NTNU
Norwegian University of
Science and Technology

Applications of Paillier□s Cryptosystem.

Nina Pettersen

Master of Science in Physics and Mathematics
Submission date: August 2016
Supervisor: Kristian Gjøsteen, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Preface

This document is the Master thesis of Nina Pettersen, prepared during spring and summer 2016, as the final part of my Master of Science degree from the Norwegian University of Science and Technology. I would like to thank my supervisor Kristian Gjøsteen for all the guidance given, the good advices which always turned out to be true and the patience to let my understand details. I would also like to thank my parents for all the hours of watching my child while I had to study, and not the least for moral support.

Contents

List of Figures

1 Introduction

Voting is as old as society. The simplest and probably one of the earliest methods of voting is done by a "show of hands" which means raising one's hand if one agrees with the proposition put forth. Nowadays voting is usually associated with the election of governments, election of boards or similar elections.

Electronic voting has been a topic of great discussion the last couple of decades, and more and more elections are being converted from conventional methods to the electronic ones. There are several reasons for moving from conventional to electronic voting. One is the economic point of view – it is easy to see that making the election process electronic could cut down costs and save time. Another point of view is to achieve a higher voter turnout – one hopes that when voting is made more convenient considerably more voters will turn out.

When we are moving from conventional voting to electronic voting, we are trying to mimic conventional voting. This is important to make electronic voting understandable and trustworthy for both voters and authorities. But different countries implement elections in different ways. Electronic voting must take many different, and sometimes mutual excluding, considerations into account. Some of these considerations might not be an issue some years from now. When voters and authorities are used to electronic elections there might be new and completely revolutionary ways to put electronic voting into practice. But until this might happen, we will need to mimic conventional voting.

Money and convenience do not cover every aspect of the voting process. The most important requirement for a voting process is that the election result reflects the opinion of the authenticated voters. Amongst others, conventional voting has been designed to prevent vote coercion, and depending on the choice of how to cast a vote, several security issues arise. Two of the possibilities for the vote casting process are either voting from a

mobil platform (as a personal computer or a mobil phone) or voting electronically from a polling place. These two different approaches have very different impact when it comes to vote coercions. So when looking at the security of electronic voting it is possible to divide security issues into at least to categories: The security of the cryptographic voting protocol and the security of the practical implementation.

The scope of this thesis is to study a set of cryptographic voting protocols, where the only difference between them is the number of candidates one can vote for. To be able to do this we will need some knowledge of public key cryptography, both of the actual encryption schemes and on how we can analyze their security. In particular we need to study Paillier's cryptosystem [24] and a generalized version of this cryptosystem by Damård, Jurik and Nielsen [7]. The latter will be the base for the voting protocols. The practical implementation of the voting protocols is out of scope for this thesis. We will assume that the practical issues are solved adequately, and when a practical problem occurs which has direct relevans for our protocol, we will let a bulletin board solve it.

The outline of the thesis is as follows: In Chapter 2 we introduce the mathematics which will be needed for the rest of the thesis. It will require only a basic prior understanding of algebra and number theory. In Chapter 3 public key cryptography is introduced, and we describe some of the classic cryptosystems. In Chapter 4 we discuss the security of cryptosystems and give a formal definition of semantical security which will be of great interest for the next chapters. In Chapter 5 we introduce Paillier's cryptosystem and prove the security of it, and in Chapter 6 we introduce a generalization of it by Damgård, Jurik and Nielsen. This cryptosystem will be the base for Chapter 7, where we introduce electronic voting. We then describe the voting protocols and introduce the concepts of zero knowledge, non-interactive zero knowledge and threshold decryption, before we end the chapter with security analyzes of the voting protocols. In Chapter 8 we conclude the thesis with an informal discussion on how a particular security issue could be addressed, namely the trust we are giving the dealer.

2 Preliminaries and notation

We will start by looking at some basic results from algebra and number theory which will be used throughout this thesis. They are all well-known, so for the most of them we will not include proofs here. The interested reader can find the proofs in any undergraduate textbook in algebra and number theory.

2.1 Primes and Composites

We will start with a short introduction of primes, as they are essential to every cryptosystem discussed within this thesis.

Definition 2.1. Let a, b be integers with $0 < b \leq a$. Then $\exists$ unique k and t with $0 \leq r < b$ such that

$$a = kb + t,$$

where k is called the quotient and t is called the residue. If $t = 0$ we say that b *divides* a and denote it by $b \mid a$.

Definition 2.2. We divide the natural numbers into three categories:

- 1 is the only number with exactly one divisor, namely 1 itself.
- *Primes* are integers $p > 1$ with exactly two natural number divisors, namely 1 and p itself.
- *Composites* are integers $n > 1$ which are not primes, i.e. have more than one prime divisor.

PROPOSITION 2.3 (Fundamental theorem of arithmetic).
Every positive integer n can be expressed as a unique product of primes. That is:

$$n = \prod_{i=1}^{t} p_i^{k_i},$$

for some positive integers j, k_i, where p_i are primes with $p_1 < \cdots < p_t$.

DEFINITION 2.4. The *greatest common divisor* of two integers a and b with $ab \neq 0$, denoted $\gcd(a, b)$, is defined to be the largest positive integer which divides both a and b. We say that a and b are *relatively prime*, or *coprime*, if $\gcd(a, b) = 1$.

DEFINITION 2.5. The *least common multiple* of two integers a and b with $ab \neq 0$, denoted $\operatorname{lcm}(a, b)$, is defined to be the smallest positive integer which is divisible by both a and b.

PROPOSITION 2.6.
Let a, b be integers. Then

$$ab = \gcd(a, b) \operatorname{lcm}(a, b). \tag{2.1}$$

PROPOSITION 2.7 (Extended Euclidean algorithm).
Let a and b be positive integers. Then the equation

$$ax + by = \gcd(a, b)$$

always has a solution in integers x and y.

2.2 Groups

DEFINITION 2.8. A *group* G is an algebraic structure consisting of a set G and a binary operation which satisfy closure, associativity, identity and invertibility. We can also list the following for a group G:

- G is called an *abelian* group if the binary operation is commutative.
- The *order* of G is the number of elements in G, and is denoted $|G|$.
- The *order of an element* $a \in G$, is the smallest positive integer k such that $a^k = identity$. If such a k exists, we say that a has *finite order*, if not we say that a has *infinite order*.

- G is *cyclic* if it contains an element $g \in G$ which generate the entire group. That is, $G = \{g^k\}$. In this case, g is called a *generator* of G and we write $\langle g \rangle = G$.
- The order k of a finite cyclic group is the order of its generator, and we write $|\langle g \rangle| = |G| = k$.

We will now look at some properties for groups.

PROPOSITION 2.9.

- *Every group of prime order is cyclic.*
- *Every cyclic group is abelian.*
- *Let G be a finite group and let $a \in G$. Then $|\langle a \rangle|$ divides $|G|$.*

We will continue defining subgroups.

DEFINITION 2.10. Let G be a group and $H \subseteq G$. If H is closed under the group operation, and is itself a group with the induced group operation, then H is a subgroup of G.

PROPOSITION 2.11 (Lagrange's theorem).
Let H be a subgroup of a finite group G. Then $|H|$ divides $|G|$.

All the groups we will be studying in this thesis are abelian. We will then have the following important structure:

DEFINITION 2.12. Let H be a subgroup of an abelian group G. Then G/H is the *factor group* defined by

$$G/H = \{aH \mid a \in G\},$$

under the binary operation

$$(aH)(bH) = (ab)H.$$

At last we will take a look at mappings with some particular properties.

DEFINITION 2.13. Let G and G' be two groups with binary operations $*$ and $*'$ respectively. Let $\mu : G \to G'$, and consider the following two properties:

- For all $a, b \in G$

$$\mu(a * b) = \mu(a) *' \mu(b).$$

- The mapping μ is bijective.

If the the first condition holds, we have a group *homomorphism* between G and G'. If both conditions hold, we have a group *isomorphism*, denoted $G \simeq G'$.

2.3 Congruences

DEFINITION 2.14. Let a, b, n be integers with $n \neq 0$. Then

$$a \equiv b \pmod{n}$$

iff $a = b + kn$ for some integer k. We call this a *congruence*, and say that a *is congruent* b *modulo* n.

The congruence relation is an equivalence relation, and thus partitions the integers into disjoint *equivalence classes*, such that two integers are congruent modulo n if and only if they lie in the same class.

DEFINITION 2.15. The *residue class* of a modulo n, denoted $\langle a \rangle$, is defined by

$$\langle a \rangle = \{b \in \mathbb{Z} \mid a \equiv b \pmod{n}\} = a + n\mathbb{Z} = a + \langle n \rangle.$$

The *integers modulo* n is the set of all residue classes modulo n given by

$$\mathbb{Z}_n = \{0 + \langle n \rangle, 1 + \langle n \rangle, \ldots, (n-i) + \langle n \rangle\},$$

where the group operations addition and multiplication are inherited from $\mathbb{Z}$ and where $\{0, 1, \ldots, (n-1)\}$ are called the *residues* mod$n$. When $n \neq 0$, then $|\mathbb{Z}_n| = n$.

It is possible to think of $\mathbb{Z}_n$ as the set of residues classes of $\mathbb{Z}$ modulo n, or the set of residues modulo n. Without loss of generality, we will denote an element in $\mathbb{Z}_n$ both as a or $a + \langle n \rangle$, depending on the context.

It is easy to see that $\mathbb{Z}_n$ is an abelian group under addition, but it is not a group under multiplication.

PROPOSITION 2.16.
Let a and n be integers. Then for some integer b,

$$ab \equiv 1 \pmod{n} \iff \gcd(a, n) = 1.$$

If such an integer b exists, it is unique, and we say that b is the multiplicative inverse *of a modulo n.*

Since every residue $a \bmod n$ with $\gcd(a, n) = 1$ has an inverse, they form the following abelian group under multiplication, called the *multiplicative group of integers modulo n*:

$$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \gcd(a, n) = 1\}.$$

We will now have a look at the group $\mathbb{Z}_p$, where p is a prime. As above, this is a group under addition. But since p is prime, then $\gcd(a, p) = 1 \ \forall \ a \in \mathbb{Z}_p$, and hence it is also a group under multiplication. A set that is a group under both addition and multiplication is called a *field*.

We will end this section with the Chinese remainder theorem. It is a result about congruences in number theory and can be generalized to abstract algebra. We will include a proof, and then let the generalization follow as a corollary.

PROPOSITION 2.17 (Chinese remainder theorem).
Let $n_1, n_2, \ldots, n_t$ be positive integers such that for all i, j with $i \neq j$, we have that $\gcd(n_i, n_j) = 1$, and let $n = \prod_{i=1}^{t} n_i$. Then the system of linear congruences*,*

$$\begin{aligned} a &\equiv b_1 \pmod{n_1} \\ a &\equiv b_2 \pmod{n_2} \\ &\cdots \\ a &\equiv b_t \pmod{n_t} \end{aligned}$$

has a simultaneous solution which is unique modulo n. This is called the Chinese remainder theorem*, abbreviated* CRT*.*

Proof. We will first construct a solution a and then show that it is unique. So let $\tilde{n}_j = \frac{n}{n_j}$ for $1 \leq j \leq t$. That is,

$$\tilde{n}_j = n_1 n_2 \cdots n_{j-1} n_{j+1} \cdots n_t.$$

Since $\gcd(n_i, n_j) = 1$ for each $i \neq j$, we get $\gcd(\tilde{n}_j, n_j) = 1$. From Proposition 2.16 we then have that the equation $\tilde{n}_j x \equiv 1 \pmod{n_j}$ has a unique solution modulo n_j. We will call this solution a_j. This leads to the following two equations:

$$\tilde{n}_i a_i \equiv \begin{cases} 1 \pmod{n_j} & \text{for } i = j, \\ 0 \pmod{n_j} & \text{for } i \neq j. \end{cases}$$

We now have the solution a to our system of linear congruences as follows:

$$a = b_1\tilde{n}_1 a_1 + \cdots + b_t\tilde{n}_t a_t,$$

and from the arguments above we know that a satisfies all the congruences. That is,

$$a \equiv b_j\tilde{n}_j a_j \equiv b_j \pmod{n_j}.$$

We know have to show that a is unique modulo n. So suppose a' is another solution to the linear system. Since a_j is the unique solution for each congruence separately, we have that for each $1 \leq j \leq t$,

$$\begin{gathered} a' \equiv a \pmod{n_j} \\ \Updownarrow \\ a' - a \equiv 0 \pmod{n_j} \\ \Updownarrow \\ n_j \mid a' - a. \end{gathered}$$

Since each of the moduli are pairwise relatively prime, this leads to

$$\begin{gathered} \prod_{i=1}^{t} n_i \mid (a' - a) \\ \Updownarrow \\ a' \equiv a \pmod{n}. \end{gathered}$$

□

The next corollary follows directly from CRT by the two mappings

$$\begin{aligned} \mu \ : \ \mathbb{Z}_n &\longrightarrow \ \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_t} \\ x \bmod n &\longmapsto \ (x \bmod n_1, x \bmod n_2, \cdots, x \bmod n_t), \end{aligned}$$

and

$$\begin{aligned} \mu^* \ : \ \mathbb{Z}_n^* &\longrightarrow \ \mathbb{Z}_{n_1}^* \times \cdots \times \mathbb{Z}_{n_t}^* \\ x \bmod n &\longmapsto \ (x \bmod n_1, x \bmod n_2, \cdots, x \bmod n_t). \end{aligned}$$

It is easy to show by CRT that both the mappings are isomorphisms.

Corollary 2.17.1. *Let $n_1, \ldots, n_t$ be positive integers such that for all i, j with $i \neq j$, $\gcd(n_i, n_j) = 1$, and let $n = \prod_{i=1}^{t} n_i$. Then*

$$\mathbb{Z}_n \simeq \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_t}, \text{ and}$$
$$\mathbb{Z}_n^* \simeq \mathbb{Z}_{n_1}^* \times \cdots \times \mathbb{Z}_{n_t}^*.$$

2.4 Fermat, Euler and Carmichael

This section provides some important definitions and results for this thesis, and all of them will be followed by proofs. We will start with Euler's phi function.

Definition 2.18. *Euler's phi function*, also known as *Euler's totient function*, is a function $\varphi(n)$ that assigns to n the number of integers between 1 and n that are relatively prime to n. That is,

$$\varphi(n) = |\mathbb{Z}_n^*|.$$

Proposition 2.19. *Let p be a prime. Then*

$$\varphi(p) = p - 1.$$

Proof. The natural numbers less then a prime p are all relatively prime to p. □

Proposition 2.20. *Let $\tilde{n}, n$ be positive integers such that $\gcd(\tilde{n}, n) = 1$. Then we have*

$$\varphi(\tilde{n}n) = \varphi(\tilde{n})\varphi(n).$$

Proof. This follows from CRT by $\mathbb{Z}_{\tilde{n}n}^* \simeq \mathbb{Z}_{\tilde{n}}^* \times \mathbb{Z}_n^*$. □

Proposition 2.21. *Let p^k be a prime power. Then*

$$\varphi(p^k) = p^k \left(1 - \frac{1}{p}\right) = p^{k-1}(p-1).$$

Proof. We have that

$$\varphi(p^k) = \left| \left\{ a \;\middle|\; 1 \leq a < p^k \ \wedge \ \gcd(p^k, a) = 1 \right\} \right|.$$

Since $\gcd(p^k, p^i) = p^i$ for $0 \leq i \leq k$, we get $\gcd(p^k, a) \neq 1 \Leftrightarrow a = tp$ where $1 \leq t \leq p^{k-1}$, since $p^{k-1}p = p^k$. There are p^{k-1} such numbers. This leads to the following:

$$\varphi(p^k) = p^k - p^{k-1} = p^k \left(1 - \frac{1}{p}\right) = p^{k-1}(p-1).$$

□

PROPOSITION 2.22. *Let $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ be the unique prime factorization of n, with $p_1 < \cdots < p_k$. Then*

$$\varphi(n) = \varphi\left(p_1^{e_1}\right) \varphi\left(p_2^{e_2}\right) \cdots \varphi\left(p_k^{e_k}\right) = n \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right).$$

Proof. Since $\gcd(p_i, p_j) = 1$ for all $1 \leq i, j \leq k$ with $i \neq j$, we get

$$\varphi(n) = \varphi(p_1^{e_1}) \cdots \varphi(p_k^{e_k}) = p_1^{e_1} \left(1 - \frac{1}{p_1}\right) \cdots p_k^{e_k} \left(1 - \frac{1}{p_k}\right) = n \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right).$$

□

PROPOSITION 2.23 (Euler's theorem).
If $a, n \in \mathbb{Z}$ with $\gcd(a, n) = 1$, then

$$a^{\varphi(n)} \equiv 1 \pmod{n}.$$

Proof. We have $\gcd(a, n) = 1$ when $a \in \mathbb{Z}_n^*$. From proposition 2.9 we then have $t = |\langle a \rangle| \mid |\mathbb{Z}_n^*|$ which means that $\exists\, k \in \mathbb{Z}$ such that $kt = |\mathbb{Z}_n^*| = \varphi(n)$. Then

$$a^{\varphi(n)} \equiv a^{kt} \bmod n \equiv 1^k \bmod n \equiv 1 \bmod n.$$

□

If we let n be a prime in Euler's theorem, we get Fermat's little theorem (stated below). This might seem strange, but Fermat published this theorem in 1640 whereas Euler in 1736 was the first to publish a proof of the theorem. Euler himself found a generalization of Fermat's theorem in the same year, which is Euler's theorem above.

COROLLARY 2.23.1 (Fermat's little theorem)**.** *If $a \in \mathbb{Z}$ and p is a prime such that $\gcd(a, p) = 1$, then*

$$a^{p-1} \equiv 1 \pmod{p}.$$

Proposition 2.24. *Let p, q be primes, $n = pq$ and a, k integers with $0 \leq a < n$. We then have*

$$a(a^{k\varphi(n)}) \equiv a \pmod{n}.$$

Proof. Since $a < n$ and $n = pq$ have three possible cases:

$\gcd(a, n) = 1$: From Euler's theorem we have $a^{\varphi(n)} \equiv 1 \pmod{n}$, so $a(a^{k\varphi(n)}) \equiv a(1^k) \equiv a \pmod{n}$.

$\gcd(a, n) = p$: 1. $\gcd(a, p) = p$, so by triviality $a \equiv 0 \pmod{p}$ and $a(a^{k\varphi(n)}) \equiv 0 \pmod{p}$ which leads to $a(a^{k\varphi(n)}) \equiv a \pmod{p}$.

2. $\gcd(a, q) = 1$ so by Euler we get $a^{\varphi(q)} \equiv 1 \pmod{q}$ which leads to $a(a^{k\varphi(n)}) = a(a^{k\varphi(p)\varphi(q)}) \equiv a(1^{k\varphi(p)}) \equiv a \pmod{q}$.

1.+2.: We have $a(a^{k\varphi(n)}) \equiv a \pmod{p}$ and $a(a^{k\varphi(n)}) \equiv a \pmod{q}$, so by CRT we get $a(a^{k\varphi(n)}) \equiv a \pmod{n}$.

$\gcd(a, n) = q$: We can make the same argument as above, only switching p and q.

□

We will now take a look at Carmichael's function.

Definition 2.25. The *Carmichael function* is the smallest positive integer $\lambda(n)$ such that

$$a^{\lambda(n)} \equiv 1 \pmod{n},$$

for all integers a with $\gcd(a, n) = 1$.

While Euler's phi function defines the order of the multiplicative group of integers, Carmichael's function defines maximum order of the elements in the group. If the multiplicative group is cyclic, then there will be an element which generates the group and hence has maximum order. In this case Euler's phi function and Carmichael's function will have the same value. In the case where no element generates the multiplicative group, and since the order of an element divides the group order, the value of Carmichael's function will be less then Euler's phi function.

Proposition 2.26. *Let p be a prime. Then*

$$\lambda(p) = \varphi(p) = (p - 1).$$

Proof. Follows from $\mathbb{Z}_p$ being a cyclic group and the discussion above. □

PROPOSITION 2.27. *Let $\tilde{n}, n$ be integers with $\gcd(\tilde{n}, n) = 1$. Then*

$$\lambda(\tilde{n}n) = \operatorname{lcm}(\lambda(\tilde{n}), \lambda(n)).$$

In particular, if p, q are two distinct odd primes and $n = pq$, then

$$\lambda(n) = \operatorname{lcm}(\lambda(p), \lambda(q)) = \operatorname{lcm}(p-1, q-1).$$

Proof. From CRT we know that for $a \in \mathbb{Z}^*_{\tilde{n}n}$ we have

$$a^k \equiv 1 \pmod{\tilde{n}n} \iff a^k \equiv 1 \pmod{\tilde{n}} \wedge a^k \equiv 1 \pmod{n}.$$

So k must be a multiple of $\lambda(\tilde{n})$ and $\lambda(n)$, and the smallest such is $\operatorname{lcm}(\lambda(\tilde{n}), \lambda(n))$. The second part follows from the first part. □

PROPOSITION 2.28. *Let p, q be two distinct odd primes, let $n = pq$ and let $1 \leq s$ be an integer. Then For each $a \in \mathbb{Z}^*_{n^{s+1}}$ we have*

$$a^{\lambda(n)} \equiv 1 \pmod{n}$$
$$a^{n^s \lambda(n)} \equiv 1 \pmod{n^{s+1}}.$$

Proof. The first part follows from the second part by letting $s = 0$, so we only need to prove the second part. So let $a \in \mathbb{Z}^*_{n^{s+1}}$. We then get

$$a^{n^s \lambda(n)} \equiv \left(a^{\varphi(p^{s+1})}\right)^{q^s \frac{\lambda(n)}{\lambda(p)}} \equiv 1 \pmod{p^{s+1}}, \text{ and}$$
$$a^{n^s \lambda(n)} \equiv \left(a^{\varphi(q^{s+1})}\right)^{p^s \frac{\lambda(n)}{\lambda(q)}} \equiv 1 \pmod{q^{s+1}},$$

and since $\gcd(p^{s+1}, q^{s+1}) = 1$ we can combine them by CRT to get

$$a^{n^s \lambda(n)} \equiv 1 \pmod{n^{s+1}}.$$

□

2.5 Notation

The notation for the rest of this thesis is stated in Figure 2.1. Notice that when p, q are large and randomly chosen, n will be admissible except with negligible probability. If p, q is of equal length, this will be guaranteed.

- p, q will denote two distinct large primes.
- n is a composite such that $n = pq$.
- n is called *admissible* if $n = pq$ and $\gcd(n, \varphi(n)) = 1$.
- $\mathbb{Z}_n$ is the additive group of integers modulo n.
- $\mathbb{Z}_n^*$ is the multiplicative group of integers modulo n.
- $\varphi(n)$ is Euler's phi function.
- $\lambda(n)$ is Carmichael's function.

Figure 2.1: The notation for the rest of this thesis.

3 | Public Key Cryptography

The most basic problem in cryptography is the following: Alice wants to send a secret message to Bob. But an eavesdropper, Eve, wants to know the secret message, see Figure 3.1. Eve can intercept the message Alice and Bob send to each other and can also tamper with it. Alice and Bob want to be assure that Eve do not tamper with the message. They both want *integrity*. And Alice does not want Eve to eavesdrop on her messages to Bob. She wants *confidentiality* [14].

To solve this problem we can lock the message using some sort of a key, and then open it again using the same or another key. If we lock and open the message with the same key, it is called *symmetric cryptography*, since the keys are symmetric. If we use two different keys, it is called *public key cryptography*, because we have one public key to lock the message and one private key to open it. Let us illustrate this with an example.

Alice wants to send a secret message to Bob, and the only way they can communicate is via postal mail. Unfortunately, the postman Eve is reading the mail Alice sends. So Alice needs to find a way to send messages to Bob without Eve, or anybody else, being able to read them. Of course she could put the message in a locked box, and give one key to Bob while keeping the other to herself, see Figure 3.2. This would be a symmetric key situation. The problem is that Alice and Bob can not meet to share keys, as they live too far apart. So they come up with the following solution: Bob buys a padlock and a matching key, sends the unlocked padlock to Alice and keeps the key himself, see Figure 3.3a. Now Alice can lock the message inside the box simply by locking the padlock. Then she can mail the locked box to Bob, knowing that Eve can not read the message as she has no way of opening the padlock, and when Bob receives the locked box, he can open it with his private key, see Figure 3.3b. This is a public key situation, and we can think of the padlock as the public key and Bob's key as the private key.

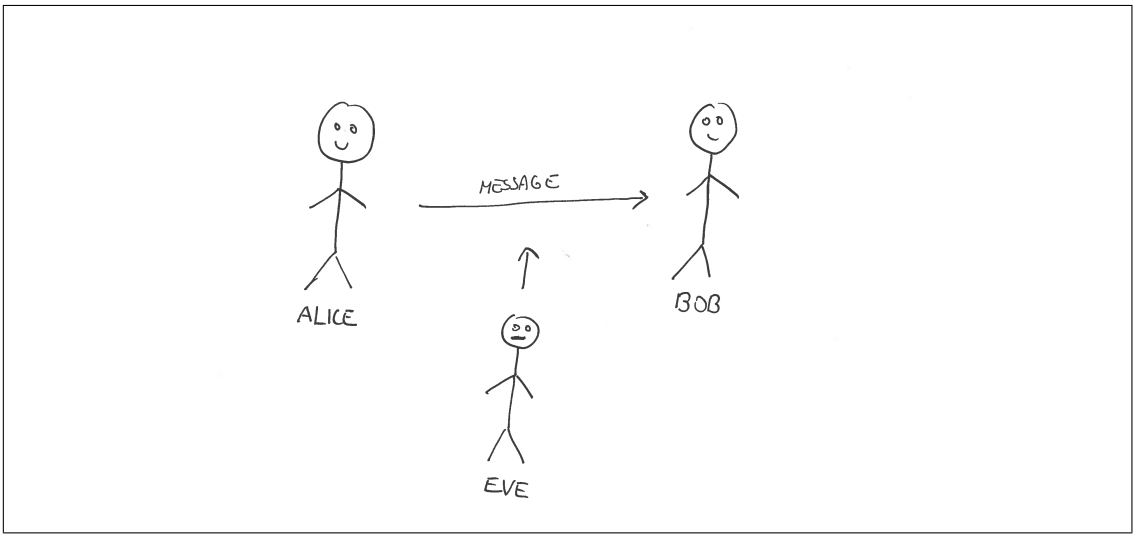


Figure 3.1: Alice wants to send a secret message to Bob, but Eve is eavesdropping.

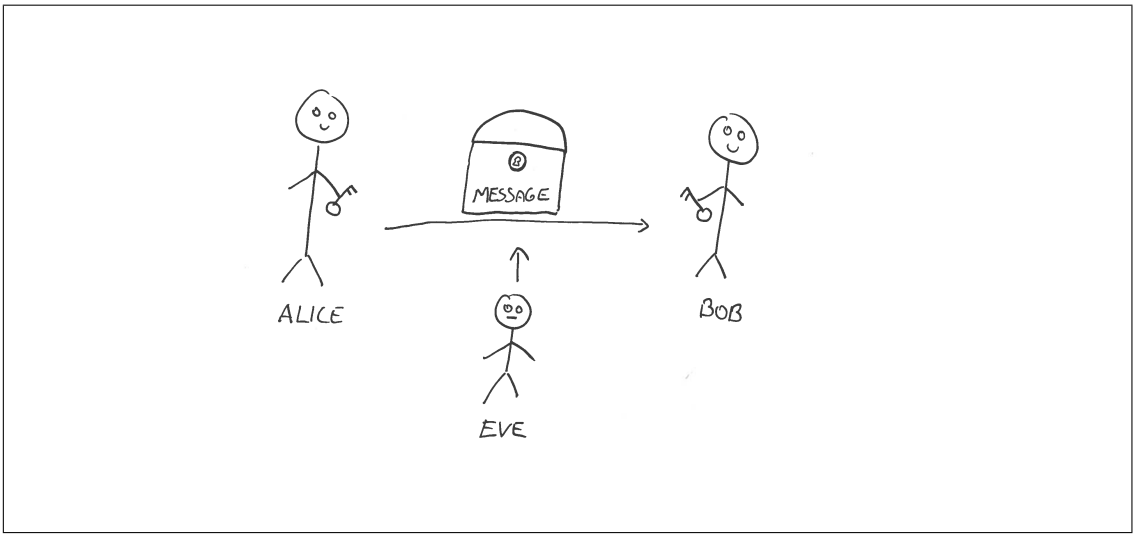
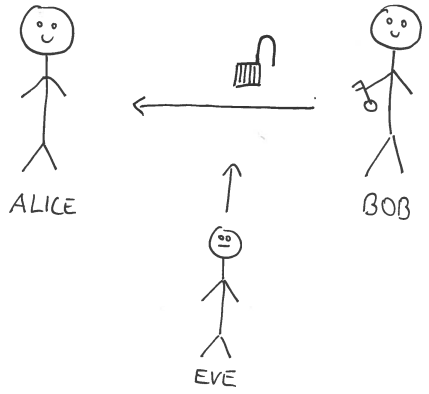
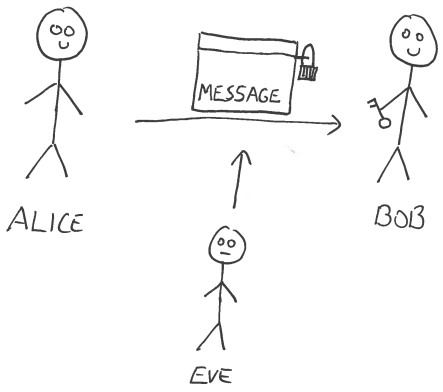


Figure 3.2: A figurative example of symmetric cryptography.



(a) The open padlock is Bob's public key.



(b) Bob can open the locked message with his private key.

Figure 3.3: A figurative example of public key cryptotography.

To draw some parallells to the language we use in cryptography, the locking process is called *encrypting* a message, while the unlocking process is called *decrypting* a message. The public key is also called the *encryption key*, and the private key is also called the *decryption key* . The parallell to the padlock is what we call a *trapdoor one-way function*. This is a function which is feasible (easy) to compute, but infeasible (hard) to inverse, unless you have some secret information called the trapdoor. The terms "feasible" and "infeasible" will be precisely defined in Section 4.2, but for now we can think of "feasible" as "sufficiently easy", and "infeasible" as "sufficiently hard". For an encryption scheme to have any sense at all, somebody must be able to decrypt the encrypted message. It is here the trapdoor comes into play. The trapdoor is some piece of information which makes the one-way function feasible to inverse, usually the decryption key. Or, not completely. We do not need the complete inversion, we only need access to the message. See Figure 3.4 for an illustration of public key encryption.

The example above is very limited and do not describe public key cryptography completely. As an example, all but one of the cryptosystems we will be studying in this thesis have a homomorphic property which will be defined below. And this property compromises the integrity. We will discuss this in further details in Section **??**.

The great breakthrough for public key cryptography came in 1976 with Diffie and Hellman's seminal article "New Directions in Cryptography" [8], and the first sentence of the article is on the point: "We stand today on the brink of a revolution in cryptography". The revolution they talked about, was a method to establish a shared value by using a trapdoor one-way function. This method is described in the Diffie-Hellmann protocol, see Section 3.1.

Another prominent breakthrough at about the same time as Diffie and Hellman, was the RSA cryptosystem introduced by Rivest, Shamir and Adleman in 1977 [27]. Although other had similar ideas as Diffie-Hellman and RSA already from the early 1970's, cryptography was often considered classified information, and so many of these publications were not declassified until late 1990's. RSA is related to the second part of the example above: It is based on a one-way trapdoor function, and uses a public encryption key to encrypt a message and then a private decryption key to open it. We will study RSA in Section 3.3.

We have used Alice, Bob and Eve to denote the communicating parties and the attacker. This is common practice in cryptography, at least at a basic level when we are trying to

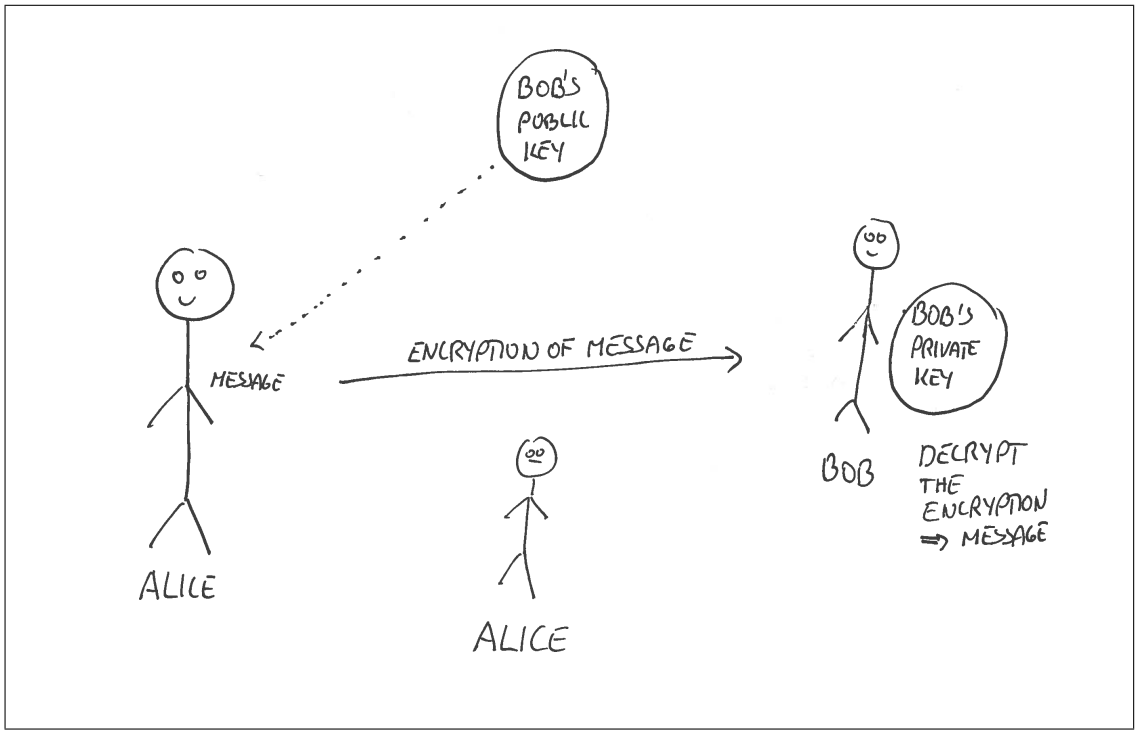


Figure 3.4: An illustration of public key encryption.

describe the concepts. But it is important to keep in mind that the parties are usually not humans. We could just as well be describing a communication being carried out between two autonomous machines. In fact, the need of machine-to-machine communication was one of the starting points of public key cryptography [1].

