\infty} \Pr[E_i]$.

Definition 2.3.3. Given a σ-field $(\Omega, \mathscr{F})$ and a probability measure Pr defined on $\mathscr{F}$, we call $(\Omega, \mathscr{F}, \Pr)$ a *probability space*. We refer to Ω as a *sample space*, and the elements of $\mathscr{F}$ as *events*.

Let $(\Omega, \mathscr{F}, \Pr)$ be a probability space, with Ω a finite set, and $\mathscr{F}$ the set of all subsets of Ω. Suppose Q is a predicate over Ω. Let $A_Q = \{\omega \in \Omega : Q(\omega)\} \in \mathscr{F}$. Then,

$$\Pr[A_Q] \overset{\text{def}}{=} \sum_{\omega \in A_Q} \Pr[\{\omega\}].$$

If $\Pr[\{\omega\}] = 1/|\Omega|$, for all $\omega \in \Omega$, then we say that $(\Omega, \mathscr{F}, \Pr)$ is a *uniform distribution*.

If S is an arbitrary set,

$$\Pr_{x \leftarrow S}[Q(x)]$$

denotes the probability $\Pr[Q(x)]$ in the uniform distribution $(S, \mathscr{P}(S), \Pr)$.

Finally, we often use an algorithmic instruction dealing with randomness. Let S be a set. We use the notation $x \in_{\mathrm{U}} S$ to mean that x is an element chosen uniformly at random from S.

2.4 Algebra

We now review some basic facts about modern algebra. As algebra deals with the study of binary operations over sets, we must first define what we mean by "binary operation."

Definition 2.4.1. Let S be a set. A *binary operation* over S is a function $\star\colon S \times S \longrightarrow S$. If $a, b \in S$, we write '$a \star b$' for '$\star(a, b)$'.

Groups are the fundamental algebraic structure that we shall concern ourselves with.

Definition 2.4.2. A *group* is a pair $(G, \star)$, where G is a set and $\star$ is a binary operation over G, such that the following axioms hold.

1. For all $x, y, z \in G$, it holds that $(x \star y) \star z = x \star (y \star z)$ (*associativity*).
2. There exists an $e \in G$ such that $x \star e = e \star x = x$, for all $x \in G$ (*identity*).
3. For all $x \in G$, there exists a $y \in G$ (often denoted x^{-1}) such that $x \star y = x \star y = e$ (*inverse*).

Example 2.4.3. Examples of groups include $(\mathbb{Z}, +), (\mathbb{R} \setminus \{0\}, \cdot), (\mathbb{Q}, +)$. An example of a non-group is $(\mathbb{N}, +)$ (not every element has an inverse). ◊

Instead of writing out the $\star$ operation explicitly, we usually denote the group operation by juxtaposition ($a \star b$ is written as ab), and refer to the operation as "multiplication." Predictably, we use the notation g^x to refer to the successive multiplication of g by itself x times.

Definition 2.4.4. Let $(G_1, \star)$ and $(G_2, \circ)$ be groups. A map $\varphi\colon G_1 \longrightarrow G_2$ is called a *homomorphism* if $\varphi(g \star h) = \varphi(g) \circ \varphi(h)$, for all $g, h \in G_1$. A bijective homomorphism is called an *isomorphism*. If $\varphi\colon G_1 \longrightarrow G_2$ is an isomorphism, we say that G_1 and G_2 are *isomorphic* and we write $G_1 \cong G_2$.

Homomorphism and isomorphism are two notions of equality for groups. Two groups are homomorphic if they have the same algebraic structure, even if the elements themselves are not the same. Two groups are isomorphic if they are identical, up to a renaming of the elements.

Definition 2.4.5. Let G be a group under a binary operation $\star$, and let $H \subseteq G$. We say H is a *subgroup* of G, and write $H \leq G$, if H is also a group under $\star$.

It is important to notice that there is a distinction between *subgroups* and *subsets*. An arbitrary subset of a group may not itself be a group. Furthermore, even if a subset of a group is itself a group, it must be a group under the *same operation* in order to be considered a subgroup.

Definition 2.4.6. A group G is called *abelian* if $xy = yx$, for all $x, y \in G$ (that is, the operation is commutative). In this case, we often write the operation as addition instead of multiplication.

Example 2.4.7. An example of an abelian group is $(\mathbb{Z}, +)$. An example of a non-abelian group is $(M_2(\mathbb{R}), \cdot)$, the group of 2×2 real matrices under matrix multiplication. ◊

Definition 2.4.8. Let G be a group, and let $H \leq G$. Then, for any $g \in G$, the set

$$g + H \overset{\text{def}}{=} \{x + h \mid h \in H\}$$

is called the *coset of H containing g*.

Let us define an operation on cosets as follows. If $a + H$ and $b + H$ are two cosets of H, define $(a + H) + (b + H)$ to be the coset $(a + b) + H$. It is easily shown that this operation defines a group, with H as the identity. We call this the *quotient group* of G with respect to H, and write it as G/H.

Suppose $(G_1, +_1), \ldots, (G_n, +_n)$ are groups. Consider the Cartesian product of $G_1, \ldots, G_n$ as sets:

$$G_1 \times \cdots \times G_n \overset{\text{def}}{=} \{(g_1, \ldots, g_n) \mid g_1 \in G_1, \ldots, g_n \in G_n\}.$$

For any tuples $(g_1, \ldots, g_n)$ and $(h_1, \ldots, h_n)$, define an operation $+$ such that

$$(g_1, \ldots, g_n) + (h_1, \ldots, h_n) \overset{\text{def}}{=} (g_1 +_1 h_1, \ldots, g_n +_n h_n).$$

Again, this easily defines a group, called the *direct product* of $G_1, \ldots, G_n$.

Definition 2.4.9. A group G is called *cyclic* if there exists a $g \in G$, such that for all $h \in G$, there exists an $x \in \mathbb{Z}$ such that $g^x = h$. In this case, g is called the *generator* of G, and we write $G = \langle g \rangle$.

In the study of cryptography, nearly all groups that we encounter will be abelian. Furthermore, the majority of them will be cyclic, as well. Observe that if G is a group, and if $a \in G$, then the cyclic group $\langle a \rangle$ generated by a is a subgroup of G. A common theme will be the analyses of groups in terms of their cyclic subgroups.

Example 2.4.10. Let $(\mathbb{Z}_n, +)$ be the group of integers modulo n under addition. Then, $(\mathbb{Z}_n, +)$ is a cyclic group. $\Diamond$

We refer to $(\mathbb{Z}_n, +)$ as *the* cyclic group of order n. Typically, we write this group as $\mathbb{Z}_n$, and the operation is understood to be addition. The next theorem implies that the *only* finite cyclic groups are of the form $\mathbb{Z}_n$, for some integer n.

Theorem 2.4.11. *If G is cyclic, then either $G \cong \mathbb{Z}_n$, for some n, or $G \cong \mathbb{Z}$.*

Proof. Let $G = \langle g \rangle$, for some $g \in G$. Suppose for all positive integers n, it holds that $g^n \neq e$. Furthermore, suppose that there exist integers x, y such that $g^x = g^y$. But then, $g^{x-y} = e$, which contradicts the fact that there is no n such that $g^n = e$. Thus, every element of G can be written as g^x, for some unique x. Let $\varphi\colon G \longrightarrow \mathbb{Z}$ be given by $g^x \mapsto x$. By our preceding argument, φ is well-defined (since G is cyclic) and bijective. Additionally, $\varphi(g^x g^y) = \varphi(g^{x+y}) = x + y = \varphi(g^x) + \varphi(g^y)$, so φ is a homomorphism. Thus, $G \cong \mathbb{Z}$.

Now, suppose $g^n = e$, for some positive integer n. Let n be the smallest such integer such that $g^n = e$. For any $s \in \mathbb{Z}$, suppose $s = nr + q$. Then,

$$\begin{aligned} g^s &= g^{nq+r} \\ &= (g^n)^q g^r \\ &= g^r. \end{aligned}$$

But then $s - r < n$ and $g^{s-r} = e$, contradicting the fact that n was the least integer such that $g^n = e$. Thus, the elements $e, g, g^2, \ldots, g^{n-1}$ are all distinct, and every element of G is one of these. So, the map $\varphi\colon \mathbb{Z}_n \longrightarrow G$ given by $i \mapsto g^i$ is a bijection. Additionally,

$$\begin{aligned} \varphi(a+b) &= g^{a+b} \\ &= g^a g^b \\ &= \varphi(a)\varphi(b), \end{aligned}$$

so φ is a homomorphism. Thus, $G \cong \mathbb{Z}_n$. $\square$

Definition 2.4.12. Let G be a group. The *order* of G, denoted $\#G$, is the number of elements in the set G.

Definition 2.4.13. Let G be a group, and let $g \in G$. Then, the *order* of g, denoted $\operatorname{ord}(g)$ is defined to be $\#\langle g \rangle$.

An alternative way of thinking about the order of a group element is that $\operatorname{ord}(g)$ is the smallest $x \in \mathbb{Z}$ such that $g^x = 1$.

Occasionally, groups will not provide enough algebraic structure for our purposes. If we augment a group with a second binary operation, and relate the two operations in the most natural way, we get a *ring*.

Definition 2.4.14. A triple $(R,+,\cdot)$ of a set R together with two binary operations (called "addition" and "multiplication") is called a *ring* if the following axioms hold.

1. $(R,+)$ is an abelian group.
2. For all $x,y,z \in R$, it holds that $x\cdot(y+z) = (x\cdot y)+(x\cdot z)$ and $(x+y)\cdot z = (x\cdot z)+(y\cdot z)$ (*distributive law*).
3. For all $x,y,z \in R$, it holds that $(x\cdot y)\cdot z = x\cdot(y\cdot z)$ (*multiplicative associativity*).
4. There exists a $1 \in R$ such that $1\cdot x = x\cdot 1 = x$, for all $x \in R$ (*multiplicative identity*).

One of the most interesting examples of rings will turn out to be extremely useful, as well.

Definition 2.4.15. Let R be a ring. Then, the set of formal sums

$$R[\mathrm{X}] \overset{\text{def}}{=} \left\{ \sum_{i=0}^{\infty} a_i \mathrm{X}^i \mid a_i \in R, \text{all but finite } a_i \text{ equal to } 0 \right\}$$

is called the *ring of polynomials over R with indeterminate* X. It can easily be verified to be a ring under the operations of polynomial addition and multiplication.

We refer to polynomials as "formal sums" in the sense that they should be viewed as strings, rather than as numbers. The operations of polynomial addition and multiplication are then defined over these sets of strings, to correspond with their natural interpretations, as if they were acting on functions.

Definition 2.4.16. A ring $(F,+,\cdot)$ is called a *field* if the following two axioms hold.

1. For all $x,y \in F$, it holds that $x\cdot y = y\cdot x$ (*multiplicative commutativity*).
2. For all $x \in F$, there exists a $y \in F$ such that $x\cdot y = y\cdot x = 1$. In this case, we denote y as x^{-1}. Additionally, we write x/y for $x\cdot y^{-1}$, for all $x,y \in F$ (*multiplicative inverse*).

We mention that all of the traditional results of linear algebra hold in a vector space over a field.

Definition 2.4.17. Let p be a prime number. We denote by $\mathbb{F}_p$ the finite field $(\mathbb{Z}_p, +, \cdot)$.

The following theorem, due to Lagrange, will prove invaluable later on.

Theorem 2.4.18. *Let $x_1, \ldots, x_n$ be distinct elements of $\mathbb{F}_q$, and let $f : \mathbb{F}_q \longrightarrow \mathbb{F}_q$. Then, there exists a unique polynomial $p \in \mathbb{F}_q[\mathrm{X}]$ of degree $n-1$ such that $f(x_j) = p(x_j)$, for $j = 1, \ldots, n$.*

Proof. Let

$$p(\mathrm{X}) \overset{\text{def}}{=} \sum_{i=1}^{n} f(x_i)\lambda_i(\mathrm{X}), \qquad \lambda_j(\mathrm{X}) \overset{\text{def}}{=} \prod_{i=1, i\neq j}^{n} \frac{\mathrm{X} - x_i}{x_j - x_i}, j = 1, \ldots, n.$$

Observe that p has degree $n-1$, since each $f(x_i)\lambda_j(\mathrm{X})$ has degree $n-1$, and the sum of two degree $n-1$ polynomials has degree $n-1$. It is easy to see that

$$\lambda_i(x_j) = \begin{cases} 0 & \text{if } i \neq j, \\ 1 & \text{otherwise.} \end{cases}$$

For each x_j, note that

$$\begin{aligned} p(x_j) &= \sum_{i=1}^{n} f(x_i)\lambda_i(x_j) \\ &= f(x_j). \end{aligned}$$

It remains to show that p is unique. Suppose $q \in \mathbb{F}_q[\mathrm{X}]$ with degree $n-1$ such that $f(x_j) = q(x_j)$, for $j = 1, \ldots, n$. Then, $r(\mathrm{X}) = p(\mathrm{X}) - q(\mathrm{X})$ is a polynomial of degree $n-1$. Also,

$$\begin{aligned} r(x_j) &= p(x_j) - q(x_j) \\ &= f(x_j) - f(x_j) \\ &= 0. \end{aligned}$$

The only way a degree $n-1$ polynomial can have n roots of 0 is if it is identically 0, so it follows that $p = q$. $\square$

Definition 2.4.19. Let R be a ring. An additive subgroup I of R is called an *ideal* if $aI \subseteq I$, for all $a \in R$.

As with groups, we can define a notion of cosets for ideals.

Definition 2.4.20. Let R be a ring, and let $I \subseteq R$ be an ideal. Then, for any $r \in R$, the set

$$r + I \stackrel{\text{def}}{=} \{r + i \mid i \in I\}$$

is called the *coset of I containing r*.

We can then define two operations on cosets given by $(a+I)+(b+I) = (a+b)+I$ and $(a+I)(b+I) = ab+I$, and we can check that the set of cosets of I form a ring under these operations. We call this ring the *quotient ring* of R with respect to I, and write R/I. Again, as with groups, we can also define notions of homomorphism, isomorphism, and direct product of rings in the natural way.

Definition 2.4.21. Let R be a ring, and let $I, J \subseteq R$ be ideals. We say I and J are *relatively prime* if $I + J = R$.

Observe that the above definition is a generalization of the concept of relative primality in $\mathbb{Z}$. For instance, if I and J are relatively prime, then for any $r \in R$, it holds that $r = aI + bJ$, for some $a, b \in R$.

Theorem 2.4.22 (Chinese Remainder Theorem)**.** *Let R be a ring, and let $I_1, \ldots, I_k$ be pairwise relatively prime ideals of R, with I their product. Then, $R/I \cong R/I_1 \times \cdots \times R/I_k$, the isomorphism given by $x + I \mapsto (x + I_1, \ldots, x + I_k)$.*

Example 2.4.23. Let $n = pq$, where p and q are prime. Then, consider the ring $\mathbb{Z}_n$. Observe that $I = \{nq \mid n \in \mathbb{Z}_n\}$ and $J = \{np \mid n \in \mathbb{Z}_n\}$ are ideals of $\mathbb{Z}_n$. Furthermore, $\mathbb{Z}_n/I \cong \mathbb{Z}_p$, and $\mathbb{Z}_n/J \cong \mathbb{Z}_q$, and since $p \perp q$, it holds that I and J are relatively prime. Thus, by the Chinese Remainder Theorem, $\mathbb{Z}_n \cong \mathbb{Z}_p \times \mathbb{Z}_q$. $\Diamond$

2.5 Number Theory

Definition 2.5.1. For $n \in \mathbb{N}$, define the *Euler totient function* $\phi(n)$ as the number of naturals less than n and relatively prime to n.

Proposition 2.5.2. *We can inductively define $\phi(n)$ as follows.*

$$\phi(n) = \begin{cases} p^e - p^{e-1} & \text{if } n = p^e, \text{ with } p \text{ prime,} \\ \prod_{i=1}^{k} \phi(p_i^{e_i}) & \text{if } n = p_1^{e_1} \cdots p_k^{e_k}, \text{ with } p_i \text{ prime.} \end{cases}$$

Notice that the group $\mathbb{Z}_n^*$ has order $\phi(n)$.

Definition 2.5.3. For $n \in \mathbb{N}$, define the *Carmichael function* $\lambda(n)$ as the smallest natural number that satisfies $a^{\lambda(n)} = 1$, for all $a \in \mathbb{Z}_n^*$.

Proposition 2.5.4. *We can inductively define $\lambda(n)$ as follows.*

$$\lambda(n) = \begin{cases} \phi(n) & \text{if } n = 2^e, e \leq 2, \\ \phi(n)/2 & \text{if } n = 2^e, e \geq 3, \\ \phi(n) & \text{if } n = p^e, p \geq 3 \text{ prime}, \\ \operatorname{lcm}_{i=1,\ldots,k}\{\lambda(p_i^{e_i})\} & \text{if } n = p_1^{e_1} \cdots p_k^{e_k}, \text{ with } p_i \text{ prime}. \end{cases}$$

Definition 2.5.5. A number $z \in \mathbb{Z}_n^*$, for some $n \in \mathbb{N}$, is said to be an *ith residue modulo nresidue* if there exists a $y \in \mathbb{Z}_n^*$ such that $z = y^i$. If $z = y^2$, for some $y \in \mathbb{Z}_n^*$, then z is called a *quadratic residue*. The set of all quadratic residues modulo n is denoted by QR_n.

Definition 2.5.6. For each prime p and for any $x \in \mathbb{Z}_p^*$, let

$$(x \mid p) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } x \in \mathrm{QR}_p, \\ 0 & \text{otherwise.} \end{cases}$$

Then, $(x \mid p)$ is called the *Legendre symbol* of x modulo p. If $n = p_1 p_2 \cdots p_k$, where $p_1, \cdots, p_k$ are prime, then

$$(x \mid n) \overset{\text{def}}{=} (x \mid p_1)(x \mid p_2) \cdots (x \mid p_k)$$

is called the *Jacobi symbol* of x modulo n.

Proposition 2.5.7. *Let $n \geq 1$, and let $x \in \mathbb{Z}_n^*$. Then, $(x \mid n)$ can be computed in polynomial time in* $\mathrm{len}(x) + \mathrm{len}(n)$.

The reader is referred to [Sho05] for the details of the algorithm.

The following theorem is a corollary of Theorem 2.4.22, and is itself often called the "Chinese Remainder Theorem."

Corollary 2.5.8. *Let $n = n_1 \cdots n_k$, where $n_1, \ldots, n_k$ are pairwise relatively prime, and let $r_1, \ldots, r_k$ be integers. Then, there is a unique $r \in \mathbb{Z}$ such that $r \equiv r_i \pmod{n_i}$, for $i = 1, \ldots, k$.*

2.6 Complexities of Arithmetic Operations

We summarize the time complexities of arithmetic operations.

Operation	Complexity
$a+b$	$\Theta(\lg a + \lg b) = \mathrm{O}(\lg n)$
$a-b$	$\Theta(\lg a + \lg b) = \mathrm{O}(\lg n)$
$a \cdot b$	$\Theta(\lg a \lg b) = \mathrm{O}(\lg(n)^2)$
a^{-1}	$\mathrm{O}(\lg(n)^2)$
g^e	$\mathrm{O}(\lg e \lg(n)^2)$
$\gcd(a,b)$	$\mathrm{O}(\lg(n)^2)$
$a \bmod n$	$\mathrm{O}(\lg(n)^2)$

Table 2.1: Time complexities of arithmetic operations in $\mathbb{Z}_n$

Chapter 3

Cryptographic Preliminaries

We now introduce several of the main objects of study. The cryptographic systems discussed in future chapters rely on several difficult problems. In this chapter, we establish these problems.

3.1 Adversarial Advantage

In designing a cryptographic system, we aim toward ensuring that no adversary can compromise it. Realistically, we do not require that the system be secure against any *possible* adversary. Rather, we require that the system be secure against any *realistic* adversary. Furthermore, it may not be reasonable to require that the system is *always* secure. A system that is *almost always* secure may be good enough. Thus, we design our systems so that they are *almost always* secure against any *realistic* adversary.

The preceding requires some formalization. First, when we talk about "realistic" adversaries, we are making a claim about the computational ability of the adversary. Specifically, we view realistic adversaries as those adversaries that can be modeled by probabilistic polynomial-time Turing machines. Second, when we say that a system is "almost always secure," we mean that the probability that an adversary can defeat the system is "negligible." This term has a formal definition, given below.

Definition 3.1.1. A function $f : \mathbb{N} \longrightarrow \mathbb{R}^+$ is said to be *negligible* if $f \in o(1/n^k)$, for all $k \in \mathbb{N}$.

Intuitively, we are trying to capture the notion that the probability that the system can be defeated is negligible if the odds cannot be beaten within the lifetime of any polynomially-bounded adversary.

Remark 3.1.2. We sometimes use the notation "negl" to refer to an anonymous negligible function.

We would now like to develop a formalization of what it means to talk about how likely it is for an adversary to defeat the system.

Definition 3.1.3. Let $E = \{E_i\}_{i \in \mathbb{N}}$ and $F = \{F_i\}_{i \in \mathbb{N}}$ be sequences of distributions, called *ensembles*, over a sample space Ω. Define, for any PPT $\mathscr{A}$ running in time polynomial in $\lg |\Omega|$,

$$\mathrm{Adv}_{\mathscr{A}}^{(E,F)}(1^\lambda) \overset{\text{def}}{=} \Big| \Pr_{a \leftarrow E_\lambda}[\mathscr{A}(a) = 1] - \Pr_{a \leftarrow F_\lambda}[\mathscr{A}(a) = 1] \Big|.$$

If, for all PPT $\mathscr{A}$, it holds that $\mathrm{Adv}_{\mathscr{A}}^{(E,F)}(1^\lambda)$ is negligible for some λ, we say that E and F are *indistinguishable*, and write $E \approx F$.

3.2 The Discrete Logarithm

Modern cryptography is based on the difficulty of solving certain problems in computational number theory. Thus, the security of all our cryptosystems will be shown by reduction to these hard problems. The first of these problems is the discrete logarithm problem, defined below.

Definition 3.2.1. Given $h \in \langle g \rangle$, the problem of finding x such that $g^x = h$ is called the *discrete logarithm problem* (DLOG). In this case, x is denoted by $\log_g h$.

It is important to note that the discrete logarithm only bears a passing resemblance to logarithms in the real numbers. In particular, none of the techniques for computing and approximating logarithms in the reals works for logarithms in arbitrary cyclic groups.

3.2.1 Algebraic Setting

When we consider the discrete logarithm problem, it is important that we are aware of the fact that the problem may not be equally difficult in all groups. As an example, consider a group whose order has many small prime factors. Recall that $\#\mathbb{Z}_p^* = \phi(p) = p-1$. Suppose that $p-1 = q_1 \cdot q_2 \cdots q_n$, where $q_1, q_2, \ldots, q_n$ are "small" compared to p. Based on this factorization of $p-1$, we see that for each q_i, there is a subgroup $G_i \leq \mathbb{Z}_p^*$ whose order is small compared to that of $\mathbb{Z}_p^*$.

