Notes on Cryptography

INSTITUTO
SUPERIOR
TÉCNICO

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Paulo Mateus, Amílcar Sernadas,

André Souto and Luís Antunes[1]

2012

[1]Para se fazer o livro à posteriori

To the guardian

Caution

The themes presented in this work are in a very preliminary phase of maturation. It contains lots of typos, examples that are not yet complete, definitions that are not written with a unified notations and theorems that are not properly written. In fact, there are lots of topics still to be written and others that will deserve a profound reflection before publishing. Many exercises and examples are part of other books are meant to be used in classes. They will be substituted in mean time by others that fulfill our purposes of presentations and focus on crucial aspects that we have in mind.

Reading this lecture notes requires the reader to be prepared to find some problems of comprehension and we suggest that it should be complemented with the reading of standard books used to teach the materials presented here. We suggest for example the readings of [Sti06], [MVO96], [KL07] and the other references that we put on along the text for a full understanding of cryptography.

Preface

This work is meant to compile in a book the programmatic contents of the introductory course Criptografia e Protocolos de Segurança of the Master of Bologne in Mathematics and applications and also from the Doctoral program me on Security of Information lectured in the mathematical department of Instituto Superior Técnico of Universidade Técnica de Lisboa. It also uses material lectured in the course Criptografia lectured at computer science department of Faculdade de Ciências of Universidade do Porto.

The idea of this work is provide a very useful self contained book about cryptography using a perspective of a mathematician and also a perspective of a computer scientist in order to be able to a tool that others can use to teach a similar course.

For financial support the author André Souto is deeply thankful to FCT through the grant SFRH/BPD/76231/2011.

We are grateful to the SQIG members for the nice working ambient and all the encouragement that they have us.

A special thanks is due to several people with which we had very helpful discussions, feedback and advising on the themes and the approaches choosen.

2012.

Mathematical Department

Instituto Superior Técnico, Universidade Técnica de Lisboa

Computer Science Department

Faculdade de Ciências da Universidade do Porto

Security and Quantum Information Group

Instituto de Telecomunicações

Contents

CHAPTER 1

Intro

To be done later...

"Cryptography is the art of writing codes and solving codes..."

Need to say that the material collected in this work is original and was influenced by many excellent books containing their own perspective of Cryptography. We were influenced mainly by the following references: [Sti06], [KL07], [HPS08], [MVO96],

Part I

Basic concepts

In this part it is presented some basic concepts and constructions that will crucial and widely used in due course on the presentation of the themes of these notes and are fundamental in modern cryptography has they are they are their cornerstones.

It is presented, in the next Chapter and, in particular, in Section2.1, the material related with algebra. In Section 2.2 it is presented the foundation of number theory and in Section 2.3 its is presented some notions concerning modular algebras. In Chapter 3 is presented the topics about Information theory and in particular, in Section 3.1 it is introduced the theory of probabilities and in Section 3.2 it is presented the concept and basic results related with Shannon entropy.

CHAPTER 2

Algebraic structures and number theory

This chapter begins by presenting the common algebraic structures that will be consider in these notes, namely, groups, rings and fields. They are important for the subsequent sections where the basic notions of number theory and modular algebras is introduced. It is suggested the reading of the book [Lan02] of Lang to the interested reader for a complete study on algebra topics.

.....

SECTION 2.1

Groups, rings and fields

Cryptography is all about processes of transforming messages into another messages among a finite space of possible messages. Thus, the set of messages have certain algebraic structures in order to be closed under the transformation operations.

In this section it is surveyed some of the basic algebraic structures, namely groups, rings and fields, that are central concepts of abstract algebra, and are also basic tools for modern cryptography.

Roughly speaking, a group is a set of objects with an operation defined between any two objects in the set with certain properties, a ring is a set with two operations that is a group for the

first operation and the second operation is distributive with respect to the group operation. Finally, a field is commutative ring with unity in which every non null element has inverse (with respect to the second operation).