3.1 Diffie-Hellman Protocol

As mentioned above, Diffie-Hellmann is a *cryptographic protocol* for establishing a shared secret. A cryptographic protocol can be thought of as a conversation between some parties, in this case Alice and Bob. The shared secret can be a symmetric key.

The protocol utilizes a classic example of a one-way function, namely discrete exponentiations versus discrete logarithms [14].

Definition 3.1. The *discrete logarithm* of x to the base g, is the smallest integer $a \neq 0$ such that

$$x = g^a.$$

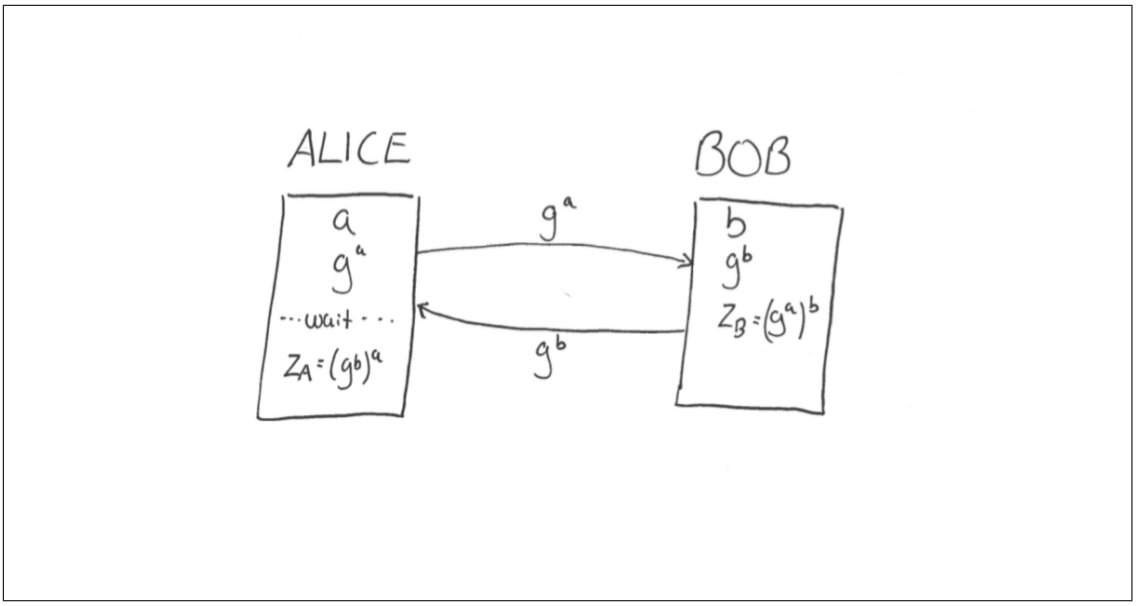


Figure 3.5: An illustration of Diffie-Hellman protocol, based on a similar illustration in [14].

We write

$$\log_g x = a.$$

The *discrete logarithm problem* in a cyclic group G is to find the discrete logarithm of x to the base g, when x has been chosen uniformly at random from the group.

So the feasible computations is $x = g^a$, and the infeasible computation is $log_g x = a$. Notice that g in this section is not to be confused with g in Chapter 5. To get an idea of what is happening, consider the value g^{ab} as their final shared secret. They can establish this as follows (see Figure 3.5 for an illustration):

1. Alice chooses a uniformly at random, computes g^a and sends the value to Bob.
2. Bob chooses b uniformly at random, computes g^b and sends the value to Alice.
3. Now Alice can compute $(g^b)^a$ and Bob can compute $(g^a)^b$, and they both share a secret.

The reason for this being a secret, is that Eve only sees g^a and g^b. But for computing g^{ab} she needs either a or b. Computing one of these from g^a or g^b, as they are chosen uniformly at random, would be the same as breaking the discrete logarithm problem,

Diffie-Hellman protocol

Common input: A finite cyclic group G, the order of the group n and a generator g.

1. Alice $\rightarrow$ Bob:
   (a) Chooses a number a uniformly at random from the set $\{0, 1, \ldots, n-1\}$.
   (b) Computes $x = g^a \bmod n$.
   (c) Sends x to Bob.
2. Bob $\rightarrow$ Alice:
   (a) Chooses a number b uniformly at random from the set $\{0, 1, \ldots, n-1\}$.
   (b) Computes $y = g^b \bmod n$.
   (c) Computes $z_B = x^b \bmod n$.
   (d) Sends y to Alice.
3. Alice: Computes $z_A = y^a \bmod n$.

Figure 3.6: The Diffie-Hellman protocol.

which is thought to be an infeasible computation. We will discuss this in further details in Chapter 4.

We will now formalize the Diffie-Hellman protocol, see Figure 3.6 [14].

For cryptographic protocols we, amongst other requirements, consider *completeness*. Completeness means that the result of the protocol has to be consistent. For Diffie-Hellman it is trivial to show completeness. It is enough noticing that

$$z_A \equiv y^a \equiv (g^b)^a \equiv (g^a)^b \equiv x^b \equiv z_B \pmod{n},$$

so Alice and Bob have established a shared secret.

Even though Diffie-Hellman is only a protocol, it is possible to make a cryptosystem based on Diffie-Hellman. This cryptosystem is called ElGamal and will be defined in Section **??**.

We must mention one more thing concerning Diffie-Hellman protocol. The group to be used has to be chosen wisely. Alice and Bob want to spend as little effort as possible to establish the shared secret, both with respect to computation and communication.

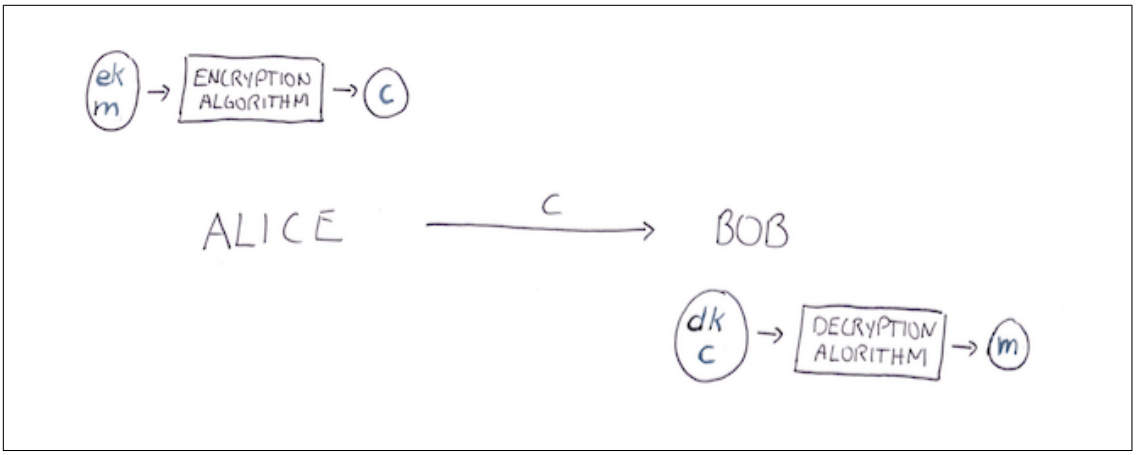


Figure 3.7: An illustration of a public key encryption scheme.

Computing the group elements should be reasonably fast, and group elements should have a reasonably compact representation. And solving the discrete logarithm problem must be an infeasible computation. In particular we have to take into notion the *Pohlig-Hellman algorithm*, which is an algorithm for computing discrete logarithms in multiplicative groups, whose orders factors completely into small prime numbers. For further details see [14].

3.2 Public Key Encryption Schemes

We will now formalize what we mean when we are talking about a public key cryptosystem, or equivalently, a public key encryption scheme, see Figure 3.7 for an illustration.

DEFINITION 3.2. An *algorithm* is a set of step-by-step rules which takes an input and then produce an output according to these rules.

DEFINITION 3.3. A *public key encryption scheme* consists of the three algorithms $\mathcal{K}$, $\mathcal{E}$ and $\mathcal{D}$ defined by [14]:

- The *key generation* algorithm $\mathcal{E}$ takes no input and outputs an *encryption key* ek and a *decryption key* dk. To each encryption key ek there is an associated set of messages, also called *plaintexts*.
- The *encryption* algorithm $\mathcal{E}$ takes as input an encryption key ek and a message m in the set of plaintexts, and outputs a ciphertext c.

- The *decryption* algorithm $\mathcal{D}$ takes as input a decryption key dk and a ciphertext c and outputs either a message m or the special symbol $\perp$ indicating decryption failure.

Definition 3.4. A public key encryption scheme is *correct* if for any key pair (ek, dk) output by $\mathcal{K}$ and any message m in the set of plaintexts,

$$\mathcal{D}\left(dk, \mathcal{E}(ek, m)\right) = m, \tag{3.1}$$

and this computation is solvable in probabilistic polynomial time.

Probabilistic polynomial time will be defined in Section 4.2, but for now we can say that 3.1 must be a feasible computation. There is a very good reason for this: For the cryptosystem to have any practical implementation it has to be sufficiently easy to both encrypt messages and decrypt ciphertexts. For the cryptosystems we will study in this thesis, it goes without saying that the 3.1 is solvable in probabilistic polynomial time.

We will study the security of public key cryptosystems in Chapter 4, but for now we can just notice that at least it has to be an infeasible computation to decrypt a ciphertext without knowing the private decryption key.

We are now ready to leave the world of Alice, Bob and Eve, and will not bother defining the parties in our cryptosystem anymore. It is enough knowing that someone or something encrypts messages and decrypts ciphertexts, and against the system there are adversaries.

3.3 RSA

We will start by giving a formal description of a cryptosystem called Textbook RSA, see Figure 3.8 [14].

It is easy to see that

$$\mathcal{D}\left(dk, \mathcal{E}(ek, m)\right) = \mathcal{D}\left(dk, m^e \bmod n\right) = (m^e)^d \bmod n = m^{ed \bmod \lambda(n)} = m.$$

We can divide cryptosystems into two cathegories:

Textbook RSA

Textbook RSA is given by $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ as follows:

Key generation algorithm $\mathcal{K}$:

(1) Choose two large primes p, q.

(2) Compute $n = pq$.

(3) Compute $\lambda(n) = \text{lcm}(p-1, q-1)$.

(4) Choose e and find d such that $ed \equiv 1 \pmod{\lambda(n)}$.

Output: $ek = (n, e)$ and $dk = (n, d)$.

Encryption algorithm $\mathcal{E}$:

Input: $ek = (n, e)$ and $m \in \mathbb{Z}_n$.

1. Compute $c = m^e \bmod n$.

Output: $c \in \mathbb{Z}_n$.

Decryption algorithm $\mathcal{D}$:

Input: $dk = (n, d)$ and $c \in \mathbb{Z}_n$.

1. Compute $m = c^d \bmod n$.

Output: $m \in \mathbb{Z}_n$.

Figure 3.8: A formal description of textbook RSA.

Deterministic cryptosystems are cryptosystems in which the encryption algorithm uses an injective mapping $m \mapsto c$ to encrypt messages. RSA as first introduced by Rivest, Shamir and Adleman in 1977 [27], was a deterministic cryptosystem. This cryptosystem is what we now call Textbook RSA, and it is easy to see that $f(m, e) = m^e$ is an injective mapping.

Probabilistic cryptosystems was introduced by Goldwasser and Micali in 1984 [18]. These cryptosystems use some kind of random padding in the encryption algorithm, so that the mapping $m \mapsto c$ is not injective. About every public key cryptosystem is probabilistic, as this is a crucial requirement for the security of the system. This will be further discussed in Section 4.6.

There are several ways to turn Textbook RSA into a probabilistic encryption scheme. One of the methods is to make a random padding of the message by adding some random bits to the message. Another method is to combine RSA with a symmetric cryptosystem, as is also done with Diffie-Hellman protocol to get ElGamal cryptosystem defined below. We will not discuss this further here, but refer the interested reader to [14].

There are several other security issues to take into account when it comes to RSA One of them is the choice of p, q. We will not discuss the security issues of RSA in further details here, but refer the interested reader to [14].

3.4 Homomorphic Encryption

A cryptosystem is said to be *homomorphic* if it has the following property:

Definition 3.5. Let $\mathcal{E}$ be an encryption algorithm and let $\otimes$ and $\oplus$ be the group operations in the corresponding group of ciphertexts and plaintexts respectively. If

$$\mathcal{E}(m) \otimes \mathcal{E}(m') = \mathcal{E}(m \oplus m'),$$

for all distinct m, m' in the set of plaintexts, we say that $\mathcal{E}_D$ has a *homomorphic property.*

Textbook RSA is obviously a homomorphic cryptosystem since

$$\mathcal{E}(ek, m)\mathcal{E}(ek, m') = m^e m'^e = (mm')^e = \mathcal{E}(ek, mm').$$

Pailler's cryptosystem and the generalized version by Damgård, Jurik and Nielsen which we will study in Chapter 5 and 6 both have a homomorphic property. This property will be crucial for the application to electronic voting in Chapter 7.

3.5 ElGamal

ElGamal cryptosystem was introduced in 1985 by ElGamal [9]. It is based on the Diffie-Hellman protocol and a very simple symmetric cryptosystem called *Shift cipher*. We will not go into the theory of symmetric cryptosystems here, so for Shift cipher we will just say that for a symmetric key z we encrypt a message m by

$$\mathcal{E}_{sym}(z, m) = mz = w,$$

and decrypt it by

$$\mathcal{D}_{sym}(z, w) = mz^{-1} = m.$$

To establish the key z, we do the following steps based on Diffie-Hellman:

1. The key generation establishes $ek = g^a = y$ and $dk = a$.
2. The encryption algorithm esablishes $x = g^b$ and symmetric key $z = y^b$.
3. The decryption algorithm esablishes symmetric key $z = x^a$.

ElGamal is formalized in Figure 3.9 [14]. Notice that also in this section, g is not to be confused with g in Chapter 5.

ElGamal is obviously correct since:

$$z = y^b = g^{ab} = x^a,$$

and

$$\mathcal{D}\left(dk, \mathcal{E}(ek, m)\right) = \mathcal{D}\left(dk, mz\right) = mzz^{-1},$$

where z^{-1} exists since G is cyclic.

There are several other security issues to take into account when it comes to ElGamal. One of them is the choice of the group G. We will not discuss the security issues of ElGamal in further details here, but refer the interested reader to [14].

ElGamal encryption scheme

ElGamal encryption scheme is given by $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ as follows:

Key generation algorithm $\mathcal{K}$:

(1) Choose a finite cyclic group G with order n, and a generator g.

(2) Choose uniformly at random $a \in \mathbb{Z}_n$.

(3) Compute $y = g^a \bmod n$.

Output: $ek = (n, y)$ and $dk = (n, a)$.

Encryption algorithm $\mathcal{E}$:

Input: $ek = (n, y)$ and $m \in G$.

1. Choose uniformly at random random $b \in \mathbb{Z}_n$.
2. Compute $x = g^b \bmod n$.
3. Compute $z = y^b \bmod n$.
4. Encrypt $\mathcal{E}_{sym}(z, m) = mz \bmod n = w$.

Output: $c = (x, w)$.

Decryption algorithm $\mathcal{D}$:

Input: $dk = (n, a)$ and $c = (x, w)$.

1. Compute $z = x^a \bmod n$.
2. Decrypts $\mathcal{D}_{sym}(z, w) = mz^{-1} \bmod n = m$.

Output: $m \in \mathbb{Z}_n$.

Figure 3.9: A formal description of ElGamal.

3.6 Hash Functions

Briefly, a *hash function* is a function h which takes inputs of some length and compress them into shorter, fix-length outputs. Hash-functions have been around for a long time, but one of their first use in cryptography were in protocols for signing messages. In this thesis we will use it in a similar construction, namely to prove that parties of the voting protocols in Chapter 7 follow the protocols properly.

In an ideal world, we would like the hash functions to be one-way and injective. Obviously a hash function can not be injective as the domain is larger then the range. And hash functions are not one-way in the classic sense. So we need a more appropriate definition of the properties we want our hash functions to possess.

DEFINITION 3.6. Let $h : S \to T$ be a function. We then have the following [14]:

I A *preimage* of $t \in T$ is an element $s \in S$ such that $h(s) = t$.

II A *second preimage* for $s_1 \in S$ is an element $s_2 \in S$ such that $s_1 \neq s_2$ while $h(s_1) = h(s_2)$.

III A *collision* for h is a pair of distinct elements $s_1, s_2 \in S$ such that $h(s_1) = h(s_2)$.

We say that h is *one-way* if it is an infeasible computation to find a preimage for $t \in T$ and a second preimage for a random $s \in S$ (I and II). And we say that a h is *collision resistant* If it is an infeasible computation to find collisions for h and to find a second preimage for a random $s \in S$ (II and III). If you can find second preimages, you can also find collisions. So it follows that if finding a collision is an infeasible computation, the hash function will be collision resistant and it will behave like an injective function [14].

We sometimes also want our hash functions to be "random-looking", but non of the above concerns this property. We will return to this question in Section 4.4.1.

4 | Security

Auguste Kerckhoffs stated an important principle in the 19th century. It is called Kerckhoffs' principle [10]:

"The security of the encryption scheme must depend only on the secrecy of the key, and not on the security of the algorithm".

What he meant was that every cryptosystem should be secure even if it is public. There are several very good reasons for this principle, and we will state two of them here, in simplified terms. A cryptosystem can have millions or more users, and it takes only one dishonest user to leak a secret. It is easy to change the keys of a cryptosystem, but much more difficult to change the whole system. It might also be reasonable to have cryptosystems publicly available, because then there will be more people to test and analyze them.

In 1945, Claude Elwood Shannon wrote a seminal article called "A Mathematical Theory of Cryptography" [30]. In this article he reformulated Kerckhoffs' principle as: "The enemy knows the system", and went on asking: "How secure is a system against cryptanalysis when the enemy has unlimited time and manpower available (...)?" . He continued by defining *perfect secrecy*, which was later denoted information-theoretical security, as oppose to computational-theoretical security which we will discuss in Section 4.6. Perfect secrecy is a property of not being able to get any information about the plaintext given the ciphertext, except possibly the length of the message, even if the adversary has unlimited computational power. But the real world do not offer unlimited time, manpower or computational power, as we will discuss in Section 4.2, and perfect secrecy has some great limitations which makes it very unpractical in most real world settings.

We will make one more notion before we continue. When discussing the security of a

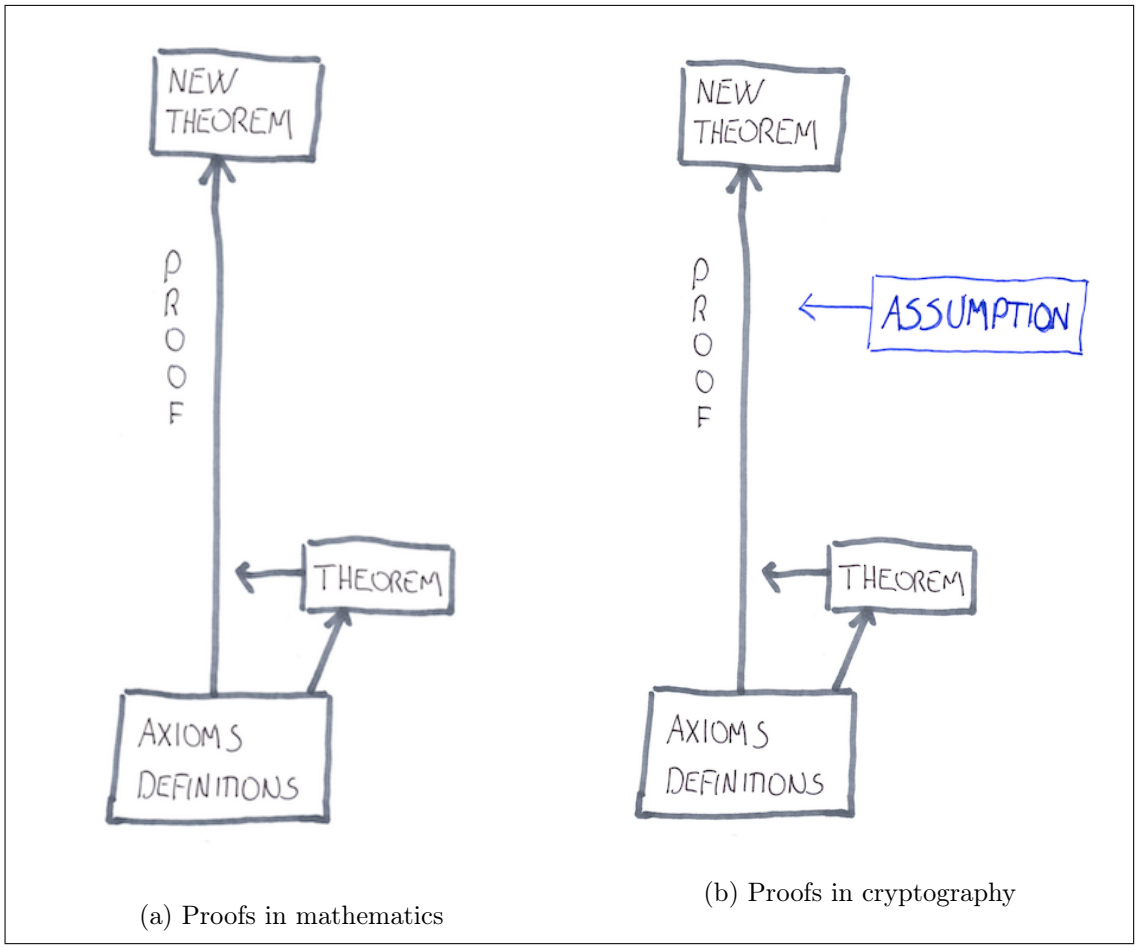
cryptosystem we can use many levels of formality. If we use the theory of computational complexity and statistics, we can formalize almost every discussion. But it is also possible to make security discussions in a more informal way. In the rest of this thesis we will make informal discussions and explore concepts rather than going into details. We will use terms like *feasible*, *infeasible*, *tractable* and *intractable*, all of them which will be explained in this chapter. We will discuss what it means for a cryptosystem or a cryptographic protocol to be secure, and draw lines to the security discussions in the next chapters. As we discuss this informally, there will not be citations for every statement, but the theory can be found in numerous books and articles, amongst them [14], [10], [23] and [21].

4.1 Security Proofs

There is a fundamental difference between proving a statement in mathematics and "proving" a statement in a non-axiomatic system as the real world. In mathematics we start with a set of axioms, and then form a deductive argument which we call a proof. When the starting point is not axiomatic, we can not prove anything in the rigorous meaning of the word, but only form inductive arguments based on evidence.

Modern cryptography is a combination of abstract algebra, number theory, probability theory and computational complexity theory, applied to computer science. We develop cryptosystems for actual security situations in the real world, and we would like them to be secure from any threat of both today and the future. But the threats depend on unpredictable variables such as the capacity of the human mind, the capacity of computer systems and probably other variables we know nothing about yet. And our adversaries are intelligent, clever, malicious and devious - they will try to do things no one has ever thought of before.

It is not difficult to understand that it is impossible to take into account all possible security threats. So as in the other branches of applied mathematics, we solve this problem by making models. We build our systems from smaller instances, often named *primitives*. For each of this instances we define exactly what we are trying to achieve, and from which types of attacks the instance is secure, including assumptions on the computational power of the adversary (see Section 4.2). In this manner we can build a system which fulfill our requirements, and if a new threat should appear we can immediately find out which part of the system that might be broken.



(a) Proofs in mathematics

(b) Proofs in cryptography

Figure 4.1: The use of assumptions in security proofs in cryptography.

When we define from which types of attacks the instance is secure, we also need to know what we mean by being secure. The security proofs in cryptography are formed in the same deductive manner as ordinary mathematical proofs, but there is an important difference between them: In our models we include computational hardness assumptions, i.e. assumptions on the existence of trapdoor one-way functions, see Figure 4.1. These hardness assumptions are not proven. Instead they started out as conjectures, and after years of studying them we now often think of them as assumptions.

There are several ways of thinking about security and forming the actual models and proofs. One of them is the one we will use in this thesis. In simplified terms it is based on the idea that if we are not able to distinguish two things, then we do not know anything about them. This idea applied to cryptosystems under a particular attack model called chosen-plaintext attacks, is often thought of as the computational-theoretical analoge to Shannon's perfect secrecy. It is based on the adversary's ability to tell which message a given ciphertexts belongs to under given circumstances. We often do this by simulating a game between an adversary and a challenger, and then analyze the outcome of the game. If the adversary is not able to distinguish, then we have security against chosen plaintext attacks. We will discuss this further in Section 4.6, and see examples of such games in Chapter 5 and 6. When we apply the indistinguishability idea to cryptographic protocols, instead of simulating a game, we can simulate a run of the protocol. This simulation might require more facilities, such as for example a random oracle, see Section 4.4.1. If the adversary can not tell the difference between a real run of the protocol and a simulated run, then we conclude that the protocol is secure in the model made of the given facilities. We will see many examples of such simulations in Chapter 7.

We are now ready to define many of the terms we need in the rest of this thesis.

4.2 Time and efficiency

As we mentioned in the beginning of this chapter, Shannon talked about adversaries with infinite computational power. This is the *information-theoretical approach*. But information-theoretical secure cryptosystems are not very practical. So for the modern cryptosystems we are discussing in this thesis, computational power is the fundament. Because infinite computational power does not exist: There will probably always be limitations. So instead we are making a *complexity-theoretical approach*.

The limitations we are talking about leads us to a discussion on time. Suppose we have access to all the computational power in the world of today, and suppose we have two algorithms $\mathcal{I}$ and $\mathcal{J}$ that run using this computational power. If algorithm $\mathcal{I}$ needs one minute to fininsh, and algorithm $\mathcal{J}$ needs thousands of years to finish, it is obvious that algorithm $\mathcal{J}$ is not very interesting to us, neither when it comes to encryption, decryption nor adversaries. But to do this reasoning we made an assumption on all the computational power in the world of today. This is not a very precise or scalable term. What about the running times with less computational power? And what about the running times required in ten years, when the computational power has increased? We need a more precise way to define the *time complexity* of an algorithm.

The answer can be found in the branch of computational complexity theory. Here we define the time complexity of an algorithm as the number of instructions, or steps, a machine need to execute during the running time of the algorithm. The machine we talk about here is a *Turing machine* named after the english mathematician and computer scientist Alan Turing. We will not go into the details of the Turing machine here, but only state that it is an abstract machine that manipulates symbols on a strip of tape according to a table of rules, like a computer in its simplest form. We refer the interested reader to [34].

There are several ways to calculate the time complexity, but in cryptography we use an asymptotical approach since the size of the inputs (n and c) are very large, and since it can be difficult to derive the exact running times (and the exact running times is not important in this discussion). This asymptotical approach is denoted by *big O notation*, $\mathcal{O}$, and it characterizes functions according to their growth rates. If an algorithm needs for example $8n^5 + 7n^3 + 4n^2$ steps, it has a time complexity of $\mathcal{O}(n^5)$.

If the time complexity of an algorithm can be described by $\mathcal{O}(n^k)$ for some positive constant k, we say that the algorithm is of *polynomial time*. In modern cryptography (at least for the systems we are discussing in this thesis), we are only interested in algorithm with polynomial running time.

At this stage it is important to mention that we are not giving the complete picture here. We are only trying to grasp the concept of time complexity, and are not looking into the details. For example, some polynomial time algorithms can run way too slow to be useful in a cryptographic context, depending on the constant k. But we will not go into this here. One thing to worry about though, is that in cryptography it is essential

to know the average-case time complexity. There are many problems which do not have polynomial running time on worst-case, but though have polynomial running time on average-case.

We can sort algorithms into three categories, where two of them are the categories "deterministic" and "non-deterministic". A *deterministic algorithm*, abbreviated DA, is an algorithm that follows the same execution path each time it is invoked with the same input. Everything we have discussed so far in this section concerns deterministic algorithms. A *non-deterministic algorithm*, abbreviated NDA, is an algorithm that makes decisions at certain points in the execution. So its execution path may differ each time it is invoked with the same input. The real difficult part of understanding the concept of NDA, is the algorithm's decisions. How does a NDA make its choice? We can think of the NDA as the "luckiest possible guesser". It always picks a transition that eventually leads to an accepting state, if this exists (we must imagine that the algorithm can "magically" make the choices that leads to success) [26]. It is worth mentioning that NDAs are not real algorithms. They are abstract constructions we use to try understanding the time complexity of algorithms.

We are now ready to define out first two complexity classes. As mentioned above, $\mathcal{O}$ represents a way to characterizes functions according to their growth rates, and a *complexity class* is a set of decision problems with related time complexity. We will start by defining the two complexity classes **P** and **NP**[28] :

DEFINITION 4.1. The complexity class **P** is the set of all decision problems which can be solved by a deterministic polynomial time algorithm.

DEFINITION 4.2. The complexity class **NP** is the set of all decision problems for which the answer can be *verified* in polynomial time, which is the same as the set of all decision problems which can be solved by a non-deterministic polynomial time algorithm (this is so because if a solution exists, the non-deterministic polynomial time algorithm will always find it).

One of the great unresolved question in computer science, is about the relation between **P** and **NP**: Is $\mathbf{P} = \mathbf{NP}$? This problem has been open since the beginning of 1970, and it is believed by most theorist that $\mathbf{P} \neq \mathbf{NP}$ [13]. One can think of public key cryptography as being founded on the assumption that the intractable problems which we know lies in **NP**, does not also lie in **P**. But this does not give the complete picture. Our only real requirement is that it takes significantly more time to break the cryptosystem than to

use it.

Above we made a notion concerning worst case complexity versus average case complexity. We will look into this now. First of all; our intractable problems are not that hard. Or, at least they are not at all the hardest problems to solve of all decision problems. We assume they are not possible to solve in polynomial time. But we have several complexity classes with problems harder and much harder than our intractable problems. Amongst them, there are complexity class called **NP-complete**, which consist of the the decision-problems that are both in **NP** and **NP-hard** (which are the decision problems that are at least as hard as the hardest problems in **NP**). We know that the intractable problems we are discussing in this thesis are not even in **NP-complete**. And we also know about problems which are in **NP-complete**, and thereby harder to solve, which we could have used. Amongst them there are some problems called the *knapsack* problems (for details see [33]. But so far the knapsack problems have always turned out to be average cases, and average-case time complexity for these problems are polynomial time. And so far, no one has been able to exploit the worst-case-complexity of them.

Let us continue the discussion on worst case versus average case time complexity. If this was of no concern, we could have stopped this section by stating that the problem of computing our one-way functions lies in **P** and the problem of computing inverses lies in **NP**. But this would be a crucial mistake as the average case of computing inverses could be polynomial as the above example, and hence it would be easy to decrypt the ciphers. The solution to this problem comes with random self-reducibility in Section 4.3.

But there is also another issue to be solved. Our encryption schemes are probabilistic, which means that the encryption algorithm gets as input a random integer. Also the key generation algorithm need efficient algorithms for tasks such as prime number generation, so also these algorithms are probabilistic.

Probabilistic algorithms, PAs, are similar to DAs, except that they have decision points during their run, and on decision point they chooses the successor path uniformly at random from the possible ones. Informally we can say that the algorithm is allowed to *toss coins* during the execution. We can think of them as DAs which takes the coin tosses as input. As it turns out, tossing coins is a very powerful attribute. PAs run in probabilistic polynomial time, abbreviated ppt. The polynomial time bound is the same as for polynomial time DAs, but the PAs are usually much faster. It is worth mentioning that when it is possible to make PAs decide on the uniform distribution, is is also possible

to make PAs decide on other distributions. PAs give rise to another complexity class:

DEFINITION 4.3. The complexity class **BPP** is the set of all decision problems which can be solved by a probabilistic polynomial time algorithm. (Actually, **BPP** stands for bounded-error probabilistic polynomial time, and an algorithm in **BPP** has an error probability of less than or equal to $\frac{1}{3}$. For details see [28].)

It is obvious that the set of all DAs running in polynomial time is included in the set of PAs, so $\mathbf{P} \subseteq \mathbf{BPP}$. We are now ready to define some efficiency terms.

DEFINITION 4.4.

- A computation which lies in **BPP** is called a *feasible* or *easy* computation. A problem which can be solved with a feasible computation is called *tractable*.
- A computation which do not lie in **BPP** is called an *infeasible* or *hard* computation. A problem which on average case can be solved by nothing faster than an infeasible computation is called *intractable*.