Lemma 3.2.2. *Let p be a prime such that $p - 1 = q_1 \cdot q_2 \cdots q_n$. Let G_i be the subgroup of $\mathbb{Z}_p^*$ of order q_i. Then, for each $i \in \{1, \ldots, n\}$, there is a homomorphism $f_i \colon \mathbb{Z}_p^* \longrightarrow G_i$ given by*

$$x \mapsto x^{(p-1)/q_i}.$$

Suppose we are given $y = g^x \in \mathbb{Z}_p^*$, and we would like to compute $x = \log_g y$. First, we can find the projection of y in each G_i by computing $x_i = f_i(y)$. Then, we can solve the discrete logarithm in G_i by brute force, since $\#G_i$ is small. Now, observe that we have the system of equations $x \equiv x_i \pmod{q_i}$. Finally, we can apply the Chinese Remainder Theorem to this system to get x.

Obviously, to avoid this sort of attack, we should have selected p such that $p - 1$ has very few small factors.

Definition 3.2.3. A prime p is called a *safe prime* if $p = 2q + 1$, for some prime q.

If p is a safe prime, then $p - 1$ has only two factors: 2 and q. Thus, $\mathbb{Z}_p^*$ has only two non-trivial subgroups: one of order 2, and one of order q. The q-order subgroup turns out to be QR_p, and it is in this group that we wish to perform all of our computations, as it has *no* non-trivial subgroups. The distribution of safe primes among the prime numbers is not well understood. It is possible that they are considerably less dense. In fact, it is unknown whether or not there are even an infinite number of safe primes.

3.2.2 The Diffie-Hellman Problems

Suppose Alice and Bob want to agree on a secret key. Furthermore, they wish to perform this agreement over an insecure channel, so that no one who is privy to the transcript of their communication can deduce the key. Diffie and Hellman propose [DH76] the protocol in Figure 3.1.

Now, suppose that Eve, an eavesdropper, wishes to compute k for herself. That is, she knows g^x and g^y (these were sent over the channel), and she wishes to compute g^{xy}. This problem can be stated as follows.

Definition 3.2.4. Given $g^x, g^y \in \langle g \rangle$, the problem of finding g^{xy} is called the *computational Diffie-Hellman problem* (CDH).

It is clear that an adversary who can solve DLog can certainly break the DH key exchange. Indeed, this is the main intuition behind its security. However, the CDH is a more accurate task for the adversary who wishes to recover the

1. Alice and Bob agree on a cyclic group $\mathbb{Z}_p^*$ and a generator g of a q-order subgroup.
2. Alice chooses $x \in_{\mathrm{U}} \mathbb{Z}_q$, and Bob chooses $y \in_{\mathrm{U}} \mathbb{Z}_q$.
3. Alice sends Bob g^x, and Bob sends Alice g^y.
4. Alice computes $k \leftarrow (g^y)^x$, and Bob computes $k \leftarrow (g^x)^y$.

Figure 3.1: Diffie-Hellman Key Exchange

exchanged key. In principle, it may be possible to solve CDH without solving DLog, using some unknown technique.

The question then arises: is CDH itself the necessary task for an adversary to break in order to compromise the DH key exchange? Suppose an adversary is unable to recover the full key, but it can recover, say, one half of the bits of the key. This adversary has not solved CDH, but it has in some sense compromised the system. Ideally, the adversary would be able to recover *no* bits of the key. The most general formulation of this goal would be to say that the adversary cannot distinguish between a random transcript of the protocol, and a transcript in which an actual key is exchanged. If this property is satisfied, then it is clear that the adversary could not have gained *any* useful information about the key from witnessing the transcript of the protocol, as it cannot even tell if a key has been exchanged or not. We state this formally below.

Definition 3.2.5. Given $g^x, g^y, g^z \in \langle g \rangle$, the problem of determining whether or not $z = xy$ is called the *decisional Diffie-Hellman problem* (DDH).

We can formally state the relationships between the three discrete log problems.

Theorem 3.2.6. *In any cyclic group,* DDH $\leq_{\mathrm{P}}$ CDH $\leq_{\mathrm{P}}$ DLog.

Proof. Suppose an adversary can solve DLog. Then, given (g^a, g^b), the adversary can compute both $a = \log_g g^a$ and $b = \log_g g^b$ and then compute g^{ab} directly. Thus, the adversary can solve CDH. Now, given (g^a, g^b, g^c), the adversary, being able to compute CDH, can compute g^{ab} and compare it with g^c, thus solving DDH. □

Theorem 3.2.6 says that DLog is harder than both CDH and DDH. That is, if an adversary is able to solve DLog, it can easily solve the other two. With this in mind, we see that the assumption that DDH is hard is in some sense the strongest assumption we can make. If it turns out to be true, then we know that all three problems must be difficult to solve.

Now that we have stated the problems, we must formalize what it means to assume that these problems are difficult.

Definition 3.2.7. Let $G = \langle g \rangle$ be a cyclic group. The *description* of G, denoted $\mathrm{desc}(G)$, is a string representation of the group, consisting of: an efficient membership test, a description of the group operation, a representation of the identity, and an efficient algorithm for computing the inverse of a group element[1]. Suppose $\mathscr{G}$ is a PPT that on input 1^λ, where λ is a *security parameter*, returns $\mathrm{desc}(G)$ (of length polynomial in λ), for some group G. We call $\mathscr{G}$ a *group generator* for G.

It is important to notice that $\mathrm{desc}(G)$ contains no information about the *structure* of G. It only contains the necessary information to perform computations within the group. This is a necessary restriction, as most cryptographic systems rely on the fact that the adversary does not have too much knowledge of the structure of the underlying group.

Definition 3.2.8. Let $\mathscr{G}$ be a group generator that on input 1^λ returns $\mathrm{desc}(G)$, where G has generator g of order q. We say $\mathscr{G}$ satisfies the *decisional Diffie-Hellman assumption* if, for all PPT $\mathscr{A}$,

$$\Big| \Pr_{x,y \leftarrow [q]}[\mathscr{A}(\mathrm{desc}(G), g^x, g^y, g^{xy}) = 1] - \\ \Pr_{x,y,z \leftarrow [q]}[\mathscr{A}(\mathrm{desc}(G), g^x, g^y, g^z) = 1] \Big| \leq \varepsilon(\lambda),$$

where ε is a negligible function. That is, the DDH problem is hard to solve in G.

It should be noted that the DDH assumption does *not* hold in all groups.

Proposition 3.2.9. *The DDH assumption does not hold in $\mathbb{Z}_p^*$, for any prime p.*

Proof. We wish to construct a PPT $\mathscr{A}$ that can distinguish between DDH tuples and random tuples. Given (a, b, c), the adversary $\mathscr{A}$ will return 1 if either all three of $(a \mid p), (b \mid p)$, and $(c \mid p)$ equal -1, or if both $(a \mid p)$ and $(b \mid p)$ equal 1 but $(c \mid p)$ equals -1. Thus, $\Pr_{x,y,z \leftarrow [q]}[\mathscr{A}(g^x, g^y, g^z) = 1] = 1$ and $\Pr_{x,y \leftarrow [q]}[\mathscr{A}(g^x, g^y, g^{xy}) = 1] = 1/2$. Since the Jacobi symbol is computable in polynomial time, $\mathrm{Adv}_{\mathscr{A}}^{\mathrm{DDH}} = 1/2$. $\square$

[1]It is outside the scope of this thesis to explain precisely how this information can be represented. It should suffice to say that there exists some standard encoding that can be interpreted by any Turing machine that needs to.

The group $\mathbb{Z}_p^*$ fails the DDH assumption because it is easy to determine the property of quadratic residuosity. This leads us to believe that if we use a group in which it is not so easy to distinguish between elements, the DDH assumption might hold.

Assumption 3.2.10. *Let $\mathcal{G}$ be a group generator that on input 1^λ, selects a random safe prime $p = 2q + 1$ such that $\operatorname{len} q = \lambda$, and $g \in_{\mathrm{U}} \mathrm{QR}_p$, and outputs $\operatorname{desc}(\mathrm{QR}_p)$. Then, $\mathcal{G}$ satisfies the decisional Diffie-Hellman assumption.*

This assumption says that the DDH problem is difficult to solve in $\mathrm{QR}_p \leq \mathbb{Z}_p^*$, for sufficiently large primes p. Realistically, we demand that the size of p be on the order of 1024 bits or larger.

We conclude the section on the Diffie-Hellman problems by providing one final assumption which will be used later on.

Definition 3.2.11. Suppose p and q are prime numbers of approximately the same length. Then, $n = pq$ is called an *RSA composite*.

Assumption 3.2.12 (Composite-DDH Assumption)**.** *Let $\mathcal{G}$ be a group generator that on input 1^λ, selects an RSA composite $n = pq$ such that $\operatorname{len} p = \lambda$, an element $g \in_{\mathrm{U}} \mathrm{QR}_n$, and outputs $\operatorname{desc}(\mathrm{QR}_n)$. Then, $\mathcal{G}$ satisfies the decisional Diffie-Hellman assumption.*

3.3 The Composite Residuosity Problem

There is another class of number theoretic problems that we will study. These are based on the difficulty of determining residuosity classes in groups of composite order. In the remainder of this section, it is assumed that $n = pq$ is an RSA composite, $n \perp \phi(n)$, and $s \geq 1$. Then,

$$\begin{aligned}
\phi(n^{s+1}) &= \phi(p^{s+1}q^{s+1}) \\
&= \phi(p^{s+1})\phi(q^{s+1}) \\
&= \phi(p^s q^s (p-1)(q-1)) \\
&= n^s \phi(n).
\end{aligned}$$

Thus, by the Chinese Remainder Theorem, $\mathbb{Z}_{n^{s+1}}^* \cong \mathbb{Z}_{n^s} \times \mathbb{Z}_n^*$.

The fundamental computational problem that we consider is the following.

Definition 3.3.1. The problem of distinguishing nth residues from nth non-residues is called the *composite residuosity problem*, and is denoted by $\mathrm{CR}[n]$.

As with the Diffie-Hellman problem, we define an property that we expect group generators to have if they are to be considered secure. The following assumption is known as the *decisional composite residuosity assumption* (DCRA).

Assumption 3.3.2 (Decisional Composite Residuosity Assumption). *Let $\mathcal{G}$ be a PPT that, on input 1^{λ}, outputs a random RSA composite $n = pq$ such that* $\operatorname{len} p = \lambda$. *Then, for all PPT $\mathscr{A}$,*

$$\left| \Pr_{x \leftarrow \mathbb{Z}^*_{n^{s+1}}} [\mathscr{A}(n,x)=1] - \Pr_{x \leftarrow \mathbb{Z}^*_{n^{s+1}}/\mathbb{Z}_{n^s}} [\mathscr{A}(n,x)=1] \right| \leq \varepsilon(\lambda),$$

where ε is a negligible function. That is, $\mathrm{CR}[n]$ *is a difficult problem.*

We further explore the group structure of $\mathbb{Z}^*_{n^{s+1}}$ with the following lemma.

Lemma 3.3.3. *Provided $s < p, q$, the element $1+n$ has order n^s in $\mathbb{Z}^*_{n^{s+1}}$.*

Proof. We would like to find the minimum value of i for which $(1+n)^i \equiv 1 \pmod{n^{s+1}}$. By the binomial theorem, we have that

$$(1+n)^i = \sum_{j=0}^{i} \binom{i}{j} n^j,$$

which is $1 \pmod{n^{s+1}}$ if and only if

$$\sum_{j=1}^{i} \binom{i}{j} n^{j-1} \equiv 0 \pmod{n^s}.$$

Now, if $i = n^s$, then

$$\begin{aligned} \sum_{j=1}^{n^s} \binom{n^s}{j} n^{j-1} &= \binom{n^s}{1} n^0 + \binom{n^s}{2} n + \cdots + \binom{n^s}{n^s} n^{n^s-1} \\ &\equiv 0 \pmod{n^s}. \end{aligned}$$

Thus, it follows that $\operatorname{ord}(1+n) \mid n^s$. That is, $\operatorname{ord}(1+n) = p^{\alpha} q^{\beta}$, with $\alpha, \beta \leq s$. Let $a = \operatorname{ord}(1+n) = p^{\alpha} q^{\beta}$. We now claim that each term in $\sum_{j=1}^{a} \binom{a}{j} n^{j-1}$ is divisible by a. Suppose $j \leq s$. Then, $j! \leq s!$, and since $s < p, q$, we have that $j!$ is less than both $p!$ and $q!$. So, neither p nor q divides $j!$. Thus,

$$\begin{aligned} \binom{a}{j} &= \frac{a!}{j!(a-j)!} \\ &= \frac{a(a-1)\cdots(a-j)!}{j!(a-j)!} \\ &= \frac{a(a-1)\cdots(a-j+1)}{j!}, \end{aligned}$$

which is divisible by a. So, each term in $\sum_{j=1}^{a} \binom{a}{j} n^{j-1}$ is divisible by a. Now, suppose that $a = p^\alpha q^\beta < n^s$, and without loss of generality that $\alpha < s$. We now want to show that $\sum_{j=1}^{a} \binom{a}{j} n^{j-1}$ is divisible by n^s. This follows from the fact that $\sum_{j=1}^{a} \binom{a}{j} n^{j-1} \equiv 0 \pmod{n^s}$, by the way we chose a. Now, dividing both sides by a, we see that that n^s/a divides $\left(\sum_{j=1}^{a} \binom{a}{j} n^{j-1}\right)/a$. But this implies that p divides $\left(\sum_{j=1}^{a} \binom{a}{j} n^{j-1}\right)/a$, which is $1 \pmod{p}$, a contradiction. Thus, $a \geq n^s$, and since $a \mid n^s$, it follows that $a = n^s$. That is, $\mathrm{ord}(1+n) = n^s$. □

Since $\#H = \phi(n)$, and $\phi(n) \perp n^s$, it follows that the coset $(1+n)H$ is a generator of $\mathbb{Z}^*_{n^{s+1}}/H$. Thus, we can order the cosets of H in $\mathbb{Z}^*_{n^{s+1}}$ as

$$H, (1+n)H, (1+n)^2 H, \ldots, (1+n)^{n^s-1} H.$$

Lemma 3.3.4. *The map* $\psi_s : \mathbb{Z}_{n^s} \times \mathbb{Z}^*_n \longrightarrow \mathbb{Z}^*_{n^{s+1}}$ *given by* $(x, r) \mapsto (1+n)^x r^{n^s}$ *is an isomorphism.*

Proof. Let $(x_1, r_1), (x_2, r_2) \in \mathbb{Z}_{n^s} \times \mathbb{Z}^*_n$. Then,

$$\begin{aligned} \psi_s\left[(x_1, r_1) \cdot (x_2, r_2)\right] &= \psi_s(x_1 + x_2, r_1 r_2) \\ &= (1+n)^{x_1+x_2} (r_1 r_2)^{n^s} \\ &= (1+n)^{x_1} r_1^{n^s} (1+n)^{x_2} r_2^{n^s} \\ &= \psi_s(x_1, r_1) \cdot \psi_s(x_2, r_2). \end{aligned}$$

Thus, ψ_s is a homomorphism. Now, suppose $(1+n)^{x_1} r_1^{n^s} = (1+n)^{x_2} r_2^{n^s}$. Then, $(1+n)^{x_1-x_2} = (r_2/r_1)^{n^s}$. But, $r_2/r_1 \in \mathbb{Z}^*_n$, so $(r_2/r_1)^{n^s} = 1$, and $r_2 = r_1$. Then, $(1+n)^{x_1-x_2} = 1$, and so $x_1 = x_2$, implying that ψ_s is injective. Finally, let $g \in \mathbb{Z}^*_{n^{s+1}}$. Then, g is a member of some coset $(1+n)^x H$. Let $r \in H$. Then, $(1+n)^x r^{n^s} = g$. So, ψ_s is surjective, and is thus an isomorphism. □

Furthermore, we can see that the isomorphism is easy to compute in both directions. This fact is essential for producing the encryption and decryption functions of the Paillier cryptosystem. The proof of the following lemma can be found in [DJ00].

Lemma 3.3.5. *The map* ψ_s *can be inverted in polynomial time given* $\lambda(n)$.

Proof. First, observe

$$\mathrm{L}((1+n)^i \bmod n^{s+1}) = (i + \binom{i}{2} n + \cdots + \binom{i}{s} n^{s-1}) \bmod n^s.$$

Define i_j to be $i \bmod n^j$. Notice that $i_j = i_{j-1} + kn^{j-1}$, for some $0 \le k < n$. Now, observe that

$$\mathrm{L}((1+n)^i \bmod n^{j+1}) = (i_j + \binom{i_j}{2}n + \cdots + \binom{i_j}{j}n^{j-1}) \bmod n^j.$$

For $0 < t < j$, we have that $\binom{i_j}{t+1}n^t = \binom{i_{j-1}}{t+1}n^t \bmod n^j$. Now,

$$\begin{aligned}&\mathrm{L}((1+n)^i \bmod n^{j+1}) = \\ &\qquad (i_{j-1} + kn^{j-1} + \binom{i_{j-1}}{2}n + \cdots + \binom{i_{j-1}}{j}n^{j-1}) \bmod n^j.\end{aligned}$$

Thus,

$$i_j = \mathrm{L}((1+n)^i \bmod n^{j+1}) - (\binom{i_{j-1}}{2}n + \cdots + \binom{i_{j-1}}{j}n^{j-1}) \bmod n^j.$$

Thus, given $c = (1+n)^x r^{n^s}$, we compute $c^\lambda = (1+n)^{\lambda x}$. Given the above relation, we can then find x. $\square$

Definition 3.3.6. For $w \in \mathbb{Z}^*_{n^{s+1}}$, the *nth residuosity class* of w, denoted $[w]$, is the unique $x \in \mathbb{Z}_{n^s}$ for which there exists a $y \in \mathbb{Z}^*_n$ such that $\psi_s(x,y) = w$.

From the definition of the factor group $\mathbb{Z}^*_{n^{s+1}}/\mathbb{Z}^*_n$, we notice that $[w] = 0$ if and only if w is an nth residue modulo n^{s+1}. Furthermore, for all $w_1, w_2 \in \mathbb{Z}^*_{n^{s+1}}$, it follows that $[w_1w_2] = [w_1] + [w_2]$. That is, the class function $w \mapsto [w]$ is a homomorphism from $\mathbb{Z}^*_{n^{s+1}}$ to $\mathbb{Z}_n$.

3.4 Encryption

Suppose Alice wishes to send a message to Bob over a public channel in such a way that no party who intercepts the message can read it. To perform this task, Alice and Bob should use an *encryption scheme*.

3.4.1 Formal Definition

We can formalize the notion of encryption as follows.

Definition 3.4.1. A *public-key cryptosystem*, known henceforth as a *cryptosystem*, is a 7-tuple $(M, C, K, R, \mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$, such that

1. M is the *message space*.

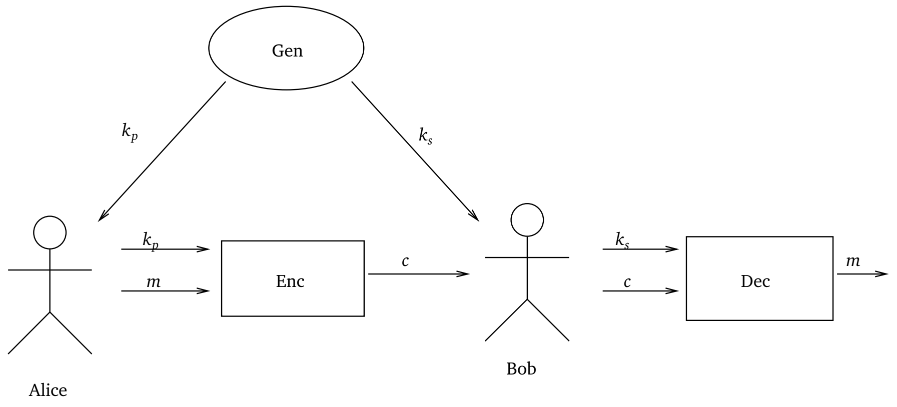

Figure 3.2: Diagram of encryption. Alice wants to send a message m to Bob, and encrypt it as a ciphertext c.

2. C is the *ciphertext space*.
3. K is the *key space*.
4. R is the *randomness space*.
5. Gen: $\mathbb{N} \times R \longrightarrow K \times K$ is a *key generation algorithm*.
6. Enc: $M \times K \times R \longrightarrow C$ is an *encryption algorithm*.
7. Dec: $C \times K \longrightarrow M$ is a *decryption algorithm*.

If $m \in M$ and $k \in K$, we denote by $\mathrm{Enc}_k(m; r)$ the ciphertext $\mathrm{Enc}(m, k, r)$. Likewise, for $c \in C$ and $k \in K$, we denote by $\mathrm{Dec}_k(c)$ the plaintext $\mathrm{Dec}(c, k)$.

Remark 3.4.2. The reader will note that we will always use the word "cryptosystem" to refer to asymmetric, probabilistic cryptosystems.

3.4.2 Security Definitions

To evaluate the security of a public key cryptosystem, we must have a clear definition of "security." When we formulate security definitions, we consider an idealized "game" that we play with the adversary. The game must capture exactly the security requirements of a real-life execution of the system. The system is considered secure if the probability that the game succeeds is independent of the abilities of the adversary.

$\mathrm{Game}^{\mathscr{A},\mathfrak{C}}_{\text{IND-CPA}}(1^\lambda)$	Random variables
$(k_p, k_s) \leftarrow \mathrm{Gen}(1^\lambda; \rho)$	$\rho \leftarrow R$
$(m_0, m_1, aux) \leftarrow \mathscr{A}_1(k_p)$	
$c \leftarrow \mathrm{Enc}_{k_p}(m_b; r)$	$b \leftarrow \{0,1\}$, $r \leftarrow R$
$b^* \leftarrow \mathscr{A}_2(c, aux)$	
if $b = b^*$ **then return** 1	
else return 0	

Figure 3.3: The IND-CPA game

IND-CPA Security

Our first notion of security for public-key cryptosystems is *indistinguishability under chosen plaintext attack* [GM84] (IND-CPA). This notion means that the adversary is unable distinguish between the encryptions of two plaintexts that it has chosen itself. We can view this as the minimum necessary security for a cryptosystem, as it ensures that the adversary cannot recover any information about a plaintext by inspecting the ciphertext.

Given a public key cryptosystem, we can define the following game, called the *IND-CPA game*. The game is parametrized by an adversary $\mathscr{A} = (\mathscr{A}_1, \mathscr{A}_2)$.

Definition 3.4.3. A cryptosystem $\mathfrak{C}$ is said to be *IND-CPA-secure* if for all PPT $\mathscr{A} = (\mathscr{A}_1, \mathscr{A}_2)$,

$$\Pr\left[\mathrm{Game}^{\mathscr{A},\mathfrak{C}}_{\text{IND-CPA}}(1^\lambda) = 1\right] = \frac{1}{2} + \varepsilon(\lambda),$$

where ε is a negligible function of λ.

ROR-CPA Security

We now define a notion of security that seems to be a loosening of IND-CPA. In this model, called *real or random chosen plaintext attack*, the adversary is faced with the task of determining whether a ciphertext encrypts a message of its choosing, or a random message.