Definition 2.1.1 (*Monoid*)

Let G be a nonempty set and $\circ : G \times G \to G$ and binary operator in G. The pair $(G, \circ)$ is called a monoid if and only if the following properties hold:

Identity: *There is $e \in G$ such that for all $a \in G$, $a \circ e = a$.*

Associativity: *For all $a, b, c \in G$, $a \circ (b \circ c) = (a \circ b) \circ c$.*

Examples of monoinds are $(\mathbb{N}, +)$, $(\mathbb{N}, \cdot)$, $(\mathbb{Z}, +)$, $(\mathbb{Z}, \cdot)$, $(\mathbb{Q}, +)$, $(\mathbb{Q}, \cdot)$, $(\mathbb{R}, +)$, $(\mathbb{R}, \cdot)$, $(\mathbb{C}, +)$ and $(\mathbb{C}, \cdot)$, where $+$ and $\cdot$ are the usual sum and product operations in those sets.

Definition 2.1.2 (*Group*)

Let $(G, \circ)$ be a monoid. The monoid is called a group if and only if every $a \in G$ has inverse, i.e, there is an element $b \in G$ such that $a \circ b = e$.
A group $(G, \circ)$ is commutative or Abelian if in addition, one has, for all $a, b \in G$, $a \circ b = b \circ a$.

Examples of groups are $(\mathbb{N}, +)$, $(\mathbb{Z}, +)$. Another important example of groups are $(\mathbb{N}^*, \cdot)$, $(\mathbb{Z}^*, \cdot)$, $(\mathbb{Q}^*, \cdot)$, $(\mathbb{R}^*, \cdot)$, where $C^* = C - \{0\}$.

Example 2.1.1. *Let n be a natural number and let $\mathbb{Z}_n$ be the set of all possibles remainders by n, i.e.,*

$$\mathbb{Z}_n = \{0, \cdots, n-1\}$$

and given $a, b \in \mathbb{Z}_n$ define $a + b$ as the remainder of $a + b$ by n. Then $(\mathbb{Z}_n, +)$ is an example of a finite group, i.e., a group over a finite set. This particular type of groups will be discussed in more detail later on (in Section 2.3) and will widely used.

Definition 2.1.3 (*Subgroup*)

A subgroup of a group $(G, \circ)$ is a non-empty subset H of G which is itself a group under the same operation $\circ$.

Clearly $(\mathbb{N}, +)$ is a subgroup of $(\mathbb{Z}, +)$. Also $(\{e\}, \circ)$ is a subgroup of $(G, \circ)$.

Definition 2.1.4

Let $(G, \circ)$ be a finite group. The number of elements in G is called the order of G and is denoted by $ord(G)$.

Theorem 2.1.1 (Lagrange)

Let G be a finite group. If H is a subgroup of G then $ord(H)$ divides $ord(G)$.

Proof: To prove this theorem first consider the sets:

$$aH = \{a \circ b : b \in G\}.$$

usually called lateral classes of H.

Claim 2.1.2. *$aH = bH$ if and only if $a \circ b^{-1} \in H$.*

Proof:

($\Rightarrow$) Since $a \in aH = bH$ there exists $h \in H$ such that $a = b \circ h$. Thus, $b^{-1} \circ a = b^{-1} \circ (b \circ h) = (b^{-1} \circ b) \circ h = e \circ h = h$. Hence $b^{-1} \circ a \in H$.

($\Leftarrow$) Since $b^{-1} \circ a \in H$ then there exists $h \in H$ such that $b^{-1} \circ a = h \Leftrightarrow a = b \circ h$, hence $a \in bH$. Thus if $g \in aH$ then $g = a \circ h'$ for some $h' \in H$, hence $g = b \circ h \circ h' = b \circ h''$ Since H is a subgroup $h'' \in H$ and therefore $g \in bH$, which means that $aH \subset bH$. The other inclusion is obtained observing that, since $b^{-1} \circ a = h$ then $b = a \circ h^{-1}$. ■

Claim 2.1.3. *If $aH \cap bH \neq \emptyset$ then $aH = bH$.*

Proof: If $aH \cap bH \neq \emptyset$ then there exists $g \in G$ such that $g = a \circ h = b \circ h'$, hence $b^{-1} \circ a = h' \circ h^{-1}$. Since $h' \circ h^{-1} \in H$ then by the previous item $aH = bH$. ■