The algorithms constituting a cryptographic scheme are thus all understood to be ppt algorithms. The adversaries are also viewed as ppt algorithms. Hence it does not suffice to show that the security of a cryptographic scheme is guaranteed if the underlying problem is not in **P**: we need problems that are not in **BPP**, and we need that they are not in **BPP** on average.

There are several other complexity classes which we will not look into here. The interested reader can find more information in textbooks on computational complexity theory. We will now move forward and look into how we can decide to which complexity class a problem belong.

4.3 Reductions

When facing a computational decision problem we have to be able to decide which complexity class it belongs to. To do this, we can use reductions [23]:

DEFINITION 4.5. Let P_1 and P_2 be two decision problems, and let $\mathcal{A}_2$ be an algorithm that solves P_2. P_1 is said to be *polynomial (time) reducible* to P_2, written $P_1 \leq P_2$, if

there exists an algorithm $\mathcal{A}_1$ that solves P_1 which uses $\mathcal{A}_2$ as a subroutine, and where $\mathcal{A}_1$ runs in polynomial time if $\mathcal{A}_2$ does.

If $P_1 \leq P_2$ we can informally say that P_2 is at least as difficult as P_1, or equivalently, P_1 is no harder than P_2. To see this we can think of $\mathcal{A}_1$ with the subroutine $\mathcal{A}_2$ as one of many solutions to P_1, and that there can possible be other more efficient solutions to P_1. But if P_2 has a more efficient solution, P_1 has automatically a more efficient solution with P_2's algorithm as a subroutine.

Definition 4.6. Let P_1 and P_2 be two decision problems. If $P_1 \leq P_2$ and $P_2 \leq P_1$ we say that P_1 and P_2 are *computationally equivalent*, and hence belong to the same complexity class.

We will also discuss another kind of reduction, namely the random self-reductions. We can imagine a problem as a pile of instances, a pile of solutions and a correspondence between them. The idea is that we have an algorithm which solves the problem for a subset of the instances, and we want to use it to solve all instances. A random self-reduction takes an instance of a problem and make a new random instance, and supply a kind of instruction which link the solution to both the new and the old instance, such that if you find a solution to the new instance, you can easy find a solution to the old one. Then it is possible to solve every instance of the problem. Get an instance, make it into another instance and try to solve it. If the new instance ended up amongst the instances solvable by the algorithm we win, else we loose. So the algorithm has to be able to solve the problem for sufficient many instances, such that the probability of the new instances belonging to the solvable ones are sufficiently large.

Definition 4.7. Suppose x is a variable of a problem P. P is *random self-reducible* over x if we can show that solving P for a random instance of x implies that we can solve P for any given instances of x with equal probability.

When it comes to the reductions in this thesis they are very simple, and so it is obvious that the probability distribution going from a random instance of a variable to any other instance of that variable is uniform.

We have the following proposition which connect random self-reducible problems to other complexity problems:

PROPOSITION 4.8.
Any random self-reducible problem that is infeasible to compute on the worst case is also infeasible to compute on the average case.

Proof. Suppose a computational problem is random self-reducible and is infeasible to compute on the worst case. Suppose the problem is feasible to compute on the average. Then there cannot be any infeasible instances at all, since any such instance can be computed by transforming it to a uniformly random instance due to the random self-reducibility. And this is a contradiction. □

4.4 Security Models

We will use two models in this thesis, the standard model and the oracle model. The *standard model* is the most basic model. When we are proving security without defining any model, we are usually in the standard model.

4.4.1 The Random Oracle Model

Let us start by defining a random orcale.

DEFINITION 4.9. A *random oracle* is a black box containing a *random function*:

$$\mathcal{H} : \{0,1\}^* \to \{0,1\}^t$$

such that when the oracle is queried on an input value x, it will return the output value $\mathcal{H}(x)$.

If the same query is asked twice, identical answers are obtained since $\mathcal{H}$ is a function. $\{0,1\}^*$ means that the domain values can be infinite.

There are several ways to think of such a random function, but one way is to think of it as a table which initially is empty. Every time the oracle gets a query x, it will first check wether $x = x_i$ for some pair (x_i, y_i) in the tablet. If yes, then y_i is returned. If no, then a uniform string $y \in \{0,1\}^t$ is chosen, the pair (x, y) is stored in the table and the value y is returned, see Figure 4.2.

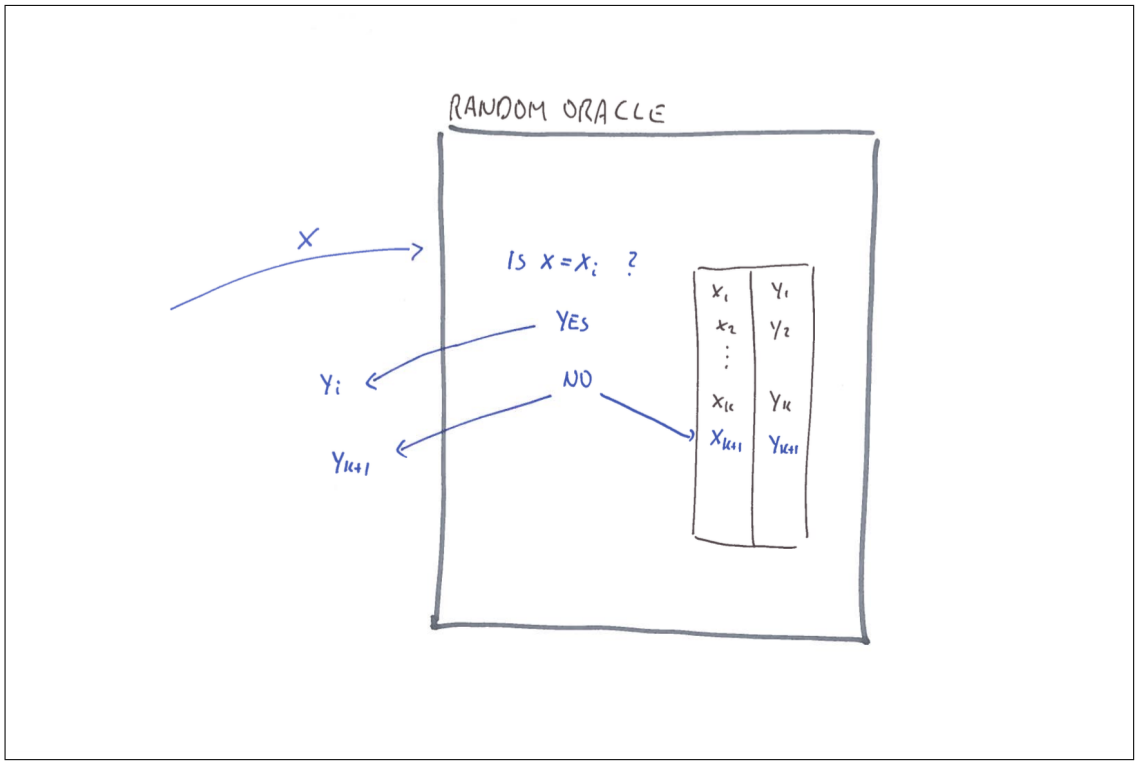


Figure 4.2: A visualization of a random oracle being queried on the value x.

So our problem is as follows: There are many functions which have properties that are similar to those of random oracles. But we have no way to model them. As an example we know that hash functions can be both one-way and collision resistant, but non of these properties tells us anything about wether they are random-looking or not.

The solution is the random oracle model, introduced by Bellare and Rogaway in 1993 [3]. The *random oracle model* is an idealized model where we assume that the hash functions are random oracles. All parties, both honest and adversaries, have access to this random oracle. Queries to the oracle are assumed to be private, so that if some party queries the oracle on x, then no one else learns x, or even learns that this party queried the oracle at all [28].

It is important to notice that this model is a *heuristic*. No one claims that a random oracle exists. If we prove a protocol secure in the random oracle model, it does not implicate that the same protocol instantiated with an actual hash function is secure. In fact, there are schemes that can be proven secure in the random oracle model but are insecure no matter how the random oracle is instantiated. These schemes are though contrived. And there has never been a successful attack on real schemes proven secure in the random oracle model, as long as the oracle was instantiated properly [21]. So the evidences speak for this heuristic.

4.5 Adversarial model

The best available measure of security (with a few exceptions as the one-time pad), is the complexity of the best (currently) known attacks. The idea is as follows: to be able to prove security of a cryptosystem, we have to know from what threats we are proving the security. This is usually done by defining an *adversarial model*, also called an *attacker model*. In this model we limit the attacks that may be executed, by defining what kind of access the adversary has to the system. We can then use this limitation to prove the security against those specific attacks. There are many possible attacks we know about, and there are probably plenty more we do not know anything about. We will list the most common here, but as we will see in the end, it is only one of them that are of particular interest in this thesis. The selection of these adversarial models comes from [10]. We will once again introduce Alice who is sending messages to Bob, and the eavesdropper Eve:

The ciphertext-only model is the situation in which Alice are encrypting her data, and all Eve get to see is the ciphertext. Trying to decrypt a message if Eve only knows the ciphertext is called a *ciphertext-only attack*, and is what most people mean when talking about breaking a cryptosystem. Included in this model is the *brute force attack*, where every possible key is tried until the correct one is found.

The known-plaintext model is the situation where Eve knows both a plaintext and its ciphertext. This can easily happen in a e-mail system. Eve might know that every e-mail starts with a "Hi", or "I'm away on holiday". Responding on such a e-mail can generate the ciphertext. Trying to decrypt a message in this model is called a *known-plaintext attack*.

The chosen-plaintext model is the situation where Eve can choose any number of plaintext and get the corresponding cipertexts. There are both a offline and a online version of a *chosen-plaintext attack*, but the most powerful is the online attack where Eve is trying to decrypt a message by choosing new plaintext depending on the ciphertexts she has already received. The chosen-plaintext attack will be abbreviated *CPA*.

The chosen-ciphertext model is the situation where Eve can choose both plaintext and ciphertext values. That is, for every plaintext she choose, she get the corresponding ciphertext, and for every ciphertext she choose, she gets the corresponding. Trying to decrypt a message in this model is called a *chosen-ciphertext attack*, and will be abbreviated *CCA*.

In the ciphertext-only model, the security relies on the decryption being an intractable problem. It is the most basic and obvious claim: We will always have to show that the function used by the encryption algorithm is trapdoor one-way, hence the encryption schemes are are secure in the ciphertext-only model see Section 4.1. When it comes to the brute force attack, every encryption scheme is vulnerable to this attack except the information-theoretically secure ones defined in the beginning of this chapter. But it is not any real threat: it is based on an infeasible calculation of exponential time complexity: If the decryption key has t bits, there are 2^t possible keys to try, so the time complexity of a brute-force attack is on worst case $\mathcal{O}(2^t)$ and on average case $\mathcal{O}(2^{t-1})$.

The known-plaintext model is contained in the chosen-plaintext model. This model is relevant for the semantic security of the system, and will be discussed in Section 4.6.

The chosen-ciphertext model is another story, as it turns out that many public key cryptosystem is vulnerable to CCA. In particular, all of the cryptosystems we are discussing in this thesis are in this category. This comes from the homomorphic property of the cryptosystems (defined in Section **??**). This is easy to see: When we operate in this model, Eve can get the decryption of any ciphertext, except the one she is trying to attack. So suppose she is trying to attack the cipher c to get the message m. She can now ask the decryption for a ciphertext c' to get the corresponding m'. Then, she can form another ciphertext as cc' and get the decryption of this which is $m + m'$ because of the homomorphic property. It is then easy to caluculate m.

We will mention here that we can differentiate between adaptive and non-adaptive attacks. An similar idea can be found in Section 7.6, but there we talk about adaptive adveraries versus static adversaries. The idea is as follows: In an *adaptive attack*, the adversary can choose to use her power both before and after a challenge ciphertext is given, whereas in a *non-adaptive attack*, this can only be done before. When it comes to CPA in public key cryptography, we can not differentiate between adaptive and non-adaptive attacks, as all of them are adaptive because of the public key (see Section 4.6). But for CPA in symmetric cryptography, which we are not studying in this thesis, it has an impact.

In all the attacks we have been studying so far, we have been worried about the confidentiality for Alice and Bob. But we also want them to have integrity. The integrity of homomorphic cryptosystem can be destroyed by something called *malleability*. It is defined as even if somebody do not know the decryption of a ciphertext, it is still possible to create new ciphertexts that decrypt to the same or related messages [14]. So all the encryption schemes we are studying in this thesis are malleable due to the homomorphic property.

When an adversary attack a cryptosystem by altering a ciphertext in the above manner, it is called a *man-in-the-middel attack*. This is an attack where the adversary, which we often name *Mallory*, secretly intercepts and relays messages between two parties who believe they are communicating directly with each other. It is an example of an *active* attack (changes the information in some way), whereas the other attacks we have been discussing here are examples of *passive* attacks (does neither affect the information nor disrupt the communication channel). Man-in-the-middel attack can also compromise the confidentiality. We will show how by giving an example: Suppose Alice wants to communicate with Bob, and Bob sends his public key to Alice. The man-in-the-middel

attack can then begin by Mallory intercepting this communication. She keeps the key for herself, and sends a forged message to Alice that give the impression of coming from Bob. Instead this is Mallory's public key. Alice, believing this public key to be Bob's, encrypts her message with Mallory's key and sends the encrypted message back to Bob. Mallory again intercepts, decrypts the ciphertext using her private key, possibly alters it if she wants, and re-encrypts it using the public key Bob originally sent to Alice. When Bob receives the newly encrypted message, he believes it came from Alice.

Both the chosen-ciphertext attacks and the man-in-the-middel attacks can be encountered for, or they may be of no concern in the environment in which the public key encryption scheme is to be used, but subject is out of scope for this thesis.

4.6 Semantic Security

A public key cryptosystem *has* to be secure in the chosen-plaintext model, if it is supposed to offer any security at all. This is so because the encryption key is public, so if the adversary wants to encrypt some messages of her choosing, she can just do it. So in the rest of this thesis, if nothing else is said, we are situated in the chosen-plaintext adversarial model, and the standard security model.

The ideal encryption scheme would be one that for every ciphertext, the probability of finding the corresponding message only given the public key, should be negligible.

Definition 4.10. A function f from the natural numbers to the non-negative real numbers is *negligible* if for every positive polynomial *pol* there is a T such that for all integers $t > T$ it holds that $f(t) < \frac{1}{pol(t)}$, [21].

As we talked about in the beginning of this chapter, it is not possible to use such ideal encryption schemes in practice. Instead we need another definition of security in the chosen-plaintext model. The answer to this is the notion of semantic security.

Semantic security is the computational complexity analogue to Shannon's concept of information-theoretical security, and it is often called *complexity-theoretical security.* What we want to show is that given the ciphertext of a certain message m, and maybe also the length of the message, we cannot determine any partial information of the message with probability non-negligibly higher than all other ppts that only have access to

the length of the message (and not the ciphertext). This is the same as saying that knowledge of the ciphertext (and maybe the length) of some unknown message does not reveal any additional information on the message that can be feasibly extracted. The notion of semantic security intuitively says that a ppt adversary cannot effectively *distinguish* between the encryption of two messages of his choosing. In [32], Victor Shoup formalize this as a game between an adversary and a challenger:

DEFINITION 4.11. A public key cryptosystem defined by $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is *indistinguishability under chosen plaintext attack*, abbreviated *IND-CPA* if the outcome of the following game between a challenger and a ppt bounded adversary satisfies the probability requirements defined below:

1. The challenger generate a pair of keys (ek, dk) by running $\mathcal{K}$.
2. The adversary (who knows dk and can perform any ppt bounded number of operations) chooses two distinct messages m_0 and m_1 from the possible plaintexts, and gives them to the challenger.
3. The challenger selects a bit $b \in \{0, 1\}$, computes $c = \mathcal{E}(dk, m_b)$ and gives c to the adversary.
4. The adversary (still able to perform any ppt bounded number of operations) output the value b' for which she thinks the message m'_b is the decryption of c.

The advantage of the adversary, defined as $P(b = b') - \frac{1}{2}$, is negligible.

DEFINITION 4.12. We define a public key cryptosystem to be *semantically secure* if it satisfies the IND-CPA requirement. In this case, we assume that the cryptosystem is secure in the chosen-ciphertext model.

It is important to notice that an encryption algorithm that are not probabilistic, will not be secure in this model. It is enough noticing that in this case, the adversary can just compute the ciphertext of m_0 and m_1, and by this be able to choose which message the ciphertext c belongs to with probability one.

The IND-CPA proves in this thesis are very simple, so it goes without saying that the advantage of the adversary is negligible.

There is also a similar concept which is known to be equivalent to IND-CPA.

DEFINITION 4.13. A public key cryptosystem defined by $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is *real-or-random*

secure, abbreviated *RoR* if the outcome of the following game between a challenger and a ppt bounded adversary satisfies the probability requirements defined below:

1. The challenger generates a pair of keys (ek, dk) by running $\mathcal{K}$.
2. The adversary (who knows dk and can perform any ppt bounded number of operations chooses a message m_0 and gives it to the challenger.
3. The challenger chooses a random message m_1, a bit $b \in \{0, 1\}$, computes $c = \mathcal{E}(dk, m_b)$ and gives c to the adversary.
4. The adversary outputs the value $b = 0$ if she thinks the message m'_b is the decryption of m_0 and the value $b = 1$ if she thinks the message m'_b is the decryption of a random message.

The advantage of the adversary, defined as $P(b = b') - \frac{1}{2}$, is negligible.

Proposition 4.14.
For a public key cryptosystem, IND-CPA $\Leftrightarrow$ RoR.

Proof. We will now show an example of how we can use games to prove a statement.

$\Rightarrow$: This is the same as proving that if we have an adversary, $\mathcal{A}_{\text{ROR}}$ with non-negligible advantage in the RoR game, then we can create an adversary $\mathcal{A}_{\text{IND}}$ with non-negligible advantage in the IND-CPA game. We do as follows:

1. $\mathcal{A}_{\text{ROR}}$ start a RoR game by choosing a message m_0.
   (a) We start a IND-CPA game by choosing two messages $\tilde{m}_0 = m_0$ and $\tilde{m}_1$ and send it to the challenger.
   (b) The challenger selects a bit $b \in \{0, 1\}$, computes $c = \mathcal{E}(dk, m_b)$ and outputs c.
   (c) We send c to $\mathcal{A}_{\text{ROR}}$.
2. $\mathcal{A}_{\text{ROR}}$ decides if c is an encryption of m_0 or a random message.
   (a) If $\mathcal{A}_{\text{ROR}}$ says m_0, we say m_0.
   (b) If $\mathcal{A}_{\text{ROR}}$ says random message, we say m_1.

If $\mathcal{A}_{\text{ROR}}$ win, we win. So if $\mathcal{A}_{\text{ROR}}$ has an non-negligible advantage then we have a non-negligible advantage.

$\Rightarrow$: We will not prove this here, but refer the interested reader to [15]

□

5 | Paillier's Cryptosystem

In 1999 Pascal Paillier published the article *"Public-Key Cryptosystems Based on Composite Degree Residuosity Classes"*, where he introduced a new cryptosystem [24]. It is based on the problem of computing n'th residuosity classes, which is thought to be an intractable problem. The security of the cryptosystem relies on an assumption on the hardness of distinguishing n'th residues from non n'th residues. This assumption is related to the hardness of factoring, though it is not known to be equivalent. As it turns out, this encryption scheme is of particularly interest because it possesses some nice homomorphic properties, which will be crucial for the applications in Chapter 7.

In this chapter we will describe Paillier's cryptosystem and analyze the security of it. This will be the foundation for the next chapter, where we present a generalization of Pailliers cryptosystem introduced in 2010 by Damgård, Jurik and Nielsen [7].

5.1 Paillier's Cryptosystem

Paillier utilizes the multiplicative group $\mathbb{Z}_{n^2}^*$, given by

$$\mathbb{Z}_{n^2}^* = \left\{1 \leq i \leq n \;\middle|\; \gcd(n^2, i) = 1\right\},$$

where n admissible (i.e. the usual product of two large primes p, q such that $\gcd(n, \varphi(n)) = 1$). The function of the encryption algorithm is closely related to n'th residues, and is given by

$$\begin{aligned} \gamma_g : \mathbb{Z}_n \times \mathbb{Z}_n^* &\longrightarrow \mathbb{Z}_{n^2}^* \\ (m, r) &\longmapsto g^m r^n \pmod{n^2}. \end{aligned} \tag{5.1}$$

This function is believed to be one-way, as we will see in Section 5.4. The trapdoor of this function is the factorization of n as will be shown in Section 5.3.

Paillier states that if the order of g is a nonzero multiple of n in $\mathbb{Z}_{n^2}^*$, then γ_g is bijective and $L(g^{\lambda(n)} \bmod n^2)$ has an multiplicative inverse in $\mathbb{Z}_n^*$ (as will be shown below in Section 5.1.1), where L denotes the function

$$\begin{aligned} L : S_n &\to \mathbb{Z}_n \\ u &\mapsto \frac{u-1}{n}, \end{aligned}$$

where S_n is the multiplicative subgroup of $\mathbb{Z}_{n^2}^*$ given by

$$S_n = \left\{ u < n^2 \;\middle|\; u \equiv 1 \pmod{n} \right\}.$$

In Figure 5.1 we formalize Paillier's probabilistic encryption scheme PA_g, given by $(\mathcal{K}_\mathrm{P}, \mathcal{E}_{\mathrm{P}g}, \mathcal{D}_{\mathrm{P}g})$. The choice of g in step (4) of $\mathcal{K}_\mathrm{P}$ will be discussed in further details below.

5.1.1 Paillier's g

The description of Paillier's cryptosystem above is identical to the description in [24]. Paillier let g be a general element in $\mathbb{Z}_{n^2}^*$ with order kn where $1 \leq k \leq \lambda(n)$, and stated that this is equivalent to $L(g^{\lambda(n)} \bmod n^2)$ being invertible (as needed for the decryption algorithm). Paillier also showed that the semantic security is independent of the choice of g. Later, in 2010, Damgård, Jurik and Nielsen made a particular choice of g, namely the simplest possible letting $g = (n+1)$ [7]. We will continue this thesis with the same choice of g as Damgård-Jurik, but first we will look a bit closer into Paillier's g. We will begin by stating a proposition:

PROPOSITION 5.1. *For any integer $a \geq 0$, we have*

$$(n+1)^a = 1 + an \bmod n^2,$$

and the order of $(n+1)$ in $\mathbb{Z}_{n^2}^*$ is n.

Proof. For the first part of the proposition, we can use the binominal expansion to get $(n+1)^a = \sum_{i=0}^{a} \binom{a}{i} n^i$. On the right-hand side, all terms with $i \geq 2$ becomes $0 \bmod n^2$,

Paillier's encryption scheme

Paillier's probabilistic encryption scheme PA_g is given by $(\mathcal{K}_\mathrm{P}, \mathcal{E}_{\mathrm{P}g}, \mathcal{D}_{\mathrm{P}g})$ as follows:

Key generation algorithm $\mathcal{K}_\mathbf{P}$:

(1) Choose two large primes p, q, with $\gcd(pq, \varphi(pq)) = 1$.

(2) Compute $n = pq$.

(3) Compute $d = \lambda(n)$.

(4) Choose $g \in \mathbb{Z}^*_{n^2}$ s.t. the order of g in $\mathbb{Z}^*_{n^2}$ is an nonzero multiple of n .

Output: $ek = (n, g)$ and $dk = d$.

Encryption algorithm $\mathcal{E}_{\mathbf{P}g}$:

Input: $ek = (n, g)$ and $m \in \mathbb{Z}_n$.

1. Choose random $r \in \mathbb{Z}^*_n$.
2. Compute $c = \gamma_g(m, r) = g^m r^n \bmod n^2$.

Output: $c \in \mathbb{Z}^*_{n^2}$.

Decryption algorithm $\mathcal{D}_{\mathbf{P}g}$:

Input: $dk = d$ and $c \in \mathbb{Z}^*_{n^2}$.

1. Compute $m = \dfrac{L(c^{\lambda(n)} \bmod n^2)}{L(g^{\lambda(n)} \bmod n^2)} \bmod n$.

Output: $m \in \mathbb{Z}_n$.

Figure 5.1: Paillier's probabilistic encryption scheme, PA_g, with general g.

so $(n+1)^a = 1 + an \bmod n^2$. The second part is a consequence of the first part of the proposition, since $a = n$ is the smallest nonzero a such that $1 + an = 1 \bmod n^2$. □

When $(n+1)$ has order n in $\mathbb{Z}^*_{n^2}$, $(n+1)^x$ will also have order n except when $\gcd(x, n) \neq 1$. In this case the order will be $\frac{n}{\gcd(x,n)}$. This will be the case when x is a nonzero multiple of p or q. From Carmichael's Theorem, we know that $r^{\lambda(n)n} = 1 \bmod n^2$, so the order of r^n in $\mathbb{Z}^*_{n^2}$ will be an integer k such that $1 \leq k \leq \lambda(n)$. We then get the following description of Paillier's g:

$$g = (n+1)^x r^n, \tag{5.2}$$

for $r \in \mathbb{Z}^*_n$ and $x \in \mathbb{Z}_n$ with x relatively prime to n (in Section 5.2, Proposition 5.2, we will see that in fact $(n+1)^x r^n \in \mathbb{Z}^*_{n^2}$).

To show the equivalence between this choice of g and $L(g^{\lambda(n)} \bmod n^2)$ being invertible, Paillier defined a class function similar to the function f_{Class} which will be defined by (6.2.2) in Section 5.2. The class function of an element $z \in \mathbb{Z}^*_{n^2}$ with respect to g, is the element m such that $z = g^m r^n \bmod n^2$. He proved that $L(g^{\lambda(n)} \bmod n^2)$ was the same as $\lambda(n)$ multiplied by the class function of g with respect to $(n+1)$. He then needed to show that the class function of g with respect to $(n+1)$ was invertible. He did this by showing an equality given below, but it suffices to see that when g is specified with respect to $(n+1)$, the class function is the x from (5.2), and this x is invertible iff $\gcd(x, n) = 1$ as stated.

To see that the security of PA_g is independent of g, we will use a general result involving two different g's, namely g_1 and g_2 given by:

$$g_1 = (n+1)^{x_1} r_1^n$$
$$g_2 = (n+1)^{x_2} r_2^n.$$

If we now want to specify g_1 with respect to g_2, and g_2 with respect to g_1, we get:

$$g_1 = ((n+1)^{x_2} r_2^n)^{\frac{x_1}{x_2}} \tilde{r_1}^n$$
$$g_2 = ((n+1)^{x_1} r_2^n)^{\frac{x_2}{x_1}} \tilde{r_2}^n.$$

So the class functions of g_1 with respect to g_2 is $\frac{x_1}{x_2}$, which exist because $\gcd(x_2, n) = 1$ from the definition of g_2. The same goes for g_2 with respect to g_1. We also notice that these two class functions are multiplicative inverses of each other. So it is easy to

change one into the other, and this shows that the security of PA_g, which is equivalent to calculating the class function (see Section 5.4), is random self-reducible over g.

To make the discussion easier, we will in the rest of this thesis let $g = (n+1)$. We will then redefine the function from (5.1) as:

$$\begin{aligned} \gamma : \mathbb{Z}_n \times \mathbb{Z}_n^* &\longrightarrow \mathbb{Z}_{n^2}^* \\ (m, r) &\longmapsto (n+1)^m \pmod{n^2}. \end{aligned} \tag{5.3}$$

We will also have a redefined version of Paillier's probabilistic encryption scheme PA given by $(\mathcal{K}_\mathrm{P}, \mathcal{E}_\mathrm{P}, \mathcal{D}_\mathrm{P})$, as seen in Figure 5.2. Notice that the decryption algorithm $\mathcal{D}_\mathrm{P}$ is also made easier by this choice of g, as we do not need the function L.

Paillier's probabilistic encryption scheme, PA

Paillier's probabilistic encryption scheme $\mathbf{PA}_g$ is given by $(\mathcal{K}_\mathrm{P}, \mathcal{E}_\mathrm{P}, \mathcal{D}_\mathrm{P})$ as follows:

Key generation algorithm $\mathcal{K}_\mathbf{P}$:

(1) Choose two large primes p, q, with $\gcd(pq, \varphi(pq)) = 1$.

(2) Compute $n = pq$.

(3) Compute $d = \lambda(n)$.

Output: $ek = n$ and $dk = d$.

Encryption algorithm $\mathcal{E}_\mathbf{P}$:

Input: $ek = n$ and $m \in \mathbb{Z}_n$.

1. Choose random $r \in \mathbb{Z}_n^*$.
2. Compute $c = \gamma(m, r) = (n+1)^m r^n \bmod n^2$.

Output: $c \in \mathbb{Z}_{n^2}^*$.

Decryption algorithm $\mathcal{D}_\mathbf{P}$:

Input: $dk = d$ and $c \in \mathbb{Z}_{n^2}^*$.

1. Compute $m = \dfrac{c^d \bmod n^2 - 1}{nd} \bmod n$.

Output: $m \in \mathbb{Z}_n$.

Figure 5.2: Paillier's probabilistic encryption scheme, PA, with $g = (n+1)$.

The further details on the encryption algorithm $\mathcal{E}_\mathrm{P}$ and the decryption algorithm $\mathcal{D}_\mathrm{P}$ will

be given in Section 5.2 and 5.3 respectively. In Section 5.3 we will also prove the required equality

$$\mathcal{D}_{\mathrm{P}}(dk, \mathcal{E}_{\mathrm{P}}(ek, m)) = m.$$

5.2 Pailliers encryption algorithm

Paillier's cryptosystem utilize the function γ which gets as input a message m and a randomized integer r. In this way, each particular message m can be encrypted into $|\mathbb{Z}_n^*|$ different ciphertexts (as will be proven below), where the probability of each ciphertext is uniformly distributed, i.e. each of the $|\mathbb{Z}_n^*|$ ciphertexts is equally likely to appear.

We will now take a closer look at the set $\mathbb{Z}_{n^2}^*$, and we start by proving that γ, defined by (6.1), defines an important isomorphism which includes an important homomorphic property.

5.2.1 The isomophism γ

PROPOSITION 5.2. *γ is a bijection.*

Proof. Since the two sets $\mathbb{Z}_n \times \mathbb{Z}_n^*$ and $\mathbb{Z}_{n^2}^*$ have the same cardinality,

$$|\mathbb{Z}_{n^2}^*| = \varphi(n^2) = p(p-1)q(q-1) = n\varphi(n) = |\mathbb{Z}_n||\mathbb{Z}_n^*| = |\mathbb{Z}_n \times \mathbb{Z}_n^*|,$$

it is only necessary to show that γ is injective, that is, for $m_1, m_2 \in \mathbb{Z}_n$ and $r_1, r_2 \in \mathbb{Z}_n^*$,

$$\gamma(m_1, r_1) = \gamma(m_2, r_2) \text{ in } \mathbb{Z}_{n^2}^* \implies m_1 = m_2 \text{ in } \mathbb{Z}_n \ \wedge \ r_1 = r_2 \text{ in } \mathbb{Z}_n^*.$$

So we begin with

$$(n+1)^{m_1} r_1^n \equiv (n+1)^{m_2} r_2^n \pmod{n^2}.$$

Since $r_1 \in \mathbb{Z}_n^* \Rightarrow r_1 \in \mathbb{Z}_{n^2}^*$, r_1 has an multiplicative inverse and we get

$$(n+1)^{m_2-m_1} \left(\frac{r_2}{r_1}\right)^n \equiv 1 \pmod{n^2} \tag{5.4}$$

$$(n+1)^{(m_2-m_1)\lambda(n)} \left(\frac{r_2}{r_1}\right)^{n\lambda(n)} \equiv 1 \pmod{n^2} \tag{5.5}$$

$$(n+1)^{(m_2-m_1)\lambda(n)} \equiv 1 \pmod{n^2} \tag{5.6}$$

Note that (5.5) implies (5.6) because of Carmichael's Theorem. Since the order of $(n+1)$ is n due to Proposition (5.1), we then get

$$(m_2 - m_1)\lambda(n) \equiv 0 \pmod{n} \tag{5.7}$$
$$m_2 - m_1 \equiv 0 \pmod{n} \tag{5.8}$$
$$m_2 \equiv m_1 \pmod{n} \tag{5.9}$$

where (5.7) is equivalent to (5.8) since $\gcd(n, \lambda(n)) = 1$. When substituting (5.9) into (5.4) we get

$$\left(\frac{r_2}{r_1}\right)^n \equiv 1 \pmod{n^2}$$
$$r_2^n \equiv r_1^n \pmod{n^2} \tag{5.10}$$
$$r_2 \equiv r_1 \pmod{n^2} \tag{5.11}$$
$$r_2 \equiv r_1 \pmod{n}.,$$

where (5.10) implies (5.11) because $\gcd(\lambda(n), n) = 1$ and hence n has a multiplicative inverse modulo $\lambda(n)$. □

Proposition 5.3. *γ defines an isomorphism from $\mathbb{Z}_n \times \mathbb{Z}_n^*$ to $\mathbb{Z}_{n^2}^*$.*

Proof. We already know that γ is a bijection, so we only need to show the homomorphic property

$$\gamma(m_1 + m_2, r_1 r_2) = \gamma(m_1, r_1)\gamma(m_2, r_2),$$

for $m_1, m_2 \in \mathbb{Z}_n$ and $r_1, r_2 \in \mathbb{Z}_n^*$, and with left-hand side taking place in modulo n^2 and right-hand side taking place in modulo n. So we have

$$\gamma(m_1 + m_2 \bmod n, r_1 r_2 \bmod n) \equiv (n+1)^{m_1+m_2 \bmod n}(r_1 r_2 \bmod n)^n \pmod{n^2}.$$

We know that $(n+1)$ has order n in $\mathbb{Z}_{n^2}^*$, so

$$(n+1)^{m_1+m_2 \bmod n} = (n+1)^{m_1+m_2} \bmod n^2.$$

We also know that $r_1 r_2 \bmod n \equiv r_1 r_2 + kn$ for some integer k, so by the binomial

expansion we get

$$\begin{aligned}(r_1 r_2 \bmod n)^n &\equiv (r_1 r_2 + kn)^n \\ &\equiv \sum_{i=0}^{n} \binom{n}{i} (r_1 r_2)^{n-i} (kn)^i \\ &\equiv (r_1 r_2)^n + n(r_1 r_2)^{n-1}(kn) \\ &\equiv (r_1 r_2)^n \pmod{n^2}.\end{aligned}$$

We then get as desired,

$$\begin{aligned}\gamma(m_1 + m_2 \bmod n, r_1 r_2 \bmod n) &\equiv (n+1)^{m_1+m_2}(r_1 r_2)^n \\ &\equiv (n+1)^{m_1}(r_1)^n (n+1)^{m_2}(r_2)^n \\ &\equiv \gamma(m_1, r_1)\gamma(m_2, r_2) \pmod{n^2}.\end{aligned}$$ □

We see that that the group operations for $m \in \mathbb{Z}_n$ and $c \in \mathbb{Z}_{n^2}^*$ are not the same: the product of two ciphertext will decrypt to the sum of their plaintexts. In comparison, the product of two RSA ciphertexts decrypt to the product of their plaintext. Hence Paillier's probabilistic encryption scheme, PA, is additively homomorphic whereas RSA is multiplicatively homomorphic. This property of PA, or actually the corresponding property of the generalization of PA which will be described in Section 6.2, will be essential for the voting schemes in Chapter 7.