Definition 3.4.4. A cryptosystem $\mathfrak{C}$ is *ROR-CPA-secure* if for all PPT $\mathscr{A} = (\mathscr{A}_1, \mathscr{A}_2)$,

$$\Pr\left[\mathrm{Game}^{\mathscr{A},\mathfrak{C}}_{\text{ROR-CPA}}(1^\lambda) = 1\right] = \frac{1}{2} + \varepsilon(\lambda),$$

where ε is a negligible function of λ.

$\mathrm{Game}_{\text{ROR-CPA}}^{\mathscr{A},\mathfrak{C}}(1^\lambda)$	Random variables
$(k_p, k_s) \leftarrow \mathrm{Gen}(1^\lambda; \rho)$	$\rho \leftarrow R$
$(m_0, aux) \leftarrow \mathscr{A}_1(k_p)$	
$m_1 \in_{\mathrm{U}} M$	
$c \leftarrow \mathrm{Enc}_{k_p}(m_b; r)$	$b \leftarrow \{0,1\}$, $r \leftarrow R$
$b^* \leftarrow \mathscr{A}_2(c, aux)$	
if $b = b^*$ **then return** 1	
else return 0	

Figure 3.4: The ROR-CPA game

Proposition 3.4.5. *If a cryptosystem is ROR-CPA-secure, then it is also IND-CPA-secure.*

Proof. Let $G_{\mathscr{A}}^{\mathscr{O}}$ be a modification of the IND-CPA game, on input 1^λ, where the adversary $\mathscr{A}$ receives the ciphertext from the oracle $\mathscr{O}$. We define four oracles as follows. Oracle $\mathscr{P}$ takes as input a plaintext, and encrypts the given plaintext. Oracle $\mathscr{R}$ takes as input a plaintext, and encrypts a random plaintext, ignoring its input. Oracle $\mathscr{P}_1$ takes as input a pair of plaintexts, and encrypts the first one. Oracle $\mathscr{P}_2$ takes as input a pair of plaintexts, and encrypts the second one. Let $\mathscr{A}$ be an IND-CPA adversary. Let $\mathscr{A}^{(1)}$ be an adversary who simulates $\mathscr{A}$ and at the first stage returns the first plaintext given by $\mathscr{A}$. Likewise, let $\mathscr{A}^{(2)}$ be an adversary who simulates $\mathscr{A}$ and at the first stage returns the second plaintext given by $\mathscr{A}$. It is then clear that

$$\Pr[G_{\mathscr{A}^{(i)}}^{\mathscr{P}} = 1] = \Pr[G_{\mathscr{A}}^{\mathscr{P}_i} = 1],$$

for $i = 1, 2$. Now, by the assumption that the cryptosystem is ROR-CPA-secure, we have that for all adversaries $\mathscr{B}$,

$$\left|\Pr[G_{\mathscr{B}}^{\mathscr{P}} = 1] - \Pr[G_{\mathscr{B}}^{\mathscr{R}} = 1]\right| = \mathrm{negl}.$$

Thus,

$$\left|\Pr[G_{\mathscr{A}^{(i)}}^{\mathscr{R}} = 1] - \Pr[G_{\mathscr{A}}^{\mathscr{P}_i} = 1]\right| = \varepsilon,$$

for $i = 1, 2$, where ε is negligible. Then,

$$\begin{aligned}
\mathrm{Adv}_{\mathscr{A}}^{\text{IND-CPA}}(1^\lambda) &= \left|\Pr[G_{\mathscr{A}}^{\mathscr{P}_2} = 1] - \Pr[G_{\mathscr{A}}^{\mathscr{P}_2} = 1]\right| \\
&= |\Pr[G_{\mathscr{A}}^{\mathscr{P}_1} = 1] - \Pr[G_{\mathscr{A}^{(1)}}^{\mathscr{R}} = 1] - \\
&\qquad \Pr[G_{\mathscr{A}}^{\mathscr{P}_2} = 1] + \Pr[G_{\mathscr{A}^{(2)}}^{\mathscr{R}} = 1]\big| \\
&\leq 2\varepsilon,
\end{aligned}$$

which is negligible. Thus, ROR-CPA security implies IND-CPA security. □

$\text{Game}^{\mathscr{A},\mathfrak{C}}_{\text{IND-CCA2}}(1^\lambda)$	Random variables
$(k_p, k_s) \leftarrow \text{Gen}(1^\lambda; \rho)$	$\rho \leftarrow R$
$(m_0, m_1, aux) \leftarrow \mathscr{A}_1^{\mathscr{D}}(k_p)$	
$c \leftarrow \text{Enc}_{k_p}(m_b; r)$	$b \leftarrow \{0,1\}$, $r \leftarrow R$
$b^* \leftarrow \mathscr{A}_2^{\mathscr{D}'}(c, aux)$	
if $b = b^*$ **then return** 1	
else return 0	

Figure 3.5: The IND-CCA2 game

IND-CCA2 Security

The strongest notion of security that we present is the *indistinguishability under adaptive chosen ciphertext attacks* [RS92] (IND-CCA2). In an adaptive chosen ciphertext attack, the adversary is allowed access to a decryption oracle in two steps of the game. First, it is allowed to inspect the public key, make as many queries as needed to the decryption oracle, and is required to produce two plaintexts. Then, it is presented with a random ciphertext, and is again allowed to query the decryption oracle, with the restriction that it cannot directly ask it to decrypt the challenge ciphertext. Finally, it must determine which of its two plaintexts is the decryption of the challenge ciphertext.

The game is shown in Figure 3.5. Observe that the adversary $\mathscr{A}$ has access to two decryption oracles, $\mathscr{D}$ and $\mathscr{D}'$. The oracle $\mathscr{D}$ functions as a normal decryption algorithm, which can decrypt any ciphertext. The oracle $\mathscr{D}'$ is the same except that it will return a failure if used to decrypt the ciphertext $\text{Enc}_{k_p}(m_b)$ that is given to $\mathscr{A}$.

At first glance, an adversary that can conduct an adaptive chosen ciphertext attack would seem to be unrealistically powerful. Surprisingly, however, a real-life attack was demonstrated that follows this pattern. In 1998, Daniel Bleichenbacher showed [Ble98] that an attacker can perform decryption or signing with RSA keys by exploiting flaws in the PKCS #1 encryption standard. The practical significance of this discovery is that it leaves SSL version 3 open to adaptive chosen ciphertext attacks.

Thus, IND-CCA2 would appear to be the "correct" notion of security for asymmetric cryptosystems. Nonetheless, this definition of security, as will be shown in Chapter 4, is too powerful for the type of voting we are interested in, as will be shown in Proposition 4.1.2.

3.5 Conclusion

We have introduced several important ideas in this chapter. First, we have described what it means for a system to be considered *secure*. That is, assuming the difficulty of some computational problem, a system is secure against an adversary if the adversary cannot distinguish between a real version of the system and an idealized version of the system, except with negligible probability. We have presented two problems which are believed to be computationally difficult, and upon which cryptographic systems can be constructed. Finally, we have introduced the idea of public-key encryption, and given three definitions of security for public-key encryption schemes. In the coming chapters, we will see how these ideas can be used to construct several different cryptographic primitives.

Chapter 4

Homomorphic Encryption Schemes

In this chapter, we explore a desirable property of encryption schemes for voting.

4.1 Introduction

One of our goals is to design cryptographic systems that naturally lend themselves to voting. The basic primitive we use is known as a *homomorphic encryption scheme*, introduced by Benaloh [Ben87]. To illustrate the behavior of this primitive, imagine that we have an encryption function Enc, and a sequence of ciphertexts $\mathrm{Enc}(m_1), \mathrm{Enc}(m_2), \ldots, \mathrm{Enc}(m_n)$. Suppose we would like to compute $\mathrm{Enc}(m_1 + m_2 + \cdots + m_n)$. This would appear impossible, as we do not posses the messages $m_1, m_2, \ldots, m_n$, and thus cannot add them and compute the encryption. However, if we have a homomorphic encryption function, this becomes possible. Presuming that there is a operation over ciphertexts (say, multiplication), it might hold that $\mathrm{Enc}(m_1)\,\mathrm{Enc}(m_2) \cdots \mathrm{Enc}(m_n) = \mathrm{Enc}(m_1 + m_2 + \cdots m_n)$. That is, we can add the messages without even knowing what they are, provided we can compute the operation defined over the ciphertext space.

Definition 4.1.1. Let $\mathfrak{C} = (M, C, K, R, \mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ be a cryptosystem. Suppose $(M, +)$, $(R, \oplus)$, and $(C, \cdot)$ are abelian groups with Enc_k a homomorphism from $M \times R$ into C. Suppose that if m and m' are drawn uniformly from M, and r and r' are drawn uniformly from R, then $\mathrm{Enc}_k(m + m'; r \oplus r')$ is drawn uniformly from C. Then, we say that $\mathfrak{C}$ is a *homomorphic* cryptosystem.

Intuitively, given two ciphertexts, a homomorphic cryptosystem lets us produce a third ciphertext encrypting some function of the two plaintexts. Why is this useful for voting? If we treat votes as being group elements, we would like to be able to add encryptions of these elements without knowing the corresponding plaintexts. A homomorphic encryption system allows us to add the ciphertexts, according to the group operation of the ciphertext space, which will then be mirrored in the plaintext group. Thus, we can preserve the privacy of voters, while still performing computations with their votes.

It should be noted that homomorphic encryption does not come without a price. Specifically, there is a reduction in security that makes any homomorphic encryption unsuitable for secure data transmission.

Proposition 4.1.2. *If $\mathfrak{C}$ is a homomorphic cryptosystem, then $\mathfrak{C}$ is* not *IND-CCA2 secure.*

Proof. Recalling Figure 3.5, suppose that $\mathscr{A}_2$ is given the ciphertext c. Consider the following behavior of $\mathscr{A}_2$. First, it will choose $m \in M$, and query the decryption oracle to get $c' = \mathrm{Enc}_{k_p}(m; r)$. Then, it will use the homomorphic property of the cryptosystem to compute $d = c \cdot c' = \mathrm{Enc}_{k_p}(m; r) \cdot \mathrm{Enc}_{k_p}(m_b; r') = \mathrm{Enc}_{k_p}(m + m_b; r + r')$. Finally, it will use the decryption oracle again to retrieve $m + m_b$. Subtracting m will reveal the plaintext m_b. Note that the decryption oracle will successfully decrypt d, as it is not the challenge ciphertext. □

Thus, the requirements of IND-CCA2 are too strong for the homomorphic approach to voting. Accordingly, we restrict ourselves to examining only IND-CPA secure cryptosystems. In this chapter, we will explore three such systems: those of Elgamal, Paillier and Damgård/Jurik.

4.2 The Elgamal Cryptosystem

Elgamal [Elg85] proposed a cryptosystem based on the discrete log problem. The cryptosystem is illustrated in Figure 4.1.

Efficiency. The key generation algorithm of Elgamal runs in an unspecified amount of time. This is because it is necessary to generate a safe prime, which could take a while. Fortunately, the safe prime itself is not secret, and can be reused. Assuming that the prime is already generated, the process involves two modular exponentiations, encryption involves three modular exponentiations, and decryption involves one.

Proposition 4.2.1. *The Elgamal cryptosystem is IND-CPA-secure under the decisional Diffie-Hellman assumption.*

Key generation Alice chooses a random prime number p, and selects g as a generator of an order q subgroup of $\mathbb{Z}_p^*$. She then selects $x \in_{\mathrm{U}} \mathbb{Z}_q$ and sets $h \leftarrow g^x$. She publishes (g, p, q, h) as her public key, and keeps x as her private key.

Encryption Bob selects a message $m \in \mathbb{Z}_p^*$, and $r \in_{\mathrm{U}} \mathbb{Z}_q$. He then computes $(G, H) \leftarrow (g^r, h^r m)$, and sends (G, H) to Alice.

Decryption Alice recovers m as H/G^x.

Figure 4.1: The Elgamal cryptosystem

Proof. Given a PPT $\mathscr{A}$ such that $\mathrm{Adv}_{\mathscr{A}}^{\text{IND-CPA}}$ is non-negligible, we wish to construct a PPT $\mathscr{B}$ such that $\mathrm{Adv}_{\mathscr{B}}^{\text{DDH}}$ is non-negligible. On input (g, a, b, c), we will have $\mathscr{B}$ pass the input g, a into $\mathscr{A}_1$. Then, $\mathscr{A}_1$ will return two messages, m_0 and m_1. We then select a random bit b, and send the ciphertext (b, cm_b) to $\mathscr{A}_2$, and return 1 if and only if $\mathscr{A}_2$ guessed the correct bit.

Observe that if $\mathscr{B}$ is input a tuple of the form (g, g^x, g^y, g^{xy}), then $\mathscr{A}$ will receive a correct encryption of one of m_0 or m_1, and hence the probability that $\mathscr{A}_2$ outputs the correct bit is $1/2$ plus some non-negligible quantity. If $\mathscr{B}$ is input a tuple of the form (g, g^x, g^y, g^z), then the probability that $\mathscr{A}_2$ is correct is only $1/2$, as the ciphertext does not encrypt one of the selected values. Thus, the difference between the two probabilities is non-negligible, and we have constructed a DDH distinguisher. □

Elgamal's original cryptosystem is multiplicatively homomorphic, but not additively homomorphic, and it is the additive property that we are after when designing voting schemes. However, the system easily lends itself to an additively homomorphic version. To see how, consider the modification to Elgamal described in Figure 4.2. Note that this cryptosystem requires Alice to compute m from g^m. There are efficient ways to perform this computation, since in an election the range of m will be limited. For example, Shanks' baby-step giant-step algorithm [Sha69] can be performed in time $\mathrm{O}(\sqrt{\ell})$, where ℓ is the number of possible values taken by m.

Proposition 4.2.2. *The additively homomorphic Elgamal cryptosystem is additively homomorphic.*

Proof. Suppose we have two Elgamal ciphertexts, $\mathrm{Enc}_h(r, m)$ and $\mathrm{Enc}_h(r', m')$.

Key generation Alice chooses a random prime number p, and selects g as a generator of an order-q subgroup of $\mathbb{Z}_p^*$. She then selects $x \in_U \mathbb{Z}_q$ and sets $h \leftarrow g^x$. She publishes (g, p, q, h) as her public key, and keeps x as her private key.

Encryption Bob selects a message $m \in \mathbb{Z}_q$, and $r \in_U \mathbb{Z}_q$. He then computes $(G, H) \leftarrow (g^r, h^r g^m)$, and sends (G, H) to Alice.

Decryption Alice recovers g^m as H/G^x. She then uses some efficient algorithm to compute m from g^m.

Figure 4.2: The additively homomorphic Elgamal cryptosystem

Multiplying ciphertexts pairwise, we see that

$$\begin{aligned}\mathrm{Enc}_h(r,m) \cdot \mathrm{Enc}_h(r',m') &= (g^r, h^r g^m) \cdot (g^{r'}, h^{r'} g^{m'}) \\ &= (g^{r+r'}, h^{r+r'} g^{m+m'}) \\ &= \mathrm{Enc}_h(r+r', m+m'). \end{aligned}$$ □

4.3 The Paillier Cryptosystem

Damgård and Jurik devised a generalization of Paillier's scheme, based on the composite residuosity assumption, shown in Figure 4.3. We note that the Damgård-Jurik version of Paillier's scheme is used in this thesis only for the sake of generality. In practice, it is not necessary in the case of voting to use anything value for s other than 1, and so the cryptosystem reduces to that of Paillier.

Efficiency. To choose d, Alice will use the Chinese Remainder Theorem. Encryption involves two modular exponentiations, and decryption uses the algorithm given in Lemma 3.3.5, with one modular exponentiation.

We now show that the Paillier cryptosystem is correct. That is, given a ciphertext, it is possible to recover the plaintext given the secret key.

Proposition 4.3.1. *The Paillier cryptosystem is correct.*

Proof. Suppose c is a ciphertext. Then, Alice can recover the message m by

For each $s \in \mathbb{N}$, define a cryptosystem as follows.

Key generation Alice chooses a random RSA composite $n = pq$, and sets $g \leftarrow 1 + n$. Additionally, she chooses d such that $d \perp n$, and $d \equiv 0 \pmod{\lambda}$. She publishes (n, g) as her public key, and keeps d as her private key.

Encryption Bob selects a message $m \in \mathbb{Z}_{n^s}$, and $r \in_{\mathrm{U}} \mathbb{Z}_n^*$. He then computes $c = g^m r^{n^s}$ and sends c to Alice.

Decryption Alice receives a ciphertext $c \in \mathbb{Z}_{n^{s+1}}^*$. She recovers the message by computing $\mathrm{L}_s(c^d)^{-1}$.

Figure 4.3: The Paillier cryptosystem

computing

$$\begin{aligned} c^d &= (g^m r^{n^s})^d \\ &= ((1+n)^m r^{n^s})^d \\ &= (1+n)^{md} r^{dn^s} \\ &= (1+n)^{md}. \end{aligned}$$

By Lemma 3.3.5, she can easily compute md from $(1+n)^{md}$. Finally, she can recover m by computing $(md) \cdot d^{-1}$. □

Proposition 4.3.2. *The Paillier cryptosystem is additively homomorphic.*

Proof. This follows easily from Lemma 3.3.4. □

Proposition 4.3.3. *Under the DCRA, the Paillier cryptosystem satisfies IND-CPA security.*

Proof. We only present a proof for the case when $s = 1$, as shown by Paillier [Pai99]. The reader is referred to [DJ00] for the case when s is polynomially bounded by the security parameter. Suppose $\mathscr{A}$ is an IND-CPA adversary, and we would like to construct a nth residue distinguisher $\mathscr{B}$. Given a candidate nth residue c, we receive two messages m_0 and m_1 from $\mathscr{A}$. Then, we pick a random bit b and send $\mathscr{A}$ the ciphertext cg^{-m_b}. Observe that a ciphertext c is the encryption of m if and only if cg^{-m} is an nth residue modulo n^2. □

4.4 The Damgård-Jurik Cryptosystem

Key generation Alice chooses an RSA composite $n = pq$ such that $p = 2p' + 1$ and $q = 2q' + 1$, where p and q are safe primes. She then selects $g \in_U \mathrm{QR}_n$, and $\alpha \in_U \mathbb{Z}_N$, where $N = \lfloor n/4 \rfloor$, and sets $h = g^\alpha$. She sets her public key as (n, g, h), and her private key as α.

Encryption Bob selects a message $m \in \mathbb{Z}^+$, and $r \in_U \mathbb{Z}_N$. He then computes $c = (g^r, (h^r)^{n^s}(1+n)^m) \in \mathbb{Z}_n^* \times \mathbb{Z}_{n^{s+1}}$ and sends c to Alice.

Decryption Alice receives a ciphertext (G, H). She recovers the message by computing

$$m = \mathrm{L}_s(H(G^\alpha)^{-n^s}).$$

Figure 4.4: The Damgård-Jurik cryptosystem

Damgård and Jurik have proposed a cryptosystem which is a hybrid of the Paillier and Elgamal cryptosystems. It is shown in Figure 4.4.
Efficiency. Key generation uses one modular exponentiation, and also involves generating two safe primes. In contrast to Elgamal, these safe primes *are* secret, and must be generated again each time a new key-pair is made. Encryption and decryption are the same as in Paillier.

Proposition 4.4.1. *The Damgård-Jurik cryptosystem is correct.*

Proof. Observe that

$$\begin{aligned} m &= \mathrm{L}_s(H(G^\alpha)^{-n^s}) \\ &= \mathrm{L}_s((g^{\alpha r})^{n^s}(n+1)^m(g^{r\alpha})^{-n^s}) \\ &= \mathrm{L}_s((n+1)^m). \end{aligned}$$

Thus, m is easily computed. □

Proposition 4.4.2. *The Damgård-Jurik cryptosystem is additively homomorphic.*

Proof. Suppose we have two ciphertexts, $\mathrm{Enc}_h(r, m)$ and $\mathrm{Enc}_h(r', m')$. Multiplying ciphertexts pairwise, we see that

$$\begin{aligned}\mathrm{Enc}_h(r,m)\cdot\mathrm{Enc}_h(r',m') &= (g^r,(h^r)^{n^s}(n+1)^m)\cdot(g^{r'},(h^{r'})^{n^s}(n+1)^{m'})\\ &= (g^{r+r'},(h^{r+r'})^{n^s}(n+1)^{m+m'})\\ &= \mathrm{Enc}_h(r+r',m+m'). \qquad \square\end{aligned}$$

One point must be clarified immediately. Ideally, Alice would choose the private key α randomly from $\mathbb{Z}_{p'q'}$, observing that $\#\mathrm{QR}_n = p'q'$. However, knowledge of $p'q'$ implies knowledge of the factorization of n, and we would potentially like to have n be a system constant, not generated by Alice. To solve this problem, we select α from $\mathbb{Z}_N$, where N is sufficiently large enough so that g^α is chosen from a distribution statistically indistinguishable from $\langle g\rangle$. The following claim formalizes our goal.

Lemma 4.4.3. *Let D_1 be the uniform distribution over $\mathbb{Z}_{p'q'}$, and let D_2 be the uniform distribution over $\mathbb{Z}_N$, where $N = \lfloor n/4\rfloor$. Let X and Y be random variables distributed according to D_1 and D_2, respectively. Then, $X \approx Y$.*

We would now like to argue the security of the Damgård-Jurik cryptosystem. Before we do this, we prove that modifying the cryptosystem somewhat does not change the security.

Lemma 4.4.4. *The Damgård-Jurik cryptosystem with $r \in_{\mathrm{U}} \mathrm{QR}_n$ is ROR-CPA secure if and only if the cryptosystem where $r \in_{\mathrm{U}} \mathbb{Z}_n^*$ is ROR-CPA secure.*

Proof. We shall refer to the Damgård-Jurik cryptosystem with $r \in_{\mathrm{U}} \mathrm{QR}_n$ as $\mathfrak{Q}$. Suppose there exists an adversary $\mathscr{A}$ who can break the security of $\mathfrak{Q}$. We then construct an adversary $\mathscr{B}$ who can break the security of the Damgård-Jurik cryptosystem . This adversary will operate as follows. On input public key n, it will forward n to $\mathscr{A}$, receive a message m, and return m. It will then get input c, where c is an encryption of either m or a random element of $\mathbb{Z}_{n^s}$. It will compute $c' = (c(n+1)^{-m})^2(n+1)^m \bmod n^{s+1}$, and send c' to $\mathscr{A}$, returning the output b.

Suppose c is an encryption of the message m. Then,

$$\begin{aligned}c' &= \left(c(n+1)^{-m}\right)^2(n+1)^m\\ &= \left(r^{n^s}(n+1)^{m-m}\right)^2(n+1)^m\\ &= (r^2)^{n^s}(n+1)^m\\ &= \mathrm{Enc}(m,r^2).\end{aligned}$$

Now, suppose that c is an encryption of an element $\hat{m}$ chosen uniformly at random from $\mathbb{Z}_{n^s}$. Then,

$$\begin{aligned} c' &= \left(c(n+1)^{-m}\right)^2 (n+1)^m \\ &= \left(r^{n^s}(n+1)^{\hat{m}-m}\right)^2 (n+1)^m \\ &= r^{2^{n^s}}(n+1)^{2\hat{m}-m} \\ &= \mathrm{Enc}(2\hat{m}-m, r^2). \end{aligned}$$

In either case, c' is the encryption of a random element of $\mathbb{Z}_{n^s}$ with randomness chosen from QR_n. Thus, $\mathrm{Adv}_{\mathscr{A}} = \mathrm{Adv}_{\mathscr{B}}$. □

Based on this fact, we can demonstrate the security of the Damgård-Jurik cryptosystem.

Proposition 4.4.5. *The Damgård-Jurik hybrid cryptosystem satisfies IND-CPA security under the composite-DDH and DCRA assumptions.*

Proof. Consider the two pairs $P_1 = (g^k, (h^k)^{n^s}(n+1)^m)$ and $P_2 = (g^k, r^{n^s}(n+1)^m)$, where r is chosen uniformly at random from QR_n. Suppose there is an adversary $\mathscr{A}$ who has a non-negligible probability of distinguishing between these two pairs. We would like to build an adversary $\mathscr{B}$ who can break the composite-DDH assumption. The adversary $\mathscr{B}$ will operate as follows. On input (n, g, g^a, g^b, y), it will pass the public key (n, g, g^a) to $\mathscr{A}$, get a message m, send the ciphertext $(g^b, y^{n^s}(n+1)^m)$, and return the output b. There are thus two possibilities: either $y = g^{ab}$, or y is uniformly at random from QR_n. But this is exactly the difference between P_1 and P_2, and thus can be distinguished by $\mathscr{A}$. Hence, under the composite-DDH assumption, P_1 and P_2 are indistinguishable.

Let P_3 be the pair $(g^k, r^{n^s}(n+1)^{m'})$, where m' is a random element in $\mathbb{Z}_{n^s}$. We claim now that pairs P_2 and P_3 are indistinguishable. This is an immediate consequence of the fact that $r^{n^s}(n+1)^m$ is indistinguishable from $r^{n^s}(n+1)^{m'}$, by the DCRA and Lemma 4.4.4. Thus, P_2 is indistinguishable from P_3.

Finally, let P_4 be the pair $(g^k, (h^k)^{n^s}(n+1)^{m'})$. Pairs P_3 and P_4 are again indistinguishable, by the same argument as with pairs P_1 and P_2. Thus, P_1 is indistinguishable from P_4. Since distinguishing these two pairs is precisely the task faced by an ROR-CPA adversary, we can conclude that under the composite-DDH and DCRA assumptions, no such adversary can exist. By Proposition 3.4.5, the cryptosystem is thus IND-CPA-secure. □

4.5 Conclusion

In this chapter, we have seen three important cryptosystems. All of these cryptosystems share the important property of being additively homomorphic, that is, they admit the computation of the sum of the plaintexts given only the corresponding ciphertexts. Additionally, all three satisfy IND-CPA security.

Table 4.1 compares the previously discussed cryptosystems in terms of the assumptions upon which they depend. It should be noted that the Damgård-Jurik cryptosystem can be modified [DJ03] to depend solely on the DCRA.

Name	Assumptions
Elgamal	DDH
Paillier	DCRA
Damgård-Jurik	Composite-DDH,DCRA

Table 4.1: Comparison of Cryptosystems

Chapter 5

Proofs of Knowledge

This chapter introduces the notion of proofs of knowledge*, a technique whereby one can prove knowledge of some witness to a predicate without revealing the witness itself.*

5.1 Motivation

It may seem paradoxical that one can prove the knowledge of something without revealing any information. Quisquater, et al., give an amusing example [QGAB89] to demonstrate the intuition behind why this is possible. The core idea behind of their example is as follows. There are two parties, Alice and Bob. Alice knows the magic word to open a door inside a cave. The cave is shaped like a circle, with only one entrance, and a magic door in the middle. Alice would like to prove to Bob that she knows the magic word, but she does not want Bob to learn the magic word for himself. Thus, Bob cannot simply follow Alice into the cave and listen to her open the door.

To solve this problem, they employ the following scheme. Alice goes into the cave, and Bob waits outside. Alice enters the cave through a random path. Once she is inside, Bob enters the cave, and shouts out to Alice the path he wants her to return by. If he requested the same path by which she entered, she returns by that path. Otherwise, she uses the magic word to return by the other path. However, if she does not know the magic word, then there is only a 50% chance that she can return by the path Bob requested. If this experiment is repeated several times, the chances that Alice can fool Bob each time become tiny. Thus, Alice has demonstrated to Bob that she indeed knows the magic word. Furthermore, she has not revealed any information about what that word might be, as Bob did not immediately follow her into the cave.

5.2 Required Properties

Zero-knowledge proofs were first introduced by Goldwasser, Micali, and Rackoff [GMR85]. A zero-knowledge proof must satisfy three properties.

Completeness Assuming that the statement is true, an honest prover will always convince an honest verifier.

Soundness Assuming that the statement is false, a cheating prover cannot convince an honest verifier.

Zero-knowledge Assuming that the statement is true, an honest verifier learns nothing else, except for this fact. That is, no knowledge is transferred during the execution of the protocol, except for the proof itself.

When we say that a party is *honest* we mean that it is following the protocol as written. A *cheating* party may deviate from the protocol in order to gain an advantage over the other party.

5.3 Interactive Proof Systems

Before we present a formal definition of a zero-knowledge proof, we must introduce a model of computation that allows for interacting parties.

Definition 5.3.1. An *interactive proof system* $\Pi = (\mathscr{P}, \mathscr{V})$ is a protocol run between two parties, $\mathscr{P}$ and $\mathscr{V}$. Party $\mathscr{P}$ is called the *prover*, and party $\mathscr{V}$ the *verifier*. Both parties are modeled by PPT machines. The proof system, on input x, produces an output in $\{0, 1\}$ subject to the following restrictions.

1. Both parties receive input x.
2. The computation proceeds as a sequence of exchanged messages

   $$m_1, m_2, m_3, \ldots,$$

   where each message m_i satisfies $|m_i| \leq |x|^k$, for some $k \in \mathbb{N}$.
3. Each of the two parties is unaware of the internal state of the other, including the randomness used at each stage of the conversation.

The proof system Π *decides* (that is, it is "complete") a language L if, for all strings x,

$$\Pr[\Pi(x) = 1 \mid x \in L] \geq 1 - \frac{1}{2^{|x|}},$$

and for all probabilistic Turing machines $\mathscr{P}^*$, which may run in exponential time,

$$\Pr[(\mathscr{P}^*, \mathscr{V})(x) = 1 \mid x \notin L] \leq 1 - \frac{1}{2^{|x|}},$$

(that is, it is "sound").

Observe that the first two properties of zero-knowledge proof systems also apply to any general interactive proof system. The third property is the distinguishing property for zero-knowledge proofs.

5.4 Formal Definition

The general form of zero-knowledge proof that we will use is the *three-move protocol*. The structure of a three-move protocol is shown in Figure 5.4. We will use such diagrams to show the sequence of events in an interactive protocol. Execution proceeds downwards sequentially, with each party sending data via arrows to the other.

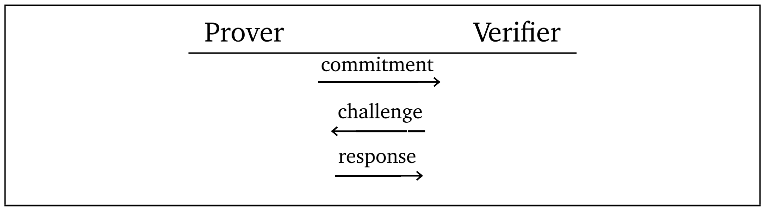

Figure 5.1: General form of three-move zero-knowledge protocols

Lemma 5.4.1 (Alternative definition of NP). *Let $\Sigma = \{0, 1\}^*$. For any language L over alphabet Σ, it holds that $L \in \mathrm{NP}$ if and only if*

$$L = \left\{x \in \Sigma^* \mid \textit{there exists a } w \in \Sigma^* \textit{ such that } |w| \leq p(|x|) \textit{ and } \langle x, w\rangle \in L\right\},$$

for some polynomial p.

Remark 5.4.2. In the above lemma, the notation $\langle x, w\rangle$ refers to a string representation of the pair (x, w).

By Lemma 5.4.1, we get an alternative formulation of the class NP. Let L be a language in NP. Then, for each $x \in L$, there exists a set W_x such that W_x is the set of "witnesses" for x. Thus, there is a polynomial-time computable relation $R_L = \{(x, w) \mid x \in L, w \in W_x\}$. We term R_L an NP-*relation* for the language L.

Definition 5.4.3. A *three-move proof of knowledge* for an NP-relation R_L is an interactive proof system $(\mathscr{P}, \mathscr{V})$ satisfying the following properties. Both parties receive a common input $x \in L$, and the prover $\mathscr{P}$ wishes to prove to the verifier $\mathscr{V}$ that it knows a w such that $(x, w) \in R_L$. A proof of knowledge must have the following basic properties.

1. *Completeness*: For all $(x, w) \in R_L$,

$$\Pr[(\mathscr{P}(w), \mathscr{V})(x) = 1] = 1 - \varepsilon(|x|),$$

   where ε is a negligible function.

2. *Special soundness*: For all provers $\mathscr{P}^*$, there exists a PPT *extractor* $\mathscr{X}$ such that given two accepting conversations (y, c, s) and (y, c', s') with the same first move and different second moves, is able to extract the witness w.

3. *Honest-verifier zero-knowledge*: There exists a simulator $\mathscr{S}$ such that

$$\{\text{view}(\mathscr{P})\}_{x \in L} \approx \{\mathscr{S}(x)\}_{x \in L},$$

   where $\{\text{view}(\mathscr{P})\}_{x \in L}$ represents a sequence consisting of all data viewable by $\mathscr{P}$ during the execution of the protocol for all inputs x.

Intuitively, soundness ensures that the provers actually knows a witness, completeness ensures that the verifier must accept a valid proof, and zero-knowledge ensures that the verifier cannot recover the witness.

It turns out that the class of languages that can be proved with three-move protocols is quite large.

Theorem 5.4.4. *If* $L \in \text{NP}$*, then there exists a three-move zero-knowledge protocol for* L*.*

5.5 Basic Protocols

We provide several examples of proofs of knowledge, which will serve as building blocks for more complex protocols.

5.5.1 Knowledge of Discrete Logs

Suppose g is a generator of an order-q subgroup of $\mathbb{Z}_p^*$, for some prime p. The verifier is given input (p, q, g, h), and the prover wishes to prove that it knows $w \in \mathbb{Z}_q$ such that $h = g^w$. Schnorr [Sch90] presents the following protocol to accomplish this task.

$$\begin{array}{ccc}
\text{Prover} & & \text{Verifier} \\
\hline
r \in_{\mathrm{U}} \mathbb{Z}_q & & \\
y \leftarrow g^r & \xrightarrow{y} & \\
 & \xleftarrow{c} & c \in_{\mathrm{U}} \mathbb{Z}_q \\
s \leftarrow r + wc & \xrightarrow{s} & g^s \overset{?}{=} yh^c
\end{array}$$

Figure 5.2: The Schnorr proof of knowledge of discrete logs.

Proposition 5.5.1. *The Schnorr protocol satisfies completeness, special soundness, and honest-verifier zero-knowledge.*

Proof. To show completeness, observe that $g^s = g^{r+wc} = yh^c$. Thus, if both the prover and verifier are honest, the verifier will be convinced with probability 1. To show special soundness, we construct an extractor that operates on two accepting conversations (y, c, s) and (y, c', s') from the prover that have the same first move but different second moves. Given these two conversations, observe that $g^s = yh^c$ and $g^{s'} = yh^{c'}$, so it follows that $x = (s - s')/(c - c')$, which is the desired witness. Finally, to show honest-verifier zero-knowledge, we construct a simulator that on input (g, p, q, h), chooses $c, s \in_{\mathrm{U}} \mathbb{Z}_q$, and outputs $(g^s h^{-c}, c, s)$. □

5.5.2 Equality of Discrete Logs

Suppose again that g is a generator of an order-q subgroup of $\mathbb{Z}_p^*$, and wishes to prove that it knows $\alpha \in \mathbb{Z}_q$ such that $x = g^\alpha$ and $y = h^\alpha$. Chaum and Pedersen [CP93] give a protocol to prove the equality of two discrete logarithms.

Proposition 5.5.2. *The Chaum-Pedersen protocol satisfies completeness, special soundness, and honest-verifier zero-knowledge.*

$$\begin{array}{ccc}
\text{Prover} & & \text{Verifier} \\
\hline
w \in_{\mathrm{U}} \mathbb{Z}_q & & \\
(a,b) \leftarrow (g^w, h^w) & \xrightarrow{a,b} & \\
& \xleftarrow{c} & c \in_{\mathrm{U}} \mathbb{Z}_q \\
r \leftarrow w + \alpha c & \xrightarrow{r} & g^r \overset{?}{=} a x^c \\
& & h^r \overset{?}{=} b y^c
\end{array}$$

Figure 5.3: The Chaum-Pedersen proof of equality of discrete logs

The proof is quite similar to that of Proposition 5.2, and it thus omitted.

5.6 Composition of Proofs

Feige and Shamir [FS90] introduced the idea of *witness indistinguishability*.

Definition 5.6.1. A proof of knowledge is *witness indistinguishable* if conversations generated with the same x but with different witnesses w have indistinguishable distributions.

Cramer, et al., [CDS94] provide a method for deriving the OR-composition of proofs of knowledge.

Theorem 5.6.2. *Let Π be a three-round, public coin, honest-verifier proof of knowledge for an NP-relation R_L, satisfying special soundness. Then, for any integers n, d with $n \geq d$, there exists a three-round, public coin, honest-verifier proof of knowledge in which the prover shows that it knows d out of n witnesses for elements in R_L, without revealing which d witnesses it knows.*

Proof. Suppose $\Pi = (\mathscr{P}, \mathscr{V})$. Recall that since Π has the honest-verifier zero-knowledge property, there exists a simulator $\mathscr{S}$ that presents a conversation indistinguishable from the view of $\mathscr{V}$ during a real execution of the protocol. We now proceed to construct a new protocol $\Pi' = (\mathscr{P}', \mathscr{V}')$ as follows.

1. For each i for which the prover does *not* know a witness w_i for statement x_i, the prover runs the simulator $\mathscr{S}$ to produce a conversation (y_i, c_i, s_i). For each i for which the prover *does* know a witness, it runs $\mathscr{P}$ to receive a commitment y_i. The prover then sends y_i, for $i = 1, \ldots, n$, to the verifier.

2. The verifier chooses a t-bit string s at random and sends it to the prover.

3. The prover chooses “shares” $c_1, \ldots, c_n$ such that $c_1 \oplus \cdots \oplus c_n = s$. For each i, if the prover knows a witness w_i, it runs $\mathscr{P}$ on c_i to get s_i. Otherwise, it sets s_i to be the response of the ith conversation generated by the simulator. It then sends $s_1, \ldots, s_n$ and $c_1, \ldots, c_n$ to the verifier.

4. The verifier checks that all conversations (y_i, c_i, s_i) are accepted by $\mathscr{V}$, that $s = c_1 \oplus \cdots \oplus c_n$, and accepts if and only if the checks are satisfied.

It remains to show that the above protocol satisfies completeness, special soundness, and witness indistinguishability. Completeness trivially holds, since the simulator $\mathscr{S}$ must produce accepting conversations. To show soundness, suppose we have two accepting conversations $(\{y_i\}_1^n, s, \{s_i\}_1^n)$ and $(\{y_i\}_1^n, s', \{s_i'\}_1^n)$ with the same first move and different second moves. Suppose further that the challenges s and s' give rise to shares $c_1, \ldots, c_n$ and $c_1', \ldots, c_n'$, respectively. Then, for each set of indices such that the prover knows witness for elements in the set, there must exist an i such that $c_i \neq c_i'$. Since Π satisfies special soundness, we can then extract a witness w_i from $\mathscr{P}$. Finally, to show witness indistinguishability, we must show that the distribution of the conversation is independent of which witnesses the prover knows. Since Π is honest-verifier zero-knowledge, we have that the distribution of each y_i is dependent only on x_i, and is independent of the known witnesses. Next, the construction of the challenge shares ensures randomness. Finally, the honest-verifier zero-knowledge property again implies that each s_i is dependent only on x_i, y_i, and c_i, and so is independent of the known witnesses. □

Figure 5.4 illustrates the construction given in the previous proof, in the case where the prover wishes to prove the disjunction of two predicates.

Prover		Verifier
select ρ, d', s' at random		
$a \leftarrow P_1(\rho, x)$		
$a' \leftarrow \mathscr{S}(d', s')$	$\xrightarrow{a,a'}$	
	$\xleftarrow{c}$	$c \leftarrow V_1(1^\lambda)$
compute d such that $c = d \oplus d'$		
$s = P_2(\rho, x, a, d)$	$\xrightarrow{s,s'}$	$V_2(a, d, s) \overset{?}{=} 1$
		$V_2'(a', d', s') \overset{?}{=} 1$
		$d \oplus d' \overset{?}{=} c$

Figure 5.4: Proof of the disjunction of two predicates

5.7 Non-interactive Proofs

Using the *Fiat-Shamir heuristic* [FS86], we can produce non-interactive versions of any three-round zero-knowledge protocol. Before we introduce the procedure, we first define a basic primitive that is used in its construction.

Definition 5.7.1. A *cryptographic hash function* (we shall refer to these simply as "hash functions") is a function $H\colon \{0,1\}^* \longrightarrow \mathbb{Z}_q$, for some integer q, with the following properties.

1. It is computationally difficult to find strings x and y such that $x \neq y$ but $H(x) = H(y)$. That is, H is *collision-resistant*.

2. Given $H(x)$, for some string x, it is difficult to find x. That is, H is *hard to invert*.

3. The function H should be computable in polynomial time with respect to $|x|$. That is, H is *easy to compute*.

Remark 5.7.2. A word on notation: if H is a hash function, and x is a group element, we denote by $H(x)$ the application of H to some unique binary representation of x. If $x_1, \ldots, x_n$ are group elements, we denote by $H(x_1, \ldots, x_n)$ the application of H to the concatenation of the binary representations of $x_1, \ldots, x_n$.

Several candidate hash functions have been proposed in the literature and used in practice. As of this writing, the two most popular hash functions used in practice are SHA-1 and MD5. However, both functions have recently been shown to contain security flaws. Whether or not there exist functions that truly satisfy the required properties is currently a matter of speculation. Nonetheless, we take as an operating assumption that there does exist such a function.

Given a hash function H, we can construct a non-interactive version of any three-move zero-knowledge protocol. Suppose the prover wishes to prove the validity of statement x. It first generates a commitment y, followed by a challenge $c = H(y,x)$. It then computes a response s based on y and c. The triple (y,c,s) serves as a static proof of the validity of x. The intuition here is that the hash function ensures that the challenge is computed after the commitment has already been chosen. The security of this method is based on the fact that it is difficult to find collisions for h, so an attacker who wishes to generate its own challenge with a different commitment would need to find $y' \neq y$ such that $H(y,x) = H(y',x)$.

Example 5.7.3. We can construct a non-interactive version of Schnorr's proof as follows. The prover would set its commitment to be $y = g^t$, the challenge as $c = H(h, g, y)$, and the response as $s = t + cw$. It would then send (y, c, s) as its proof. The verifier then checks that $c = H(g, h, g^s h^{-c})$. $\Diamond$

5.8 The Random Oracle Model

When we construct a cryptographic scheme using a hash function, the proof of security often requires that we use a special model. This model, called the *random oracle model*, assumes that the cryptographic hash function behaves as a *random oracle*. That is, a deterministic function whose output is uniformly distributed in its range.

In a proof based on the random oracle model, it is assumed that the environment is controlled by a simulator, and that the simulator also has control over every random oracle called by each player. Suppose, for instance, that the players in the protocol all use a hash function H. When players query H, the simulator is then responsible for replying with a suitable value for H applied to the input.

Observe that it is possible to simulate a deterministic random oracle in probabilistic polynomial time. The simulator will behave as follows. Upon receiving a query for a string (say, s_1), the simulator will sample a random element from the range of H (say, v_1), and return v_1 to the caller. It will then store the pair (s_1, v_1) in a table. Upon receiving a second query for a different string s_2, it will again sample a random value v_2, return v_2, and store (s_2, v_2) in the table, sorted by string. As each new string is queried, the simulator will search the table for the string. If the string is found, it will return the value that it already sampled; otherwise, it will sample a new value. Since the table is sorted, the look-up time is $\mathrm{O}(\log n)$, where n is the size of the table. Thus, the random oracle is perfectly simulated in polynomial time.

It is unknown whether currently used cryptographic hash functions have this property. Thus, it cannot be said that the random oracle model provides an accurate representation of the real world. However, several existing hash functions appear to provide near-uniform behavior, and so the random oracle model remains a good heuristic to argue security.

5.9 Proofs About Encryptions

In this section, we provide several examples of proofs of knowledge for various properties relating to the cryptosystems we have previously discussed.

5.9.1 Elgamal Proofs

As our first example of technique in §5.6, we present a protocol that will allow a prover to prove that a given homomorphic Elgamal ciphertext $(g^r, h^r g^m)$ encrypts either 0 or 1. We have already done most of the work to develop this protocol. To prove that an Elgamal ciphertext (G, H) encrypts a value m, we must show that $(G, H) = (g^r, h^r g^m)$, for some r. That is, we must show that $\log_g G = \log_h(H/g^m)$. This is exactly the Chaum-Pedersen protocol discussed earlier. Thus, we construct the Elgamal proof of encryption of 0 or 1 as the composition of two executions of the Chaum-Pedersen protocol, combined with the technique of §5.6. The combined protocol is shown in Figure 5.5.

$$\begin{array}{ccc}
\text{Prover} & & \text{Verifier} \\ \hline
t_0, s_1, c_1 \in_{\text{U}} [0, q] & & \\
y_0 = g^{t_0}, z_0 = h^{t_0} & \xrightarrow{y_0, z_0, y_1, z_1} & \\
y_1 = g^{s_1} G^{-c_1} & & \\
z_1 = h^{s_1} (H/f)^{-c_1} & & \\
 & \xleftarrow{c} & c \in_{\text{U}} [0, q) \\
c_0 = c - c_1 & \xrightarrow{s_0, s_1, c_0, c_1} & g^{s_0} \overset{?}{=} y_0 G^{c_0} \\
s_0 = t_0 + c_0 r & & h^{s_0} \overset{?}{=} z_0 H^{c_0} \\
 & & g^{s_1} \overset{?}{=} y_1 G^{c_1} \\
 & & h^{s_1} \overset{?}{=} z_1 (H/g)^{c_1} \\
 & & c \overset{?}{=} c_0 + c_1
\end{array}$$

Figure 5.5: Proof that an Elgamal ciphertext encrypts 0 or 1

5.9.2 Paillier Proofs

Damgård and Jurik construct some zero-knowledge proofs for the Paillier cryptosystem. As any element in $\mathbb{Z}^*_{n^{s+1}}$ is a valid Paillier ciphertext, there is no need for a proof of well-formedness. Thus, we present proofs for the encryption of 0 and the combined encryption of either 0 or 1. Figure 5.6 shows the proof that a Paillier ciphertext encrypts the value 0.