5.2.2 N'th Residues

As mentioned in Section 5.1, n'th residues are closely related to Paillier's encryption function. In fact, the security of Paillier's cryptosystem relies on the intractability of distinguishing n'th residues from non n'th residues.

We will begin by defining two new sets, M_{PA} and R_{PA}, in the following way:

$$\begin{aligned}M_{\text{PA}} &= \{(n+1)^m \mid m \in \mathbb{Z}_n\} \\ R_{\text{PA}} &= \{r^n \mid r \in \mathbb{Z}_n^*\}.\end{aligned}$$

PROPOSITION 5.4. *M_{PA} and R_{PA} are subgroups of $\mathbb{Z}_{n^2}^*$.*

Proof.

- Closed under multiplication:

$$(n+1)^{m_1}, (n+1)^{m_2} \in M_{\text{PA}} \;\Rightarrow\; (n+1)^{m_1}(n+1)^{m_2} = (n+1)^{m_1+m_2} \in M_{\text{PA}}.$$
$$r_1^n, r_2^n \in R_{\text{PA}} \;\Rightarrow\; r_1^n r_2^n = (r_1 r_2)^n \in R_{\text{PA}}.$$

- Identity elements are included:

$$(n+1)^0 = 1 \in M_{\text{PA}}.$$
$$1^n = 1 \in R_{\text{PA}}.$$

- Inverses are included:

$$(n+1)^m \in M_{\text{PA}} \;\Rightarrow\; ((n+1)^m)^{-1} = (n+1)^{-m} \in M_{\text{PA}}.$$
$$r^n \in R_{\text{PA}} \;\Rightarrow\; \left(r^{-1}\right)^n \in M_{\text{PA}}.$$

□

Proposition 5.5. *$M_{PA} \simeq \mathbb{Z}_n$ and $R_{PA} \simeq \mathbb{Z}_n*$.*

Proof. We define the following functions

$$\mu_1 : \mathbb{Z}_n \to M_{\text{PA}}$$
$$m \longmapsto (n+1)^m \bmod n$$

and

$$\mu_2 : \mathbb{Z}_n^* \to R_{\text{PA}}$$
$$r \longmapsto r^n \bmod n.$$

The homomorphic properties follow directly from proposition 5.3, and surjectivity follows by the definition of M_{PA} and R_{PA}, so we only need to prove that the functions are injective.

$$(n+1)^{m_1} = (n+1)^{m_2} \pmod n \;\Rightarrow\; m_1 = m_2 \pmod n,$$

because $|\mathbb{Z}_n| = n$.

$$r_1^n = r_2^n \pmod n \;\Rightarrow\; (r_1^n)^{\lambda}(n) = (r_2^n)^{\lambda}(n) \pmod n \;\Rightarrow\; r_1 = r_2 \pmod n,$$

due to Carmichael's Theorem. □

We will now look at the *n'th residues* and their characteristics.

DEFINITION 5.6. A number c is said to be a *n'th residue modulo* n^2 if there exists a number $z \in \mathbb{Z}_{n^2}^*$ such that

$$c = z^n \bmod n^2.$$

The set of n'th residues will be denoted by R_{PA}, that is

$$R_{\text{PA}} = \left\{ c \mid \exists\, z \in \mathbb{Z}_{n^2}^* \text{ with } c = z^n \bmod n^2 \right\}.$$

The *problem of deciding n'th residuosity*, that is, distinguishing n'th residues from non n'th residues, will be denoted by $DCR[n]$.

Let us look at the set R_{PA}. We know that $r \in \mathbb{Z}_n^* \Rightarrow r \in \mathbb{Z}_{n^2}^*$, so obviously $R_{\text{PA}} \subseteq R_{\text{PA}}$. We will now prove the following:

PROPOSITION 5.7. $R_{PA} = R_{PA}$.

Proof. Let

$$\begin{aligned} \psi_1 : \mathbb{Z}_{n^2}^* &\longrightarrow \mathbb{Z}_{n^2}^* \\ z &\longmapsto z^n \bmod n^2. \end{aligned}$$

An element $z \in \mathbb{Z}_{n^2}^*$ has the representation $(n+1)^m r^n$ for some $m \in \mathbb{Z}_n, r \in \mathbb{Z}_n^*$, so

$$\psi_1(z^n) = ((n+1)^m r^n)^n = r^{n^2} \bmod n^2,$$

because $(n+1)$ has order n in $\mathbb{Z}_{n^2}^*$. The image of ψ_1 is the set R_{PA}, that is

$$\psi_1[\mathbb{Z}_{n^2}^*] = \left\{ r^{n^2} \;\middle|\; r \in \mathbb{Z}_n^* \right\} = R_{\text{PA}}.$$

We now define a function

$$\begin{aligned} \psi_2 : R_{\text{PA}} &\longrightarrow R_{\text{PA}} \\ r^{n^2} \bmod n^2 &\longmapsto r^{n^2 b} = r^n \bmod n \\ r^{n^2} \bmod n^2 &\longleftarrow\!\shortmid\, r^n \bmod n, \end{aligned}$$

where b is an integer with $nb = 1 \pmod{\varphi(n)}$, which exists since $\gcd(n, \varphi(n)) = 1$. From the previous discussion, ψ_2 is clearly a bijective homomorphim. Since $R_{\text{PA}} = \left\{ r^{n^2} \;\middle|\; r \in \mathbb{Z}_n^* \right\}$ contains all n'th residues and $R_{\text{PA}} \simeq R_{\text{PA}}$, it follows that $R_{\text{PA}} = R_{\text{PA}}$. □

M_{PA} and R_{PA} give rise to two factor groups, namely

$$\bar{M}_{\mathrm{PA}} = \mathbb{Z}_{n^2}^* / R_{\mathrm{PA}} = \{(n+1)^m R_{\mathrm{PA}} \mid m \in \mathbb{Z}_n\}$$

with elements $\bar{M}_{\mathrm{PA}_i} = \{(n+1)^{m_i} r^n \mid r \in \mathbb{Z}_n^*\}$, for $1 \leq i \leq n$ (see Figure 5.3a), and

$$\bar{R}_{\mathrm{PA}} = \mathbb{Z}_{n^2}^* / M_{\mathrm{PA}} = \{M_{\mathrm{PA}} r^n \mid r \in \mathbb{Z}_n^*\},$$

with elements $\bar{R}_{\mathrm{PA}_i} = \{(n+1)^m r_i^n \mid m \in \mathbb{Z}_n\}$, for $1 \leq i \leq \varphi(n)$ (see Figure 5.3b).

Let us look at the function ψ_1. It is clear that each element in R_{PA} is independent of m. That is, each n'th residue has n roots, and the set of roots of r_i^n is the set $\bar{R}_{\mathrm{PA}_i}$.

Let us now look at set $\bar{M}_{\mathrm{PA}}$. As mentioned in the beginning of this section, each message m can be encrypted into $\varphi(n)$ different ciphertext. The set of all possible encryptions of the message m_i is the set $\bar{M}_{\mathrm{PA}_i}$, with $m_i = 0$ being the set of n'th residues. The factor group $\bar{M}_{\mathrm{PA}}$ gives rise to two functions, which we can combine as follows:

$$\begin{array}{lccccc} f_{\mathrm{Class}}: & \mathbb{Z}_{n^2}^* & \xrightarrow{f_1} & \bar{M}_{\mathrm{PA}} & \xrightarrow{f_2} & \mathbb{Z}_n \\ & (n+1)^m r^n & \longmapsto & (n+1)^m R_{\mathrm{PA}} & \longmapsto & m. \end{array}$$

Definition 5.8. The *n'th Residuosity Class Problem*, denoted *Class*[n], is the problem of computing $f_{\mathrm{Class}}(c)$ for a given $c \in \mathbb{Z}_{n^2}^*$.

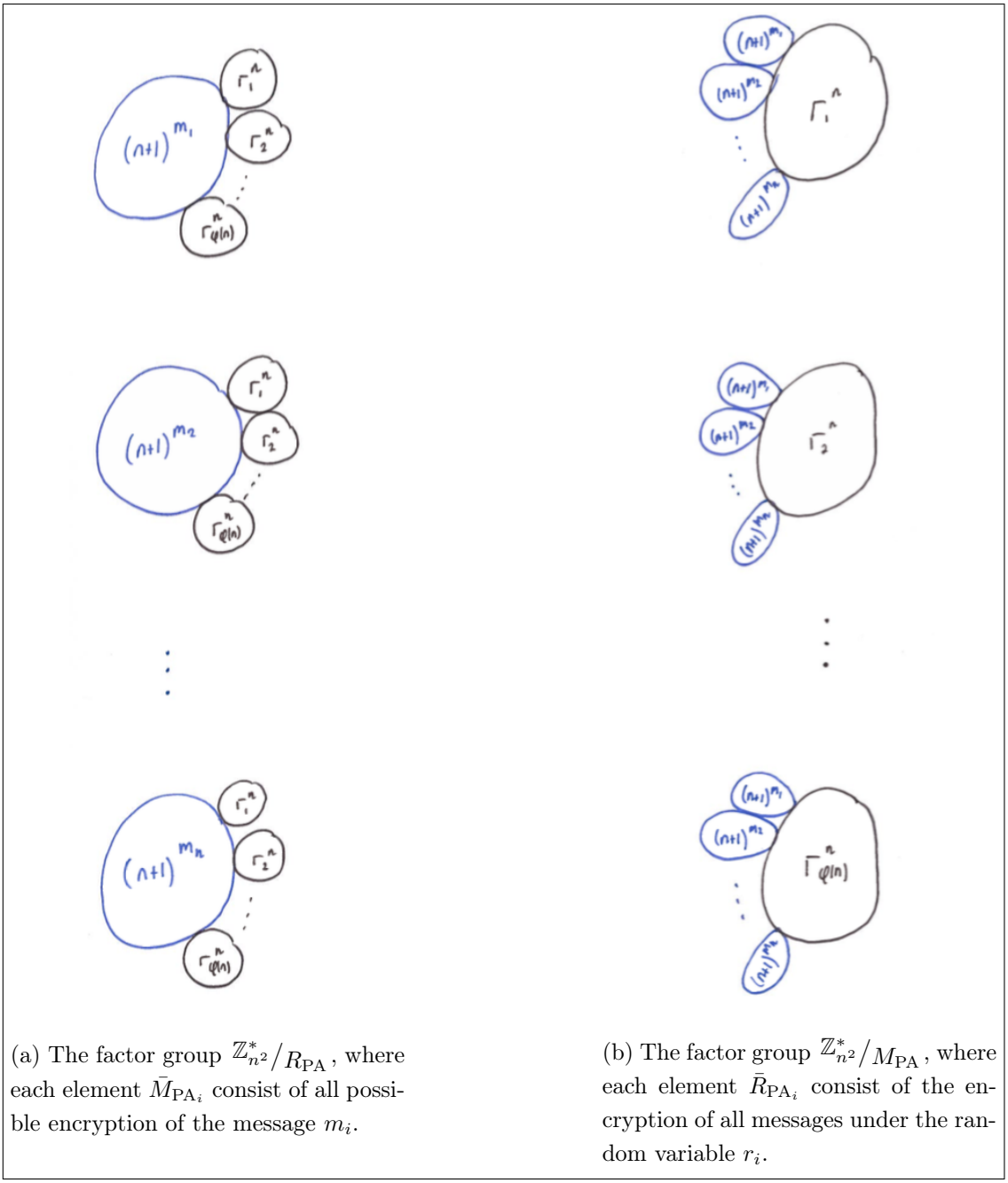
5.3 The Decryption Algorithm $\mathcal{D}_{\mathbf{P}}$

As mentioned in Section 5.1, the decryption algorithm with $g = (n+1)$ is easier than in Paillier's original system (for details on the original system see [24]).

We know from Proposition 5.1 that

$$(n+1)^a = 1 + an \bmod n^2.$$

Combining this with Carmichael's Theorem we get:



(a) The factor group $\mathbb{Z}^*_{n^2}/R_{\text{PA}}$, where each element $\bar{M}_{\text{PA}_i}$ consist of all possible encryption of the message m_i.

(b) The factor group $\mathbb{Z}^*_{n^2}/M_{\text{PA}}$, where each element $\bar{R}_{\text{PA}_i}$ consist of the encryption of all messages under the random variable r_i.

Figure 5.3: Dividing $\mathbb{Z}^*_{n^2}$ into two different factor groups.

$$\begin{aligned} c &= (n+1)^m r^n \\ c^{\lambda(n)} &\equiv (n+1)^{m\lambda(n)} r^{n\lambda(n)} \\ &\equiv (n+1)^{m\lambda(n)} \\ &\equiv 1 + mn\lambda(n) \pmod{n^2} \\ \Rightarrow \frac{c^{\lambda(n)} - 1}{n} &\equiv m\lambda(n) \pmod{n}. \end{aligned}$$

From Section 5.1 we have the decryption key $d = \lambda(n)$, and so we get the decryption m of c by:

$$\frac{(c^d \bmod n^2) - 1}{n} d^{-1} \bmod n = m.$$

So the trapdoor is knowing the factorization of n to compute $d = \lambda(n)$, and with this information the problem of decrypting m from c becomes tractable. We then get

$$\mathcal{D}_{\mathrm{P}}(dk, \mathcal{E}_{\mathrm{P}}(ek, m)) = \mathcal{D}_{\mathrm{P}}(dk, c) = m,$$

as required.

5.4 The Security of PA

We will first look at the problem DCR[n], which is the problem of deciding n'th residuosity defined in Section 5.2.

Proposition 5.9. *DCR[n] is random self-reducible over $c \in \mathbb{Z}^*_{n^2}$.*

Proof. Let $c \in \mathbb{Z}^*_{n^2}$ be a particular ciphertext. We have to show that if we can solve DCR[n] for a random $\tilde{c} \in \mathbb{Z}^*_{n^2}$, then we can also solve DCR[n] for any $c \in \mathbb{Z}^*_{n^2}$. So let

$$c = (n+1)^m r^n.$$

We can transform c into a random instance $\tilde{c}$ simply by randomly choosing $\tilde{m} \in \mathbb{Z}_n$ and $\tilde{r} \in \mathbb{Z}^*_n$ such that

$$\tilde{c} \equiv c(n+1)^{\tilde{m}} \tilde{r}^n \equiv (n+1)^{m+\tilde{m}} (r\tilde{r})^n \pmod{n^2}.$$

Suppose we can solve DCR[n] for $\tilde{c}$, that is, we know whether $m + \tilde{m} \equiv 0 \pmod{n}$ or $m + \tilde{m} \not\equiv 0 \pmod{n}$. Since $n = pq$ where p, q are very large primes, the probability that $m \equiv \tilde{m} \pmod{n}$ is negligible. This leads to the following:

$$\begin{aligned} m + \tilde{m} &\equiv 0 \pmod{n} \;\Rightarrow\; m = 0 \\ m + \tilde{m} &\not\equiv 0 \pmod{n} \;\Rightarrow\; m \neq 0. \end{aligned}$$ □

From the discussion in the previous sections, we know that $c \mapsto c(n+1)^{\tilde{m}}\tilde{r}^n$ is a bijection from $\mathbb{Z}^*_{n^2}$ to $\mathbb{Z}^*_{n^2}$ (since m has an additive inverse in $\mathbb{Z}_n$ and r has as multiplicative inverse in $\mathbb{Z}^*_n$), which means that our random instances will cover all of $\mathbb{Z}^*_{n^2}$.

Since DCR[n] is random self-reducible, we know that the problem is either uniformly intractable or uniformly polynomial. The problem DCR[n] is a well-studied mathematical problem, and it is believed to be computationally hard, i.e. it is intractable. This leads to the following assumption.

Assumption 1*: *Decisional Composite Residuosity Assumption, DCRA. *There exists no polynomial time distinguisher for n'th residues modulo n^2, that is, DCR[n] is intractable.*

We will continue by proving the computational hierarchy of the problems related to Pailler's encryption scheme. We start by looking at the problem Class[n]. From Section 5.1.1 we know that Class[n] is random self-reducible over g.

PROPOSITION 5.10. *Class[n] is random self-reducible over $c \in \mathbb{Z}^*_{n^2}$.*

Proof. Let $c \in \mathbb{Z}^*_{n^2}$ be a particular ciphertext. We have to show that if we can solve Class[n] for a random $\tilde{c} \in \mathbb{Z}^*_{n^2}$, then we can also solve Class[n] for any $c \in \mathbb{Z}^*_{n^2}$. So let

$$c = (n+1)^m r^n.$$

We can transform c into a random instance $\tilde{c}$ simply by randomly choosing $\tilde{m} \in \mathbb{Z}_n$ and $\tilde{r} \in \mathbb{Z}^*_n$ such that

$$\tilde{c} = c(n+1)^{\tilde{m}}\tilde{r}^n = (n+1)^{m+\tilde{m}}(r\tilde{r})^n.$$

Suppose we can solve Class[n] for $\tilde{c}$. Then we can solve Class[n] for c simply by computing $f_{\text{Class}}(\tilde{c}) - \tilde{m}$. Following the proof of Proposition 5.9, we know that our random instances will cover all of $\mathbb{Z}^*_{n^2}$. □

The computational hierarchy of the problems is as follows:

Proposition 5.11. *DCR*$[n] \leq$ *Class*$[n] \leq$ *Fact*$[n]$.

Proof. What this proposition says, is that if you can solve Fact$[n]$, then you can solve Class$[n]$, and if you can solve Class$[n]$, then you can solve DCR$[n]$.
DCR$[n] \leq$ Class$[n]$: Suppose we have an algorithm $\mathcal{A}$ that solves Class$[n]$. That is, $\mathcal{A}$ computes the function f_{Class}. When we get an element c from DCR$[n]$, we can then use $\mathcal{A}$ to compute $f_{\text{Class}}(c)$. If the answer is 0, then we have an n'th residue, and if the answer is $\neq 0$ we have a not.
Class$[n] \leq$ Fact$[n]$: This follows directly from Section 5.1.1 and 5.3 (and is the base for our decryption algorithm). □

This leads to our second assumption:

Assumption 2*: *Computational Composite Residuosity Assumption, CCRA.
There exists no probabilistic polynomial time algorithm solving f_{Class}, that is, CCRA is intractable.

Theorem 5.12. *Paillier's encryption scheme, as defined in Section 5.1, is one-way if and only if CCRA holds.*

Proof. From the discussion in Section **??**, solving Class$[n]$ is equivalent to inverting γ. □

Theorem 5.13. *Paillier's encryption scheme PA, as defined in Section 5.1, is semantically secure if and only if DCRA holds.*

Proof. We will use the IND-CPA game to show semantic security here.

$\Leftarrow$: This is the same as proving that if PA is not secure, then DCRA do not hold. So we have an adversary $\mathcal{A}_1$ against PA. We can then create an adversary against DCR$[n]$ as follows:

1. $\mathcal{A}_1$ start a IND-CPA game in PA by choosing two messages $m_0, m_1 \in \mathbb{Z}_n$.
2. The challenger chooses $b \in \{0, 1\}$ and outputs $c = (n+1)^{m_b} r^n \bmod n^2$.
   (a) We choose an instance x of DCR$[n]$, and want to find out if x is a n'th residue or not.

(b) We compute $c' = cx$.

(c) We sen c' or $\mathcal{A}_1$.

3. $\mathcal{A}_1$ decides if c' is an encryption of m_0 or m_1 and answers $b' \in 0, 1$.
   (a) If $b' = b$ we say n'th residue.
   (b) If $b' \neq b$ we say not n'th residue.

We then get two cases:

- x is a n'th residue: Then c' is an encryption of either m_0 or m_1, and so $\mathcal{A}_1$ has an non-negligible advantage, which means that it has a probability non-negligible more then $1/2$ to choose the right one. When $\mathcal{A}_1$ win, we win, and so we have a probability non-negligible higher than $1/2$ to find n'th residues.
- x is a non n'th residue: Then x is an encryption of some random message and hence c' is an encryption of something random. Now $\mathcal{A}_1$ choose with zero advantage, which means that it has only probability $1/2$ to choose the right one. So when $\mathcal{A}_1$ win, we loose, and this happens with probability $1/2$.

This means that we more often choose right when x is a n'th residue, than wrong when x is a non n'th residue, so we are able to solve DCR$[n]$ with probability non-negligible more than $1/2$, and so we break the DCRA assumption.

$\Rightarrow$: This is the same as proving that if DCRA do not hold, then PA is not semantically secure. So we now have an adversary $\mathcal{A}_{\text{DCR}}$ which solves DCR$[n]$ with probability more then $1/2$. We can then create an adversary $\mathcal{A}_1$ against PA as follows:

1. $\mathcal{A}_{\text{DCR}}$ takes as input an instance x of DCR$[n]$ and then outputs n'th residue or non n'th residue.
   (a) We choose two messages $m_0 = 0$ and $m_1 = *$, where $*$ is a random element in $\mathbb{Z}_n$.
   (b) We send m_0, m_1 to the challenger.
   (c) The challenger chooses $b \in \{0, 1\}$ and outputs $c = (n+1)^{m_b} r^n \bmod n^2$.
   (d) We send $x = c$ to the oracle.
2. If the oracle outputs n'th residue, then we say $b' = 0$.
3. If the oracle outputs non n'th residue, then we say $b' = 1$.

When the oracle wins we win, and since the oracle has a probability of winning which is non-negligible more than $1/2$, $\mathcal{A}_1$ has an non-negligible advantage of choosing right.

□

6 A Generalization of Paillier's Cryptosystem

We will now introduce a generalization of Paillier's cryptosystem, described by Damgård, Jurik and Nielsen in [7], namely the *Damgård-Jurik-Nielsen* cryptosystem which we will abbreviate DJN_s. Paillier uses multiplication modulo n^2, whereas Damgård-Jurik-Nielsen use multiplications modulo n^{s+1} for any $s \geq 1$. It is worth noticing that DJN_1, that is $s = 1$, is the same as Paillier's cryptosystem.

6.1 Damgård-Jurik-Nielsen cryptosystem

As already mentioned, the Damgård-Jurik-Nielsen cryptosystem uses multiplications modulo n^{s+1} for any $s \geq 1$. The corresponding multiplicative group is described by

$$\mathbb{Z}^*_{n^{s+1}} = \left\{ 1 \leq i \leq n^{s+1} \;\middle|\; \gcd(n^{s+1}, i) = 1 \right\},$$

where n is admissible.

As in the previous chapter, we will simplify the system by letting $g = (n+1)$. As for PA (PA_g with $g = (n+1)$), the public key can consist of only the modulus n. We then get the function related to the encryption algorithm of DJN_s as follows:

$$\xi_s : \mathbb{Z}_{n^s} \times \mathbb{Z}^*_n \longrightarrow \mathbb{Z}^*_{n^{s+1}} \tag{6.1}$$

$$(m, r) \longmapsto (n+1)^m r^{n^s} \pmod{n^{s+1}}. \tag{6.2}$$

In the same manner as in the previous chapter, we will show that the function is feasible to compute but infeasible to reverse, unless we have a trapdoor, which then makes the

function feasible to reverse. The trapdoor is once again the factorization of n, from which we can compute $\lambda(n)$.

We will continue by formalizing the Damgård-Jurik-Nielsen probabilistic encryption scheme, DJN_s (see Figure 6.1). The figure needs some comments. If we first look at the key generation algorithm, (1) has a security parameter t. We will use this when we apply the system to electronic voting in Chapter 7. (2) is straightforward since we know both p and q, while (3) needs some more discussion. It is not difficult to compute d, as we can apply the Chinese Remainder Theorem. But this differs from the choice of d in Paillier's original scheme since he used $d = \lambda(n)$ which is the smallest possible value. The reason we need other choices than $\lambda(n)$ will be revealed when we come to a threshold decryption scheme in Section 7.6. The encryption algorithm is straightforward, but we will do the calculations of (1) in the decryption algorithm:

$$c^d = \left((n+1)^m r^{n^s}\right)^d \bmod n^{s+1} = (n+1)^{md \bmod ns} r^{n^s d \bmod \lambda(n)} = (n+1)^{md \bmod ns}.$$

In (3) we know that the multiplicative inverse of d exists because

$$d \in \mathbb{Z}_n^* \;\Rightarrow\; d \in \mathbb{Z}_{n^s}* \;\Rightarrow\; d \text{ has a multiplicative inverse in } \mathbb{Z}_{n^s}.$$

We will discuss the details on the encryption algorithm $\mathcal{E}_D$ and the decryption algorithm $\mathcal{D}_D$ in Section 6.2 and 6.3 respectively. In Section 6.3 we will also prove the required equality

$$\mathcal{D}_D(dk, \mathcal{E}_D(ek, m)) = m.$$

Before we end this section, we will prove a proposition corresponding to Proposition 5.1 of Section 5.1

PROPOSITION 6.1. *For any admissible n and $s < p, q$, the element $(n+1)$ has order n^s in $\mathbb{Z}_{n^{s+1}}^*$.*

Proof. For any integer $a \geq 0$, let us look at the integer $(n+1)^a$.

$$\begin{aligned}(n+1)^a &= \sum_{i=0}^{a} \binom{a}{i} n^i \\ &= 1 + \sum_{i=1}^{a} \binom{a}{i} n^i.\end{aligned}$$

Damgård-Jurik-Nielsen's encryption scheme

Damgård-Jurik-Nielsen's probabilistic encryption scheme DJN_s is given by $(\mathcal{K}_D, \mathcal{E}_D, \mathcal{D}_D)$ as follows:

Key generation algorithm $\mathcal{K}_D$:

(1) Choose admissible $n = pq$ of length t bits.

(2) Compute $\lambda(n)$.

(3) Choose d such that $d \bmod n \in \mathbb{Z}_n^*$ and $d = 0 \bmod \lambda(n)$.

Output: $ek = n$ and $dk = d$.

Encryption algorithm $\mathcal{E}_D$:

Input: $ek = n$ and $m \in \mathbb{Z}_{n^s}$.

(1) Choose random $r \in \mathbb{Z}_n^*$.

(2) Compute $c = \xi_s(m, r) = (n+1)^m r^{n^s} \bmod n^{s+1}$.

Output: $c \in \mathbb{Z}_{n^{s+1}}^*$.

Decryption algorithm $\mathcal{D}_D$:

Input: $dk = d$ and $c \in \mathbb{Z}_{n^{s+1}}^*$.

(1) Compute $c^d = (n+1)^{md \bmod n^s} = (n+1)^{m'}$.

(2) Use algorithm from Figure **??** to extract m'.

(3) Compute $m = \dfrac{m'}{d} \bmod n^s$

Output: $m \in \mathbb{Z}_{n^s}$.

Figure 6.1: Damgård-Jurik-Nielsen's probabilistic encryption scheme, DJN_s, with $g = (n+1)$.

We then get

$$\begin{aligned}
(n+1)^a &\equiv 1 \pmod{n^{s+1}} \\
&\Updownarrow \\
\sum_{i=1}^{a} \binom{a}{i} n^i &\equiv 0 \pmod{n^{s+1}} \\
&\Updownarrow \\
\sum_{i=1}^{a} \binom{a}{i} n^{i-1} &\equiv 0 \equiv kn^s \pmod{n^s}, \qquad (6.3)
\end{aligned}$$

for some integer k. Equation (6.3) is clearly satisfied if $a = n^s$, so it follows that the order of $(n+1)$ is a divisor of n^s. The order then has to be the smallest number $b = p^l q^m$, where $l, m \leq s$, such that

$$\sum_{i=1}^{b} \binom{b}{i} n^{i-1} = 0 \pmod{n^s}. \qquad (6.4)$$

Let us now assume that $b < n^s$, and without loss of generality we can assume that $l < s$ (if $l = s$ then $m < s$ and we can follow the same argument as below). Let us now look at a term $\binom{b}{i} n^{i-1}$ in (6.4). We claim that each such term is divisible by b: If $i > s$ this is trivial. If $i \leq s$, it follows from the fact that $s < p, q$ leads to p, q not being prime factors of $i!$, and hence b must divide $\binom{b}{i}$. We know from (6.4) that n^s must divide $\sum_{i=1}^{b} \binom{b}{i} n^{i-1}$, and so clearly p must divide $\sum_{i=1}^{b} \binom{b}{i} n^{i-1}/b$. The first term in this sum is 1, and all the other terms are divisible by p, and so $\sum_{i=1}^{b} \binom{b}{i} n^{i-1}/b = 1 \pmod{p}$ which leads to a contradiction. So $b \not< n^s$ and n^s is the order of $(n+1)$ in $\mathbb{Z}^*_{n^{s+1}}$. □

6.2 The Encryption Algorithm $\mathcal{E}_D$

As for Paillier's original cryptosystem, the randomized integer r leads to being a probabilistic encrytion scheme (see Section 4. We will continue to follow the lines from the previous chapter, and take a closer look at the function ξ_s and the set $\mathbb{Z}^*_{n^{s+1}}$.

6.2.1 The isomophism γ

Proposition 6.2. *ξ_s is a bijection.*

Proof. Like in the previous chapter, the two sets $\mathbb{Z}_{n^s} \times \mathbb{Z}_n^*$ and $\mathbb{Z}_{n^{s+1}}^*$ have the same cardinality,

$$|\mathbb{Z}_{n^{s+1}}^*| = \varphi(n^{s+1}) = p^n(p-1)q^n(q-1) = n^s\varphi(n) = |\mathbb{Z}_{n^s}||\mathbb{Z}_n^*| = |\mathbb{Z}_{n^s} \times \mathbb{Z}_n^*|,$$

due to Theorem **??**, so we need to show that ξ_s is injective, that is, for $m_1, m_2 \in \mathbb{Z}_{n^s}$ and $r_1, r_2 \in \mathbb{Z}_n^*$,

$$\xi_s(m_1, r_1) = \xi_s(m_2, r_2) \text{ in } \mathbb{Z}_{n^{s+1}}^* \implies m_1 = m_2 \text{ in } \mathbb{Z}_{n^s} \ \wedge \ r_1 = r_2 \text{ in } \mathbb{Z}_n^*.$$

With the same arguments as in the previous chapter, we get:

$$\begin{aligned}
(n+1)^{m_1} r_1^{n^s} &\equiv (n+1)^{m_2} r_2^{n^s} \pmod{n^{s+1}} \\
(n+1)^{(m_2-m_1)\lambda(n)} &\equiv 1 \pmod{n^{s+1}} \\
(m_2-m_1)\lambda(n) &\equiv 0 \pmod{n^s} \\
m_2 - m_1 &\equiv 0 \pmod{n^s} \\
m_2 &\equiv m_1 \pmod{n^s},
\end{aligned}$$

and substitution then gives:

$$\begin{aligned}
\left(\frac{r_2}{r_1}\right)^{n^s} &\equiv 1 \pmod{n^{s+1}} \\
r_2^{n^s} &\equiv r_1^{n^s} \pmod{n^{s+1}} \\
r_2 &\equiv r_1 \pmod{n^{s+1}} \\
r_2 &\equiv r_1 \pmod{n}.
\end{aligned}$$

□

Proposition 6.3. *ξ_s defines an isomorphism from $\mathbb{Z}_{n^s} \times \mathbb{Z}_n^*$ to $\mathbb{Z}_{n^{s+1}}^*$.*

Proof. As in the previous chapter we only need to show the homomorphic property

$$\xi_s(m_1 + m_2, r_1 r_2) = \xi_s(m_1, r_1)\xi_s(m_2, r_2),$$

for $m_1, m_2 \in \mathbb{Z}_{n^s}$ and $r_1, r_2 \in \mathbb{Z}_n^*$, with left-hand side taking place in modulo n^{s+1} this time, and right-hand side taking place in modulo n^s for the messages and n for the random numbers. We then get:

$$\xi_s(m_1 + m_2 \bmod n^s, r_1 r_2 \bmod n) \equiv (n+1)^{m_1 + m_2 \bmod n^s} (r_1 r_2 \bmod n)^{n^s} \pmod{n^{s+1}}.$$

By Proposition 6.1 we know that $(n+1)$ has order n^s in $\mathbb{Z}^*_{n^{s+1}}$, so

$$(n+1)^{m_1+m_2 \bmod n^s} = (n+1)^{m_1+m_2} \bmod n^{s+1}.$$

We continue with

$$\begin{aligned}(r_1 r_2 \bmod n)^{n^s} &\equiv (r_1 r_2 + kn)^{n^s} \\ &\equiv \sum_{i=0}^{n^s} \binom{n^s}{i} (r_1 r_2)^{n^s - i} (kn)^i \\ &\equiv (r_1 r_2)^{n^s} + n^s (r_1 r_2)^{n^s - 1} (kn) \\ &\equiv (r_1 r_2)^{n^s} \pmod{n^{s+1}},\end{aligned}$$

and get as desired:

$$\begin{aligned}\xi_s(m_1 + m_2 \bmod n^s, r_1 r_2 \bmod n) &\equiv (n+1)^{m_1+m_2} (r_1 r_2)^{n^s} \\ &\equiv (n+1)^{m_1} (r_1)^{n^s} (n+1)^{m_2} (r_2)^{n^s} \\ &\equiv \xi_s(m_1, r_1) \xi_s(m_2, r_2) \pmod{n^{s+1}}.\end{aligned}$$ □

So the product of ciphertexts will decrypt to the sum of the messages. This will be a crucial property for the voting protocols in the next chapter.