Proposition 5.9.1. *The protocol in Figure 5.6 satisfies completeness, special soundness, and honest-verifier zero-knowledge.*

Proof. To show completeness, observe that since $u \in \mathbb{Z}^*_{n^{s+1}}$, it follows that $u \perp n^{s+1}$, and so $u \perp n$. Similarly, as $v, z \in \mathbb{Z}^*_n$, we have that $v \perp n$ and $z \perp n$.

$$\begin{array}{ccc}
\text{Prover} & & \text{Verifier} \\
\hline
r \in_{\mathrm{U}} \mathbb{Z}_n^* & & \\
a \leftarrow \mathrm{Enc}(0, r) & \xrightarrow{a} & \\
 & \xleftarrow{c} & c \in_{\mathrm{U}} [2^{\mathrm{len}(n)/2}] \\
z \leftarrow r v^c & \xrightarrow{z} & u \perp n \\
 & & a \perp n \\
 & & z \perp n \\
 & & \mathrm{Enc}(0, z) \overset{?}{=} a u^c
\end{array}$$

Figure 5.6: Proof that a Paillier ciphertext encrypts 0

To show soundness, suppose we have two accepting conversations, (a, c, z) and (a, c', z'). Then, $\mathrm{Enc}(0, z) = z^{n^s} = (r v^c)^{n^s} = a u^c$ and $\mathrm{Enc}(0, z') = a u^{c'}$. Additionally,

$$\begin{aligned}
\mathrm{Enc}(0, z/z') &= (z/z')^{n^s} \\
&= \left(\frac{r v^c}{r v^{c'}}\right)^{n^s} \\
&= (v^{n^s})^{c-c'} \\
&= u^{c-c'}.
\end{aligned}$$

Now, since $2^{\mathrm{len}(n)/2} \perp n$, it follows that $(c-') \perp n^s$. So, using the extended Euclidean algorithm, we can find α, β such that $\alpha n^s + \beta(c - c') = 1$. Next, let $\bar{u} = u \bmod n$ and $v = \bar{u}^\alpha (z/z')^\beta$. Notice that $u^{n^s} = \mathrm{Enc}(0, \bar{u})$. Thus,

$$\begin{aligned}
\mathrm{Enc}(0, v) &= \left(\bar{u}^\alpha (z/z')^\beta\right)^{n^s} \\
&= (\bar{u}^{n^s})^\alpha (z/z')^{\beta n^s} \\
&= \mathrm{Enc}(0, \bar{u})^\alpha \, \mathrm{Enc}(0, z/z')^\beta \\
&= (u^{n^s})^\alpha u^{\beta(c-c')} \\
&= u^{\alpha n^s + \beta(c-c')} \\
&= u.
\end{aligned}$$

That is, v was the correct randomness for the ciphertext u, completing the extraction.

Finally, we show honest-verifier zero-knowledge by performing a simulation as follows. First, we select $z \in_{\mathrm{U}} \mathbb{Z}_n^*$. Then, we select $c \in_{\mathrm{U}} [2^{\mathrm{len}(n)/2}]$. Finally, we compute $a \leftarrow \mathrm{Enc}(0, z) u^{-c}$ and output the conversation (a, c, z). □

Using the technique of §5.6, we can construct a protocol that allows the prover to prove that a ciphertext encrypts either 0 or 1. In this construction we make use of the simulator $\mathscr{S}$ described in the proof of Proposition 5.9.1.

$$\begin{array}{ccc}
\text{Prover} & & \text{Verifier} \\
\hline
r_1 \in_{\text{U}} \mathbb{Z}_n^* & & \\
(a_2, e_2, z_2) \leftarrow \mathscr{S}(n, u_2) & & \\
a_1 \leftarrow \text{Enc}(0, r_1) & \xrightarrow{a_1, a_2} & s \in_{\text{U}} [2^{\text{len}(n)/2}] \\
 & \xleftarrow{s} & \\
e_1 \leftarrow s - e_2 \bmod 2^{\text{len}(n)/2} & & \\
z_1 \leftarrow r_1 v_1^{e_1} & \xrightarrow{e_1, z_1, e_2, z_2} & s \overset{?}{=} e_1 + e_2 \bmod 2^{\text{len}(n)/2} \\
 & & \text{Enc}(0, z_1) \overset{?}{=} a_1 u_1^{e_1} \\
 & & \text{Enc}(0, z_2) \overset{?}{=} a_2 u_2^{e_2} \\
 & & u_1 \perp n,\ u_2 \perp n \\
 & & a_1 \perp n,\ a_2 \perp n \\
 & & z_1 \perp n,\ z_2 \perp n
\end{array}$$

Figure 5.7: Proof that a Paillier ciphertext encrypts 0 or 1

Proposition 5.9.2. *The protocol in Figure 5.7 satisfies completeness, special soundness, and honest-verifier zero-knowledge.*

Proof. The result follows immediately from Proposition 5.9.1 and Theorem 5.6.2. □

5.9.3 Damgård-Jurik Proofs

As above, we can construct proofs about Damgård-Jurik ciphertexts as well. However, we need to make a modification to the cryptosystem before this will work. Damgård and Jurik present a modification of their cryptosystem that allows values to be squared before making proofs, to ensure that elements are in QR_n. The modified cryptosystem is shown in Figure 5.8.

We can now use the standard methods to create a proof that a Damgård-Jurik ciphertext encrypts one of a list of values. The reader is referred to [Jur03] for proofs of the correctness and security of these protocols.

Key generation This is identical to the original cryptosystem.

Encryption Bob selects a message $m \in \mathbb{Z}^+$, and $r \in_U \mathbb{Z}_N$, and $b_0, b_1 \in_U \{0,1\}$. He then computes

$$\mathrm{Enc}_s^{\pm}(m, r, b_0, b_1) = ((-1)^{b_0} g^r, (-1)^{b_1} (h^r)^{n^s} (n+1)^m)$$

and sends c to Alice.

Decryption Alice receives a ciphertext (G,H). She only decrypts (G,H) if both $(G \mid n) = 1$ and $(H \mid n) = 1$. She computes $d = G^\alpha$ as in the original cryptosystem, and computes

$$\begin{aligned} H' &= H^2 d^{-2n^s} \\ &= (n+1)^{2m}. \end{aligned}$$

She recovers the message by computing

$$m = \mathrm{L}_s(H')/2.$$

Figure 5.8: The modified Damgård-Jurik cryptosystem

5.10 Conclusion

This chapter has seen the introduction of a crucial cryptographic primitive: the proof of knowledge. This primitive is essential in building cryptographic systems in which there are multiple parties mutually distrustful of the information they send and receive. It allows parties to keep secrets, while at the same time ensuring to others that they actually possess these secrets. We have seen how various properties of encryption schemes can be proved using zero-knowledge proofs.

$$\begin{array}{ccc}
\text{Prover} & & \text{Verifier} \\
\hline
r' \in_{\mathrm{U}} \{0,\ldots,2^{|N|+2k_2}\} & & \\
c' \leftarrow (G',H') = \mathrm{Enc}_s^{\pm}(m',r',0,0) & \xrightarrow{c'} & \\
 & \xleftarrow{e} & e \in_{\mathrm{U}} \{0,\ldots,2^{k_2}-1\} \\
\hat{r} \leftarrow r' + er & & \\
\hat{m} \leftarrow m' + em \bmod n^s & \xrightarrow{\hat{r},\hat{m}} & \\
 & & G \perp n,\, G' \perp n \\
 & & H \perp n,\, H' \perp n \\
 & & (G \mid n) \overset{?}{=} 1,\, (G' \mid n) \overset{?}{=} 1 \\
 & & (H \mid n) \overset{?}{=} 1,\, (H' \mid n) \overset{?}{=} 1 \\
 & & \mathrm{Enc}_s^{\pm}(2\hat{m},2\hat{r},0,0) \overset{?}{=} (c')^2 c^{2e}
\end{array}$$

Figure 5.9: Proof that a Damgård-Jurik ciphertext is well-formed. Here, we assume 2^{k_2} is less than the smallest prime factor of n.

$$\begin{array}{ccc}
\text{Prover} & & \text{Verifier} \\
\hline
r' \in_{\mathrm{U}} \{0,\ldots,2^{|N|+2k_2}\} & & \\
c' \leftarrow (G',H') = \mathrm{Enc}_s^{\pm}(m',r',0,0) & \xrightarrow{c'} & \\
 & \xleftarrow{e} & e \in_{\mathrm{U}} \{0,\ldots,2^{k_2}-1\} \\
\hat{r} \leftarrow r' + er & & \\
\hat{m} \leftarrow m' + em \bmod n^s & \xrightarrow{\hat{r},\hat{m}} & \\
 & & G \perp n,\, G' \perp n \\
 & & H \perp n,\, H' \perp n \\
 & & (G \mid n) \overset{?}{=} 1,\, (G' \mid n) \overset{?}{=} 1 \\
 & & (H \mid n) \overset{?}{=} 1,\, (H' \mid n) \overset{?}{=} 1 \\
 & & \mathrm{Enc}_s^{\pm}(2em,2\hat{r},0,0) \overset{?}{=} (c')^2 c^{2e}
\end{array}$$

Figure 5.10: Proof that a Damgård-Jurik ciphertext encrypts m

Chapter 6

Distribution of Trust

A key decision to be made in the construction of any secure system is where to place trust.

6.1 Motivation

Consider the classical example of a government preparing for nuclear war. A missile is aimed at the enemy country, and it can only be launched by the insertion of a special key. However, it would be devastating for the key to fall into the wrong hands, so no single person is trusted with it. Instead, several officials are each given separate keys, and the missile launch depends on the successful insertion of all of them. Thus, no single person can take it into his or her own hands to launch the missile. There remains a problem, however. Suppose an enemy spy kidnaps one of the officials. Then, the ability to launch the missile has effectively been destroyed, as one of the keys is now missing. To avoid this calamity, we require only that a certain *threshold* of keys be present. For example, if there are a total of ten keys, we might estimate our enemy's capacity to kidnap our officials to be limited to two individuals, and only require the presence of eight keys.

The placement of trust is a recurring concept in computer security. As an example, consider the goal of sending authenticated email. There are two main standards for this task: OpenPGP [CDFT98] and S/MIME [DHR+98]. Both of these standards allow the authentication of email using public-key cryptography, but each relies on a different method of trust. OpenPGP is based on a *web of trust*, where each participant has a public keypair. If Alice has a copy of Bob's public key, and she has independent verification that this is actually Bob's key, she can use her secret key to sign it. Thus, if Dave obtains a copy of Bob's public key (as signed by Alice), he can trust that the key is authentic, provided that he trusts Alice. In this way a web is constructed, where every participant has a list of other participants that he or she trusts, and these trusts are public information.

This is in contrast to the trust model used by S/MIME. In S/MIME, each participant is issued a *certificate* by a *certification authority* (CA). The certificate is signed by the CA, and thus anyone who has a copy of the CA's public key can verify signatures produced by it. Thus, if Alice knows the CA's public key, and Bob's certificate is signed by the CA, Alice can verify that Bob's email is valid. Notice that this scheme assumes that Alice trusts the CA. If the CA were malicious, an impostor could claim to be Bob, get a signed certificate from the CA, and impersonate Bob. In practice, CAs are normally large companies who charge for their services, and require identification before issuing certificates.

6.2 Shamir's Scheme

Often, it is desirable to share a secret among several parties, such that no one party can recover the secret alone. Schemes that accomplish this task are called *secret-sharing schemes*. Shamir [Sha79] gives a polynomial interpolation-based scheme for secret-sharing.

Definition 6.2.1. A (k,n)-*threshold scheme* is a scheme in which a piece of data D is "divided" into n shares $D_1,\dots,D_n$ such that

1. Knowledge of at least k shares makes D easily computable.
2. Knowledge of fewer than k shares leaves D impossible to determine.

Suppose we wish to share a secret $D \in \mathbb{Z}$ among n trustees. We begin by choosing a prime number p such that $p > D$ and $p > n$. Recall, by Theorem 2.4.18, that we can perform polynomial interpolation in the field $\mathbb{F}_p$. That is, if we have k points $(x_1,y_1),\dots,(x_k,y_k)$, there is exactly one polynomial $q(\mathrm{X})$ of degree $k-1$ such that $q(x_i)=y_i$, for all $i=1,\dots,k$.

With this in mind, the dealer chooses a random polynomial $q(\mathrm{X}) \in \mathbb{F}_p[\mathrm{X}]$ of degree $k-1$, setting $q(0)=D$. That is, it chooses coefficients $a_1,\dots,a_{k-1} \in_{\mathrm{U}} \mathbb{F}_p$, and sets $a_0 = D$. The n shares are then computed and distributed as $D_i = q(i)$, for each $i=1,\dots,n$. When the trustees wish to recover the secret, they pool k of their shares together, use interpolation to recover the complete $k-1$ degree polynomial, and evaluate it at 0.

Proposition 6.2.2. *In a (k,n)-threshold scheme, an adversary corrupting fewer than k trustees cannot recover the secret.*

Proof. Since the polynomial is of degree $k-1$, an adversary is required to obtain k points on the polynomial to apply interpolation. If the adversary possesses fewer than k shares, the polynomial remains completely undetermined. Furthermore, $q(0)$ is independent from the other shares, so without using interpolation, the secret cannot be recovered. □

Based on the previous discussion of Shamir's scheme, we can isolate several concerns.

1. There is trust in a single dealer. In applications such as elections, where all parties may be mutually distrustful of each other, it may not be possible to find a party whom everyone trusts. The private information is thus required to be generated *jointly* by all players, in such a way that no one party is aware of the secrets of any other party.

2. There is no way to catch misbehaving players. A player who submits invalid shares in the final stage can completely ruin the procedure and prevent the interpolation from being carried out correctly. There should be some way to verify that each submitted share is actually correct.

3. It may be desirable to perform computations with the secret without actually revealing it to any one party. In the above setting, the secret information is eventually revealed to one party, who then presumably uses it for some computational task. In many settings, such as elections, it is definitely not acceptable for *any* party to have access to the secret (a decryption key that would reveal each voter's ballot). In this case, we would like to perform the necessary computations "in the dark," and then reveal only the outcome.

In the schemes that follow, we will address these concerns to varying degrees. Concern 1 can be solved by making various efficiency trade-offs (namely in communication complexity). Concern 2 can be solved using *verifiable secret sharing* and zero-knowledge proofs. Concern 3 is a definite requirement, and will be addressed by all of our schemes.

6.3 The Communication Model

Rather than allow parties to communicate over private channels, we employ a *bulletin board* model. This model, introduced by Benaloh [Ben87], ensures that all communication is performed in the open.

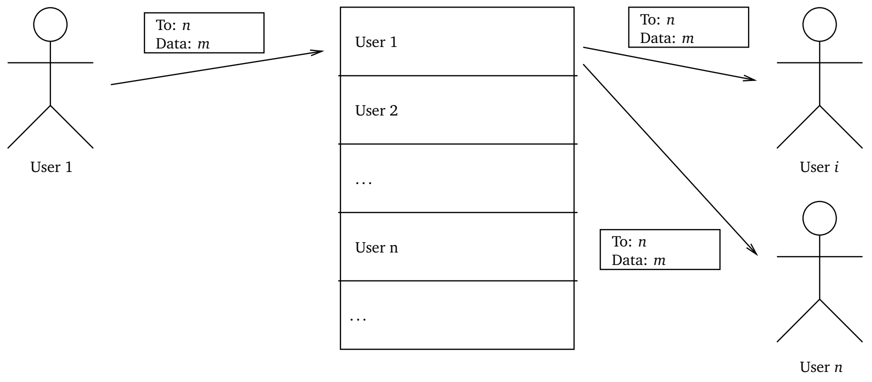

Figure 6.1: The bulletin board model of communication. User 1 wishes to send a message m to user n. Note that user i can also read the message.

Definition 6.3.1. A *bulletin board* is a public memory shared by multiple *users*. The bulletin board is divided into *areas*, one for each user. Every user has read access to every area on the bulletin board, and append access to its own. We define the *transcript* of the bulletin board to be the entire contents of the bulletin board.

Observe that no user has delete access to any area of the bulletin board.

We make one final assumption about the communication model. This is that every user who posts to the bulletin board is *authenticated*. That is, we never have to be concerned that a message is posted in a user's area that did not come from that user.

6.4 Verifiable Secret Sharing

To address the issue of cheating parties, we use secret-sharing methods known as *verifiable secret sharing* (VSS).

6.4.1 The Model

We suppose the bulletin board model of communication, with n users. Additionally, we assume that there is one adversary, who is not one of the n users. We assume that the adversary can corrupt up to t out of the n parties, for $t < n/2$, at the start of the communication. The adversary is *static*, that is, it cannot change which parties it has corrupted at any point during the execution.

Furthermore, we assume that the corrupted parties are completely under the adversary's control (that need not follow the written protocol).

We provide the following definition of security for verifiable secret sharing. We claim that a VSS protocol is *secure* if, given a static adversary who has corrupted t players at the beginning of the protocol, it is computationally infeasible for the adversary to determine the shared secret, given only the shares of the t compromised players and the public transcript of the bulletin board. Specifically, there exists a simulator that, given a shared secret, produces a bulletin board transcript (for all areas other than the t compromised players) that is indistinguishable from a bulletin board transcript that is the result of a proper execution of the protocol.

There is a weakness in this model, however. We are only assuming the existence of a static adversary. That is, the adversary is only permitted to corrupt players at the beginning of the game. We have ignored the possibility of an *active adversary*, who may change the corrupted players as the game progresses. The static model may therefore be unrealistic. Nonetheless, for simplicity, this is the model we assume.

6.4.2 Feldman's VSS

A simple VSS scheme is that of Feldman [Fel87], shown in Figure 6.3. As this scheme can detect cheating parties, it will become our basic primitive for building more complex secret sharing protocols.

Proposition 6.4.1. *Let $\mathscr{A}$ be an adversary who has compromised t or fewer players in Feldman's VSS. Then, $\mathscr{A}$ cannot learn any information about σ beyond what can be derived from g^{σ}.*

Proof. We prove this claim by a simulation argument. That is, we construct a simulator $\mathscr{S}$ that presents to $\mathscr{A}$ a view of the protocol that is indistinguishable from a real-world execution of the protocol. Specifically, given a secret σ, and the compromised shares $s_1, \ldots, s_t$ of t parties, $\mathscr{S}$ must generate verification values $A_1, \ldots, A_t$ such that

$$g_i^s = \prod_{k=0}^{t} (A_k)^{i^k}.$$

As each A_k should equal g^{a_k}, for $k = 0, \ldots, t$, we need to find values for each a_k. Let $\boldsymbol{f} = (f(0), \ldots, f(1))$, and let $\boldsymbol{a} = (a_0, \ldots, a_t)$. Let A be the $(t+1) \times (t+1)$ Vandermonde matrix, that is, $\mathsf{A}_{ik} = i^k$. It follows from linear algebra that this matrix is invertible. Let $\mathsf{A}^{-1} = (\lambda_{ki})$ be the inverse of A. Then, since

1. The dealer generates a random polynomial $f(\mathrm{X}) = \sigma + a_1\mathrm{X} + \cdots + a_t\mathrm{X}^t \in \mathbb{Z}_q[\mathrm{X}]$. It sends $s_i = f(i)$ to each party P_i, and broadcasts $A_k = g^{a_k}$, for $k = 0, \ldots, t$.
2. Each party P_i verifies the equality
$$g^{s_i} = \prod_{k=0}^{t} (A_k)^{i^k}.$$
If the verification fails, then P_i broadcasts a complaint against the dealer.
3. For each complaining party P_i, the dealer reveals s_i. If any of the revealed shares fails the verification, then the dealer is disqualified.
4. Before reconstructing the secret, the above equality is used again to detect invalid shares submitted by the players.

Figure 6.2: Feldman's Verifiable Secret Sharing

$f(i) = \sigma + a_1 i + \cdots + a_t i^t$, for $i = 0, \ldots, t$, it follows that $\boldsymbol{f} = \mathsf{A}\boldsymbol{a}$, or $\boldsymbol{a} = \mathsf{A}^{-1}\boldsymbol{f}$. Thus, we can define each A_k as

$$\begin{aligned} A_k &= g^{a_k} \\ &= \prod_{i=0}^{t} \left(g^{f(i)}\right)^{\lambda_{ki}}. \end{aligned}$$

This produces a distribution on all public values that is identical to that in the real execution of the protocol, so the simulation is complete. □

6.5 Distributed Key Generation

Consider now the problem of distributing secret decryption keys, so that no one player has access to the key, and a threshold number of players must be present to use it. We define the following security requirements for distributed key generation protocols.

Correctness We define correctness in terms of three properties.

1. All subsets of $t+1$ shares given by honest players define the same unique secret key.
2. At the end of the protocol, all honest players have the same public key.
3. The secret key is uniformly distributed in the secret key space, and the public key is uniformly distributed in the public key space.

Secrecy For every PPT $\mathscr{A}$, that corrupts up to t parties, there exists a PPT simulator $\mathscr{S}$ that, on input y, produces an output distribution that is indistinguishable from $\mathscr{A}$'s view of a real-world execution of the protocol that ends with public key y.

We would ideally like to construct such a scheme for all of the homomorphic cryptosystems we have discussed. However, this is not necessarily an easy task. We focus on the discrete log-based encryption schemes, notably Elgamal.

6.5.1 Pedersen's DKG Protocol

The first attempt at a secure distributed key generation protocol for discrete log keys was made by Pedersen [Ped91], shown in Figure 6.3. The protocol is based on several concurrent executions of Feldman's VSS protocol.

The protocol appears secure, but Gennaro, et al. [GJKR99], describe an attack that allows an adversary to influence the distribution of the shared secret. Their attack proceeds as follows. Suppose the adversary has compromised two parties, P_1 and P_2. Furthermore, suppose it wishes to bias the distribution of the public key so that it is more likely to have a least significant bit of 0. At the end of step 1, the adversary will compute $\alpha = \prod_{i=1}^{n} A_{i0}$ and $\beta = \prod_{i=2}^{n} A_{i0}$. That is, it computes the value of the public key both with and without the participation of player P_1. If the least significant bit of α is 0, then the adversary will do nothing. Otherwise, it will have player P_2 broadcast a complaint against P_1, forcing its disqualification. In this case, the public key will be set to the value β. Since both α and β have least significant bits of 0 with probability $1/2$, the adversary can influence the outcome of the least significant bit of the public key with probability $3/4$. Clearly, the fault in this scheme is that is allows the adversary to have some influence over the set Q.

6.5.2 Public DKG

Fouque and Stern [FS01a] devise a simple distributed key generation protocol that eliminates the need for private channels. Furthermore, their scheme is

1. Each party P_i generates a random polynomial $f_i(\mathrm{X}) = a_{i0} + a_{i1}\mathrm{X} + \cdots + a_{it}\mathrm{X}^t \in \mathbb{Z}_q$. It sends $s_{ij} = f_i(j)$ to each party P_j, and broadcasts $A_{ik} = g^{a_{ik}}$, for $k = 0, \ldots, t$.