6.2.2 The structure of $\mathbb{Z}^*_{n^{s+1}}$

As in Section 5.2.2, we can define two subgroups of $\mathbb{Z}^*_{n^{s+1}}$, namely:

$$\begin{aligned}M_D &= \{(n+1)^m \mid m \in \mathbb{Z}_{n^s}\} \\ R_D &= \left\{r^{n^s} \;\middle|\; r \in \mathbb{Z}^*_n\right\}.\end{aligned}$$

The proofs of this fact follows the exact same arguments as of the proof of Proposition 5.4 in Section 5.2.2. We can also follow the same arguments to show that $M_D \simeq \mathbb{Z}_{n^s}$ and $R_D \simeq \mathbb{Z}^*_n$.

We can then make the correspondently two factor groups, namely $\bar{M}_D$ and $\bar{R}_D$, defined by:

$$M_D = \mathbb{Z}^*_{n^{s+1}} / R_D = \{(n+1)^m R_D \mid m \in \mathbb{Z}_{n^s}\}$$

with elements $\bar{M}_{D_i} = \left\{(n+1)^{m_i} r^{n^s} \;\middle|\; r \in \mathbb{Z}^*_n\right\}$, for $1 \leq i \leq n^s$, and

$$\bar{R}_D = \mathbb{Z}^*_{n^{s+1}} / R_D = \left\{M_D r^{n^s} \;\middle|\; r \in \mathbb{Z}^*_n\right\},$$

with elements $\bar{R}_{D_i} = \left\{(n+1)^m r_i^{n^s} \;\middle|\; m \in \mathbb{Z}_{n^s}\right\}$, for $1 \leq i \leq \varphi(n)$.

We then get, as before:
The set of roots of $r_i^{n^s}$ in $\mathbb{Z}^*_{n^{s+1}}$ is the set $\bar{R}_{D_i}$, and all possible encryptions of the message m_i under the encryption algorithm $\mathcal{E}_D$ is the set $\bar{M}_{D_i}$. The factor group $\bar{M}_D$ gives rise to two functions, which we can combine as follows:

$$\begin{array}{rccccc} f_{\text{Class}_s}\colon & \mathbb{Z}^*_{n^{s+1}} & \xrightarrow{f_{s_1}} & \bar{M}_D & \xrightarrow{f_{s_2}} & \mathbb{Z}_{n^s} \\ & (n+1)^m r^{n^s} & \longmapsto & (n+1)^m R_D & \longmapsto & m. \end{array}$$

Definition 6.4. The problem $Class_s[n]$, is the problem of computing $f_{\text{Class}_s}(c)$ for a given $c \in \mathbb{Z}^*_{n^{s+1}}$.

It is worth noticing that each $m \in \mathbb{Z}_{n^s}$ can be written uniquely in n-adic notation as an s-tuple:

$$(m_s, m_{s-1}, \ldots, m_1) \;\in\; \mathbb{Z}_n \times \mathbb{Z}_n \times \cdots \times \mathbb{Z}_n,$$

with $m = \sum_{i=1}^{s} m_i n^{i-1}$.

We then get

$$m \in \mathbb{Z}_{n^s} \equiv m_1 \pmod{n} = m_1 \in \mathbb{Z}_n.$$

So starting with $m \in \mathbb{Z}_{n^s}$it is easy to find the corresponding $m_1 \in \mathbb{Z}_n$, but the other way around is not so easy. On the other hand, if we start with a message $m \in \mathbb{Z}_n$, we have:

$$m \in \mathbb{Z}_n \Rightarrow m \in \mathbb{Z}_{n^s}.$$

This will come useful later on.

6.3 The Decrypton Algorithm $\mathcal{D}_D$

The first part of $\mathcal{D}_D$ is decribed in Section 6.1 as:

$$c^d = \left((n+1)^m r^{n^s}\right)^d \bmod n^{s+1} = (n+1)^{md \bmod n^s} r^{n^s d \bmod \lambda(n)} = (n+1)^{md \bmod n^s}.$$

After making this computation, we need a way to extract $m' = md \bmod n^s$, before we end the algorithm with:

$$m = \frac{m'}{d} \bmod n^s.$$

To extract m', we can use Proposition 6.1 with $m' = a$ to get

$$\begin{aligned} c^d &= (n+1)^{m'} \\ &= 1 + \sum_{i=1}^{m'} \binom{m'}{i} n^i \\ &\Downarrow \\ \frac{c^d - 1}{n} &= \sum_{i=1}^{m'} \binom{m'}{i} n^{i-1} \\ &\equiv \sum_{i=1}^{s} \binom{m'}{i} n^{i-1} \pmod{n^s}, \end{aligned}$$

because $i > m' \Rightarrow \binom{m'}{i} = 0$, and $i > s \Rightarrow n^{i-1} \equiv 0 \pmod{n^s}$.

We will continue extracting m' from this number by induction on

$$\begin{aligned} m'_j &= m' \bmod n^j \\ &= \frac{c^d - 1}{n} \bmod n^{j+1} - \sum_{i=2}^{j} \binom{m'_j}{i} n^{i-1} \bmod n^j, \end{aligned}$$

for $1 \leq j \leq s$. We observe that

$$\begin{aligned} m_j &= m' \bmod n^j \\ &= m' \bmod n^{j-1} + kn^{j-1} \\ &= m'_{j-1} + kn^{j-1}, \end{aligned}$$

for some integer k.

The first induction step is easy because $j = 1 \Rightarrow \sum_{i=2}^{j} \binom{m'_j}{i} n^{i-1} \bmod n^j = 0$:

$$\begin{aligned} m'_1 &= m' \bmod n \\ &= \frac{c^d - 1}{n} \bmod n^2. \end{aligned}$$

For the j'th induction step, we already know the value m'_{j-1}. If we look at a term in $\sum_{i=2}^{j} \binom{m'_j}{i} n^{i-1} \bmod n^j$, we see that for $2 \leq i \leq j$

$$\begin{aligned}\binom{m'_j}{i} n^{i-1} \bmod n^j &\equiv \binom{m'_{j-1} + kn^{j-1}}{i} n^{i-1} \pmod{n^j} \\ &= \binom{m'_{j-1}}{i} n^{i-1} \bmod n^j,\end{aligned}$$

because the contribution from kn^{j-1} vanish modulo n^j after multiplication by n. From this we get the j'th induction step:

$$\begin{aligned}m'_j &= \frac{c^d - 1}{n} \bmod n^{j+1} - \sum_{i=2}^{j} \binom{m'_j}{i} n^{i-1} \bmod n^j \\ &= \frac{c^d - 1}{n} \bmod n^{j+1} - \sum_{i=2}^{j} \binom{m'_{j-1}}{i} n^{i-1} \bmod n^j.\end{aligned}$$

We can gather this induction in Algorithm 1.

Algorithm 1 Algorithm to extract m' from $(n+1)^{m'}$ by induction.

```
m' ← 0;
for j := 1 to s do                          ▷ The induction steps for 1 ≤ j ≤ s.
    t1 ← (c^d − 1)/n mod n^{j+1};
    t2 ← m'
    for i := 2 to j do                      ▷ Computing Σ_{i=2}^{j} C(m'_{j−1}, i) n^{i−1} mod n^j.
        m' ← m' − 1;
        t2 ← t2 · m' mod n^j;
        t1 ← t1 − t2·n^{i−1}/i! mod n^j;
    m' = t1;
return m'                                   ▷ Now we have m' from (n + 1)^{m'}
```

So it is clear that knowing the trapdoor d makes it possible to decrypt c in polynomial time. It is also possible to find $\xi_s(c)^{-1}$, that is, both decrypting c and finding the random variable r. For details on finding r, see [7]. The only thing remaining now is to state the following:

$$\mathcal{D}_D(dk, \mathcal{E}_D(ek, m)) = \mathcal{D}_D(d, c) = m.$$

6.4 The Security of DJN$_s$

As in the previous chapter, we have to show that the function ξ_s is one-way and that DJN$_s$ is semantically secure. We will do this by showing that ξ_s is one-way if γ of PA is one-way, and show that the semantic security is equivalent to DCRA (Assumption 1), as is also the case for PA.

PROPOSITION 6.5. *Class$_s[n]$ is random self-reducible over $c \in \mathbb{Z}^*_{n^{s+1}}$.*

Proof. Let $c \in \mathbb{Z}^*_{n^{s+1}}$ be a particular ciphertext. We have to show that if we can solve Class$_s[n]$ for a random $\tilde{c} \in \mathbb{Z}^*_{n^{s+1}}$, then we can also solve Class$_s[n]$ for any $c \in \mathbb{Z}^*_{n^{s+1}}$. So let

$$c = (n+1)^m r^{n^s}.$$

We can transform c into a random instance $\tilde{c}$ simply by randomly choosing $\tilde{m} \in \mathbb{Z}_{n^s}$ and $\tilde{r} \in \mathbb{Z}^*_n$ such that

$$\tilde{c} = c(n+1)^{\tilde{m}}\tilde{r}^{n^s} = (n+1)^{m+\tilde{m}}(r\tilde{r})^{n^s}.$$

Suppose we can solve Class$_s[n]$ for $\tilde{c}$. Then we can solve Class$_s[n]$ for c simply by computing $f_{\text{Class}_s}(\tilde{c}) - \tilde{m}$. Following the proof of Proposition 5.9, we know that our random instances will cover all of $\mathbb{Z}^*_{n^2}$. □

Following the lines of Section 5.1.1, we can show that Class$_s[n]$ is also random self-reducible over $g \in \mathbb{Z}^*_{n^{s+1}}$. For details, we refere to [7].

THEOREM 6.6. *If for some integer $j > 0$ the scheme DJN$_j$ is one-way, then DJN$_s$ is one-way for any integer $s > j$. Especially, DJN$_s$ is one-way for any s if Paillier's original scheme DJN$_1$ is one-way.*

Proof. The second part of the proposition follows from the first with $j = 1$, so it is enough to show that if DJN$_j$ is one-way then DJN$_s$ is one-way, with $s > j$. We will prove this by contradiction, so we assume that DJN$_j$ is one-way and DJN$_s$ is not, that is, there is an algorithm $\mathcal{A}$ which can compute $f_{\text{Class}_s}(c)$ for sufficient many $c \in \mathbb{Z}^*_{n^{s+1}}$. So let c_j be a ciphertext in $\mathbb{Z}^*_{n^{j+1}}$. We know that $c_j \in \mathbb{Z}^*_{n^{j+1}} \Rightarrow c \in \mathbb{Z}^*_{n^{s+1}}$, so let c_j now be a ciphertext in $\mathbb{Z}^*_{n^{s+1}}$. We can randomize c_j as described above, several times if necessary, until we find a randomized version of c_j which can be decrypted by $\mathcal{A}$. We can then get the decryption m of c_j modulo n^s, and by calculating m mod n^j we get the

decryption of c_j in $\mathbb{Z}^*_{n^{j+1}}$, which contradicts our assumption (for further details see the proof of Theorem 6.7). □

Theorem 6.7. *For any integer $s > 0$, the cryptosystem DJN$_s$ is semantically secure if and only if DCRA is true.*

Proof. We will use the RoR game to show semantic security here. Notice that DCRA is true $\Leftrightarrow$ PA is semantically secure, so we need to show that for all $s > 0$,

$$\text{DJN}_s \text{ is semantically secure} \Leftrightarrow \text{DJN}_1 \text{ is semantically secure.}$$

$\Rightarrow$: This is the same as proving that if DJN$_1$ is not semantically secure, then DJN$_s$ is not semantically secure. So assume that we have an adversary $\mathcal{A}_1$ against DJN$_1$. We will now create an adversary against DJN$_s$ as follows:

1. $\mathcal{A}_1$ start a RoR game in DJN$_1$ by choosing a message $\tilde{m}$ from $\mathbb{Z}_n$.
2. $\mathcal{A}_1$ send $\tilde{m}$ to us.
   (a) We transform the message $\tilde{m} \in \mathbb{Z}_n$ to a message $m \in \mathbb{Z}_{n^s}$ by choosing a $m \equiv \tilde{m} \pmod{n^s}$, i.e $m_(\tilde{m}, *, *, \ldots, *)$, where $* \in \mathbb{Z}_n$ is a random element.
   (b) We play the RoR game in with a challenger in DJN$_s$ to get a ciphertext $c \in \mathbb{Z}^*_{n^{s+1}}$.
   (c) We transform the ciphertext $c \in \mathbb{Z}^*_{n^{s+1}}$ to a ciphertext $\tilde{c} \in \mathbb{Z}^*_{n^2}$ by reducing in modulo n^2, i.e. $\tilde{c} = c \bmod n^2$.
   (d) We send $\tilde{c} \in \mathbb{Z}^*_{n^2}$ to $\mathcal{A}_1$.
3. $\mathcal{A}_1$ gets $\tilde{c}$ as response to the message $\tilde{m}$.
4. $\mathcal{A}_1$ no decides if $\tilde{c}$ is an encryption of $\tilde{m}$ or a random message with non-negligible advantage. Then
   (a) If $\mathcal{A}_1$ says $\tilde{c}$ is an encryption of $\tilde{m}$, then we say m.
   (b) If $\mathcal{A}_1$ says $\tilde{c}$ is an encryption of random message, then we say random message.

Now, if c is an encryption of m, then $\tilde{c}$ is an encryption of $\tilde{m}$. If c is an encryption of random message, then $\tilde{c}$ is an encryption of a random message. Since $\mathcal{A}_1$ has an advantage, then also we gets an advantage and in this way we have created an adversary against DJN$_s$.

$\Leftarrow$: We will not go into all the technical details here but rather sketch the concept, and refer the reader to [7] for details.

We have an adversary against DJN_s, lets call her $\mathcal{A}_s$. That is, $\mathcal{A}_s$ can choose a message $m \in \mathbb{Z}_{n^s}$, output it, then get a ciphertext $c \in \mathbb{Z}^*_{n^{s+1}}$ which is either a ciphertext of m or a random message $\tilde{m}$, and with more than negligible probability guess the right answer. That is, the advantage of $\mathcal{A}_s$ is non-negligible. We now want to create an adversary against DJN_1. Without loss of generality, we can assume that s is even. We will now start by making an adversary against $\text{DJN}_{s/2}$ and then use this to sketch the proof of how to get an adversary against DJN_1. (If s is odd, we will start by making an adversary against $DJN_{(s+1)/2}$ or $DJN_{(s-1)/2}$.) So we have the following:

1. $\mathcal{A}_s$ start a RoR game in DJN_s by choosing a message m from $\mathbb{Z}_{n^s}$.
2. $\mathcal{A}_s$ send m to us.
   (a) We transform the message $m \in \mathbb{Z}_{n^s}$ to a message $\tilde{m} \in \mathbb{Z}_{n^{s/2}}$.
   (b) We play the RoR game in with a challenger in $\text{DJN}_{s/2}$ to get a ciphertext $\tilde{c} \in \mathbb{Z}^*_{n^{(s/2)+1}}$.
   (c) We transform the ciphertext $\tilde{c} \in \mathbb{Z}^*_{n^{(s/2)+1}}$ to a ciphertext $c \in \mathbb{Z}^*_{n^{s+1}}$.
   (d) We send $c \in \mathbb{Z}^*_{n^{s+1}}$ to $\mathcal{A}_s$.
3. $\mathcal{A}_s$ gets c as response to the message m.
4. $\mathcal{A}_s$ no decide if c is an encryption of m or a random message with non-negligible advantage.

What we want is to turn the non-negligible advantage of $\mathcal{A}_s$ into a non-negligible advantage for us in the RoR game in $\text{DJN}_{s/2}$. Two problems arises:

1. How should we transform the message $m \in \mathbb{Z}_{n^s}$ to a message $\tilde{m} \in \mathbb{Z}_{n^{s/2}}$?
2. How should we transform the ciphertext $\tilde{c} \in \mathbb{Z}^*_{n^{(s/2)+1}}$ to a ciphertext $c \in \mathbb{Z}^*_{n^{s+1}}$?

Let $(m_1, \ldots, m_s)$ be the n-adic representation of m. We do not know how $\mathcal{A}_s$ is able to tell if a ciphertext is the encryption of m or random element, but imagine she is able to decrypt the ciphertext and then look at some of the positions in the $n-adic$ representation of the decryption, minimum one of the positions. She can then compare these positions to the corresponding positions in $(m_1, \ldots, m_s)$, and decides weather the ciphertext is a encryption of m or a random element. Our problem then becomes: When we transform the message $m \in \mathbb{Z}_{n^s}$ to a message

$\tilde{m} \in \mathbb{Z}_{n^{s/2}}$ we will loose some parts of it, i.e. some of the positions in the n-adic representation. How do we ensure that we do not loose the positions $\mathcal{A}_s$ needs? Since $\tilde{m} \in \mathbb{Z}_{n^{s/2}}$ have $\frac{s}{2}$ positions in the n-adic representation, so we can divide $(m_1, \ldots, m_s)$ in two parts:

$$(m_1, \ldots, m_{s/2}) \wedge (m_{s/2+1}, \ldots, m_s).$$

Let us assume that the positions $\mathcal{A}_s$ needs is either in the first part or in the second part[1]. Let $* \in \mathbb{Z}_n$ be a random element, and let $Enc(m_1, \ldots, m_s)$ be one instance of the distribution obtained by encryption $(m_1, \ldots, m_s)$. Les us now look at the three encryptions:

$$Enc(m_1, \ldots, m_{s/2}, m_{s/2+1}, \ldots, m_s). \tag{6.5}$$

$$Enc(m_1, \ldots, m_{s/2}, \quad * \quad, \ldots, \quad *). \tag{6.6}$$

$$Enc(\ * \ , \ldots, \quad * \ , \quad * \quad, \ldots, \quad *). \tag{6.7}$$

Since $\mathcal{A}_s$ has a non-negligible advantage of choosing between 6.5 and 6.7, we then get two cases:

1. The positions $\mathcal{A}_s$ needs is in the first half of the $n-adic$ representation of m. Then $\mathcal{A}_s$ can choose between 6.6 and 6.7 with non-negligible advantage.
2. The positions $\mathcal{A}_s$ needs is in the second half of the $n-adic$ representation of m. Then $\mathcal{A}_s$ can choose between 6.5 and 6.6 with non-negligible advantage.

In case 1:
When we get $m = (m_1, \ldots, m_s)$, we let $m \bmod n^{s/2} = (m_1, \ldots, m_{s/2})$. We output this and in return we get a ciphertext $\tilde{c} \in \mathbb{Z}^*_{n^{(s/2)+1}}$. We then have

$$\tilde{c} = Enc(m_1, \ldots, m_{s/2}) \ \vee \ \tilde{c} = Enc(*, \ldots, *).$$

We then make a random encryption $c' = Enc(0, \ldots, 0, *, \ldots, *)$, and get:

$$\tilde{c}c' = Enc(m_1, \ldots, m_{s/2}, *, \ldots, *) \ \vee \ \tilde{c}c' = Enc(*, \ldots, *, *, \ldots, *),$$

where the first part corresponds to $\tilde{c}$ being an encryption of $\tilde{m}$ and the second part corresponds to $\tilde{c}$ being an encryption of a random message in $\mathbb{Z}_{n^{s/2}}$. Since $\mathcal{A}_s$ has an

[1] If the positions are in both parts, then it is straightforward to just choose one of them, get a challenge ciphertext $\tilde{c} \in \mathbb{Z}_{n^{s/2}}$, transform it to a ciphertext $c \in \mathbb{Z}^*_{n^{s+1}}$ by adding random elements to the $\frac{s}{2}$ positions in the message part, as can be easily done by the homomorphic property of DJN$_s$, and then let $\mathcal{A}_s$ tell us if c is an encryption of m or a random message in $\mathbb{Z}_{n^s}$, which again will tell us if $\tilde{c}$ is an encryption of $\tilde{m}$ or a random message in $\mathbb{Z}_{n^{s/2}}$.

non-negligible advantage of choosing between these, we now have a non-negligible advantage of choosing between $\tilde{c} = Enc(m_1, \ldots, m_{s/2})$ or $\tilde{c} = Enc(*, \ldots, *)$.

In case 2:
When we get $m = (m_1, \ldots, m_s)$, we let $m \bmod n^{s/2} = (m_{s/2+1}, \ldots, m_s)$. We output this and in return we get a ciphertext $\tilde{c} \in \mathbb{Z}^*_{n^{(s/2)+1}}$. We then have

$$\tilde{c} = Enc(m_{s/2+1}, \ldots, m_s) \ \vee \ \tilde{c} = Enc(*, \ldots, *).$$

Since we are now looking at the second halv of the n-adic representation of $m \in \mathbb{Z}_{n^s}$, when going from the representation in $\mathbb{Z}_{n^{s/2}}$ to a representation in $\mathbb{Z}_{n^s}$, we need to "move" the values $n^{s/2}$ positions to the right. We can do this by computing $\tilde{c}^{n^{s/2}}$. We then get

$$\tilde{c}^{n^{s/2}} = Enc(0, \ldots, 0, m_{s/2+1}, \ldots, m_s) \ \vee \ \tilde{c}^{n^{s/2}} = Enc(0, \ldots, 0, *, \ldots, *).$$

We then make a random encryption $c' = Enc(m_1, \ldots, m_{s/2}, 0, \ldots, 0)$, and get:

$$\begin{aligned} \tilde{c}^{n^{s/2}} c' &= Enc(m_1, \ldots, m_{s/2}, m_{s/2+1}, \ldots, m_s) \\ &\vee \tilde{c}^{n^{s/2}} c' = Enc(m_1, \ldots, m_{s/2}, *, \ldots, *), \end{aligned}$$

where the first part corresponds to $\tilde{c}$ being an encryption of $\tilde{m}$ and the second part corresponds to $\tilde{c}$ being an encryption of a random message in $\mathbb{Z}_{n^{s/2}}$. Since $\mathcal{A}_s$ has an non-negligible advantage of choosing between these, we now have a non-negligible advantage of choosing between $\tilde{c} = Enc(m_{s/2+1}, \ldots, m_s)$ and $\tilde{c} = Enc(*, \ldots, *)$.

Combining these two cases gives us an adversary with non-negligible advantage against $\text{DJN}_{s/2}$.

To get an adversary against DJN_1, we notice that when we transform the message $m \in \mathbb{Z}_{n^s}$ to a message $\tilde{m} \in \mathbb{Z}_n$, we can only keep one of the positions. So let us assume that $\mathcal{A}_s$ only use one of the positions to choose between $Enc(m_1, \ldots, m_s)$ and $Enc(*, \ldots, *)$ (as all other possibilities will follow trivially from this).We can then start on first position and make the partition

$$(m_1) \wedge (m_2, \ldots, m_s).$$

From the discussion above, we know that $\mathcal{A}_s$ can choose with non-negligible advantage between either

1. $Enc(m_1, m_2, \ldots, m_s)$
   $Enc(m_1, *, \ldots, *)$
   or

2. $Enc(m_1, *, \ldots, *)$
   $Enc(*, * \ldots, *)$.

In the second case, we know that $\mathcal{A}_s$ need the position m_1 to get a non-negligible advantage, and so we must let $m = m_1 \bmod n$ and then create the ciphertexts $Enc(m_1, m_2, \ldots, m_s)$ or $Enc(m_1, *, \ldots, *)$ as above.
In the first case, we don't know which of the $m_2, \ldots, m_s$ that is the crucial position, so we have to make a new partition

$$(m_1, m_2) \wedge (m_3, \ldots, m_s),$$

which generates two new cases:

1. $Enc(m_1, m_2, m_3, \ldots, m_s)$
   $Enc(m_1, m_2 *, \ldots, *)$
   or
2. $Enc(m_1, m_2 *, \ldots, *)$
   $Enc(*, * \ldots, *)$.

In the second case of these new cases, we know that $\mathcal{A}_s$ need the position m_2 to get a non-negligible advantage, and so we must let $m = m_2 \bmod n$.
For the first case we once again must make two new cases.
The hierarchy of cases is illustrated in Figure 6.2. We can get an adversary against DJN_1 by combining all the cases. For a more elegant and generalized version of this proof, see the subgroup membership problem discussed by Gjøsteen in [16].

□

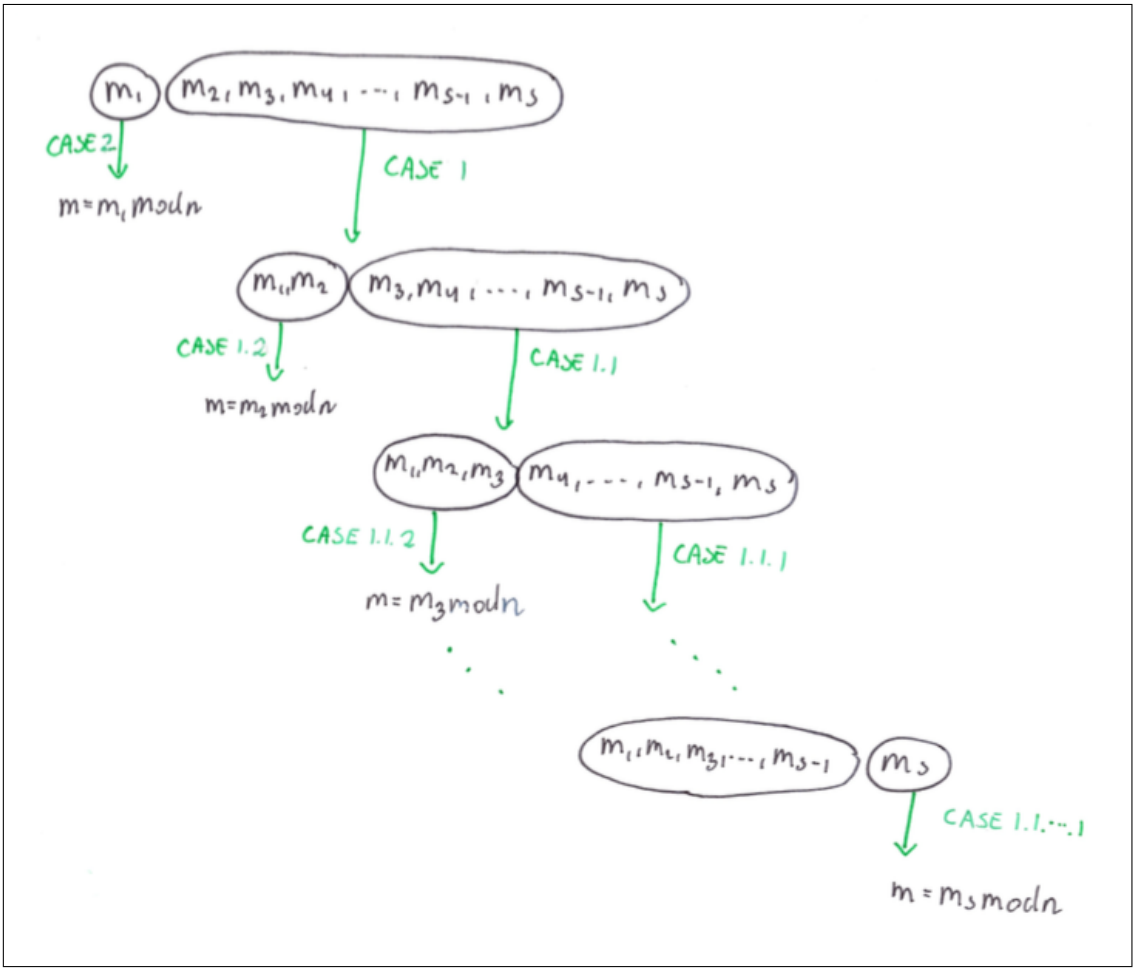


Figure 6.2: Creating an adversary against DJN_1 from an adversary against DJN_s.

7 An application to electronic voting

We will now look at an application of Damgård-Jurik-Nielsen cryptosystem, DJN_s, to electronic voting. This application is one of several kinds of electronic voting, namely homomorphic encryption-based electronic voting, and we will use the homomorphic property of DJN_s to build our voting protocols. We will start with some very simple protocols where we assume that both the voters and the authorities are honest but curious, and then continue by taking more and more security aspects, represented by malicious behavior, into account. In the end we will end up with the electronic voting protocol described in [7].

We will begin with some general notion about electronic voting.

7.1 Electronic Voting

In the literature there are many different definitions on both what an electronic voting protocol should accomplish, and on how the security of them should be analyzed. We will base this section on [20], [22] and [28].

Conventional voting is usually what we call *paper ballot* or *Australian ballot*. It was adopted in the late 19th century, and is regarded as the gold standard of voting [4]. Since the 19th century it has evolved in many different ways which vary from one country to the next, but they all have a common baseline consisting of a *set-up phase* (or *preparation phase*), a *voting phase* and a *counting phase* (or *tallying phase*). If we want to describe the voting process completely, there are many details we need to include. Instead we will

only include the most essential steps:

Set-up We need a register of valid voters, a polling station, the actual paper ballots, containers for the ballots (usually some kind of an envelope to provide secrecy of the ballot, and a ballot box that gather all the casted ballots) and trusted personell to prepare the polling station including sealing the boxes.

Voting phase First the voter is identified and authenticated, then she receives an envelope and enters a booth *alone* (this is mandatory if she wants it or not). She needs to get access to all the possible paper ballots and put the chosen ballot in the envelope in secrecy. This can be done in many different ways, but one way is to have a sufficient number of each ballot inside the booth so that she just picks the one she wants, and so that it is not possible to see which one she has taken without starting to count the ones that are left. She end her voting phase by casting the envelope in the ballot box.

Counting phase The sealed ballot boxes are counted and unsealed. Then all the envelopes are counted before the envelopes are removed. The paper ballots are examined and registered, and invalid ballots are annulled. The hole process is supervised by trusted (and also possible mistrusting) individuals. As an additional measure to preserve integrity one can check the total number of authenticated voters and match it to the total number of envelopes and total numbers of votes (including the annulled ones).

The reason for this voting process is that we want the election result to reflect the opinion of the authenticated voters. In the literature this is described with several requirements, some of them in contradiction to others. We will not take into account all the requirements in this chapter, but instead limit ourselves to the discussions in [20], [22] and [28].

Let us first look at the ideal world where both the voters and the trusted personell are honest. Then we need the following two requirements:

Completeness: The result of the election has to be consistent with the votes cast.

'Privacy': During the election, nobody will gain any information about any singel vote or subset of votes except their own private inputs and the final voting result.

Unfortunately, there is dishonesty in the world. Inputs can be leaked and outputs can be wrong. The security of an election protocol should however be similar to what can be achieved in the ideal world. So we will need some more requirements, including a rewriting of the privacy requirement:

Privacy: Only the final result is made public. No additional information about the votes will leak (except possible each voter's own private inputs).

Universal Verifiability: After the election, the result can be verified by anyone.

Robustness: Even if some voters and/or some of the entities running the election cheat, this will not effect the result. (The part of this requirement concerning the cheating voters is sometimes referd to as *soundness*.)

There are other requirements as well. One example is the *coercion-free* requirement, which means that no voter can later prove her vote. But universal verifiability means that each voter should be able to verify that their vote is correctly implemented, which in our case means that the voter will need knowledge of the random varible r from the encryption process. Knowing this varible, it is easy to prove the vote. So our system will not be coercion-free. Another example is the *eligibility* requirement, which means that only authorized parties should be able to vote. This is not difficult to implement, but we will solve it here by just stating that the bulletin board (defined below) will solve this problem. There is also a requirement called *unreusability*, which means that no voters should be able to vote more than the allowed times (usually one). We will assume that the bulletin board solves also this problem. In many actual elections it should be able to cast an empty vote, also called a NOTA (None of the Above). We will not take this into account in our systems, but we will make a notion about it in Section 7.2.3.

We will now look at the participants in our electronic voting systems, which will be the *voters* and the *authorities*. They can communicate with each other through public channels. A digital *bulletin board* serves as the place to post votes, proofs and results, as well as to authorize voters. We can define the following variables that will be used in out voting systems:

Voters: The strict upper bound of voters will be denoted by W, which means that the number of voters will be strictly less than W. An authorized voter do not need to participate in the voting. The voter i will be denoted by W_i. The totalt casted

votes are denoted v. We suppose that each voter can secretly store some amount of data in a secure place inaccessible to anyone except herself.