2. Each party P_j verifies the equality

$$g^{s_{ij}} = \prod_{k=0}^{t} (A_{ik})^{j^k},$$

for each $i = 1, \ldots, n$. If the verification fails for any i, then P_j broadcasts a complaint against player P_i.

3. For each complaining party P_j, player P_i reveals s_{ij}. If any of the shares fails the verification, then P_i is disqualified. Define the set Q to be the set of all non-disqualified players.

4. The public key is computed as $y = \prod_{i \in Q} A_{i0}$. The public verification values are $A_k = \prod_{i \in Q} A_{ik}$, for $k = 1, \ldots, t$. The secret share of player P_j is $x_j = \sum_{i \in Q} s_{ij}$, and the secret shared value is $x = \sum_{i \in Q} a_{i0}$ (this value is not actually computed).

Figure 6.3: Pedersen's Distributed Key Generation

invulnerable to the attack previously described on Pedersen's scheme, as the adversary cannot make any decisions about which players are disqualified. Observe that the above protocol relies on a non-interactive proof of knowledge. Such a "proof of fairness", showing that a Paillier ciphertext encrypts the same thing as a committed value, can be found in [FS01a].

Proposition 6.5.1. *Assuming the DCRA, the Fouque-Stern DKG is secure against static adversaries in the random oracle model.*

Proof. Let $\mathscr{A}$ be a static adversary that has compromised t players. Without loss of generality, suppose that the t compromised players are $P_1, \ldots, P_t$. We describe a simulator $\mathscr{S}$ that, on input $y \in \langle g \rangle$, produces bulletin board values for the uncompromised players $P_{t+1}, \ldots, P_\ell$. Specifically, $\mathscr{S}$ must produce, for each $i = t+1, \ldots, \ell$, values (g_i, n_i); A_{ik}, for $k = 0, \ldots, t$; y_{ij}, for $j = 1, \ldots, \ell$; and (e_{ij}, w_{ij}, z_{ij}), for $j = 1, \ldots, \ell$.

1. Each player P_i publishes a Paillier keypair (g_i, n_i), and stores the secret key $\lambda(n_i)$.

2. Player P_i generates a random s_{i0} and selects a random polynomial $f_i(\mathrm{X}) = s_{i0} + a_{i1}\mathrm{X} + \cdots + a_{it}\mathrm{X}^t \in_{\mathrm{U}} \mathbb{Z}_q[\mathrm{X}]$. It then publishes $A_{ik} = g^{a_{ik}}$, for each $k = 0, \ldots, t$, as well as $y_{ij} = g^{s_{ij}}$ and $Y_{ij} = g_j^{s_{ij}} u_{ij}^{n_i}$, followed by a proof (e_{ij}, w_{ij}, z_{ij}), where $s_{ij} = f_i(j)$.

3. Player P_i verifies, for each $j = 1, \ldots, \ell$, the equality
$$\prod_{k=0}^{t} A_{ik}^{j^k} = g^{f_i(j)}.$$
Additionally, P_i checks that $g^{f_i(j)} = y_{ij}$, verifies the proofs (e_{ij}, w_{ij}, z_{ij}), and checks that $y_{ik}^q = 1$. If any check does not pass, P_i broadcasts a complaint against P_j.

4. The set Q is formed from the players who have participated correctly. All others are disqualified.

5. Player P_j decrypts Y_{ij} to get s_{ij}, for each $i = 1, \ldots, \ell$. For each $i \in Q$, it stores s_{ij}, and computes the public key as $y = \prod_{i \in Q} A_{i0}$. The secret share of P_j is equal to $\sum_{i \in Q} s_{ij} = f(j)$, and the secret key is $s = \sum_{i \in Q} a_{i0}$.

Figure 6.4: Fouque-Stern Distributed Key Generation

For $i = t+1, \ldots, \ell - 1$, the simulator will choose a Paillier keypair (g_i, n_i) at random. Then, it will choose $a_{ik} \in_{\mathrm{U}} \mathbb{Z}_q$, for $k = 0, \ldots, t$, and set $f_i(\mathrm{X}) = \sum_{k=0}^{t} a_{ik}\mathrm{X}^k$. Next, it will set $A_{ik} = g^{a_{ik}}$, for $k = 0, \ldots, t$. Finally, it will set, for $j = 1, \ldots, \ell$, both $y_{ij} = g^{f_i(j)}$ and $Y_{ij} = g_j^{f_i(j)} u_{ij}^{n_j}$, with $u_{ij} \in_{\mathrm{U}} \mathbb{Z}_{n_j}^*$.

For player P_ℓ, the simulator will do the following. Since the desired generated key is $y = \prod_{i \in Q} A_{i0}$, the simulator will set
$$A_{\ell 0} = y \prod_{i \in Q \setminus \{P_\ell\}} (A_{i0})^{-1}.$$
Next, it will choose $f_\ell(j) \in_{\mathrm{U}} \mathbb{Z}_q$, for $j = 1, \ldots, t$, and set $y_{\ell k} = g^{f_\ell(j)}$. It will

then interpolate to get $y_{ij} = g^{f_\ell(j)}$, for $j = t+1, \ldots, \ell$. Finally, it will choose $s_{\ell j}$ and $u_{\ell j}$ at random, and set $Y_{\ell j} = g_j^{s_{\ell j}} u_{\ell j}^{n_\ell}$.

For the zero-knowledge proofs, $\mathscr{S}$ will set

$$H(g, G, y, Y, g^z y^{-e}, G^z w^N Y^{-e}) = e.$$

Observe that the distributions of the values generated by $\mathscr{S}$ are indistinguishable from the values generated by a real execution of the protocol. □

6.6 Threshold Homomorphic Cryptosystems

Based on verifiable secret sharing, we can construct distributed threshold-based versions of the cryptosystems described previously.

Definition 6.6.1. A *threshold cryptosystem* is a 9-tuple

$$(M, C, K, R, S, \mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec}, \mathrm{Com}),$$

such that

1. M is the *message space*.
2. C is the *ciphertext space*.
3. K is the *key space*.
4. R is the *randomness space*.
5. S is the *share space*.
6. $\mathrm{Gen}\colon \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times R \longrightarrow K \times \mathscr{P}(K) \times \mathscr{P}(K)$ is a *key generation algorithm*. It takes as input a security parameter, the number of players, the threshold, and randomness. It produces a public key, a list of private keys, and a list of verification keys.
7. $\mathrm{Enc}\colon M \times K \times R \longrightarrow C$ is an *encryption algorithm*.
8. $\mathrm{Dec}\colon C \times K \longrightarrow S$ is a *share decryption algorithm*. It takes as input a public key, a private key, and a ciphertext. It produces a decryption share.
9. $\mathrm{Com}\colon C \times \mathscr{P}(S) \longrightarrow M$ is a *combining algorithm*. It takes as input a ciphertext, a list of decryption shares, and produces a plaintext.

Fouque, et al., propose of definition of security [FPS01] for homomorphic threshold encryption schemes. We will refer to this definition as *indistinguishability under distributed chosen plaintext attacks* (IND-DCPA). It is a natural extension of the IND-CPA definition for non-distributed encryption. The game is shown in Figure 6.5.

```
Game$^{\mathscr{A},\mathfrak{C}}_{\text{IND-DCPA}}(1^\lambda)$                                          Random variables
$(k_p, \{\sigma_i\}_1^\ell, \{v_i\}_1^\ell) \leftarrow \text{Gen}(1^\lambda, \ell, t; \rho)$              $\rho \leftarrow R$
$(S, aux) \leftarrow \mathscr{A}_1$, such that $|S| = t, S \subset \{1,\dots,\ell\}$
$(m_0, m_1, aux') \leftarrow \mathscr{A}_2^{\mathscr{D}}(\{\sigma_i\}_{i\in S}, k_p, aux)$
$c \leftarrow \text{Enc}_{k_p}(m_b; r)$                                 $b \leftarrow \{0,1\}, r \leftarrow R$
$b^* \leftarrow \mathscr{A}_3^{\mathscr{D}}(c, aux')$
if $b = b^*$ then return 1
else return 0
```

Figure 6.5: The IND-DCPA game

In this game, the $\mathscr{D}$ oracle is a decryption oracle, which on input m, returns ℓ shares of the decryption of $\text{Enc}_{k_p}(m)$. One should also be aware that the admission of a decryption oracle to the adversary does not transform this into a chosen ciphertext attack. Instead, the adversary is only permitted to decrypt ciphertexts of chosen *plaintexts*. Observe that IND-DCPA is equivalent to IND-CPA in the case when $\ell = 1$ and $t = 0$.

6.6.1 Threshold Elgamal

Figure 6.6 illustrates a threshold Elgamal scheme with a trusted dealer.

We first show that the threshold Elgamal scheme actually defines a proper threshold encryption scheme. That is, assuming all players have followed the protocol, an encrypted message can be recovered.

Proposition 6.6.2. *The threshold Elgamal cryptosystem is correct.*

Proof. Let S be a subset of $t+1$ shares. Suppose M is a message, and $c = (G, H)$ is a ciphertext encrypting M. Then, the tallier can compute

$$\begin{aligned}\prod_{j\in S} G^{\lambda_j x_j} &= G^{\sum_{j\in S}\lambda_j x_j} \\ &= G^{P(0)}.\end{aligned}$$

Now,

$$\begin{aligned}H/G^{P(0)} &= \frac{h^r g^M}{(g^r)^{P(0)}} \\ &= g^M.\end{aligned}$$

The tallier can then use an efficient algorithm to recover M from g^M. □

Parameters Safe prime $p = 2q + 1$, generator g of q-order subgroup of $\mathbb{Z}_p^*$, set of trustees T with $|T| = n$, threshold t.

Dealing The dealer generates an Elgamal keypair (h, x), and publishes h. Then, it chooses $a_1, \ldots, a_{t-1} \in_{\mathrm{U}} \mathbb{Z}_q$. It forms the polynomial $P(\mathrm{X}) = x + a_1\mathrm{X} + a_2\mathrm{X}^2 + \cdots + a_{t-1}\mathrm{X}^{t-1}$, and sends $x_i = P(i)$ to trustee i. Finally, it publishes h to the bulletin board. The verification key v_i of player P_i is then published as g^{v_i}.

Encryption To encrypt a message, a party retrieves h from the bulletin board, and encrypts a ciphertext as $(g^r, h^r g^m)$, where $r \in_{\mathrm{U}} \mathbb{Z}_q$.

Decryption Each trustee i retrieves the ciphertext (G, H). It then posts G^{x_i} as its partial decryption, along with a proof that $\log_G G^{x_i} = \log_g g^{x_i}$.

Share combining The tallier retrieves G^{x_i} for all i. It computes each coefficient λ_i of the Lagrange interpolating polynomial. It then computes $G^{P(0)}$ as $\prod_{j=1}^{t} G^{x_j}$. Finally, it computes g^m as $H/G^{P(0)}$, and uses an efficient algorithm to recover m.

Figure 6.6: Threshold Elgamal Procedure

Proposition 6.6.3. *In the random oracle model, assuming DDH, the threshold Elgamal cryptosystem is IND-DCPA secure.*

Proof. Let $\mathscr{A}$ be an IND-DCPA threshold Elgamal adversary. We wish to construct $\mathscr{B}$, an IND-CPA Elgamal adversary. Observe that $\mathscr{B}$ needs to simulate the following data to be passed to $\mathscr{A}$: the verification keys v_i, for $i = 1, \ldots, \ell$; the secret shares of the corrupted players, x_i, for $i = 1, \ldots, t$; and, for any ciphertext c encrypting a message m, $\mathscr{B}$ must simulate shares of the decryption σ_i, and proofs of validity π_i, for $i = 1, \ldots, \ell$.

Adversary $\mathscr{B}$ will operate as follows. First, $\mathscr{A}_1$ will choose a set of S of t servers to corrupt. Without loss of generality, assume that these servers are $P_1, \ldots, P_t$. Upon receiving the Elgamal public key (p, g, h), $\mathscr{B}_1$ and pass the public key to $\mathscr{A}_2$.

Next, $\mathscr{B}_1$ must give $\mathscr{A}_1$ the secret shares of all corrupted players. In the real world, the secret shares x_i are evaluations of a random polynomial over $\mathbb{Z}_q$. Thus, in the simulation, $\mathscr{B}_1$ will choose $s_i \in_{\mathrm{U}} \mathbb{Z}_q$, for$i = 1, \ldots, t$.

The simulator must now choose the verification shares, for all players. Observe that in a real world dealing of the shares, for any set S of size $t+1$, if $i \notin S$, then $v_i = \prod_{j \in S} v_j^{\lambda_i}$. Note that $g^{P(0)}$, in the real world, is a random element of QR_p. Thus, $\mathscr{B}_1$ will choose a dummy value of $y \in_{\mathrm{U}} \mathrm{QR}_p$. So, for each of the corrupted players, the verification keys of the corrupted servers are computed as $v_i = g^{x_i}$. The verification keys of the uncorrupted servers are computed as

$$v_i = y^{\lambda_0} \prod_{j \in S \setminus \{0\}} g^{x_j \lambda_j}.$$

Finally, we turn to the simulation of the decryption oracle. On input M from $\mathscr{A}$ the simulator $\mathscr{B}$ will compute a ciphertext $c = (G, H)$. Then, the shares of the corrupted players are computed as $c_i = G^{x_i}$, for $i = 1, \ldots, t$. As with the verification shares v_i, the decryption shares of the uncorrupted players $t+1, \ldots, \ell$ will be computed as

$$c_i = Y^{\lambda_0} \prod_{j \in S \setminus \{0\}} G^{x_j \lambda_j},$$

where $Y \in_{\mathrm{U}} \mathrm{QR}_p$ in place of $G^{P(0)}$. The proofs can be simulated using the method of Proposition 5.5.2. Thus, $\mathscr{A}$ cannot detect the simulation. □

Using the Fouque-Stern DKG, we can eliminate the dealer from the threshold Elgamal scheme. This can be done by replacing the "dealing" phase of threshold Elgamal with an execution of the Fouque-Stern DKG. At the end of this phase, each player will have a share of the decryption key, which can be used in the same manner as in the threshold Elgamal scheme. As the Fouque-Stern DKG is intended for distributing discrete-log keys, it cannot be used for other encryption schemes, as discussed below.

6.6.2 Threshold Paillier

Fouque, et al., give a threshold Paillier scheme [FPS01], shown in Figure 6.8.

Proposition 6.6.4. *The threshold Paillier cryptosystem is correct.*

Proof. Suppose S is a subset of $t+1$ correct shares. First, we observe that $c^{4\Delta^2 m\beta}$ is easy to compute. To see this, notice that

$$\begin{aligned} \Delta f(0) &= \Delta m\beta \\ &= \sum_{j \in S} \mu_{0,j}^S f(j) \bmod nm, \end{aligned}$$

Parameters Set of trustees T with $|T| = \ell$ and $\Delta = \ell!$, RSA composite $n = pq$ with $p = 2p' + 1$ and $q = 2q' + 1$, $m = p'q'$, $g = (1+n)^a b^n$, $(a, b) \in_{\mathrm{U}} \mathbb{Z}_n^* \times \mathbb{Z}_n^*$.

Dealing The dealer generates a secret key βm, where $\beta \in_{\mathrm{U}} \mathbb{Z}_n^*$, and a public key (g, n, θ), where $\theta = \mathrm{L}(g^{m\beta}) = am\beta$. It then generates a random polynomial $f(\mathrm{X}) \in \mathbb{Z}_{mn}^*[\mathrm{X}]$ such that $f(\mathrm{X}) = \beta m + a_1\mathrm{X} + \mathrm{a}_2\mathrm{X}^2 + \cdots a_t\mathrm{X}^t$. It sends $s_i = f(i)$ to trustee i. Finally, it publishes (g, n, θ) to the bulletin board.

Encryption To encrypt a message M, a party retrieves (g, n, θ) from the bulletin board, and encrypts a ciphertext as $g^M x^n$, where $x \in_{\mathrm{U}} \mathbb{Z}_n^*$.

Decryption Each trustee i retrieves the ciphertext c. It then posts $c_i \leftarrow c^{2\Delta s_i}$ as its partial decryption, along with a proof that $c^{4\Delta}$ and v^{Δ} both raised to the power s_i yield c_i^2 and v_i, respectively.

Share combining The tallier retrieves c_i for all i. It computes each coefficient λ_i of the Lagrange interpolating polynomial. Finally, it recovers the message as

$$M = \mathrm{L}\Big(\prod_{j \in S} c_j^{2\Delta\lambda_j}\Big).$$

Figure 6.7: Threshold Paillier procedure

and so

$$\begin{aligned} c^{4\Delta^2 m\beta} &= \prod_{j \in S} c^{4\Delta s_j \mu_{0,j}^{s}} \\ &= \prod_{j \in S} c_j^{2\mu_{0,j}^{S}} \bmod n^2. \end{aligned}$$

Now, let M be a message, and let c be the encryption of M. Then, we can compute

$$\begin{aligned} c^{4\Delta^2 m\beta} &= g^{4\Delta^2 Mm\beta} \\ &= (1+n)^{4\Delta^2 Mam\beta} \\ &= 1 + 4\Delta^2 Mam\beta n \bmod n^2. \end{aligned}$$

Thus,

$$\begin{aligned} \mathrm{L}\Big(\prod_{j\in S} c_j^{2\mu_{0,j}^S} \bmod n^2\Big) &= 4M\Delta^2 am\beta \\ &= 4M\Delta^2\theta \bmod n. \end{aligned}$$

Dividing by $4\Delta^2\theta$, we arrive at the message M. □

Proposition 6.6.5. *In the random oracle model, and assuming DCRA, the threshold Paillier cryptosystem is IND-DCPA secure.*

Proof. Let $\mathscr{A}$ be an IND-DCPA threshold Paillier adversary. We wish to construct $\mathscr{B}$, an IND-CPA Paillier adversary. Observe that $\mathscr{B}$ needs to simulate the following data to be passed to $\mathscr{A}$: the public key parameter θ; the verification keys v_i, for $i = 1, \ldots, \ell$; the secret shares of the corrupted players, s_i, for $i = 1, \ldots, t$; and, for any ciphertext c encrypting a message m, $\mathscr{B}$ must simulate shares of the decryption σ_i, and proofs of validity π_i, for $i = 1, \ldots, \ell$.

Adversary $\mathscr{B}$ will operate as follows. First, $\mathscr{A}_1$ will choose a set of S of t servers to corrupt. Without loss of generality, assume that these servers are $P_1, \ldots, P_t$. Upon receiving the Paillier public key (n, g), $\mathscr{B}_1$ will randomly choose $a_1, b_1, \theta \in_{\mathrm{U}} \mathbb{Z}_n^*$, set $g_1 = g^{a_1} b_1^n$, and pass the public key (g_1, n, θ) to $\mathscr{A}_2$. Observe that in the real world, θ is equal to $am\beta \bmod n$, a random element in $\mathbb{Z}_n^*$. In the simulation, θ is also chosen randomly from $\mathbb{Z}_n^*$.

Next, $\mathscr{B}_1$ must give $\mathscr{A}_1$ the secret shares of all corrupted players. In the real world, the secret shares s_i are evaluations of a random polynomial over $\mathbb{Z}_{nm}$. However, m is not known to $\mathscr{B}$, thus it cannot pick from $\mathbb{Z}_{nm}$. So, in the simulation, $\mathscr{B}_1$ will choose $s_i \in_{\mathrm{U}} \{0, \ldots, \lfloor n^2/4 \rfloor\}$, for $i = 1, \ldots, t$.

The simulator must now choose the verification shares, for all players. Observe that in a real world dealing of the shares, two conditions hold. First, for any set S of size $t+1$, if $i \notin S$, then $v_i^{\Delta} = \prod_{j\in S} v_j^{\mu_{i,j}^S}$. Second, for any set S of size $t+1$, it holds that $\prod_{j\in S} v_j^{\mu_{0,j}^S} \equiv 1 \pmod n$. Again, m is unknown, so $\mathscr{B}$ cannot compute a meaningful value of $v^{m\beta}$. However, note that $v^{m\beta}$, in the real world, is a random element of QR_{n^2}, which is also 1 mod n. Thus, $\mathscr{B}_1$ will choose a dummy value of $v^{m\beta}$ as $1 + 2\alpha\theta n \bmod n^2$. So, for each of the corrupted players, the verification keys of the corrupted servers are computed as $v_i = v^{\Delta s_i}$, where $v = g_1^{2\alpha}$. The verification keys of the uncorrupted servers are computed as

$$v_i = (1 + 2\alpha\theta n)^{\mu^S_{i,0}} \prod_{j \in S \setminus \{0\}} v^{s_j \mu^S_{i,j}}.$$

Finally, we turn to the simulation of the decryption oracle. On input M from $\mathscr{A}$ the simulator $\mathscr{B}$ will compute a ciphertext $c = g_1^M x^n$. Then, the shares of the corrupted players are computed as $c_i = c^{2\Delta s_i}$, for $i = 1, \ldots, t$. As with the verification shares v_i, the decryption shares of the uncorrupted players $t+1, \ldots, \ell$ will be computing as

$$c_i = (1 + 2M\theta n)^{\mu^S_{i,0}} \prod_{j \in S \setminus \{0\}} c^{s_j \mu^S_{i,j}}.$$

To simulate the proofs, $\mathscr{B}$ will set the value of the hash function H at

$$(v, c^{4\Delta}, v_i, c_i^2, v^y / v_i^e, c^{4\Delta^2 y} / c_i^{2e})$$

to be e. Thus, $\mathscr{A}$ cannot detect the simulation. □

Removing the trusted dealer from threshold Paillier is not a trivial problem. The main issue is that if we are to jointly distribute the modulus n, we must also jointly distribute its prime factors p and q. Furthermore, we must have some assurance that the prime factors are *safe* primes.

6.6.3 Threshold Damgård-Jurik

Our last example is a threshold version of the Damgård-Jurik cryptosystem. The protocol is shown in Figure 6.8.

Proposition 6.6.6. *The threshold Damgård-Jurik cryptosystem is correct.*

Proof. Suppose S is a subset of $t+1$ correct shares. Observe that

$$\begin{aligned} d &= \prod_{i \in S} d_i^{2\lambda^S_i} \\ &= G^{4\Delta^2 \alpha} \\ &= h^{4\Delta^2 r} \bmod n. \end{aligned}$$

Then,

$$\begin{aligned} m &= \mathrm{L}_s(Hd^{-n^s}) \\ &= (1+n)^m \bmod n^{s+1}, \end{aligned}$$

and so the message is recovered. □

Proposition 6.6.7. *In the random oracle model, and assuming the DCRA and CDDH, the threshold Damgård-Jurik is IND-DCPA secure.*

We omit the proof of this proposition, as it is quite similar to the proof of Proposition 6.6.5.