Authorities: The total number of authorities will be denoted by A, and A_i represent authority i. The minimum number of honest authorities will be denoted by H, and we assume that the honest authorities will do their prescribed work correctly and honestly.

Voting options: The voting options will be called the *candidates*, and in accordance with the above, the total number of candidates will be denoted by L. Candidate i will be denoted L_i. The total votes on candidate i will be denotet v_i. If there are only two candidates, we will call it a *yes/no* election, where 1 is "yes" and 0 is "no".

When it comes to the actual voting process, it will be the same three phases as above:

Set-up In this phase we define our participants and how to cast votes. This phase will also contain the key generation algorithm from our voting schemes in use.

Voting phase Each voter will be choosing between the L pre-decided number of candidates. The votes can be placed at a polling station or through internet, and the bulletin board will ensure that the eligibility and unreusability requirements holds. When the voter has chosen her candidate, the vote will be encrypted and posted on the bulletin board, possibly together with a proof of correctness.

Counting phase The authorities will collect all the encrypted votes, multiply them and decrypt the product to get the sum of the votes. This is possible due to the homomorphic property of DJN_s. It is important that the decryption process is guaranteed to be executed in such a way that no singel vote, or proper subset of votes, will be revealed. The result, possibly together with the product of the encrypted votes and proofs of correctness, will be posted on the bulletin board.

7.2 Electronic voting protocols in the semi-honest model

So we have three parties: the voters, the authorities and the (possible) adversaries. We will refer to the voters and the authorities as the *entities* of the protocol. Thereby we will have two distinct security issues, one is from the entities within the protocol and

another one is from the adversaries outside the protocol (but who possibly can interfere with the protocol). We will start with our simplest form of electronic voting protocols. They are based on the following assumption:

Assumption 3: *The voters and the authorities are honest but curious.*

When both the voters and the authorities are honest but curious, we call this a *semi-honest model*. And since they are curious, we call them *passively corrupt*. This is opposed to *actively corrupt* entities, which we get when we add dishonesty represented by *malicious* or *cheating* behavior [28].

The honest model, where both voters and authorities are honest, will be our "ideal-world". When we add curiosity, as in this section, or dishonesty, as in the next sections, our goal is always to achieve the same level of security as in the honest model.

Since the voters and authorities are curious in this model, the voters can not just post votes on the bulletin board and hope that their privacy will remain intact. Instead, the voters must encrypt the votes before publishing it. Since both the voters and the authorities are honest, we can assumes that they will follow the voting protocols properly and that non of them will cheat. So we only need to worry about curiosity and adversaries. For curiosity, we will see that the protocols in this section do not satisfy privacy. For adversaries, we will see that the security of the protocols only depend on the security of the underlying cryptosystem which in our case is DJN_s.

We will now describe three voting protocols in the semi-honest model. The first one is a yes/no-election, where there is only two possible choices (candidates) to choose between. The second is a 1-out-of-L-election, where the voter can choose up to one out of L possible candidates. The third protocol is a K-out-of-L-election, where the voter can choose K out of L possible candidates.

7.2.1 Yes/no-election

We start with a protocol for the yes/no-election. The electronic voting protocol is described in Figure 7.1.

The security of the voting protocol in Figure 7.1 relies on the security of DJN_s and Assumption 3. So we only need to discuss completeness and the version of privacy for

Voting protocol: Yes/no-election in the semi-honest model

Set-up Let W be the strict upper bound of voters, and let $L = 2$. A vote by voter W_i for "no" is represented by $m_i = 0$ and a vote for "yes" is represented by $m_i = 1$. The authorities will be represented by A. Use the encryption scheme DJN_s, with $\mathcal{E}_D$ and $\mathcal{D}_D$ as described by Figure 6.1 in Section 6.1.

1. A chooses admissible $n = pq$.
2. A chooses d such that $d \bmod n \in \mathbb{Z}_n^*$ and $d = 0 \bmod \lambda(n)$.

Voting phase Each W_i casts a vote m_i of 0 or 1 as follows:

1. Choose random $r_i \in \mathbb{Z}_n^*$.
2. Choose $m_i = 0$ or $m_i = 1$.
3. Compute $c_i = \mathcal{E}_D(n, m_i, r_i)$.
4. Post c_i on the bulletin board.

Counting phase

1. A computes $c = \prod_{i=1}^{v} c_i$.
2. A computes $\mathcal{D}_D(d, c) = m = \sum_{i=1}^{v} m_i$.
3. A posts the voting result m and the ciphertext c on the bulletin board.

Figure 7.1: A yes/no voting protocol with honest but curious voters and authorities.

honest but curious voters and authorities. No one except each voter themselves knows the random r_i, so the privacy requirement is fulfilled except for the curious authorities. They can easily decrypt each c_i to find each and every vote, and maybe, adding factors as time and identification, they will be able to connect some of the votes to its owner. This privacy issue concerns all the voting protocols in this section, and a solution will be discussed in Section 7.6.

Proposition 7.1. *The voting protocol of Figure 7.1 is complete.*

Proof. This follows from the decryption algorithm of DJN_s and the following equation:

$$\begin{aligned}\prod_{i=1}^{v} c_i &= \prod_{i=1}^{v} \left((n+1)^{m_i} r_i^{n^s} \bmod n^{s+1}\right) \\ &= (n+1)^{\sum_{i=1}^{v} m_i \bmod n^s} \left(\prod_{i=1}^{v} r_i \bmod n\right)^{n^s} \\ &= (n+1)^{\sum_{i=1}^{v} m_i} r^{n^s},\end{aligned}$$

where the last equality follows from the fact that for all practical purposes, $n > W$. It follows that $m = \sum_{i=1}^{v} m_i$ represents the votes for "yes" and $v - m$ represents the votes for "no". □

7.2.2 1-out-of-L-election

We will continue with a voting protocol were there are L possible candidates to vote for, and each voter can select only one of the candidates. It is based on the same assumption as in the previous section, where both entities are assumed to be honest but curious. The electronic voting protocol is described in Figure 7.2.

As in Section 7.2.1, the security of the voting protocol in Figure 7.1 relies on the security of DJN_s and Assumption 3. The privacy has the same concerns as in the previous section, and will be further discussed in Section 7.6. So we only need to look at completeness.

Proposition 7.2. *The voting protocol of Figure 7.2 is complete.*

Voting protocol: 1-out-of-L-election in the semi-honest model

Set-up Let W be the strict upper bound of voters, and let L be the number of candidates. A vote for candidate number $j \in \{0, \ldots, L-1\}$ is represented by the number W^j. The total votes for candidates $(0, \ldots, L-1)$ will be denoted by $(v_0, \ldots, v_{L-1})$ respectively. The authorities will be represented by A. Use the encryptions scheme DJN_s, with $\mathcal{E}_D$ and $\mathcal{D}_D$ as described by Figure 6.1 in Section 6.1, with $s \geq L \log_n W$.

1. A chooses admissible $n = pq$.
2. A chooses d such that $d \bmod n \in \mathbb{Z}_n^*$ and $d = 0 \bmod \lambda(n)$.

Voting phase Each voter i casts a vote j_i as follows:

1. Choose random $r_i \in \mathbb{Z}_n^*$.
2. Choose $j_i \in \{0, \ldots, L-1\}$.
3. Compute $c_i = \mathcal{E}_D(n, W^{j_i}, r_i)$.
4. Post c_i on the bulletin board.

Counting phase

1. A computes $c = \prod_{i=1}^{v} c_i$.
2. A computes $\mathcal{D}_D(d, c) = m = \sum_{j=0}^{L-1} v_j W^j$.
3. Posts $(v_0, \ldots, v_L - 1)$ and c on the bulletin board.

Figure 7.2: A 1-out-of-L voting protocol with honest but curious voters and authorities.

Proof. We need to take a closer look at the decryption process:

$$\begin{aligned}
c &= \prod_{i=1}^{v} c_i \\
&= \prod_{i=1}^{v} \mathcal{E}_D(n, W^{j_i}, r_i) \\
&= \prod_{i=1}^{v} \left((n+1)^{W^{j_i}} r_i^{n^s} \bmod n^{s+1} \right) \\
&= (n+1)^{\sum_{i=1}^{v} W^{j_i} \bmod n} \left(\prod_{i=1}^{v} r_i \bmod n^s \right)^{n^{s+1}} \\
&= (n+1)^{\sum_{i=1}^{v} W^{j_i}} r^{n^{s+1}},
\end{aligned}$$

where the last two equations follows from the homomorphic property of DJN_s and the fact that

$$s \geq L \log_n W \;\; \Leftrightarrow n^s \geq W^L$$

When decrypted we get:

$$\mathcal{D}_D(d, c) = \sum_{i=1}^{v} W^{j_i} = m.$$

Since $m < W^L$, it can be represented uniquely in (W)-adic notation as

$$m = \sum_{j=0}^{L-1} v_j W^j,$$

where each v_j is an integer with $v_j < W$. So each number v_j represent the totalt votes on candidate j. □

7.2.3 K-out-of-L-election

We will continue with a voting protocol were there are L possible candidates to vote for, and each voter can select up to K of the candidates. It is very similar to the 1-out-of-L and is based on the same assumption about honest but curious entities. We will include $K-1$ dummy candidates which will get possible votes which are not placed on a real candidate, for example if a voter only wishes to vote for 1 candidate instead of all the possible K candidates. The electronic voting protocol is described in Figure 7.3. Notice that $v_i = \sum_{k=1}^{K} W^{j_{i_k}} = \sum_{k=1}^{K} W^{j_{i_k}} \bmod n^s$.

Voting protocol: K-out-of-L-election in the semi-honest model

Set-up Let W be the strict upper bound of voters, let L be the number of candidates and let K be the number of candidates each voter can vote for. We will add $K-1$ dummy candidates to collect votes which are not placed on a real candidate. A vote for candidate number $j \in \{0, \ldots, \ldots, L+K-2\}$ is represented by the number W^j. The total votes for candidates $(0, \ldots, L+K-2)$ will be denoted by $(v_0, \ldots, v_{L+K-2})$ respectively. The authorities will be represented by A. Use the encryptions scheme DJN_s, with $\mathcal{E}_D$ and $\mathcal{D}_D$ as described by Figure 6.1, with $s \geq (L+K-2)\log_n W$.

1. A chooses admissible $n = pq$.
2. A chooses d such that $d \bmod n \in \mathbb{Z}_n^*$ and $d = 0 \bmod \lambda(n)$.

Voting phase Each voter i casts the vote v_i as follows:

1. Choose random $r_i \in \mathbb{Z}_n^*$.
2. Choose $(j_{i_1}, j_{i_2}, \ldots, j_{i_K})$ from $\{0, \ldots, L+K-2\}$.
3. Compute $v_i = \sum_{k=1}^{K} W^{j_{i_k}}$.
4. Compute $c_i = \mathcal{E}_D(n, v_i, r_i)$.
5. Post c_i on the bulletin board.

Counting phase

1. A computes $c = \prod_{i=1}^{v} c_i$.
2. A computes $\mathcal{D}_D(d, c) = m = \sum_{j=0}^{L-1} v_j W^j$.
3. Posts $(v_0, \ldots, v_L - 1)$ and c on the bulletin board.

Figure 7.3: A K-out-of-L voting protocol with honest but curious voters and authorities.

When we implement the system, the interface will probably be such that the voter do not see the dummy candidates. Instead the values $j_{i_k} \in \{L, \dots, L+K-2\}$ will be chosen by the system when the candidate do not wish to vote for any more candidates. It is also possible to add only one dummy candidate which will take all the dummy votes.

Since this voting protocol is very similar to the 1-out-of-L voting protocol, we will only show completeness of the protocol.

Proposition 7.3. *The voting protocol of Figure 7.3 is complete.*

Proof. From Section 7.2.2 we know that

$$c = (n+1)^{\sum_{i=1}^{v} v_i} r^{n^{s+1}},$$

and when decrypted we get:

$$\mathcal{D}_D(d, c) = \sum_{i=1}^{v} v_i = \sum_{i=1}^{v} \sum_{k=1}^{K} W^{j_{i_k}} = m.$$

Since $m < W^{L+K-1}$ (since each voter can only vote for each candidate once), it can ble represented uniquely in $(L+K-2)$-adic notation as

$$m = \sum_{j=0}^{L+K-2} v_j W^j,$$

where each v_j is an integer with $v_j < W$. So each number v_j represent the totalt votes on candidate j. □

Adding dummy candidates is also a solution to one of the problems in Section 7.1, namely the possibility to cast an empty vote (NOTA). We can just add one dummy candidate which will then represent the empty vote. In the case of a K-out-of-L-election, we will then first add the $K-1$ dummy candidates and then one extra where all the NOTAs will be placed.

If we want the voters to vote for exactly K candidates, this can simply be solved by just removing the dummy candidates and not leaving that as an option. We could also add only one dummy candidate for NOTAs, but this is to be decided by the criteria of the election.

Voting protocol: K-out-of-L-election in the semi-honest model (modified)

Set-up Let W be the strict upper bound of voters, let L be the number of candidates and let K be the number of candidates each voter can vote for. We will add $K-1$ dummy candidates to collect votes which are not placed on a real candidate. A vote for candidate number $j \in \{0, \ldots, L, \ldots, L+K-2\}$ is represented by the number W^j. The total votes for candidates $(0, \ldots, L+K-2)$ will be denoted by $(v_0, \ldots, v_{L+K-2})$ respectively. The authorities will be represented by A. Use the encryptions scheme DJN_s, with $\mathcal{E}_D$ and $\mathcal{D}_D$ as described by Figure 6.1 in Section 6.1, with $s \geq (L+K-2)\log_n W$.

1. A chooses admissible $n = pq$.
2. A chooses d such that $d \bmod n \in \mathbb{Z}_n^*$ and $d = 0 \bmod \lambda(n)$.

Voting phase Each voter i casts the votes j_{i_k} as follows:

1. For $k \in \{1, \ldots, K\}$:
   (a) Choose random $r_{i_k} \in \mathbb{Z}_n^*$.
   (b) Choose j_{i_k} from $\{0, \ldots, L+K-2\}$.
   (c) Compute $c_{i_k} = \mathcal{E}_D(n, W^{j_{i_k}}, r_{i_k})$.
2. Post $c_{i_1}, \ldots, c_{i_K}$ on the bulletin board.

Counting phase

1. A computes $c = \prod_{i=1}^{v} \prod_{k=1}^{K} c_{i_k}$.
2. A computes $\mathcal{D}_D(d, c) = m = \sum_{j=0}^{L-1} v_j W^j$.
3. Posts $(v_0, \ldots, v_L - 1)$ and c on the bulletin board.

Figure 7.4: A K-out-of-L voting protocol with honest but curious voters and authorities, modified version.

When we now continue this chapter, we will need to make a modification in the voting protocol of Figure 7.3. We will start to add cheating and malicious behavior, and suggest ways to solve these issues. To do this we will use zero-knowledge proofs which will be described in the next section. In this K-out-of-L voting protocol we will need one encryption for each of the votes each voter cast. This adjusted voting protocol is described in Figure 7.4. Note that even though there is a small change in the voting phase and counting phase, the correctness and security discussions we made above are exactly the same for the modified protocol. For the rest of this thesis, when talking about the K-out-of-L voting protocol in the semi-honest model, we are referring to this modified protocol.

7.3 Zero-Knowledge

So far we have made the assumption that both the voters and the authorities are honest but curious. In real life this is not necessarily true, so to build an electronic voting system that has any possible usage in real life, we have to take dishonesty into account.

When the electronic voting system is based on homomorphic encryption, as in our case, there is an extensive use of proofs of knowledge. Each voter has to prove that her ciphertext actually is the encryption of a vote for one of the possible candidates. And the authorities have to prove that they have followed the protocol properly. Suppose this was not the case and that a dishonest voter voted the number 10 in our yes/no voting scheme. It is easy to see that this would interrupt the whole voting process and in the end yield a false result. So proofs of knowledge is helping to make the system fulfill the robustness requirement.

An *interactive proof* protocol is a form of conversation between a prover, P, and a verifier, V, where they exchange multiple messages. P claim knowledge of a secret, and P's objective is to convince V about the truth of this claim. V either accepts or rejects the proof. An interactive proof is said to be a *proof of knowledge* if it has the following two properties [23]:

Definition 7.4. Completeness: If P is honest (i.e. the claim is true), then the honest V (that is, one following the protocol properly) should accept the proof with probability one [1].

[1] The definition in [23] uses the word *overwhelming* probability, and states: "(i.e. the verifier accepts

DEFINITION 7.5. Soundness: If P is cheating (i.e.the claim is false) then the honest V should only have a negligible probability of actually accepting the proof. If the honest V accepts the proof by the cheating P with non-negligible probability, then there exists a polynomial-time algorithm $\mathcal{A}$ which can extract P's secret or some equivalent with probability one [1] (which then again means that P knows the secret and is not cheating).

It is easy to prove knowledge of a secret by revealing the secret to the verifier. But to fulfill our privacy requirements in our electronic voting systems, we need the individual votes to remain secret. To solve this problem we need a *zero-knowledge* proof protocol. Zero-knowledge, denoted ZK, means that the verifier does not learn anything as a result of the proving process, except the validity of the claim. So a zero-knowledge proof protocol has an additional property to the two properties above, namely the zero-knowledge property:

DEFINITION 7.6. Zero-knowledge, ZK: If P is honest (i.e. the claim is true), no possibly cheating V learns anything other than this fact.

To show that an interactive proof protocol has ZK, one can show that there exists a polynomial-time simulator S which can simulate the whole transcript given only the claim to be proved, and no access to P, and that this transcript has the exact same probability distribution as actual transcripts between P and V. From Section **??** we know that this is the same as saying that real and simulated transcripts are indistinguishable [23]. We will in the rest of this chapter let S be a polynomial-time simulator.

It is important to mention that ZK does not guarantee that a protocol is secure. Similarly, soundness does not guarantee the security either. Neither property has much value unless the underlying problem faced by an adversary is computationally hard.

Ordinary ZK protocols are often unpractical for various reasons. As an example it can be very difficult to show ordinary soundness. But this discussion on ordinary ZK protocols is out of scope for this thesis. It is however possible to make some restrictions, which will lead us to a special type of protocols, namely the Σ-*protocols*.

---

the prover's claim). The definition of overwhelming depends on the application, but generally implies that the probability of failure is not of practical significance." Other literature states that the probability should be one, and for simplicity and without loss of significance we will use the word *one* here.

7.3.1 Σ-protocols

First we have to define what we want to prove. We will use the following problem as an example: Suppose P claims that c is the encryption of m, but V is skeptical. P then wants to prove that c is the encryption of m given by $c = (n+1)^m r^{n^s} \bmod n^{s+1}$, without revealing r.

In a Σ-protocol there is first some common input x for P and V, usually an instance of some computational problem. In our case this is c, n and s. P has some secret information, the private input, called a *witness* w for x. In our case, if P actually knows the secret, $w = r$. P now wants to prove her claim, which is a binary relation R between x and w, without revealing w. In our case this is proving the binary relation $c = (n+1)^m r^{n^s}$, without revealing r. P and V will now continue with the following three-moves conversation:

1. $P \to V$: commitment a.
2. $V \to P$: challenge, a random t-bit number e.
3. $P \to V$: response z.

In the end V decides to accept or reject the claim based on x and the conversation (a, e, z). We will define Σ-protocols below, but first we will look at some properties.

We will start by assuming that V is honest. There is then two restrictions we can make to the zero-knowledge property, which gives us two new properties. The first one is called honest-verifier zero-knowledge, denoted HVZK, and makes the restriction of the verifier being honest. The second one is called special honest-verifier zero-knowledge, denoted SHVZK, and is related to the Σ-protocols.

Definition 7.7. Honest-verifier zero-knowledge, HVZK: There exists S which on input x generates a conversation (a, e, z) which has same probability distribution as the real conversations between honest P and honest V on input x [6].

Definition 7.8. Special honest-verifier zero-knowledge, SHVZK : There exists S which on input x and e outputs an accepting conversation of the form (a, e, z). This conversation must have the same probability distribution as conversations between honest P and honest V on input x, and where V's challenge is e. [6].

Note that ordinary ZK only gets the claim as input and has no other interaction with

the P. Σ-protocols are not ZK, only SHVZK as we will se below.

We can also make a restriction on the soundness property, which gives us a property called special soundness. Also this is related to Σ-protocols.

DEFINITION 7.9. Special soundness: From any x and any pair of accepting conversation (a, e, z) and $(a, \tilde{e}, \tilde{z})$, where $e \neq \tilde{e}$, computing w is feasible.

We will now define the Σ-protocol.

DEFINITION 7.10. A *Σ-protocol* is a protocol of the same three-moves form as above, and which has the three properties completeness, special soundness and SHVZK.

THEOREM 7.11. *A Σ-protocol with challenge length t, is a proof of knowledge with knowledge error 2^{-t}.*

Proof. The proof is out of scope in this thesis, but we refer the interested reader to [6]. □

7.3.2 The Fiat-Shamir Heuristic

But why do we need Σ-protocols as they are not ZK [2]? The answer is to be find in Fiat and Shamir's seminal article "How To Prove Yourself: Practical Solutions to Identification and Signature Problems" from 1986 (published in 1987) [11]. In this article they show that the verifier's part can be replaced by a random function, and thus the scheme can be made non-interactive. This is often called the *Fiat-Shamir heuristic*.

So assume that we have access to a random oracle which initially chooses a random function $\mathcal{H}$ with the following parameters:

$$\mathcal{H} : \{0,1\}^l \rightarrow \{0,1\}^t,$$

where l is the length of the binary representation of a, and t is the length of the binary representation of the challenge. We can then outline the idea as follows [6]:

[2] There are also other answers to this question. One of these is transforming Σ-protocols into a proper ZK-protocol using commitment schemes. We will not go into this here, but refer the interested reader to [19]

1. $P \rightarrow$ random orcale: a (an l-bit number).
2. random oracle $\rightarrow P$: $e = \mathcal{H}(a)$ (a t-bit number).
3. P: computes z and then outputs (a, z).

Any verifier can now call the oracle with a as input to get e and then check the answer z as above.

One important requirement for being able to use Fiat-Shamir heuristic, is that the Σ-protocol has to be *public coin*, as opposed to *private coin*. The constructions of public coin protocols was introduced in 1985 by Babai in an proving game called *Arthur-Merlin game* (Arthur is a king, and Merlin is a supernatural intellectual in whom the king do not trust, so Merlin has to convince him) [2]. The term "public" refers to a specific coin tossing situation, which is the equivalence of choosing the random challenges in our Σ-protocol. Babai made a proving system where this coin tossing was public, as opposed to the private coin protocols defined the same year by Goldwasser, Micali and Rachopp [17], were the coin tossing can be done in private. The Σ-protocol is public coin as the random challenge is public and do not depend on the commitment a.

As we have discussed in Section 3.6, the random functions we need does not exist. But we can move to the random oracle model where we assume that hash functions are random oracles, as we know from Section 4.4.1. In the original paper from Fiat and Shamir, they did not include a proof of security. But in 1996, David Pointcheval and Jacques Stern proved the Fiat-Shamir heuristic is secure against ciphertext-only attacks and chosen-plaintext attacks in this random oracle model. We will not go into the details of this proof here, but refer the interested reader to [25].

So we are now in the random oracle model, and have a hash function which we assume is a random oracle $\mathcal{H}$ as described above. We can than turn a Σ-protocol into the following one-move proof:

1. P: computes $(a, \mathcal{H}(a), z)$ and outputs (a, z).

Any verifier can now calculate $\mathcal{H}(a)$ and then check z as above[3].

[3] In practice we also include all the public values as inputs to the hash function. When this is done, we prevent that the same proof can be reused in another setting. We will include the public values in the sections to come.

In [6], Damgård states that the above proof is a ZK-proof. The reason behind this is that the random oracle can be view as a honest verifier. We will briefly look into the discussion he makes, and refer the interested reader to [6] for details:

- Completeness: Follows from the discussion above (the random oracle just replaces V's part in the protocol, and after, any verifier can then call the oracle to compute $\mathcal{H}(a)$ and check z).
- Soundness: P has no information about e before he has sent a. So the only difference here is that P is free to call the oracle as many times as she wants in the hope of getting some challenge she can answer. But if the number of challenges are exponentially large, this is not a feasible strategy (as P is ppt bounded).
- ZK: The random oracle forces V to be honest, because the e's are always randomly and independent chosen. So it is enough to show that the proof is HVZK, and by this Σ-protocols are automatically ZK in the random oracle model. More formally, we define ZK in this model by the following: we allow S to decide what the oracle response should be as long as they have the same distribution as in real life. So then one can just choose e at random and run S to get (a, e, z), and define the oracle's response on input a to be z, and output a, z.

The definition of ZK as above, is also referred to as *non-interactive ZK*, abbreviated *NIZK*.

In the following, we will denote the proofs described above by *non-interactive ZK proofs*, abbreviated *NIZK proofs*. We are now ready to look at the actual Σ-protocols and NIZK-proofs we will use in the electronic voting protocols.

7.4 Σ-protocols and NIZK Proofs

We will now look at five actual Σ-protocols and their corresponding NIZK-proofs. The first protocol will be used to prove the example in Section 7.3.1, the second protocol will be used to prove a vote in the yes/no voting protocol and the third will be used to prove votes in the 1-out-of-L and K-out-of-L voting protocols. The fourth protocol will be used in the K-out-of-L voting protocol to prove that the votes from a voter are pairwise distinct, and the last protocol will be related to the threshold decryption protocol in Section 7.6. We will first define a Σ-protocol and then its corresponding

NIZK-proof. We will then prove the Σ-protocol and from the previous section we know that the NIZK-proof follows directly in the random oracle model.

All the Σ-protocols and NIZK-proofs will have a security parameter t, which will be the lengt of the challenge, where 2^t will be less than the smallest of p, q. When we are invoking the encryption algorithm $\mathcal{E}_D$, we will omit the value n in the input. This is done to enhance the readability, and we assume $\mathcal{E}_D$ is invoked also on n.

7.4.1 Encryption of 0

Suppose P wants to prove that a ciphertext c is the encryption of a message m without revealing some secret information. That is, P wants to prove the knowledge of $\tilde{r}$ in $c = (n+1)^m \tilde{r}^{n^s}$ without revealing the actual value. This is the same as proving that $x = c(n+1)^{-m} \bmod n^{s+1}$ is an encryption of 0, or equivalently that it is an n^s'th power. The witness for this relation will be $w = \tilde{r}$.The Σ-protocol is described in Figure 7.5 and the corresponding NIZK-proof in Figure 7.6.

In the verifying phase, V has to check that x, a, z are all relatively prime to n. If this is not the case, then eiter one or more of them is 0, or non of them is 0 but one or more of them is a multiplum of p or q . In the former case, if as an example a is 0, then r is necessarily 0 and hence z is 0. Then $\mathcal{E}_D(0, z) \equiv ax^e \pmod{n^{s+1}} = 0$ and it is possible for P to get an accepting conversation without knowing w. in the latter case, somebody has to know the factoring of n, as finding numbers not relatively prime to n simply by guessing is only possible with negligible probability. And this somebody might be trying to trick the conversation. In both cases, the relatively prime requirement is to ensure that all the values are in the groups they need to be. If not, it might be possible to make an accepting proof without knowing w. This discussion goes for all the Σ-protocols in this section, and will not be repeated for each protocol.

PROPOSITION 7.12. *The protocol of Figure 7.5 is a Σ-protocol.*

Proof. **Completeness:** We have to show that if P knows the secret then the honest V will accept, that is, the equation in the verifying fase must hold.

$$\mathcal{E}_D(0, z) \equiv \mathcal{E}_D(0, rw^e) \equiv \mathcal{E}_D(0, r)\mathcal{E}_D(0, w)^e \equiv ax^e \pmod{n^{s+1}}.$$

Special soundness: Suppose (a, e, z) and $(a, \tilde{e}, \tilde{z})$ are two accepting conversations with

$e \neq \tilde{e}$. We then get:

$$\begin{aligned} &\mathcal{E}_D(0,z) \equiv ax^e \pmod{n^{s+1}} \ \wedge \ \mathcal{E}_D(0,\tilde{z}) \equiv ax^{\tilde{e}} \pmod{n^{s+1}} \\ \Rightarrow\ &\mathcal{E}_D(0, \frac{z}{\tilde{z}} \bmod n) = x^{e-\tilde{e}} \bmod n^{s+1}. \end{aligned}$$

By the assumption on 2^t we know that $\gcd((e-\tilde{e}), n) = 1$ and so there exist α, β such that $\alpha n^s + \beta(e-\tilde{e}) = 1$ and which can be calculated in polynomial time. Now, let $\tilde{x} = x \bmod n$ and $\tilde{w} = \tilde{x}^\alpha \left(\frac{z}{\tilde{z}}\right)^\beta \bmod n$. We now want to show that $\tilde{w}$ is P's secret, that is, $w = \tilde{w}$. First we will notice that:

$$\mathcal{E}_D(0,\tilde{x}) = \mathcal{E}_D(0, x \bmod n) = x^{n^s}.$$

We then get:

$$\mathcal{E}_D(0,\tilde{w}) = \mathcal{E}_D\left(0, \tilde{x}^\alpha \left(\frac{z}{\tilde{z}}\right)^\beta\right) = \mathcal{E}_D(0,\tilde{x})^\alpha \mathcal{E}_D\left(0, \frac{z}{\tilde{z}}\right)^\beta = x^{\alpha n^s} x^{\beta(e-\tilde{e})} = x \bmod n^{s+1}.$$

So we see that $x = \mathcal{E}_D(0,\tilde{w})$ and so $\tilde{w} = w$.

SHVZK: The input for the simulator S is $x \in \mathbb{Z}^*_{n^{s+1}}$, n, s and a challenge e. We can then choose a random $z \in \mathbb{Z}^*_n$ and then let

$$a = \mathcal{E}_D(0,z)x^{-e} \bmod n^{s+1},$$

to get an accepting conversation (a, e, z). This has the same probability distribution as conversations between P and V. This is so because in a real conversation P chooses uniformly at random r and computes a and z. In the simulated conversation, S chooses uniformly at random z and computes a. We have only switched the random variable from r to z, and since they both are chosen from the same group (modulo n and relatively prime to n) the probability distributions of real and simulated conversations are the same.

□

7.4.2 1-out-of-2 Is the Encryption of 0

Suppose there is a yes/no-election and a voter wants to prove that she voted for one of the two candidates without revealing which candidate it was. That is, P wants to prove that a ciphertext c is the encryption of either m_1 or m_2. If this is true, then either

Σ-protocol: Encryption of 0

Common input: $x \in \mathbb{Z}^*_{n^{s+1}}$, n and s.

Private input for P: $w \in \mathbb{Z}^*_n$ such that $x = \mathcal{E}_D(0, w)$.

1. $P \to V$: P chooses random $r \in \mathbb{Z}^*_n$ and sends $a = \mathcal{E}_D(0, r)$ to V.
2. $V \to P$: V chooses at random a t-bit number e and sends it to P.
3. $P \to V$: P sends $z = rw^e \bmod n$ to V.

Verifing phase : V checks that x, a, z are relatively prime to n and accepts if and only if $\mathcal{E}_D(0, z) \equiv ax^e \pmod{n^{s+1}}$.

Figure 7.5: Σ-protocol for x being the encryption of 0.

NIZK: Encryption of 0

1. P: computes $(a, \mathcal{H}(a, x_1, n, s), z)$ and outputs (a, z).

Any verifier can now calculate $\mathcal{H}(a, x_1, n, s)$ and check z as above.

Figure 7.6: The NIZK-proof of the encryption of 0.

$x_1 = c(n+1)^{-m_1}$ or $x_2 = c(n+1)^{-m_2}$ is the encryption of 0, as we know from the previous section. What also follows from the previous section, from the proof of Theorem 7.14, is that we have a simulator S which can simulate conversations with same probability as real conversations. We can use S to simulate a conversation for the one of x_1, x_2 which is not the encryption of 0. Without loss of generality we can assume that the prover knows w for $x_1 = c(n+1)^{-m_1}$, such that $x_1 = \mathcal{E}_D(0, w)$. The simulator S then calculates the second commitment as $a_2 = \mathcal{E}_D(0, z_2)x_2^{-e_2}$. The Σ-protocol is described in Figure 7.7 and the corresponding NIZK-proof in Figure 7.8. Step 3.1. is important because if $e_1 = \tilde{e}$, V would be able to find out for which of the relations P knows a witness.

PROPOSITION 7.13. *The protocol of Figure 7.7 is a Σ-protocol.*

Proof. **Completeness:** We have to show that if P knows the secret then the honest V will accept, that is, the equation in the verifying fase must hold.

1. It is obvious that $\tilde{e} = e_1 + e_2 \bmod 2^t$.
2. $\mathcal{E}_D(0, z_1) \equiv \mathcal{E}_D(0, r_1 w^{e_1}) \equiv \mathcal{E}_D(0, r_1)\mathcal{E}_D(0, w)^{e_1} \equiv a_1 x_1^{e_1} \pmod{n^{s+1}}$.

3. As stated in the beginning of this section, $a_2 = \mathcal{E}_D(0, z_2)x_2^{-e_2}$ and this is equivalent to $\mathcal{E}_D(0, z_2) \equiv a_2 x_2^{e_2}$.

Special soundness: Suppose $(a_1, a_2, e_1, e_2, z_1, z_2)$ and $(a_1, a_2, \tilde{e_1}, \tilde{e_2}, \tilde{z_1}, \tilde{z_2})$ are two accepting conversations with either $e_1 \neq \tilde{e_1}$ or $e_2 \neq \tilde{e_2}$. To extract witness we follow the lines of the special soundness proof in the previous section. We then get:

$$\mathcal{E}_D(0, z_1) \equiv a_1 x_1^{e_1} \pmod{n^{s+1}} \ \wedge \ \mathcal{E}_D(0, \tilde{z_1}) \equiv a_1 x_1^{\tilde{e_1}} \pmod{n^{s+1}}$$
$$\Rightarrow \mathcal{E}_D(0, \frac{z_1}{\tilde{z_1}} \bmod n) = x_1^{e_1 - \tilde{e_1}} \bmod n^{s+1}.$$

By the assumption on 2^t we know that $\gcd((e_1 - \tilde{e_1}), n) = 1$ and so we can find α, β in polynomial time such that $\alpha n^s + \beta(e_1 - \tilde{e_1}) = 1$. Now, let $\tilde{x_1} = x_1 \bmod n$ and $\tilde{w} = \tilde{x_1}^{\alpha} \left(\frac{z_1}{\tilde{z_1}}\right)^{\beta} \bmod n$. We now want to show that $\tilde{w}$ is P's secret, that is, $w = \tilde{w}$. First we will notice that:

$$\mathcal{E}_D(0, \tilde{x_1}) = \mathcal{E}_D(0, x_1 \bmod n) = x_1^{n^s}.$$

We then get:

$$\mathcal{E}_D(0, \tilde{w}) = \mathcal{E}_D\left(0, \tilde{x_1}^{\alpha}\left(\frac{z_1}{\tilde{z_1}}\right)^{\beta}\right) = \mathcal{E}_D(0, \tilde{x_1})^{\alpha}\mathcal{E}_D\left(0, \frac{z_1}{\tilde{z_1}}\right)^{\beta} = x_1^{\alpha n^s} x_1^{\beta(e_1 - \tilde{e_1})} = x_1 \bmod n^{s+1}.$$

So we see that $x_1 = \mathcal{E}_D(0, \tilde{w})$ and so $\tilde{w} = w$.

We know that $e_i - \tilde{e}_i = 0$ for one of $i = 1 \vee i = 2$, so if the above did not extract a witness, then we can extract a witness w' for the relation

$$x_2 = \mathcal{E}_D(0, w'),$$

in the exact same way as above.

SHVZK: We have already used S to simulate (x_2, e_2, z_2), so this simulation is obvious. For the simulation of a conversation on x_1, we do exactly as in the previous section: We invoke S by $x_1 \in \mathbb{Z}^*_{n^{s+1}}$, n, s and a challenge $e_1 = \tilde{e} - e_2$. The simulator can then choose a random $z_1 \in \mathbb{Z}^*_n$ and let

$$a_1 = \mathcal{E}_D(0, z_1)x_1^{-e_1} \bmod n^{s+1},$$

to get an accepting conversation (a_1, e_1, z_1). We unite these to get an accepting conversation $(a_1, a_2, e_1, e_2, z_1, z_2)$, and from the discussion in the previous section we know that simulated conversations have the same probability distribution as real conversations between P and V.

□

Σ-protocol: 1-out-of-2

Common input: $x_1, x_2 \in \mathbb{Z}^*_{n^{s+1}}$, n and s.

Private input for P: $w \in \mathbb{Z}^*_n$ such that $x_1 = \mathcal{E}_D(0, w)$.

1. $P \to V$:
   1.1. P chooses random $r_1 \in \mathbb{Z}^*_n$.
   1.2. P chooses random t-bit challenge e_2.
   1.3. P invokes S on input n, x_2, e_2 to get a conversation (a_2, e_2, z_2).
   1.4. P sends $a_1 = \mathcal{E}_D(0, r_1)$ and a_2 to V.
2. $V \to P$: V chooses random t-bit number $\tilde{e}$ and sends it to P.
3. $P \to V$:
   3.1. P computes $e_1 = \tilde{e} - e_2 \bmod 2^t$.
   3.2. P computes $z_1 = r_1 w^{e_1} \bmod n$.
   3.3. P sends e_1, e_2, z_1, z_2 to V.

Verifing phase : V checks that $x_1, x_2, a_1, a_2, z_1, z_2$ all are relatively prime to n and that:

1. $\tilde{e} = e_1 + e_2 \bmod 2^t$
2. $\mathcal{E}_D(0, z_1) \equiv a_1 x_1^{e_1} \pmod{n^{s+1}}$
3. $\mathcal{E}_D(0, z_2) \equiv a_2 x_2^{e_2} \pmod{n^{s+1}}$.

Figure 7.7: Σ-protocol for 1-out-of-2 is the encryption of 0.

NIZK: 1-out-of-2

1. P: computes $(a_1, a_2, \mathcal{H}(x_1, x_2, a_1, a_2, n, s), z_1, z_2)$ and outputs (a_1, a_2, z_1, z_2).

Any verifier can now calculate $\mathcal{H}(x_1, x_2, a_1, a_2, n, s)$ and check z_1, z_2 as above.

Figure 7.8: The NIZK-proof of 1-out-of-2 is the encryption of 0.

7.4.3 1-out-of-L Is the Encryption of 0

We will now look at an election with L candidates where a voter wants to prove that an encryption c is a vote for one of the candidates. We will follow the lines from the previous subsection, and so P wants to prove that a ciphertext c is the encryption of one of $m_1, m_2, \ldots, m_L$ (when we apply this to the voting protocols in the next sections, we will let $m_{j+1} = W^j$, where W is the strict upper bound of voters). From the previous subsections we know that this is the same as showing that one of

$$\begin{aligned} x_1 &= c(n+1)^{-m_1} \\ x_2 &= c(n+1)^{-m_2} \\ &\vdots \\ x_L &= c(n+1)^{-m_L} \end{aligned}$$

is an encryption of 0. Without loss of generality we can assume that P knows w for $x_1 = c(n+1)^{-m_1}$, such that $x_1 = \mathcal{E}_D(0, w)$. We will then let S simulate the $L-1$ conversations for $x_2, \ldots, x_L$, as we did for one conversation in the previous subsection. The Σ-protocol is described in Figure 7.9 and the corresponding NIZK-proof in Figure 7.10.

PROPOSITION 7.14. *The protocol of Figure 7.9 is a Σ-protocol.*

Proof. We will not prove this here, as the proof is almost identical to the proof of Theorem 7.13. □

Σ-protocol: 1-out-of-L

Common input: $x_1, \ldots, x_L \in \mathbb{Z}^*_{n^{s+1}}$, n and s.

Private input for P: $w \in \mathbb{Z}^*_n$ such that $x_1 = \mathcal{E}_D(0, w)$.

1. $P \to V$:
   (a) P chooses random $r_1 \in \mathbb{Z}^*_n$.
   (b) For each $i \in \{2, \ldots, L\}$:
      i. P chooses random t-bit challenges e_i.
      ii. P invokes S on input n, x_i, e_i to get a conversation (a_i, e_i, z_i).
   (c) P sends $a_1 = \mathcal{E}_D(0, r_1), a_2, \ldots, a_L$ to V.
2. $V \to P$: V chooses random t-bit number $\tilde{e}$ and sends it to P.
3. $P \to V$:
   (a) P computes $e_1 = \tilde{e} - \left(\sum_{i=2}^{L} e_i\right) \bmod 2^t$.
   (b) P computes $z_1 = r_1 w^{e_1} \bmod n$.
   (c) P sends $e_1, \ldots, e_L, z_1, \ldots, z_L$ to V.

Verifing phase : V checks that $x_1, \ldots, x_L, a_1, \ldots, a_L, z_1, \ldots z_L$ all are relatively prime to n and that:

1. $\tilde{e} = \left(\sum_{i=1}^{L} e_i\right) \bmod 2^t$.
2. $\mathcal{E}_D(0, z_i) \equiv a_i x_i^{e_i} \pmod{n^{s+1}}$ for each $i \in \{2, \ldots, L\}$.

Figure 7.9: Σ-protocol for 1-out-of-L is the encryption of 0.

NIZK: 1-out-of-L

1. P: computes $(a_1, \ldots, a_L, \mathcal{H}(x_1, \ldots, x_L, a_2, \ldots, a_L, n, s), z_1, \ldots, z_L)$ and outputs $(a_1, \ldots, a_L, z_1, \ldots, z_L)$.

Any verifier can now calculate $\mathcal{H}(x_1, \ldots, x_L, a_2, \ldots, a_L, n, s)$ and check $z_1, \ldots, z_L$ as above.

Figure 7.10: The NIZK-proof of 1-out-of-L is the encryption of 0.

7.4.4 The relation $\mathbf{ab = c \bmod n^s}$ between plaintexts

In a K-out-of-L-election we might want to ensure that no one has voted on the same candidate several times. We will explain how we can solve this in Section **??**, but to do this we need to be able to prove in ZK that the product of two plaintext equals another plaintext, that is, for plaintexts $a, b, c \in \mathbb{Z}_{n^s}$, $ab = c \bmod n^s$. We will now study a protocol which does exactly this. First we will make some comments on notation. To not confuse the commitments with the a in $ab = c$, we will denote the commitments by $\tilde{a}_1$ and $\tilde{a}_2$. We also have the following:

$$
\begin{aligned}
&x_a = \mathcal{E}_D(a, r_a) \\
&x_{ab} = \mathcal{E}_D(ab, r_{ab}) && (r_{ab} \text{ is the random variable of the encryption of } ab). \\
&x_a^b = \mathcal{E}_D(ab, r_a^b) && (\text{normal exponentiation of } x_a \text{ in } b). \\
&x_a x_b = \mathcal{E}_D(a + b, r_a r_b) && (\text{normal multiplication of ciphertexts}).
\end{aligned}
$$

The Σ-protocol is described in Figure 7.11 and the corresponding NIZK-proof in Figure 7.12

PROPOSITION 7.15. *The protocol of Figure 7.11 is a Σ-protocol.*

Proof. **Completeness:** We have to show that if P knows the secret then the honest V will accept, that is, the equations in the verifying fase must hold. We start by noticing that V knows the common input x_a, x_b, x_z, the commitments $\tilde{a}_1, \tilde{a}_2$, the challenge e and the responses z_1, z_2, z_3. Hence she can compute

$$x_a^e \tilde{a}_1 \ \wedge \ x_b^{z_1} (\tilde{a}_2 x_c^e)^{-1} \ \wedge \ \mathcal{E}_D(z_1, z_2) \ \wedge \ \mathcal{E}_D(0, z_3),$$

and check the equations of the verifying phase. We can then notice that the plaintext part of

$$x_b^{z_1} (\tilde{a}_2 x_c^e)^{-1} = x_b^{z_1} (x_{db} x_c^e)^{-1}$$

equals:

$$z_1 b - (db + ec) \equiv (ea + d)b - (db + ec) \equiv eab + db - db - ec \equiv eab - ec \pmod{n^s}.$$

It is easy to see that this equals 0 iff $ab \equiv c \pmod{n^s}$.

Special soundness: Suppose $(\tilde{a}_1, \tilde{a}_2, e, z_1, z_2, z_3)$ and $(\tilde{a}_1, \tilde{a}_2, \tilde{e}, \tilde{z}_1, \tilde{z}_2, \tilde{z}_3)$ are two accepting conversations with $e \neq \tilde{e}$. We know that an encryption uniquely determine

the plaintext, and since the conversations are accepting we get:

$$\begin{aligned} & z_1 b - db - ec \equiv 0 \equiv \tilde{z}_1 b - db - \tilde{e}c \pmod{n^s} \\ \Rightarrow & (z_1 - \tilde{z}_1)b \equiv (e - \tilde{e})c \pmod{n^s} \\ \Rightarrow & (ea + d - (\tilde{e}a + d))b \equiv (e - \tilde{e})c \pmod{n^s} \\ \Rightarrow & (e - \tilde{e})ab \equiv (e - \tilde{e})c \pmod{n^s}. \end{aligned}$$

Again, the restriction on 2^t gives us $\gcd((e - \tilde{e}), n) = 1$, and hence $(e - \tilde{e})$ is invertible in $\mathbb{Z}_{n^s}$. We then get

$$ab = c \bmod n^s.$$

SHVZK: The input for the simulator S is the common input and a challenge e. We can then choose a random $z_1 \in \mathbb{Z}_{n^s} z_2, z_3 \in \mathbb{Z}_n^*$ and then let

$$\begin{aligned} \tilde{a}_1 &= \mathcal{E}_D(z_1, z_2) \left(x_a^e\right)^{-1}. \\ \tilde{a}_2 &= x_b^{z_1} \left(x_c^e \mathcal{E}_D(0, z_3)\right)^{-1}. \end{aligned}$$

Since $z_1 \in \mathbb{Z}_{n^s}, z_2, z_3 \in \mathbb{Z}_n^*$ are chosen independently and uniformly at random from their groups, they have the same probability distribution as $d \in \mathbb{Z}_{n^s}, r_d, r_{bd} \in \mathbb{Z}_n^*$ which are also chosen uniformly at random from the same groups respectively. So as before we are only switching the variables of the conversation from d, r_d, r_{bd} to z_1, z_2, z_3. This leads to $(\tilde{a}_1, \tilde{a}_2, e, z_1, z_2, z_3)$ having the same probability distribution as real conversations between P and V.

□

Σ-protocol: $\mathbf{ab = c \bmod n^s}$

Common input: $x_a, x_b, x_c \in \mathbb{Z}^*_{n^{s+1}}$, n and s.

Private input for P: The witnesses will be denoted by $a, b, c \in \mathbb{Z}_{n^s}$ and $r_a, r_b, r_c \in \mathbb{Z}^*_n$ such that $ab = c \bmod n^s$ and $x_a = \mathcal{E}_D(a, r_a), x_b = \mathcal{E}_D(b, r_b)$ and $x_c = \mathcal{E}_D(c, r_c)$.

1. $P \to V$:
   (a) P chooses random values $d \in \mathbb{Z}_{n^s}, r_d, r_{bd} \in \mathbb{Z}^*_n$.
   (b) P computes $\tilde{a}_1 = x_d = \mathcal{E}_D(d, r_d)$.
   (c) P computes $\tilde{a}_2 = x_{bd} = \mathcal{E}_D(db, r_{db})$.
   (d) P sends the commitments $\tilde{a}_1$ and $\tilde{a}_2$ to V.
2. $V \to P$: V chooses random t-bit number e and sends it to P.
3. $P \to V$:
   (a) P opens the encryption

$$x_a^e \tilde{a}_1 = x_a^e x_d = \mathcal{E}_D(ea + d \bmod n^s, r_a^e r_d \bmod n) = \mathcal{E}_D(z_1, z_2)$$

   by sending z_1, z_2 to V.
   (b) P opens the encryption

$$x_b^{z_1} (\tilde{a}_2 x_c^e)^{-1} = x_b^{z_1} (x_{db} x_c^e)^{-1} = \mathcal{E}_D(0, r_b^{z_1} (r_{db} r_c^e)^{-1} \bmod n) = \mathcal{E}_D(0, z_3)$$

   by sending z_3 to V.

Verifing phase : V checks that $x_d, x_{bd}, z_1, z_2, z_3$ are all relatively prime to n and that:

1. $\mathcal{E}_D(z_1, z_2) \equiv x_a^e \tilde{a}_1 \pmod{n^{s+1}}$.
2. $\mathcal{E}_D(0, z_3) \equiv x_b^{z_1} (\tilde{a}_2 x_c^e)^{-1} \pmod{n^{s+1}}$.

Figure 7.11: Σ-protocol for proving the relation $ab = c \bmod n^s$ between plaintexts.

NIZK: $\mathbf{ab = c} \bmod \mathbf{n^s}$

1. P: computes $(a_1, a_2, \mathcal{H}(x_a, x_b, x_c, a_1, a_2, n, s), z_1, z_2, z_3)$
   and outputs $(a_1, a_2, z_1, z_2, z_3)$.

Any verifier can now calculate $\mathcal{H}(x_a, x_b, x_c, a_1, a_2, n, s)$ and check z_1, z_2, z_3 as above.

Figure 7.12: The NIZK-proof of the relation $ab = c \bmod n^s$ between plaintexts.

7.4.5 Equality of Discrete Logs

We want to prove the equality of two logs. That is, given

$$x_1 = y_1^u \ \wedge \ x_2 = y_2^u,$$

we want to prove that

$$\log_{y_1}(x_1) = u = \log_{y_2}(x_2).$$

Our witness for this relation is

$$w = u.$$

We will use this proof in the threshold decryption protocol in Section 7.6. The Σ-protocol is described in Figure 7.13 and the corresponding NIZK-proof in Figure 7.14. Since none of P and V know $\varphi(n)$, they can not calculate $z = r + ew \bmod \tilde{n}n^s$ in step 3 as they should have when used in the threshold decryption protocol. To encounter this problem, we let ew be small enough to not have any crucial impact on the size of z, by letting w be at most $(s+1)k$ bits, while r is $(s+2)k + t$ bits.

PROPOSITION 7.16. *The protocol of Figure 7.13 is a Σ-protocol.*

Proof. **Completeness:** We have to show that if P knows the secret then the honest V will accept, that is, the equations in the verifying fase must hold. For $i \in \{1, 2\}$:

$$y_i^z \equiv y_i^{r+ew} \equiv y_i^r \, (y_i^w)^e \equiv a_i x_i^e \pmod{n^{s+1}}.$$

Special soundness: Suppose (a_1, a_2, e, z) and $(a_1, a_2, \tilde{e}, \tilde{z})$ are two accepting conversations with $e \neq \tilde{e}$. We then get for $i \in \{1, 2\}$:

$$\begin{aligned} & y_i^z \equiv a x_i^e \pmod{n^{s+1}} \ \wedge \ y_i^{\tilde{z}} \equiv a x_i^{\tilde{e}} \pmod{n^{s+1}} \\ \Rightarrow \ & y_i^{z-\tilde{z}} \equiv x_i^{e-\tilde{e}} \pmod{n^{s+1}} \\ \Rightarrow \ & \log_{y_i}(x_i) \equiv \frac{z - \tilde{z}}{e - \tilde{e}} \end{aligned}$$

The multiplicative inverse of $e - \tilde{e}$ exists because of the restriction of 2^t which leads to $\gcd(e, n) = 1$, and since they both are of same size, and the group order is $pqn^s/4$ (as will be expalained in Section 7.6).

SHVZK: The input for the simulator S is $x_1, x_2, y_1, y_2 \in Q_{n^{s+1}}$, n, s and a challenge e. We can now produce a valid conversation by selecting random integer z of same size as r, and for $i \in \{1, 2\}$ compute $a_i = \frac{y_i^z}{x_i^e}$. We then get the conversation (a_1, a_2, e, z) which has the same probability distribution as conversations between P and V. This is so because in a real conversation P chooses uniformly at random r and computes a and z. In the simulated conversation, S chooses uniformly at random z and computes a. We have only switched the random variable from r to z, and since they both have the same probability distribution (uniformly at random and of same bit size), the probability distributions of real and the simulated conversations are the same.

□

Σ-protocol: Equality of discrete logs

Common input: $x_1, x_2, y_1, y_2 \in Q_{n^{s+1}}$ and n, s ($Q_{n^{s+1}}$ is defined in Section 7.6).

Private input for P: w such that $w = \log_{y_1}(x_1) = \log_{y_2}(x_2)$. The length of w is at most $(s+1)k$, where k is the length of n.

Proving phase:

1. $P \to V$:
   1.1. P chooses a random number r of length $(s+2)k + t$ bits where t is the security parameter.
   1.2. P computes $a_i = y_i^r \bmod n^{s+1}$ for $i \in \{1, 2\}$ and sends (a_1, a_2) to V.
2. $V \to P$: V chooses at random a t-bit number e and sends it to P.
3. $P \to V$: P computes $z = r + ew$ and sends z to V.

Verifing phase: V checks that $y_i^z \equiv a x_i^e \pmod{n^{s+1}}$ for $i \in \{1, 2\}$.

Figure 7.13: Σ-protocol for equality of discrete logs.

NIZK: Equality of discrete logs

1. P: computes $((a_1, a_2), \mathcal{H}(a_1, a_2, x_1, x_2, y_1, y_2, n, s), z)$ and outputs $((a_1, a_2), z)$.

Any verifier can now calculate $\mathcal{H}(a_1, a_2, x_1, x_2, y_1, y_2, n, s)$ and check z as above.

Figure 7.14: The NIZK-proof of equality of discrete logs

7.5 Cheating Voters and Honest but Curious Authorities

We will now implement some of the NIZK-proofs to our voting protocols. The assumption is as follows:

Assumption 4: *The voters are possibly cheating and the authorities are honest but curious.*

Since there might be cheating voters, we want to force them to follow the protocol. This can be done by including NIZK proofs to each vote, to assure that each vote casted is a proper vote. When voter i calls for the hash-function, she also include a user identity $id(W_i)$ in the input. This is done in order to prevent vote duplication. How to create and store these user identities, and how they are connected to the unreusability requirement defined in Section 7.1, is out of scope for this thesis. We will just assume that each user has a unique user identity $id(W_i)$ which is known both for the user herself and for the authorities in the counting phase.

We will now discuss how to implement the NIZK-proofs to our voting protocols, and in Section 7.7 we will discuss the security of them. For the yes/no-election we will write the protocol properly. For the other two voting protocols we will use the notation $Proof_{W_i}$(statement), where the term *statement* denotes what the voter wants to prove, and $Proof_{W_i}$(statement) denotes the output when the voter has created the stated NIZK-proof.

7.5.1 Yes/no-election

Again, this is our most basic protocol. In this protocol we will simply add the "1-out-of-2 is the encryption of 0" NIZK-proof described by Figure 7.8 in Section 7.4.2, to each vote casted. This will ensure that the ciphertext is either an encryption of 0 or of 1. The protocol is described in Figure 7.15.

It is easy to see that $c = \prod_{i=1}^{v} c_i$ for all valid votes m_i.

Voting protocol: Yes/no-election with cheating voter and semi-honest authority

Set-up Let W be the strict upper bound of voters, and let $L = 2$. A vote by voter W_i for "no" is represented by $m_i = 0$ and a vote for "yes" is represented by $m_i = 1$. The authorities will be represented by A. Use the encryptions scheme DJN_s, with $\mathcal{E}_D$ and $\mathcal{D}_D$ as described by Figure 6.1 in Section 6.1.

1. A chooses admissible $n = pq$.
2. A chooses d such that $d \bmod n \in \mathbb{Z}_n^*$ and $d = 0 \bmod \lambda(n)$.

Voting phase Each W_i casts a vote m_i of 0 or 1 as follows:

1. Choose random $r_i \in \mathbb{Z}_n^*$.
2. Choose $m_i = 0$ or $m_i = 1$.
3. Compute $c_i = \mathcal{E}_D(n, m_i, r_i)$.
4. Compute $\Big(a_1, a_2, \mathcal{H}\big(x_1, x_2, a_1, a_2, n, s, id(W_i)\big), z_1, z_2\Big)$ based on the "1-out-of-2 is the encryption of 0" NIZK-proof from Figure 7.8 in Section 7.4.2, with $x_1 = c_i, x_2 = c_i(n+1)^{-1}$ and $w = r_i$.
5. Post c_i and (a_1, a_2, z_1, z_2) on the bulletin board.

Counting phase

1. For each $1 \leq i < W$ A do:
   1.1. If $c_i \neq \emptyset$ then continue, else stop and set $i = i + 1$.
   1.2. Compute $\mathcal{H}\big(x_1, x_2, a_1, a_2, n, s, id(W_i)\big)$.
   1.3. Verify z_1, z_2 according to the "1-out-of-2 is the encryption of 0" NIZK-proof. If true then continue, else stop and set $i = i + 1$.
   1.4. Compute $c = cc_i$.
   Output: c.
2. A computes $m = \sum_{i=1}^{v} m_i = \mathcal{D}_D(d, c)$.
3. A posts the voting result m and the ciphertext c on the bulletin board.

Figure 7.15: A yes/no voting protocol for cheating voters and honest but curious authorities.

7.5.2 1-out-of-L-election

This voting protocol is very similar to the previous, except that we will use the "1-out-of-L is the encryption of 0" NIZK-proof described by Figure 7.10 in Section 7.4.3. The protocol is described in Figure 7.16.

Voting protocol: 1-out-of-L-election with cheating voter and semi-honest authority

Set-up Let W be the strict upper bound of voters, and let L be the number of candidates. A vote for candidate number $j \in \{0, \ldots, L-1\}$ is represented by the number W^j. The total votes for candidates $(0, \ldots, L-1)$ will be denoted by $(v_0, \ldots, v_{L-1})$ respectively. The authorities will be represented by A. Use the encryptions scheme DJN_s, with $\mathcal{E}_D$ and $\mathcal{D}_D$ as described by Figure 6.1 in Section 6.1, with $s \geq L \log_n W$.

1. A chooses admissible $n = pq$.
2. A chooses d such that $d \bmod n \in \mathbb{Z}_n^*$ and $d = 0 \bmod \lambda(n)$.

Voting phase Each voter i casts a vote j_i as follows:

1. Choose random $r_i \in \mathbb{Z}_n^*$.
2. Choose $j_i \in \{0, \ldots, L-1\}$.
3. Compute $c_i = \mathcal{E}_D(n, W^{j_i}, r_i)$.
4. Use "1-out-of-L is the encryption of 0" NIZK-proof from Figure 7.10, Section 7.4.3, to create

$$\mathit{Proof}_{W_i}\Big(\big(c_i(n+1)^{-W^0} \vee \cdots \vee c_i(n+1)^{-W^{L-1}}\big) \text{ is an encryption of } 0\Big).$$

   Post c_i and Proof_{W_i} on the bulletin board.

Counting phase

1. For each $1 \leq i < W$ A checks c_i and Proof_{W_i} and if approved then i is added to the set I.
2. A computes $c = \prod_{i \in I} c_i$ where I is the set of approved votes.
3. A computes $\mathcal{D}_D(d, c) = m = \sum_{j=0}^{L-1} v_j W^j$.
4. A posts $(v_0, \ldots, v_L - 1)$ and c on the bulletin board.

Figure 7.16: A 1-out-of-L voting protocol for cheating voters and honest but curious authorities.

7.5.3 K-out-of-L-election

This voting protocol is very similar to the previous one, except that each voter can vote for several candidates. But for each of the candidates she votes for, she computes the encryption $c_{i_k} = \mathcal{E}_D(n, W^{j_{i_k}}, r_{i_k})$. By including the "1-out-of-L is the encryption of 0" NIZK-proof described by Figure 7.10 in Section 7.4.3 to each vote, we are assured that each of this encryptions is a proper vote.

In addition to prove that each vote is a proper one, each voter also has to prove that she did not vote for the same candidate several times. This can be solved by noticing the following:

1. If all the votes are different, then $W^{j_{i_k}} - W^{j_{i_t}} \neq 0$ for each pair of votes $j_{i_k} \neq j_{i_t}$.
2. If so, then $\left(W^{j_{i_k}} - W^{j_{i_t}}\right)\left(W^{j_{i_k}} - W^{j_{i_t}}\right)^{-1} = 1$.
3. So we have two plaintexts which multiplied equals one, and we can use the NIZK-proof of the relation $ab = c \bmod n^s$ between plaintexts from Section 7.4.4 to prove it.

Now let c_{i_k}, c_{i_t} be the corresponding encryptions of $W^{j_{i_k}} \neq W^{j_{i_t}}$. Each voter can then do as follows with every pair of their votes:

1. Compute $(c_{i_k})(c_{i_t})^{-1}$ which is an encryption of $\left(W^{j_{i_k}} - W^{j_{i_t}}\right)$.
2. Compute $\mathcal{E}_D\left(n, \left(W^{j_{i_k}} - W^{j_{i_t}}\right)^{-1}\right)$ and $\mathcal{E}_D(n, 1)$.
3. Use the NIZK-proof of the relation $ab = c \bmod n^s$ between plaintexts, described by Figure 7.12 in Section 7.4.4, to prove that the plaintext of $(c_{i_k})(c_{i_t})^{-1}$ and $\mathcal{E}_D(n, \left(W^{j_{i_k}} - W^{j_{i_t}}\right)^{-1})$ multiplies to the plaintext of $\mathcal{E}_D(n, 1)$. The input for the NIZK-proof will be

   Common input:
   - $x_a = (c_{i_k})(c_{i_t})^{-1}$.
   - $x_b = \mathcal{E}_D\left(n, \left(W^{j_{i_k}} - W^{j_{i_t}}\right)^{-1}\right)$.
   - $x_c = \mathcal{E}_D(n, 1)$.

   Private input:
   - $a = W^{j_{i_k}} - W^{j_{i_t}}$.
   - $b = \left(W^{j_{i_k}} - W^{j_{i_t}}\right)^{-1}$.

- $c = 1$.
- r_a, r_b, r_c are the corresponding random variables.

There is a slight difference between the ordinary $ab = c \bmod n^s$ NIZK-proof, and what we need exactly. The ordinary proof just shows the relation, but we want to be assured that the plaintext of exactly $(c_{i_k})(c_{i_t})^{-1}$, multiplied by something known for the prover, equals exactly one, not just an unknown plaintext. Anyone can compute $(c_{i_k})(c_{i_t})^{-1}$, so this is easily taken care of. But for them to equal exactly one, we need to add something more. One idea could be to compute a reference value $\mathcal{E}_D(n, 1)$ in the set-up phase. But then the random variable of this encryption would have to be public (as the prover needs this value in the NIZK-proof). And this compromise this random variable being a private input for the prover. Another way to solve this is to let the prover choose a random variable and compute $\mathcal{E}_D(n, 1)$. She can then create a NIZK-proof of the encryption of 0 described by Figure 7.6 in Section 7.6, to prove that $\mathcal{E}_D(n, 1)(n + 1)^{-1}$ is the encryption of 0 and hence $\mathcal{E}_D(n, 1)$ is the encryption of 1.

The voting protocol is described in Figure 7.17-7.18. Notice that the decision of which of the votes that are accepted depends on the requirements of the election. One case is that all the votes from one voter has to be accepted or else non of them will be accepted. Another case can be that each vote is accepted independently from wether other votes are accepted or not.

Voting protocol: K-out-of-L-election with cheating voter and semi-honest authority

Set-up Let W be the strict upper bound of voters, let L be the number of candidates and let K be the number of candidates each voter can vote for. We will add $K-1$ dummy candidates to collect votes which are not placed on a real candidate. A vote for candidate number $j \in \{0,\ldots,L,\ldots,L+K-2\}$ is represented by the number W^j. The total votes for candidates $(0,\ldots,L+K-2)$ will be denoted by $(v_0,\ldots,v_{L+K-2})$ respectively. The authorities will be represented by A. Use the encryptions scheme DJN_s, with $\mathcal{E}_D$ and $\mathcal{D}_D$ as described by Figure 6.1 in Section 6.1, with $s \geq (L+K-2)\log_n W$.

1. A chooses admissible $n = pq$.
2. A chooses d such that $d \bmod n \in \mathbb{Z}_n^*$ and $d = 0 \bmod \lambda(n)$.

Voting phase Each voter i casts the votes j_{i_k} as follows:

1. For $k \in \{1,\ldots,K\}$:
   (a) Choose random $r_{i_k} \in \mathbb{Z}_n^*$.
   (b) Choose j_{i_k} from $\{0,\ldots,L+K-2\}$.
   (c) Compute $c_{i_k} = \mathcal{E}_D(n, W^{j_{i_k}}, r_{i_k})$.
   (d) Use "1-out-of-L is the encryption of 0" NIZK-proof from Figure 7.10 in Section 7.4.3, to create

   $$\mathit{Proof}_{W_i,k}\big(\big(c_{i_k}(n+1)^{-W^0} \vee \cdots \vee c_{i_k}(n+1)^{-W^{L+K-2}}\big) \text{ is an encryption of } 0\big).$$

2. For each $(K-1)!$ pair of votes $j_{i_k} \neq j_{i_t}, k,t \in \{1,\ldots,K\}$, use "the relation $ab = c \bmod n^s$ between plaintexts" NIZK-proof from Figure 7.12 in Section 7.4.4, to create

   $$\mathit{Proof}_{W_i,kt}\big(ab = c \text{ where } a,b,c \text{ is defined above}\big),$$

   and "the encryption of 0" NIZK-proof from Figure 7.6 in Section 7.6, to create

   $$\mathit{Proof}_{W_i,1}\big(\mathcal{E}_D(n,1)(n+1)^{-1} \text{ is an encryption of } 0\big).$$

3. Post $c_{i_k}, \mathit{Proof}_{W_i,k}, \mathit{Proof}_{W_i,kt}, \mathit{Proof}_{W_i,1}$ for $1 \leq k,t \leq K$ on the bulletin board.

Figure 7.17: A K-out-of-L voting protocol for cheating voters and honest but curious authorities (part 1 of 2).

Counting phase

1. For each $1 \leq i < W$ A checks c_{i_k} and the proofs, and validates them. If c_{i_k} is an accepted vote, then i_k is added to the set I.
2. A computes $c = \prod_{i_k \in I}^{K} c_{i_k}$.
3. A computes $\mathcal{D}_D(d, c) = m = \sum_{j=0}^{L-1} v_j W^j$.
4. Posts $(v_0, \ldots, v_L - 1)$ and c on the bulletin board.

Figure 7.18: A K-out-of-L voting protocol for cheating voters and honest but curious authorities (part 2 of 2).

7.6 Threshold decryption

In many real-life situations we might not believe that any given person can be trusted, and we will now look at what might happen if the authorities are cheating. We have trusted them to respect the privacy of voters, and we have trusted them to make a correct decryption of the voting result. But surely malicious behavior amongst the authorities can occur. When it does, it may have great impact on the election.