6.7 Conclusion

The powerful ideas of distributed trust introduced in this chapter will be essential in constructing voting systems. In large-scale systems, where there are many interacting parties, many of whom have opposing goals, it is necessary to distribute trust in a way that no small group of users can have an unfair influence on the overall behavior of the system. We have shown how to distribute encryption schemes, so that the secret key of the cryptosystem is not possessed by any one party. Combining this with homomorphic encryption, it is now possible to conduct a procedure where individual ciphertexts remain hidden, while the sum of the plaintexts can be computed and decrypted safely.

Parameters Set of trustees T with $|T| = \ell$ and $\Delta = \ell!$, RSA composite $n = pq$ with, $g \in \mathrm{QR}_n$, threshold $t \geq \ell/2$.

Dealing The dealer generates a secret key $\alpha \in \mathbb{Z}_\tau$, and a public key $h = g^\alpha$, where. It then generates a random polynomial $f(\mathrm{X}) \in \mathbb{Z}_\tau[\mathrm{X}]$ such that $f(\mathrm{X}) = \alpha + a_1\mathrm{X} + \mathrm{a}_2\mathrm{X}^2 + \cdots a_t\mathrm{X}^t$. It sends $s_i = f(i)$ to trustee i. Finally, it publishes (g, n, h) to the bulletin board. The verification value of player P_i is $h_i = g^{\alpha_i}$.

Encryption To encrypt a message $m \in \mathbb{Z}^+$, a party retrieves (g, n, h) from the bulletin board, chooses $s > 0$ such that $m \in \mathbb{Z}_{n^s}$, picks $r \in_{\mathrm{U}} \mathbb{Z}_N$, and encrypts a ciphertext as

$$\mathrm{Enc}_s(m, r) = \left(g^r, (h^{4\Delta^2 r})^{n^s}(n+1)^m\right).$$

Decryption Each trustee i retrieves the ciphertext (G, H). It then posts $d_i = G^{2\Delta\alpha_i}$ as its partial decryption, along with a proof that $\log_g h_i = \log_{G^{4\Delta}}(d_i^2)$.

Share combining The tallier retrieves d_i for all i. The tallier will compute each coefficient of the Lagrange interpolating polynomial as

$$\lambda_i^S = \prod_{j \in S \setminus \{i\}} \Delta \frac{j}{j-i}.$$

It then recovers the message as

$$m = \mathrm{L}_s(Hd^{-n^s}),$$

where $d = \prod_{i \in S} d_i^{2\lambda_i^S}$.

Figure 6.8: Threshold Damgård-Jurik procedure

Chapter 7

The Election Procedure

We now possess the tools necessary to design robust voting schemes. In this chapter, we explore how the techniques described thus far can be used to achieve such an end.

7.1 The Players

We can divide users of a voting system into four distinct rôles: *voters*, *trustees*, *auditors*, and *administrators*. As expected, voters are the players who cast ballots. Trustees are responsible for maintaining the integrity of the election. They possess shares of the secret decryption key needed to recover the final election result. In the literature, trustees are often called "authorities," but we feel that the term "trustees" more accurately reflects the nature of their position. Trustees do not have more power than any other users. Administrators are tasked with creating and maintaining current elections. Auditors can consist of members of the previous three groups, as well as members of the public at large. They are able to perform tasks guaranteed by the property of universal verifiability, e.g., inspecting the election transcript, double-checking the final tally, etc.

These groups may overlap arbitrarily. For instance, it may be desirable to make the trustees a subset of the voters, so that the voters themselves can ensure the integrity of the election. Furthermore, the administrators might contain some members who are trustees, and some who are neutral parties. Finally, the auditors would contain all of the above members, as well as additional neutral parties. Such a scenario is depicted in Figure 7.1.

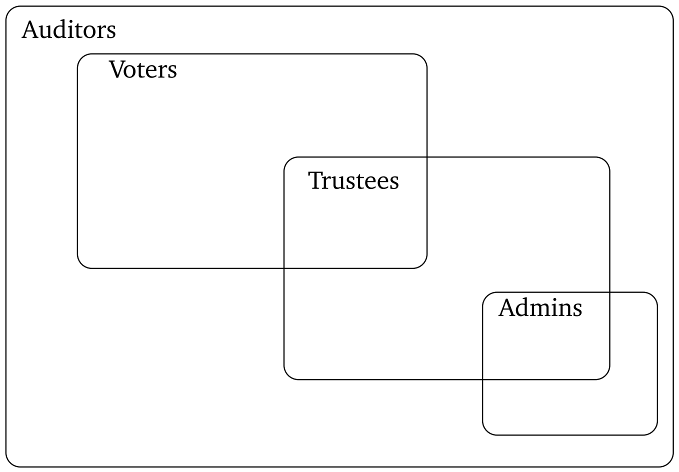

Figure 7.1: Possible separation of duties.

7.2 Ballot Encoding

We begin by discussing various ways of performing elections. Namely, we discuss how ballots can be cast, and how winners can be chosen.

There are several voting methods. We will concern ourselves with the most well-known. In the *plurality* selection method, the winner is selected to be the candidate with the most votes. It is important to observe that the winner need not have the *majority* of votes, only the most. For example, suppose there are three candidates: A, B, and C. Suppose candidate A receives 40% of the votes and candidates B and C each receive 30%. Candidate A is selected as the winner, because it received the most votes, even though it did not receive the majority.

In order to use homomorphic encryption for ballot casting, it is necessary to find some method of numerically encoding ballots. Depending on what sorts of ballots are allowed, there are numerous ways of encoding them.

7.2.1 Yes/No Elections

In a *yes/no election*, voters are asked to select one of two options. For example, there could be a proposal for which voters must accept or reject, or there could be actually be two distinct candidates. For yes/no elections, we represent the two candidates using the numbers 1 and 0. That is, the first candidate (say, YES) is assigned the number 1, and the second candidate (say, NO), is assigned

Voter	Value	Choice
V_1	0	NO
V_2	1	YES
V_3	1	YES
V_4	1	YES

the number 0. Each voter would then cast his ballot by encoding either the number 0 or 1. The votes are then added, and the result gives the number of YES votes. Subtracting the number of YES votes from the total yields the number of NO votes.

Example 7.2.1. Suppose we have four voters:

When the votes are tabulated, the result is $0+1+1+1=3$, which gives us the number of YES votes. This, subtracted from the total number of votes, gives $4-3=1$, which is the number of NO votes. ◊

7.2.2 Multi-way Elections

Many elections, however, do not have only two candidates, e.g., US presidential elections. These are called *multi-way elections*. In a multi-way election, there are c candidates, and the voter is asked to choose one. Again, we must find some way to encode a ballot as an integer. One elegant way of doing this is to encode the ballots as base-M numbers, where M is some integer larger than the total number of voters. Each power of M represents one candidate, and the voters are asked to submit a number which is a power of M. That is, if there are c candidates, the possible ballots would be $M^0, M^1, \ldots, M^{c-1}$. To vote for candidate i, a voter would submit M^i.

Example 7.2.2. Imagine a three-way election, and four voters. The candidates are labeled A, B, and C.

Voter	Value	Choice
V_1	010	B
V_2	100	C
V_3	001	A
V_4	001	A

Tabulation is base-M addition, where M is greater than the number of voters. The result is $010 + 100 + 001 + 001 = 112_M$. Each position gives us the number of votes for each candidate. That is, A received 2 votes, B received 1, and C received 1. Thus, candidate A is the winner. $\Diamond$

7.2.3 Limited Vote

There is a generalization of the multi-way election called *limited vote*. In the limited vote setting, a voter is able to choose t out of the c candidates, where $t \leq c$ is fixed. One method of approaching this problem is by treating the election as c concurrent yes/no elections. Each voter would then submit a $\mathbb{Z}^c$-vector of yes/no ballots, where each coordinate corresponds to a candidate. In position i of the vector, the voter would choose whether or not it wishes to vote for candidate i, by selecting either 0 or 1, corresponding to YES or NO. Once all ballots have been submitted, they are added component wise. The total in each coordinate of the result indicates how many votes that candidate received.

Example 7.2.3. Again, we have a three-way election, and four voters. The candidates are labeled A, B, and C, and voters are asked to choose two of the three candidates.

Voter	Value	Choice
V_1	$(0,1,1)$	B and C
V_2	$(1,1,0)$	A and B
V_3	$(1,0,1)$	A and C
V_4	$(0,1,1)$	B and C

The ballots are added component-wise. The total is then

$$(0,1,1) + (1,1,0) + (1,0,1) + (0,1,1) = (2,3,3).$$

Thus, there is a draw between candidates B and C. $\Diamond$

7.3 Ballot Casting

In a cryptographic voting system, it is crucial that the privacy of each voter is maintained. Thus, ballots cannot be cast in the clear. That is, they must be encrypted. The ballot encoding methods discussed in §7.2 lend themselves naturally to being used with homomorphic encryption schemes.

Suppose we have a homomorphic cryptosystem (Gen, Enc, Dec). If a voter wishes to cast a ballot, it will first encode its ballot using one of the previously described methods. Suppose for simplicity that the election is using one of the methods where ballots are encoded as integers. That is, the voter will choose some encoding v of its vote. Then, it will compute $\mathrm{Enc}(v; r)$, with some randomness r. Suppose now that there is another voter who has also cast a vote v', encrypted as $\mathrm{Enc}(v'; r')$. Once the two votes have been cast, they can be combined as $\mathrm{Enc}(v; r)\,\mathrm{Enc}(v'; r') = \mathrm{Enc}(v + v'; r + r')$, using the homomorphic property of Enc. Anyone who possesses the decryption key can now decrypt $\mathrm{Enc}(v + v'; r + r')$, revealing the total $v + v'$. We have thus achieved what we set out to: addition of votes without knowledge of the plaintexts.

Example 7.3.1. To illustrate voting with homomorphic encryption, let us suppose we are using the homomorphic Elgamal cryptosystem with a yes/no election.

Voter	Value	Ciphertext	Choice
V_1	0	(g^{r_1}, h^{r_1})	NO
V_2	0	(g^{r_2}, h^{r_2})	NO
V_3	1	$(g^{r_3}, h^{r_3}g)$	YES
V_4	0	(g^{r_4}, h^{r_4})	NO
V_5	1	$(g^{r_5}, h^{r_5}g)$	YES

Multiplying these ciphertexts pairwise, we get $(g^{r_1+\cdots+r_5}, h^{r_1+\cdots+r_5}g^2) = \mathrm{Enc}(2; r_1 + \cdots r_5)$. Decrypting this ciphertext yields 2, the number of YES votes. If we subtract from 5, we get 3, the number of NO votes. ◊

Similarly, we can show that any additive homomorphic cryptosystem can be used in the same way to build a voting scheme. However, the fact that the ballots are encrypted, and hence never individually revealed, leaves open the possibility of voters' tampering with the election.

Example 7.3.2. Consider a simple yes/no election to see a devastating voter misbehavior:

V_5 submits a bogus ballot, but this will never be known, as it is encrypted. When tabulation occurs, there will be 5 "yes" votes, and 0 "no" votes! ◊

The above example shows that it is crucial that voters not be allowed to submit arbitrary votes. Since the votes themselves are encrypted, it is not possible for the bulletin board to verify the validity of the votes merely by inspecting them. Thus, it is necessary for the voter to *prove* that the vote is one of

Voter	Value	Choice
V_1	0	NO
V_2	0	NO
V_3	1	YES
V_4	0	NO
V_5	4	???

the allowed choices. Luckily, we have already introduced all of the machinery necessary for this task. To form such "proofs of ballot validity," we recall the techniques of Chapter 5.

The specifics of how such proofs are constructed depends on the ballot encoding method, as well as the cryptosystem being used. We outline below the basic techniques that can be used for each encoding method.

- *Yes/No Elections*: The voter must prove that its vote is either YES or NO. Since ballots are encoded as integers in $\{0, 1\}$, the problem reduces to that of proving that the ciphertext is the encryption of 0, and also proving that it is the encryption of 1, and then forming the OR-composition of these two proofs. For example, with the Elgamal cryptosystem, the protocol in Figure 5.5 can be used for exactly this purpose. Similarly, for other cryptosystems, protocols that prove that a ciphertext encrypts either 0 or 1 can be used.

- *Multi-Way Elections*: We can simply extend the technique for yes/no elections. Suppose there are c candidates, and M is an integer greater than the number of voters. Then each vote will be one of $M^0, M^1, \ldots, M^{c-1}$. Thus, the voter must prove that its vote is in the set $\{M^0, M^1, \ldots, M^{c-1}\}$. Provided that c is small enough, it may not be inefficient to provide an OR-composition of the predicates stating that the vote is each element of this set individually. The composition can easily be obtained by Theorem 5.6.2.

- *Limited Vote*: Since the voter is submitting a vector of ballots, it must submit a vector of proofs as well. It will submit one proof for each ballot, arguing the fact that the ballot encrypts either 0 or 1 (as in the yes/no election). Additionally, it must provide a proof that it has not voted for more than the allowed number of candidates. To do this, it will compute the sum of all the components of the vector, and then form a proof that the sum encrypts the number t, where t is the allowed number of selected candidates.

We have given simple examples for constructing zero-knowledge proofs of ballot validity. In [Gro05], Groth provides many efficient examples for proving ballot validity with several different voting schemes.

7.4 A Complete Procedure

In this section, we give a complete overview of how an election can be conducted. The scheme is constructed from the Fouque-Stern distributed key generation protocol described in §6.5.2. We suppose that the vote casting is done with a threshold homomorphic cryptosystem $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$, such as the homomorphic Elgamal scheme given in §6.6. If a cryptosystem not based on the discrete logarithm is used, the distributed key generation protocol may need to change. Furthermore, we assume that the voting method is plurality with limited vote.

1. *Procedure creation*. An election begins when an administrator logs onto the bulletin board server and submits the procedure creation data. Among the parameters specified are the procedure identifier, the identities of voters and authorities that are eligible to participate, the authority threshold t, the minimum and maximum number of candidates that voters can select, the list of candidates, and the election duration. Once the administrator submits this form, the bulletin board server populates the database with the corresponding information.

   The bulletin board server subsequently generates the cryptographic values for the election.

   In the following three steps the set of authorities that are enabled for the procedure will jointly produce the public key of the system initializing a threshold encryption scheme (cf. [Ped91]). We note that not all authorities may successfully carry out the steps. In the steps below, we will use the notation A_1,A_2, and A_3 to denote the subsets of authorities that succeed in completing the stages of the system public key generation. If the authorities that complete all three stages are below a safety threshold t', the system terminates the procedure. Note that the safety threshold satisfies $t' > t$, where t is the distributing trust parameter.

2. *Authority public key generation*. Once the procedure has been created, the authorities collaborate to create the public encryption key of the system. At the end of the election, they each contribute a part to the decryption of the result. Note that no authority has the ability to decrypt a single

vote because the actual private key of the system does not exist in the private memory of any one entity. Rather, it is broken up in the form of the authorities' individual private keys. In order to decrypt a single vote, an amount of authorities greater than the authority threshold t would all have to collaborate. To counter this, election officials should designate authorities with differing political interests, so they would have no incentive to collude.

An authority i downloads the cryptographic parameters from the bulletin board. It then generates a Paillier public key (g_i, n_i), and stores the corresponding secret key $\lambda(n_i)$. The public key is stored on the bulletin board. Once all authorities have completed this stage, the public key of each authority is stored on the bulletin board. Let A_1 be the set of authorities that have completed this step. If $|A_1| < t'$ (the safety threshold), then the server will terminate the procedure here.

3. *Polynomial generation*. At this stage, the authorities will participate in a protocol that will compute the public key for the election using a distributed key generation protocol.

   An authority i logs in, reads (g_j, n_j) (the public keys) for all authorities $j \in A_1$, and the authority threshold t. The authority creates its polynomial $f_i(\mathrm{X}) = s_{i0} + a_{i1}\mathrm{X} + \cdots + a_{it}\mathrm{X}^t$. It publishes value $A_{ik} = g^{a_i k}$, for each $k = 0, \ldots, t$, as well as $y_{ij} = g^{s_i j}$ and $Y_{ij} = g_j^{s_{ij}} u_{ij}^{n_i}$, followed by a proof $(e_i j, w_{ij}, z_{ij})$, for each $j \in A_1$. This serves as a form of encryption of authority j's index evaluated in authority i's polynomial. Let $A_2 \subseteq A_1$ be the set of authorities that have completed this step. If $|A_2| < t'$, the server terminates the procedure.

4. *Verification*. Trustee i verifies, for each $j = 1, \ldots, \ell$, the equality

$$\prod_{k=0}^{t} A_{ik}^{j^k} = g^{f_i(j)}.$$

   Additionally, trustee i checks that $g^{f_i(j)} = y_{ij}$, verifies each of the proofs (e_{ij}, w_{ij}, z_{ij}), and checks that $y_{ik}^q = 1$. If any check does not pass, it broadcasts a complaint against authority j. The set Q is formed from the players who have participated correctly. All others are disqualified.

5. *Private key generation*. Each authority $j \in A_2$ connects to the bulletin board server and reads Y_{ij} for all $i \in Q$. Note that authority j can decrypt all of these values, as they were encrypted with its public key. Thus, it

retrieves its private key from its private memory. It decrypts these values, computes their sum, and stores the result in private memory. At this point, authority j has the sum of the evaluations of its index in every other authority's polynomial. Let $A_3 \subseteq Q$ be the set of authorities that have completed this step. If $|A_3| < t'$, the server terminates the procedure.

6. *Public key publication*. The bulletin board server publishes the value $y = \prod_{i \in Q} A_{i0}$. Note that h is the sum of all of the authorities' polynomials evaluated at 0. This value y serves as the public key of the procedure, which voters will use when they encrypt their ballots.

7. *Voting*. Now, the election may begin. Each voter i downloads the public key y from the bulletin board, as well as the cryptographic parameters. The voter is permitted to vote between K_{min} and K_{max} candidates, where $0 \leq K_{min} \leq K_{max} \leq L$, and L is the total number of candidates. The voter thus forms a ciphertext for each candidate, encrypting 1 if the voter votes for that candidate, and 0 otherwise. The encrypted vote is therefore a vector of ciphertexts, one for each candidate. Along with the encrypted vote, the voter computes a proof of ballot validity Π as in §7.3. Finally, the pair (V, Π) is posted on the bulletin board.

   The bulletin board server then verifies the proof. If the proof is valid, and the voter has not already voted, it posts the vote and proof on the bulletin board. Otherwise, it discards the vote and proof and returns an error.

8. *Result tabulation*. When duration of the election has expired, or when an administrator manually ends the election, result tabulation occurs. The bulletin board contains received a vote V_i from each voter i. The encrypted sum of the votes is computed by the bulletin board server combining the votes as

$$V = \bigoplus_{i=0}^{m} V_i,$$

   where $\oplus$ is the homomorphic operation on votes, and V is a vector of ciphertexts encrypting the total number of votes for each candidate. This combined sum is then posted on the bulletin board.

9. *Authority decryption*. Now, each authority logs in again, downloads the encrypted result of the election, and submits its partial decryption. These partial decryptions are posted on the bulletin board.

10. *Result decryption*. Once each authority has submitted its partial decryption, the bulletin board server combines the partial decryptions and decrypts the result.

11. *Result publication*. The end result is finally published on the bulletin board. The election is over, and no more logins are accepted. Anyone can now view the final result.

7.5 Arguments for Security and Correctness

This section can be viewed as a point-by-point argument for why the above scheme is correct and secure. The proofs are not rigorous in a mathematical sense, but should be convincing from a high-level perspective.

7.5.1 Correctness

We provide arguments for why the scheme is correct. That is, assuming no party deviates from the protocol, the election will be carried out correctly, and the proper winner chosen.

Proposition 7.5.1. *Assuming the protocol is followed correctly, the final tally is equal to the sum of each voter's vote.*

Sketch of Proof. This follows from the homomorphic property of the encryption. □

Proposition 7.5.2. *The public key posted on the bulletin and used to encrypt votes corresponds to a private key that can be reconstructed from a threshold number of trustee shares.*

Sketch of Proof. This follows from the correctness of the Feldman verifiable secret sharing scheme. □

7.5.2 Security

Next, we explain why the protocol is secure. That is, assuming that some parties are malicious and intentionally tamper with the scheme, it should be resilient up to some point. In the presence of a limited adversary, the election will be carried out correctly, and the proper winner chosen.

Proposition 7.5.3. *Under appropriate hardness assumptions, and assuming that fewer than a threshold number of trustees collude, no voter's ballot can be revealed.*

Sketch of Proof. Since a ballot is a ciphertext in an IND-DCPA secure cryptosystem, it is computationally difficult to retrieve the plaintext without knowledge of the private key. Furthermore, since the private key is shared using a secret-sharing scheme, it is impossible to determine the secret without a threshold number of shares, or by solving the computational problem that protects the shared secret. □

Proposition 7.5.4. *No less than a threshold number of trustees can misbehave and prevent the election from being carried out properly.*

Sketch of Proof. By the security of the threshold homomorphic encryption, it is impossible for any trustee to submit data that leads to a disruption of the key generation, encryption, or decryption processes. □

Proposition 7.5.5. *Short of a denial-of-service attack, the bulletin board server cannot misbehave and prevent the election from being carried out properly.*

Sketch of Proof. We assume that all parties ensure that the bulletin board server actually posts the correct data. Since the bulletin board holds no secrets, it is not possible for the bulletin board server to fabricate any meaningful data of its own accord, without being detected by at least one participant. □

Proposition 7.5.6. *No voter can misbehave and prevent the election from being carried out properly.*

Sketch of Proof. The only data submitted to the bulletin board by a voter is the encrypted ballot and the proof of validity. Assuming that the bulletin board server checks the proof of validity, the ballot will not be counted in the main tally unless it has been proved to be benign. □

Proposition 7.5.7. *No third-party, without access to any secret information held by voters or trustees, but with access to all information held on the bulletin board server, can reveal an individual voter's ballot.*

Sketch of Proof. The secret information present in the scheme is the trustee secret shares, the private election key, the randomness of each voter, and the ballot of each voter. The third party does not have access to the private memory of trustees or voters, so it cannot obtain the secret shares or the randomness. The private election key cannot be obtained without a threshold number of trustee shares. Finally, the ballot of each voter cannot be obtained without the private election key. □

7.6 Conclusion

We have now given a complete voting procedure. It uses all of the techniques described thus far, including homomorphic encryption, proofs of knowledge, distributed key generation, and a threshold cryptosystem.

Chapter 8

Other Approaches to Voting

This thesis has focused on the homomorphic encryption approach to secure electronic voting. There are, however, alternative approaches. These approaches rely on different cryptographic primitives and, especially from an implementation point of view, have different advantages and disadvantages.

8.1 Mixnets

Introduced by Chaum in 1981, the *mixnet* [Cha81] is a cryptographic primitive well suited to elections. A mixnet is comprised of a collection of servers, called *mix servers*, whose task is to shuffle a given input sequence of ciphertexts. This serves as an implementation of a robust anonymous channel. The main difficulty in designing mixnet-based protocols is in requiring the mix servers to provide proofs of correct operation. Mix servers must prove that they have performed a correct shuffle, and that they have not dropped any votes. A good survey of mixnets for electronic voting is provided by [Adi06].