One way to solve this is by simply adding the NIZK-proof from Figure 7.6 in Section (encryption of 0) . This can be done because knowing the ciphertext c, the message m and the private key d, it is possible to extract the random variable r, as is done in [7]. This r can serve as a witness w for the relation $c(n+1)^{-m}$ as usual.

But this will not solve our entire problem, at least not the privacy issues. We will therefor look at another way of solving these trust issues. Because, even though we might not believe that any given person can be trusted, even not a big fraction of all people, it is yet reasonable to assume that the majority of people are trustworthy. Similarly, we may doubt that a given server can be trusted, but we hope that the majority of servers are working properly. Based on this assumption, we can solve the problem by distributing trust amongst several entities (authorities). One method to do this is called *threshold decryption*, and in very simplified terms it can be explained as follows: We give each authority a share of the private key (set-up phase). They each authority raise the ciphertext to their share of the decryption key to get their ciphertext share (share decryption) and in the end the ciphertext shares are combined using Lagrange interpolation (share combining). See Figure 7.19 to see a very simplified model. We will also use the NIZK-proof from Section 7.4.5 (equality of discrete logs) to force the authorities to follow the threshold decryption protocol.

As always we have to define which model we are working in. Our model in this section is the standard modell together with the the following assumption:

Assumption 5: *The* dealer *(as defined below) and a minimum of H out of A authorities are honest.*

To use the NIZK-proofs we need the random oracle model, but when needed this will be stated.

So let us begin. We need a way to share a secret, in this case the private key, amongst

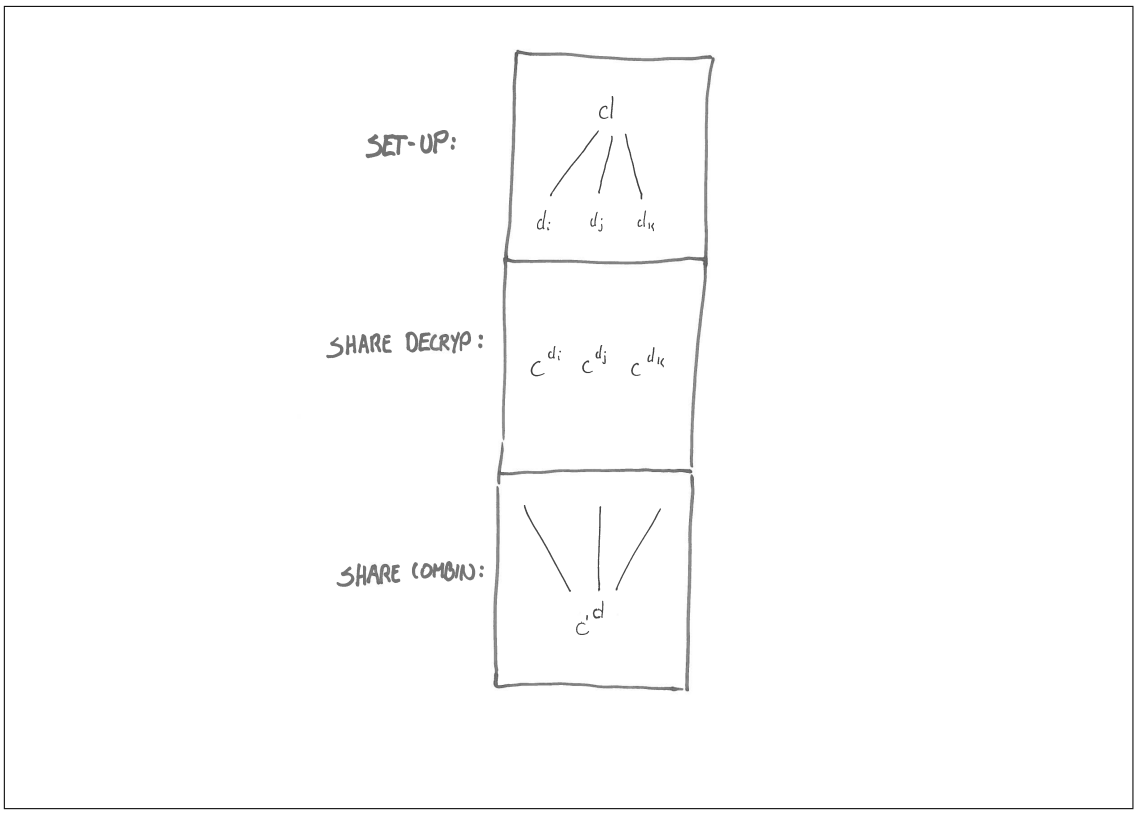


Figure 7.19: A model of a very simplified threshold cryptosystem.

all the A authorities in such a way that it can be reconstructed if at least H of the authorities are honest. To do this we can use Shamir's (H, A)-threshold scheme which he introduced in his two-page article "How to Share a Secret" in 1979 [29]. The scheme is based on the following well-known fact: Two points in the plain define a line, three points define a parabola, and generally, a polynomial of degree *deg* will be defined by $deg + 1$ number of points. To get an idea of how this concept works, we will give an example of a $(2, 4)$-threshold scheme, which means that at least 2 of the 4 authorities are honest. The polynomial will then be a straight line, that is, a polynomial of degree 1. The secret to be shared amongst the authorities is the value d.

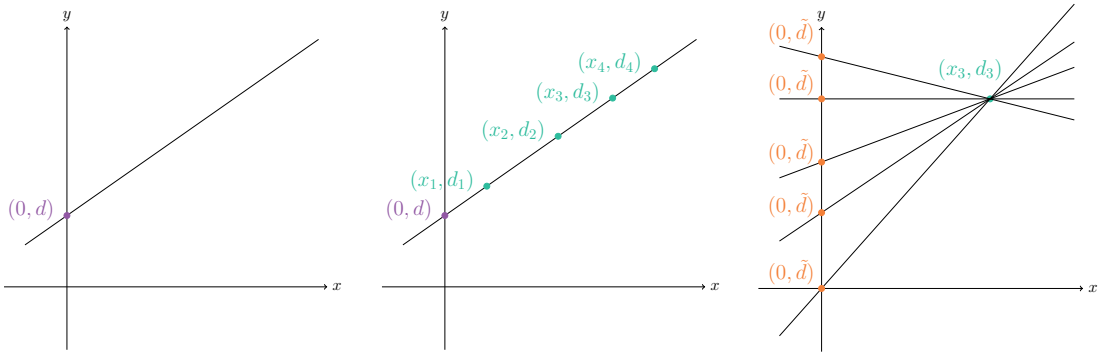
We will need an algebraic field to make these arguments, because we need multiplicative inverses. Since we have not discussed fields in this thesis we will just state that a *field* is a set which forms an abelian group under addition, and where all the nonzero elements form an abelian group under multiplication. For more on this subject, we refer to a textbook of basic abstract algebra such as [12]. Two examples of fields are the real numbers $\mathbb{R}$ and the finite set of integers $\mathbb{Z}_p^*$, where p is a prime as usual. To make the following example very intuitive we will use $\mathbb{R}$, and then move to $\mathbb{Z}_p$ in the further discussion.

So, we have 4 authorities where at least 2 of them are honest, a secret $d \in \mathbb{R}$ and we want to make a polynomial of degree 1 in the plane $\mathbb{R}^2$. We will now use a trusted instance, called a *dealer*, which will have access to the secret d and distribute the shares. We then do as follows:

1. The dealer draws a random line in $\mathbb{R}^2$ through the point $(0, d)$ (see Figure 7.20a).
2. The dealer chooses four points on the line (x_i, d_i), $1 \leq i \leq 4$, and distribute one point to each authority Ai (see Figure 7.20b).
3. Since each pair of points uniquely determine the line, each pair of honest authorities will now be able to reconstruct the line. By computing the line in $x = 0$ they will know the secret d.
4. Each authority Ai will alone not get any information about d: For each $\tilde{d} \in F$, there is a line that goes through (x_i, d_i) and $(0, \tilde{d})$ (see Figure 7.20c). Each of these lines are equally likely to appear when guessing a line through (x_i, d_i).

We will now look at the theorem, and the proof of it, which will be crucial for our threshold decryption protocol [5]:

Theorem 7.17. *Let F be a finite field and let D be a set of indices with $|D| = H$.*



(a) Random polynomial $f(x)$ with $f(0) = d$.

(b) Authority Ai gets share d_i of the secret d.

(c) Each authority Ai has no information about $f(x)$.

Figure 7.20: The concept of a $(2, 4)$-threshold scheme.

Suppose we are given a collection of H points (x_i, d_i), $i \in D$, in the plane F^2, where the x_i's are all different. Then there is a unique polynomial $f(x) \in F[X]$ (the set of all polynomials in F^2) of degree smaller than H, that passes through these H points. That is, $f(x_i) = d_i \ \forall \ i \in D$.

Proof. We need to prove both existence and uniqueness, and we will begin with the former by constructing $f(x)$ using a version of *Lagrange Interpolation*. For each $i \in D$ we define the polynomial

$$f_{D,i}(x) = \prod_{j \in D \setminus \{i\}} \frac{x - x_j}{x_i - x_j}.$$

We observe the following:

1. Each $f_{D,i}(x)$ has degree exactly $H - 1$.
2. $f_{D,i}(x_j) = \begin{cases} 1 & \text{if } j = i, \text{and} \\ 0 & \text{if } j \neq i \ . \end{cases}$

It follows from 2 that $d_i f_{D,i}(x_i) = d_i = f(x_i)$ for each $i \in D$ and so

$$f(x) = \sum_{i \in D} d_i f_{D,i}(x).$$

It is worth noting that $f(x)$ has degree *at most* $H - 1$ and that the degree actually can be strictly smaller than $H - 1$.

We will now prove uniqueness. Suppose there is a polynomial $f'(x) \in F[x]$ of degree smaller than H and where $f(x_i) = f'(x) \ \forall \ i \in D$. Then the polynomial $f(x) - f'(x) \in$

$F[x]$ is a polynomial with at least H zeros while its degree is smaller than H. It is well known that any polynomial in $F[x]$ has at most as many zeroes as the degree of the polynomial, unless it is the zero-polynomial. So $f(x) - f'(x) = 0$, and $f(x) = f'(x)$. □

In practice we do not do exactly as described above. Instead of choosing the points (x_i, d_i) freely, the dealer computes $(i, f(i))$, where f is the polynomial, and then deals the share $d_i = f(i)$ to each authority Ai. And since $d = f(0)$, we will only be interested in the value of $f(x)$ in 0. So instead of reconstructing the polynomial completely, the H authorities only reconstruct the value $f(0) = d$. This lead to the following equation:

$$d = f(0) = \sum_{i \in D} d_i \tilde{\gamma}_{0,i}^{D}, \tag{7.1}$$

where $\tilde{\gamma}_{0,i}^{D}$ will be called the *Lagrange coefficient of the interpolating polynomial* and is the function $f_{D,i}(x)$ from above, computed in $x = 0$ with $x_i = i \; \forall \; i \in D$. That is,

$$\tilde{\gamma}_{0,i}^{D} = \prod_{j \in D \setminus \{i\}} \frac{-j}{i - j} \; .$$

But there is also another issue to look into. The threshold decryption system of Shamir is not the exact one we will use to make a threshold protocol based on DJN$_s$. Instead we will make a similar one, introduced by Damgård, Jurik and Nielsen in [7]. And they base their protocol on the one Victor Shoup introduced for RSA cryptosystems in the article "Practical Threshold Signatures" in 2000 [31]. In his article, Shoup stated that he had to ensure to do all group computations in the cyclic group Q_n, defined as [4]:

$$Q_n = \left\{ c \in \mathbb{Z}_n^* \;\middle|\; c = z^2 \text{ for } z \in \mathbb{Z}_n^* \right\},$$

and the corresponding exponent arithmetic in the group orden $\tilde{n}$, which then is the group $\mathbb{Z}_{\tilde{n}}$. The integer $\tilde{n}$ is defined to be $\tilde{n} = \tilde{p}\tilde{q}$, where $\tilde{p}, \tilde{q}$ are two large primes such that $p = 2\tilde{p} + 1$ and $q = 2\tilde{q} + 1$, also called *safe primes*.

But the ciphertexts of DJN$_s$ do not belong to $\mathbb{Z}_n^*$, as the ciphertexts of RSA. Our ciphertexts belongs to $\mathbb{Z}_{n^{s+1}}^*$, and we need a similar group as Q_n only transformed to our

[4] The reason for this choice is out of scope for this thesis, but it probably has something to do with the adversary being able to reconstruct a bit of the plaintext if we are working in $\mathbb{Z}_n^*$, maybe distinguishing prime plaintexts from odd ones.

system. This group is given by:

$$Q_{n^{s+1}} = \left\{ c \in \mathbb{Z}^*_{n^{s+1}} \;\middle|\; c = z^2 \text{ for } z \in \mathbb{Z}^*_{n^{s+1}} \right\}$$

with group order $n^s\tilde{n}$. So we will do group operations in $Q_{n^{s+1}}$ and exponent operations in $\mathbb{Z}_{n^s\tilde{n}}$. But then the problem arises: To use Theorem 7.17 we need a field. And the interpolation will be done in exponent arithmetic. But in $\mathbb{Z}_{n^s\tilde{n}}$, not all nonzero elements have multiplicative inverses. So let us look at the finite field $\mathbb{Z}_p$ and see where it differs from $\mathbb{Z}_{n^s\tilde{n}}$. The only part is in the proof of the theorem, where we need to calculate the multiplicative inverses $(x_i - x_j)^{-1}$. Transformed to the interpolation we will use we will need to calculate $(i-j)^{-1}$, and we are not guaranteed that this number exists in $\mathbb{Z}_{n^s\tilde{n}}$. And if it exists, it is not possible to calculate it since no one knows $\lambda(n)$ and hence not the group order of $\mathbb{Z}^*_{n^s\tilde{n}}$. But we know that $\prod_{j\in D\setminus\{i\}}(i-j)^{-1}$ divides $i!(H-i)!$, which in turn divides $H!$. So we can let $\gamma = H!\tilde{\gamma} \bmod n^s\tilde{n}$ which guarantee us that γ is an integer in $\mathbb{Z}_{n^s\tilde{n}}$. That is:

$$H!d = H!f(0) = \sum_{i\in D} d_i \tilde{\gamma}^D_{0,i} \bmod n^s\tilde{n}, \tag{7.2}$$

with

$$\gamma^D_{0,i} = H! \prod_{j\in D\setminus\{i\}} \frac{-j}{i-j} \bmod n^s\tilde{n}.$$

The previous discussion was related to combining the shares. We will now discuss the prior part of the decryption protocol, namely the shared decryption process. Each authority A_i has to raise the encrypted election result to their share of the secret. This would be to compute $c_i = c^{d_i}$. But we want to make sure that each c_i is in $Q_{n^{s+1}}$. To to this we exponentiate c^2. This gives us $c_i = c^{2d_i}$. But we need even one more exponentiation factor, because in the proof of this protocol we will need to interpolate once more. And this leads us once again with the problem of not knowing the group order of $Q_{n^{s+1}}$. We will solve this as follows:

$$c_i = c^{2H!d_i}.$$

To be sure that each of the H honest authorities have actually raised c to Hd_i, we will use the NIZK-proof from Section 7.4.5 to show equality of two discrete logs. The idea is that we already in the key generation phase will calculate a *verification key* for each authority, which will be a value $ver_i = ver^{H!d_i}$. This can be done long time prior to

the actual election. When the election then has finished, each authority is computing $c_i = c^{2H!d_i}$. We then get $\log_{c^4}(c_i^2) = H!d_i = \log_v er(ver_i)$, and we can prove this in the random oracle model with the NIZK-proof. The only thing we need to be assured of, is that *ver* generates $Q_{n^{s+1}}$. To solve this, we choose *ver* at random from $Q_{n^{s+1}}$. Then *ver* will generate $Q_{n^{s+1}}$ with all but negligible probability-

Let us return for a minute to the share combining phase. We have no guarantee that each authority raised c to the power of two [5]. So to ensure we are working in $Q_{n^{s+1}}$, we will raise the c_i to the power of 2 in the share combining phase as well. That is,

$$c' = \prod_{i \in D} c_i^{2\gamma_{0,i}^D}.$$

We are now ready to formalize the threshold decryption protocol based on DJN_s, and this is done in Figure 7.21.

THEOREM 7.18. *The threshold protocol in Figure 7.21 is correct.*

Proof. The soundness of the NIZK-proof ensures us that the H c_i's are correct values with all but negligible probability. So will only check if the share decryption phase outputs m as stated.

$$c' = \prod_{i \in D} c_i^{2\gamma_{0,i}^D} = \prod_{i \in D} \left(c^{2H!d_i}\right)^{2\left(H! \prod_{j \in D \setminus \{i\}} \frac{-j}{i-j}\right)} = \prod_{i \in D} \left(c^{4H!^2}\right)^{\left(d_i \prod_{j \in D \setminus \{i\}} \frac{-j}{i-j}\right)}$$
$$= c^{4(H!)^2 d} = (n+1)^{4(H!)^2 m} \bmod n^{s+1},$$

where the last equality follows from

$$4(H!)^2 d \equiv 0 \pmod{n} \ \wedge \ 4(H!)^2 d \equiv 4(H!)^2 \pmod{n^s},$$

because of the choice of d. The rest follows directly. □

Before we end this section, let us look at the choice of d. If we had chosen $\lambda(n)$ as done in Paillier's original system PA, then an adversary could easily have broken the system completely. It is enough noticing that $4(H!)^2\lambda(n) \not\equiv 4(H!)^2 \pmod{n^s}$, and so we would get $(n+1)^{4(H!)^2\lambda(n)m} \bmod n^{s+1}$. Knowing the voting result would then lead to knowledge of $\lambda(n)$. Also notice that d in this protocol is a particular choice of the d in

[5] We are not only concerned of what each authority has done to c. There might as well be adversaries which are interfering with the value c_i.

DJN_s cryptosystem, as it is defined as $d \in \mathbb{Z}_n^*$ there instead of $d \equiv 1 \pmod{n^s}$ as here. But the latter requirement is essential to be able to compute m without the knowledge of d, as the former requirement would yield $(n+1)^{4(H!)^2 dm} \bmod n^{s+1}$.

In the next section we will look at the security of the voting protocols in general, and in particulare this threshold decryption protocol.

Threshold decryption protocol

Set-up The set-up fase will be added to the set-up phase of the voting protocol in use, so we will need all the values from the voting protocol as well. A dealer will execute the set-up part of the protocol. Choose two primes p, q with the additional property $\tilde{p} = \frac{p-1}{2}$ and $\tilde{q} = \frac{q-1}{2}$ (i.e. safe primes) which also gives us $\tilde{n} = \tilde{p}\tilde{q}$. We will let d be such that $d \equiv 0 \pmod{n}$ and $d \equiv 1 \pmod{n^s}$. We will let s be as required from the voting protocol.

1. Make a polynomial $f(x) = \sum_{i=0}^{H-1} b_i x^i \bmod n^s n'$, by picking b_i as random values from $\{0, \cdots, n^s(n'-1)\}$ for $1 \leq i \leq H-1$ and $b_0 = d$.
2. For each $1 \leq i \leq A$, distribute $d_i = f(i)$ to A_i.
3. Chose at random a public value ver which generates $Q_{n^{s+1}}$.
4. Fix a public verification key $ver_i = ver^{d_i} \bmod n^{s+1}$ for each A_i, $1 \leq i \leq A$.

Output: $ek = (n, s, g)$, $dk = \{d_i \mid 1 \leq i \leq A\}$ and the set of verification keys $\{ver_i \mid 1 \leq i \leq A\}$.

Voting phase We will import the voting phase from the voting protocol in use.

Share decryption From the voting protocol we get an encrypted voting result c. Each A_i will now do as follows:

1. Compute $c_i = c^{2H!d_i}$.
2. Use "equality of discrete logs" NIZK-proof from Figure 7.14, Section 7.4.5, to create
$$Proof_{A_i}\left(\log_{ver}(ver_i) = \log_{c^4}(c_i^2)\right).$$
3. Post c_i and $Proof_{A_i}$ on the bulletin board.

Share combining

1. Check that there are H or more numbers of c_i's with correct NIZK-proof.
2. Take a subset S of the c_i's with correct proofs, such that $|S| = H$, and make a set D of the indices.
3. Compute $c' = \prod_{i \in D} c_i^{2\gamma_{0,i}^D} = (n+1)^{4(H!)^2 m} \bmod n^{s+1}$, where $\gamma_{0,i}^D = H! \prod_{j \in D \setminus \{i\}} \frac{-j}{i-j}$.
4. Use Algorithm 1 from Section 6.3 on $(n+1)^{4(H!)^2 m}$ to extract $4(H!)^2 m$ and multiply this by $\left(4(H!)^2\right)^{-1} \bmod n^s$ to get m.
5. Transform m according to the voting protocol to a format which yields the voting result.
6. Post the voting result on the bulletin board.

Figure 7.21: A threshold decryption protocol based on DJN_s.

7.7 Security

We get our complete voting protocols by starting with the threshold decryption protocol and then add the voting protocols with cheating voters and honest but curious authorities from Section 7.5. The threshold decryption protocol includes a description on how to do this.

We will now look at the security of these voting protocols. The security is proved in the random oracle model, but as we will see, we only need the random oracle model for the NIZK-proofs. Except for this, the rest of the security proof is in the standard model. This is important to notice, because we might could have used interactive or non-interactive proofs that were secure in the standard model. If this was the case, then also the threshold protocol would have been secure in the standard model.

7.7.1 A Basic Level of Security

There are several ways to analyze the security of a voting protocol. The most basic level is as follows:

1. First we consider the protocols in the semi-honest model. In this model the voting protocols are semantical secure because DJN_s is semantical secure. We have to add threshold decryption to fulfill the privacy requirement though, because the authorities are curious. But since they are honest, we can add the threshold decryption protocol excluded the NIZK-proofs.
2. Next, we consider the model where the voters are cheating and the authorities are honest but curious. We then include NIZK-proves to ensure that the voters are sending encryption of valid votes. An adversary will only have negligible probability to contribute with incorrect values because of the soundness of the $\text{NIZK} - proofs$. In some sense we are forcing the voters to follow the protocol, and hence act as if they were honest.
3. In the end we assume that also the authorities are dishonest. We want to protect the encrypted election result from dishonest authorities as well, so we include NIZK-proofs to the threshold protocol to ensure that the H honest authorities does not contribute with an invalid share. Also here we are in some sense forcing the voters to act as if they were honest.

Conclusion: Since the protocols are secure in the semi-honest model and we are forcing all the parties to act as if they were honest, then the voting protocols included the NIZK-proofs must be secure in the cheating voters and cheating authorities model.

This argument does offer some level of security. And it is a very tempting idea to just make a final conclusion based on it. But in cryptography such arguments can be very dangerous. There might be technical details and subtleties which this argument does not catch. And there might also be something about the protocol we do not immediately see, but which makes these logical arguments invalid. If we rather make a simulation of the protocol, and show that an adversary can not tell the difference between the simulated run of the protocol and a real run of the protocol, then we can offer a much higher level of security.

7.7.2 Security by Simulation

We will now make a simulation of the protocol, and we start with the threshold decryption protocol.

THEOREM 7.19. *Assume that a static adversary corrupts $H-1$ players from the beginning, and assume that we know what the honest voters voted. Then the adversary's view of the threshold decryption protocol can be efficiently simulated such that the adversary can not distinguish between the real run and a simulated run of the protocol.*

Proof. The public value n is given. Let $1 \leq u \leq H$ represent the honest authorities, and $i_1, \leq i_{i'} \leq i_{H-1}$ with $i_{i'} \notin \{1, \ldots, H\}$ represent the cheating authorities. Since we know the voting result m, we also know $c^d = (1+n)^m \bmod n^{s+1}$.

We start by choosing shares $d_{i_{i'}}, \ldots, d_{i_{H-1}}$, for the cheating authorities. These shares is chosen uniformly at random modulo n^{s+1}, and they are indistinguishable from the real values, since the real values are chosen modulo $n^s\tilde{n}$.

Since d is fixed by the choice of n, the points $(o, d), (i_1, d_{i_1}), \ldots, (i_{H-1}, d_{i_{H-1}})$ completely determines the polynomial $f(x)$. Notice that $f(i_{i'}) = d_{i_{i'}}$. But we do not know d, and since it is an infeasible computation to extract d from $c^d = (1+n)^m$, we can not compute f. And hence we cannot compute the share d_u for the honest authorities. But we know the value c^d, so for each authority u it is possible to interpolate $f(x)$ in u to get $d_u = f(u)$. So instead of interpolating in 0 to get d as we do in the real threshold protocol, we will

interpolate in u to get $f(u)$. It is though not possible to get the value $f(u)$ as we do not know d, but we only need the value $c^{2H!f}$ and this is possible since we have the value c^d. This is done as follows:

From the discussion in the previous section, we know that

$$f(x) = \sum_{i \in D} d_i f_{D,i}(x) = \sum_{i \in D} d_i \prod_{j \in D \setminus \{i\}} \frac{x - x_j}{x_i - x_j},$$

where $f(x_i) = d_i$. If we let $D = \{0\} \cup \{i_1, \ldots, i_{H-1}\}$, and since $f(i_{i'}) = d_{i_{i'}}$, we get

$$f(x) = \sum_{i \in D} d_i \prod_{j \in D \setminus \{i\}} \frac{x - j}{i - j},$$

which yields

$$H!f(x) = \sum_{i \in D} d_i \gamma_{x,i}^D.$$

We do not know $f(0) = d$, but we do know $c^d = (1+n)^m \bmod n^{s+1}$. So by this we can calculate

$$c_u = c^{2H!d_u} = c^{2H! \sum_{i \in D} d_i \prod_{j \in D \setminus \{i\}} \frac{u-j}{i-j}} = c^{\sum_{i \in D} d_i \gamma_{u,i}^D 2} = \prod_{i \in D} c^{d_i \gamma_{u,i}^D 2}.$$

This interpolation is the reason behind the $H!$ in $c^{2H!d_i}$ from the threshold protocol.

We also need to find a random value $ver \in Q_{n^{s+1}}$ and verification keys ver_i. We do this by first choosing a random message $m_0 \in \mathbb{Z}_{n^s}$ and a random $r \in \mathbb{Z}_n^*$ such that $ver = (n+1)^{m_0} r^{2n^s}$.

For the cheating authorities we then calculate $ver_{i_i'} = ver^{d_{i_i'}}$. For the honest authorities we calculate as above

$$ver_u = \prod_{i \in D} ver^{d_i \gamma_{u,i}^D}.$$

For the zero knowledge proofs, we must now move to the random oracle model. For each authority i, we do the simulations of the NIZK-proof as follows:

As we know from Section 7.3.2, we can use the HVZK simulator S instead of the SHVZK one. This means that we can choose the challenge at random. We let S decide what the random oracle response should be on each input, as long as the values have the same distribution as in real life.

So now we just choose a random challenge e and then run S with $c_i^2, ver_i, c^4, ver^2, n, s$ and e. S now producees a valid conversation by selecting a random integer z and compute $a_1 = \frac{(c^4)^z}{(c_i^2)^e}$ and $a_2 = \frac{(ver^2)^z}{(ver_i^2)^e}$. S now defines that on input $a_1, a_2, c_i^2, ver_i, c^4, ver^2, n, s$ and z, the oracle's response is e. □

We will also sketch how to simulate the vote casting part of the voting protocols.

First we simulate the votes by choosing only random messages $\tilde{m}_i$ in $\mathbb{Z}_{n^s}$. We then forge the NIZK-proofs using the random oracle as above (we are now back in the random oracle model).

We take all the simulated encryptions of the votes, multiply them and get a ciphertext which we are supposed to threshold-decrypt.

The problem now is that this ciphertext does not contain the correct election result. But this does not matter, as we know what the decryption should be. So we can use the interpolated values c_u from above, where c_u is the share from honest authority u, which then will output the correct result.

8 Concluding Remarks

As we mentioned in the beginning of the previous chapter, there are several requirements we have not fulfilled in the voting protocols. The eligibility requirement is partly solved by the user identification tag we include in the NIZK-proofs, while the coersion-free requirement is not solved. We would like to make a concluding remark on another security issue, namely the dealer.

The dealer is given a lot of trust in the set-up phase of the threshold decryption protocol. This does not have to be a problem, as the set-up phase can be done long time prior to the election, and so the trust can be given to an instance with no attachments to the election. As an example, we can imagine a government election where the set-up phase is done by neutral parties outside the country. But as we mentioned in the introduction, we will conclude this thesis by sketching an idea of how this could be solved with cryptographic methods.

Threshold decryption is a particular instance of something called multiparty computation. The idea is that we can use techniques as Lagrange interpolation to jointly compute functions.

In our case we will give an example of generating a prime. This can be done by multiparty computation by first let the parties chooses an integer of a predefined size. Then each party creates a polynom as in the set-up phase of the threshold decryption protocol, and gives shares of their secret to each of the other parties. Now everybody have shares of all the secrets. Each party can then add all the shares together, to form a new share. The new shares can now be used to interpolate a polynom in a given value. But we will not be interested in finding this value, as this would reveal the secret. Instead we will interpolate directly when it is needed, for example in Fermat's primality test [14].

There is a problem with this methods in our case: If the parties were supposed to find only a regular large prime, they would might have to test some thousands candidates. But in the case of safe primes, they would have to test millions of candidates, and this could take weeks and weeks. There are other solutions to this problem, as it is possible to limit the amount of possible candidates. We will not go into this here, but leave it as an open question for the interested reader to pursue.

Bibliography

[1] Khan Academy. Journey into cryptography. `https://www.khanacademy.org/computing/computer-science/cryptography` (last checked 07/08/2016).

[2] L Babai. Trading group theory for randomness. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 421–429. ACM, 1985.

[3] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 62–73. ACM, 1993.

[4] Josh Benaloh. Rethinking voter coercion: The realities imposed by technology. *USENIX Journal of Election Technology and Systems*, 1(1):82–105, August 2013.

[5] Ronald Cramer. Introduction to secure computation. `http://homepages.cwi.nl/~cramer/papers/CRAMER_revised.ps` (last checked 14/08/2016). CWI, the Netherlands.

[6] Ivan Damgå rd. On σ-protocols. `http://www.cs.au.dk/~ivan/Sigma.pdf` (last checked 25/07/2016). Aarhus University, Denmark.

[7] Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of paillier's public-key system with applications to electronic voting. *International Journal of Information Security*, 9(6):371–385, December 2010.

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

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

[10] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. *Cryptography Engineering: Design Principles and Practical Applications.* Wiley Publishing, Inc., 2nd edition, 2010.

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

[12] J.B. Fraleigh and V.J. Katz. *A first course in abstract algebra.* Addison-Wesley world student series. Addison-Wesley, 2003.

[13] William I. Gasarch. Guest column: The p=?np poll. *ACM SIGACT News*, 33:34–47, June 2002.

[14] Kristian Gjø steen. Lecture notes for tma4160 cryptography. `https://wiki.math.ntnu.no/tma4160/2015h/notes` (last checked 10/07/2016). Norwegian University of Science and Technology.

[15] Kristian Gjø steen. *Subgroup membership problems and public key cryptosystems.* PhD thesis, NTNU, Norway., 2004.

[16] Kristian Gjøsteen. *Homomorphic Cryptosystems Based on Subgroup Membership Problems*, pages 314–327. Springer Berlin Heidelberg, 2005.

[17] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 291–304. ACM, 1985.

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

[19] Carmit Hazay and Yehuda Lindell. *Sigma Protocols and Efficient Zero-Knowledge1*, pages 147–175. Springer Berlin Heidelberg, 2010.

[20] Mads J. Jurik. Extensions to the paillier cryptosystem with applications to cryptological protocols. *BRICS Dissertation Series*, 9, 2003.

[21] Jonathan Katz and Yehuda Lindell. *Introduction to Cryptography.* CRC Press, 2nd edition, 2015.

[22] Helger Lipmaa. Secure electronic voting protocols. `http://kodu.ut.ee/~lipmaa/papers/voting4hb.pdf` (last checked 24/07/2016). University of Tartu, Estonia.

[23] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1st edition, 1996.

[24] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer Berlin Heidelberg, April 1999.

[25] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *Advances in Cryptology — Proceedings of EUROCRYPT '96*, pages 387–398. Springer Berlin Heidelberg, 1996.

[26] Harry H. Porter. Theory of computation - video course. `http://web.cecs.pdx.edu/~harry/videos/` (last checked 03/08/2016). Portland State University.

[27] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[28] Berry Schoenmakers. Lecture notes cryptographic protocols. `http://www.win.tue.nl/~berry/2DMI00/LectureNotes.pdf` (last checked 24/07/2016). Technical University of Eindhoven, The Netherlands.

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

[30] Claude E. Shannon. A mathematical theory of cryptography. `ttps://archive.org/stream/ShannonMiscellaneousWritings/ShannonMiscellaneousWritings_djvu.txt`. A re-typesetting in modern fonts by Whitfield Diffie and John O'Rourke using LaTeXcan be found at `https://www.iacr.org/museum/shannon/shannon45.pdf`., 1945.

[31] Victor Shoup. Practical threshold signatures. In *Proceedings of the 19th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'00, pages 207–220. Springer-Verlag, 2000.

[32] Victor Shoup. Sequences of games: A tool for taming complexity in security proofs. `http://www.shoup.net/papers/games.pdf` (last checked 05/08/2016), January 2006.

[33] Nigel Smart. Cryptography: An introduction. `https://www.cs.umd.edu/~waa/414-F11/IntroToCrypto.pdf` (last checked 03/08/2016). University of Maryland, USA.

[34] Alan M. Turin. On computable numbers, with an application to the entscheidungsproblem. `https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf` (last checked 03/08/2016).