8.2 Blind Signatures

Blind signature-based schemes use a method proposed by Fujioka, Okamoto, and Ohta [FOO93]. In this scheme, voters form ballots, encrypt them, and send them to an administrator. The administrator then creates a *blind signature* on the ballot., i.e., a signature that does not require knowledge of the plaintext to form. The administrator then sends the encrypted ballot along with the blind signature back to the voter. Subsequently, voters submit their blindly signed ballots through an anonymous channel to a voting bulletin board that will only accept ballots signed by the administrator. Thus, the voter has demonstrated that the ballot has actually been created by someone who is permitted to vote (and, presumably, has only voted once), while the vote tallier has no knowledge of which voter cast which ballot.

There are several implementations based on the blind signature approach, all following the same basic scheme introduced by Fujioka, et al., known as the “FOO” scheme. The protocol is as follows: There are two authorities, an administrator and a collector. To vote, the voter fills out a ballot, commits to it, signs it, and sends it to the administrator. Using a *blind signature*, the administrator signs it, and sends it back to the voter. The voter unblinds the ballot, and sends the signed committed ballot anonymously to the counter. Once all voters have submitted their ballots, the counter numbers them, and publishes the list. Each voter is then responsible for finding its ballot in the list, and sending the number along with a key needed to uncommit the vote to the counter through an anonymous channel. The counter then uncommits the votes, publishes the keys, and announces the result of the election. A disadvantage of this scheme is that it requires an additional action on the part of the voter, as it must sent the collector the decryption key.

The main disadvantage of the blind signature approach is that the voter needs to be active in at least two phases to ensure verifiability. In practice, realizing an anonymous channel is not straightforward. In current implementations of blind signature schemes that run on the Internet, for example, it is easy to correlate voters with their votes.

8.3 Other Methods

While homomorphic encryption, mixnets, and blind signatures constitute the three main voting-oriented primitives found in the cryptographic literature, several other methods have been used in practice. The reader is referred to [KKW06] for a brief summary of existing implementations of electronic voting systems.

GNU.FREE [FRE] is a free voting system that is officially part of the GNU project, and remains one of the very few free electronic voting systems in existence. It is not, however, based on any voting-oriented cryptographic primitives. In GNU.FREE There are two servers, the *electoral roll server* (ERServer), and the *regional server* (RTServer). All communications are encrypted using BlowFish encryption with a key exchanged by RSA. The voter submits authentication information to the ERServer, who validates it. The voter then fills out a ballot, and sends a key along with a timestamp to the RTServer. The RTServer then stores this, and challenges the voter for the timestamp again. If the voter provides the valid timestamp, then the key is decrypted and sent to the ERServer. The ERServer records that the voter has submitted a key and cannot vote again. The RTServer then decrypts the vote, stores it, and deletes the key. The fault in this scheme is that malicious servers can record which user voted for which candidate as the votes are submitted. It is worth mentioning that the author of GNU.FREE, has publicly discontinued his work on the project, due to disillusionment with the concept of secure Internet voting.

Another implementation, SureVote [Cha04], is based on “visual crypto.” The voting machine produces an encryption of the voter’s ballot, which is published on a bulletin board. The voter receives a receipt which contains a unique serial number. The serial number can later be used to compare with the results posted on the bulletin board. At the poll-site, two overlayed transparent sheets are printed. When overlayed, the voter can see the ballot which corresponds to his choices. The voter then chooses a layer to keep, and a layer to be discarded in a publicly viewable shredder. The voter also keeps the serial number of his ballot. The ciphertext is then mixed for anonymity and finally decrypted and counted. Thus, the election provides an element of voter-verifiability.

Chapter 9

Implementation of the Adder Voting System

This chapter describes the implementation of the Adder Internet-based voting system.

9.1 Introduction

In an attempt to investigate the theoretical ideas discussed thus far in a practical setting, we have implemented the Adder system, an Internet-based realization of a homomorphic cryptographic voting scheme. The design of the system has been presented in [KKW06], and the reader is encouraged to seek out this paper for more information. Additionally, the system is available on the World Wide Web as free software at

`http://cryptodrm.engr.uconn.edu/adder/`

One of Adder's primary advantages over existing systems is that it is free software. That is, the source code of the system is available for the public to download, run, experiment with, modify, etc., as they see fit. Given the inherent difficulties in software development, the transparency and openness of the system allows users to be confident that the developers are not hiding any secrets. Especially in the case of cryptographic software, correct behavior of the program is not the only necessary feature; it must be secure as well. Public access to the source is important to ensure that security flaws will not remain buried.

Furthermore, we hope that the freedom of Adder will allow independent experimentation and usage. From an academic perspective, we would like to see Adder tested in many different environments, so that its limitations and weaknesses can be exposed and corrected. We also hope that Adder will be of use to many organizations throughout the world that need to make decisions in a private, verifiable manner.

As of this writing, Adder is the only free electronic voting system based on strong cryptography.

9.2 System Architecture

The Adder system is organized into several components: a cryptographic library, a main server, a Web browser applet, and a graphical client.

The cryptographic library implements much of the cryptographic functionality discussed in this thesis. It is based on the homomorphic threshold Elgamal cryptosystem of §4.2, and implements a simplified version of the Fouque-Stern distributed key generation protocol of §6.5.2. It also implements a generic framework for forming disjunctions of non-interactive proofs of knowledge (cf. §5.6), and builds the proofs of knowledge for Elgamal encryption (cf. §5.9.1) on top of this framework. The integer computations are performed using the GNU MP multiple precision arithmetic library, and abstracted in a custom integer class. The cryptographic routines are constructed in terms of this integer class.

The main server is an implementation of the idealized concept of the bulletin board, discussed in §6.3. It is implemented as a multi-threaded server written in C++. The server stores all persistent data in a MySQL database including data needed for user authentication. When a user (either a voter or an authority) connects to the server, the user presents to the server a username and password, which is matched against a record in the database. If the user is authenticated successfully, then the server allows the user to carry out the intended action. The server thus maintains the behavioral requirements of the bulletin board; that is, that users are not permitted to delete information, that each user has append access to a designated area of the bulletin board, and that all areas are world-readable. The user communicates with the bulletin board through a special protocol, and the internal structure of the server and the implementation of the data storage itself are invisible to the user. Thus, the bulletin board server presents a complete abstraction of the idealized bulletin board.

The Web applet is written in Java, and allows voting over the Web. As a cryptographic voting system such as this requires heavy computation on the voter's computer, an applet of some sort is necessary if voting is to be done over the Web. The applet implements a large portion of the cryptographic functionality of the C++ cryptographic library, including encryption and the forming of proofs of knowledge.

Finally, the graphical client is written in C++ and makes use of the C++ cryptographic library. It provides a consistent graphical user interface written with Qt, and is used for administering elections, as well as performing authority functions. In addition to this, it allows read-only access to an Adder server, and can provide auditing features, so that interesting third parties can ensure the integrity of election procedures.

9.3 Outline of an Adder Procedure

An Adder procedure advances in much the same way as described in §7.4. An election procedure is initiated through an interface which allows the administrator to provide the candidate list and specify the eligible users. Such users are voters and authorities. An Adder election procedure progresses in the following manner. The authorities log into the system and participate in a protocol that results in the creation of a public encryption key for the system, and a unique private decryption key for each authority.

Next, each voter logs on, downloads the public key of the system, and uses that to encrypt the ballot, which is placed in an area of public storage specifically reserved for that voter. When the election is over, the server tallies the votes (using special encryption properties) and posts the encrypted result. Subsequently, the authorities provide some decoding information based on the encrypted result and their private keys. When enough such decoding information has been collected, the server combines the individual pieces to form the election result, which is then published. We note that Adder does not employ any user-to-user communication; instead, users of the system (in particular, the authorities) communicate indirectly through the public bulletin board that is maintained by the system. Voters are only active in one round throughout the system's operation (unless they are also playing the role of the authorities, which is possible). A graphical depiction is shown in Figure 9.1.

9.4 Limitations of Adder

As an Internet-based voting system, Adder is susceptible to a number of vulnerabilities and attacks. We list them below, along with possible solutions which will be implemented in future versions of the Adder system.

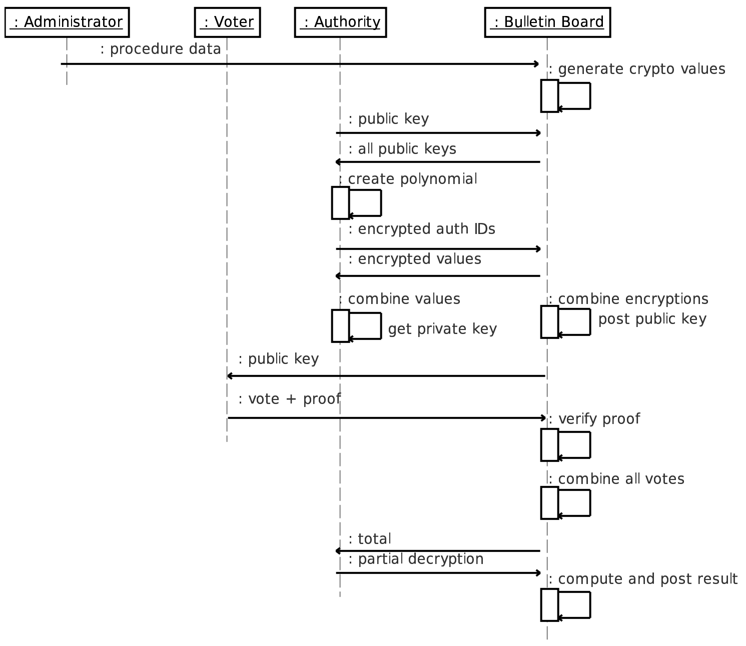

Figure 9.1: The stages of an Adder election procedure

1. Currently, the distributed key generation subsystem of Adder does not employ the verification methodology discussed previously. Thus, it is possible for authorities to cheat in this protocol, biasing the distribution of the public key, or disrupting the protocol entirely.

2. The MySQL database that Adder relies on may not provide an adequate level of protection against insider attacks.

3. Vote buying and coercion is a problem that remains difficult to solve with remote-based voting systems. It is possible that re-randomizing the ciphertext [BFP+01] before the ballot is cast could protect against voters' proving to others the accuracy of their ciphertexts.

4. As Adder does not use a voter-verifiable audit trail, it is hard for voters to be confident that there votes have been cast correctly. The danger with an audit trail is that it may make it easy for voters to prove how they voted.

5. The fact that Adder runs on standard PCs that have not been certified by election officials makes it possible that an Adder procedure could be disrupted by a virus. A possible work-around is to use a boot CD-ROM that contains a certified read-only operating system image.

In the future, we aim to implement more cryptographic tools to improve the security and functionality of Adder. These tools include the verification aspects of the Fouque-Stern distributed key generation, as well as support for a wider variety of election schemes.

9.5 Conclusion

We hope that Adder is a significant contribution in the area of publicly usable voting systems.

Chapter 10

Conclusion

We conclude this thesis by summarizing what we have discussed so far, and proposing several directions for future research in cryptographic e-voting.

10.1 Further Directions

This thesis has only touched upon a few aspects of electronic voting. The ideas we have covered here can be extended and investigated in many further ways.

10.1.1 Formalization of Security

The arguments for security given in this thesis are done in a rather ad hoc manner. Although the specific encryption schemes and zero-knowledge proofs used have formal security proofs, it has not been shown that the composition of all of these primitives yields a secure voting protocol. Future work in this area would involve formalizing the exact security definitions required by a homomorphic voting protocol, and proving the security of our protocol based on this definition. One promising approach to consider is the universal composability framework of Canetti [Can01]. Groth has initiated the investigation of analyzing voting schemes in this framework [Gro04].

10.1.2 Non-voting Applications

Although the protocols and primitives are described here in the context of voting, there may be other applications for this theory, e.g., general forms of distributed decision making where auditability is important. Another application is *contract-based computing*. This is an application to the World Wide Web, where users are asked to provide certain personal information (say, for demographic analysis), and they are presented with a privacy contract guaranteeing that their data is only used for aggregation.

10.1.3 Implementations

So far, there are very few implementations of cryptographically robust electronic voting systems. Of those, even fewer are available for public use and scrutiny. The Adder system is one such system which uses the homomorphic approach, and to this date, it remains the only publicly available system that does. More implementations of this theory would be a welcome contribution.

10.1.4 Experimental Data

In this thesis, we have focused exclusively on theory. It would also be useful to gather experimental data. Aspects that could be tested include scalability, resistance to denial-of-service and other network attacks, practical efficiency, and usability.

10.1.5 More Sophisticated Voting Methods

The voting scheme we have discussed uses the *plurality* election method. That is, the candidate with the most votes wins. Recall the earlier example where there are three candidates: A, B, and C. Candidate A received 40% of the votes and candidates B and C each receiveed 30%. Candidate A is selected as the winner, because it received the most votes, even though it did not receive the majority.

After reading the previous example, one might say there is a flaw in the plurality method, as the majority of voters actually voted *against* candidate A. To address this concern, we can use the *Borda* method of selecting a winner. In the Borda method, each voter submits as its ballot an ordered list of candidates. Various results have been proved about different voting schemes, and some are more suitable to certain types of elections than others. Expanding the cryptographic tools to work with varied elections schemes is a necessary task. Groth [Gro05] has provided examples of non-interactive zero-knowledge proofs based on various voting schemes.

It is also possible that different election outcomes may be desired. For example, in the current scheme, the outcome consists of the number of votes each candidate received. However, it may be desirable to only reveal the winner, or only the ranking of the candidates, rather that the exact number of votes.

10.2 Discussion

In this thesis, we have explored the idea of using homomorphic encryption to perform elections. We have hopefully presented a convincing argument that this approach solves many of the security problems that plague existing electronic voting techniques. With homomorphic encryption, voters are ensured that their individual ballots need not be decrypted, and yet can still be added together. The techniques of zero-knowledge proofs have been described, which prevent voters from "cheating." Finally the idea of distributed trust and threshold encryption provide a way for the trust in an election system to be distributed to a large set of trustees, so that no one player holds all of the information necessary to compromise the privacy of any individual voter.

The aforementioned goals of transparency, universal verifiability, privacy, and distributed trust have been achieved. The bulletin board model allows all communication to be performed in the open, so that any interested third party can perform an audit of the entire scheme.

We have introduced the Adder system, a free implementation of a homomorphic voting scheme. It is hoped that Adder will provide an opportunity for further experimentation with cryptographic voting.

Despite the successes achieved thus far, several challenges remain. Many of these challenges are of a non-technical nature—vote buying, voter coercion, inherent insecurities in computer networks—and are perhaps insurmountable with current technology. By exploring the limits of what cryptography can provide, and by pursuing a dialogue with those who actually conduct and participate in elections, we will hopefully reach an understanding of what aspects of technology are useful, and what remains to be explored.

Bibliography

[Adi06] Ben Adida. *Advances in Cryptographic Voting Systems*. PhD thesis, Massachusetts Institute of Technology, 2006.

[Ben87] Josh Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, Yale University, 1987.

[BFP$^+$01] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *Principles of Distributed Computing*, pages 274–283, 2001.

[BG02] Dan Boneh and Philippe Golle. Almost entirely correct mixing with applications to voting. In Vijay Atlury, editor, *Proceedings of the 9th ACM Conference on Computer and Communication Security (CCS-02)*, pages 68–77, New York, November 18–22 2002. ACM Press.

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on RSA encryption standard PKCS #1. In *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer Verlag, 1998.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[CDFT98] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer. RFC 2440: OpenPGP message format. Technical report, Network Working Group, November 1998.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo G. Desmedt, editor, *Advances in Cryptology—CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer-Verlag, 21–25 August 1994.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT '97*. Springer Verlag, 1997. Lecture Notes in Computer Science No. 1233.

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

[Cha04] David Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy*, 2(1):38–47, 2004.

[CP93] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *CRYPTO '92: Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, pages 89–105, London, UK, 1993. Springer-Verlag.

[DGS03] Ivan Damgård, Jens Groth, and Gorm Salomonsen. The theory and implementation of an electronic voting system. In D. Gritzalis, editor, *Secure Electronic Voting*, chapter 6, pages 77–99. Kluwer Academic Publishers, 2003.

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

[DHR$^+$98] S. Dusse, P. Hoffman, B. Ramsdell, L. Lundblade, and L. Repka. RFC 2311: S/MIME version 2 message specification. Technical report, Network Working Group, March 1998.

[DJ00] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Public Key Cryptography: 4th International Workshop on Practice and Theory in Public Key Cryptosystems*, volume 1992 of *Lecture Notes in Computer Science*, page 119. Springer Berlin/Heidelberg, 2000.

[DJ03] Ivan Damgård and Mads Jurik. A length-flexible threshold cryptosystem with applications. In *ACISP*, pages 350–364, 2003.

[Elg85] Taher Elgamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, 1985.

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *IEEE Symposium on Foundations of Computer Science*, pages 427–437. IEEE, 1987.

[FOO93] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *Proceedings of AUSCRYPT '92*, pages 244–251, 1993.

[FPS01] Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. In *FC '00: Proceedings of the 4th International Conference on Financial Cryptography*, pages 90–104, London, UK, 2001. Springer-Verlag.

[FRE] GNU.FREE: Heavy-Duty Internet Voting. *http://www.j-dom.org/users/re.html*.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Proc. CRYPTO'86*, LNCS-263, pages 186–194. Springer Verlag, 1986.

[FS90] U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 416–426, New York, NY, USA, 1990. ACM Press.

[FS01a] Pierre-Alain Fouque and Jacques Stern. One round threshold discrete-log key generation without private channels. In *PKC '01: Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography*, pages 300–316, London, UK, 2001. Springer-Verlag.

[FS01b] Jun Furukawa and Kazue Sako. An efficient scheme for proving a shuffle. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO ' 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 368–387. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 2001.

[GJKR99] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Lecture Notes in Computer Science*, 1592:295+, 1999.

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

[GMR85] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *STOC '85: Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 291–304, New York, NY, USA, 1985. ACM Press.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001.

[Gro04] Jens Groth. Evaluating security of voting schemes in the universal composability framework. In *Applied Cryptography and Network Security*, volume 3089 of *Lecture Notes in Computer Science*, pages 46–60. Springer Berlin/Heidelberg, 2004.

[Gro05] Jens Groth. Non-interactive zero-knowledge arguments for voting. In *Applied Cryptography and Network Security*, volume 3531/2005 of *Lecture Notes in Computer Science*, pages 467–482. Springer Berlin/Heidelberg, 2005.

[GZB$^+$02] Philippe Golle, Sheng Zhong, Dan Boneh, Markus Jakobsson, and Ari Juels. Optimistic mixing for exit-polls. In *ASIACRYPT: Advances in Cryptology – ASIACRYPT: International Conference on the Theory and Application of Cryptology*, volume 2501, pages 451–465, 2002.

[Hur05] Harri Hursti. Critical security issues with Diebold Optical Scan design. Technical report, Black Box Voting Project, July 2005. *http://www.blackboxvoting.org/BBVreport.pdf*.

[jK02] Kwang jo Kim. Killer application of PKI to Internet voting. In *IWAP 2002*. Springer Verlag, 2002. Lecture Notes in Computer Science No. 1233.

[JRSW04] D. Jefferson, A. Rubin, B. Simmons, and David Wagner. A security analysis of the secure electronic registration and voting experiment (SERVE). *http://servesecurityreport.org/*, 2004.

[Jur03] Mads J. Jurik. *Extensions to the Paillier Cryptosystem with Applications to Cryptographic Protocols*. PhD thesis, University of Aarhus, 2003.

[KKW06] Aggelos Kiayias, Michael Korman, and David Walluck. An Internet voting system supporting user privacy. In *ACSAC '06: Proceedings of the 22nd Annual Computer Security Applications Conference*, pages 165–174, Washington, DC, USA, 2006. IEEE Computer Society.

[KMR$^+$06] A. Kiayias, L. Michel, A. Russell, A. A. Shvartsman, M. Korman, A. See, N. Shashidhar, and D. Walluck. Security assessment of the Diebold Optical Scan voting terminal. Technical report, University of Connecticut VoTeR Center, October 2006. *http://voter.engr.uconn.edu/voter/Reports_files/uconn-report-os.pdf*.

[KSRW04] Tadayoshi Kohno, Adam Stubblefield, Aviel Rubin, and Dan Wallach. Analysis of an electronic voting system. In *IEEE Symposium on Security and Privacy*, May 2004.

[KSW] Chris Karlof, Naveen Sastry, and David Wagner. The promise of cryptographic voting protocols. *http://www.cs.berkeley.edu/∼daw/papers/cvop-unpub05.pdf*.

[Mao04] Wenbo Mao. *Modern Cryptography: Theory and Practice*. Prentice Hall PTR, 2004.

[Nef01] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In Pierangela Samarati, editor, *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 116–125, Philadelphia, PA, USA, November 2001. ACM Press.

[Nef03] C. Andrew Neff. Verifiable mixing (shuffling) of Elgamal pairs. *http://www.votehere.net/vhti/documentation/egshuf-2.0.3638.pdf*, December 2003.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *Advances in Cryptology – EUROCRYPT '99*, volume 1592 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 D. W. Davies, editor, *Advances in Cryptology—EUROCRYPT 91*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526. Springer-Verlag, 8–11 April 1991.

[PS00] Guillaume Poupard and Jacques Stern. Fair encryption of RSA keys. In *Advances in Cryptology – EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques*, volume 1807 of *Lecture Notes in Computer Science*, page 172. Springer Berlin/Heidelberg, 2000.

[QGAB89] Jean-Jacques Quisquater, Louis Guillou, Marie Annick, and Tom Berson. How to explain zero-knowledge protocols to your children. In *CRYPTO '89: Proceedings on Advances in cryptology*, pages 628–631, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[Riv01] Ronald L. Rivest. Electronic voting. In *Financial Cryptography '01*, volume 2339 of *LNCS*, pages 243–268. Springer-Verlag, 2001.

[RS92] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, pages 433–444, London, UK, 1992. Springer-Verlag.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO '89: Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, pages 239–252, London, UK, 1990. Springer-Verlag.

[Sch00] Bruce Schneier. Crypto-gram newsletter. Technical report, Counterpane Internet Security, Inc., December 2000. *http://www.schneier.com/crypto-gram-0012.html*.

[Sha69] Daniel Shanks. Class number, a theory of factorization, and genera. In *Proceedings of Symposia in Pure Mathematics*, volume 20, pages 415–440, 1969.

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

[Sho05] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2005.

[Sip97] Michael Sipser. *Introduction to the Theory of Computation*. MIT Press, 1997.

[Sys03] Diebold Election Systems. Checks and balances in elections equipment and procedures prevent alleged fraud scenarios. July 2003.

[Vor] Poorvi Vora. Citizen verified voting: An implementation of Chaum's voter verifiable scheme. Talk given at the DIMACS Workshop on Electronic Voting, Rutgers U., NJ, May 26-27, 2004.

Index

Colophon

This document was prepared using $\LaTeX\,2_\varepsilon$. The text was set in 11pt Charter, designed by Matthew Carter in 1987. The mathematics was set in the Math Design typeface designed to accompany Charter. The illustrations were produced with Xfig.