Claim 2.1.4. *For all $a \in G$, $\#aH = \#H$.*

Proof: It is enough to prove that for all $a \in G$, $f_a : H \to aH$ defined by $f_a(h) = a \circ h$ is a bijection. It is clear from the definition of ah that f_a is surjective. To prove that f_a is injective notice that if $a \circ h = a \circ h'$ then $a^{-1} \circ a \circ h = h'$ i.e., $h = h'$. ■

Claim 2.1.5. *All the different lateral classes of H form a partition of G.*

Proof: The thesis follows from the fact that all lateral classes are equal or disjoint. ■

Now it is derived the proof of the Lagrange Theorem. Since G is finite, then $G = a_1H \cup a_2H \cup ... \cup a_kH$ for some elements $a_i \in G$ such that $a_iH \cap a_jH = \emptyset$, and all the classes have the same number of elements, then $\#G = k\#H$ and thus $ord(H)|ord(G)$. ■

Definition 2.1.5 (*Order of group element*)

Let $(G, \circ)$ be a group and $a \in G$. The order of a in G is the least positive number i such that $a^i = \underbrace{a \circ \cdots \circ a}_{i \text{ times}} = e$, and is denoted by $ord(a)$. If such i does not exist, then $ord(a) = \infty$.

Corollary 2.1.6. *Let $(G, \circ)$ be a finite group and $a \in G$. Then $ord(a)$ divides $ord(G)$.*

Definition 2.1.6

A group $(G, \circ)$ is said cyclic if there is an element $a \in G$ such that for given $b \in G$, there is $i \in \mathbb{N}$ for which $b = a^i$. Such a is called a generator of G.

Definition 2.1.7 (*Ring*)

Let R be a set and $+$ and $\cdot$ be two binary operators over R. The triplet $(R, +, \cdot)$ is called a ring if and only if the following properties hold:

Commutative addition: *$(R, +)$ is an Abelian group with identity element 0.*

Multiplication: *$(R, \cdot)$ is a monoid with identity element 1.*

Distributivity: *For all $a, b, c \in R$, $a \cdot (b + c) = a \cdot b + a \cdot c$.*

A ring $(R, +, \cdot)$ is said to be commutative if and only if for every $a, b \in R$, $a \cdot b = b \cdot a$.

Definition 2.1.8 (*Integral domain*)

A ring is an integral domain if it has no zero divisors, i.e., $a \cdot b = 0 \Leftrightarrow a = 0 \vee b = 0$.

Examples of rings are $(\mathbb{N}, +, \cdot)$, $(\mathbb{Z}, +, \cdot)$ and $(\mathbb{Z}_n, +, \cdot)$.

Exercise 2.1.7. *Show that $(\mathbb{Z}_n, +, \cdot)$ is a integral domain if and only if n is prime.*

Definition 2.1.9 (*Field*)

A tuple $(F, +, \cdot)$ is said to be a field if and only if the following properties hold:

1. *$(F, +, \cdot)$ is an integral domain.*
2. *$(F - \{0\}, \cdot)$ is an Abelian group.*

• •

SECTION 2.2

The natural numbers

For further readings on this topic it is suggested to the reader the book [And71] of G. Andrews and the book [IR90] of K. Ireland and M. Rosen.

As usual the natural numbers are represented by $\mathbb{N}$ and the integers by $\mathbb{Z}$.

One of the centrals in Number Theory and, in particular, in Cryptography is the concept of divisibility.

Theorem 2.2.1 (Euclidean decomposition)

Let a and b be two arbitrary integers. If $a \neq 0$ then there are integers d and r, called respectively quotient e remainder, such that

$$\begin{cases} b = a \cdot d + r \\ 0 \leq r < |a| \end{cases}.$$

Such d and r are unique.

Proof: First, it is shown the existence of such integers.

If $|b| < |a|$ then $b = a \cdot 0 + b$ and thus, in this case just take $d = 0$ and $r = b$. Assume that $|b| \geq |a|$. Since $\mathbb{Z}$ is a well ordered set, and thus, in particular, each of its nonempty subsets have first element, it follows that the set

$$\{n \in \mathbb{Z} : b < a \cdot n\}$$

has a minimum element. Let n be that minimum. Take $d = n - 1$, i.e., the largest integers such that $a \cdot d$ does not exceed b. By construction $d \in \mathbb{Z}$.

On the other hand, taking $r = b - a \cdot d$ one also has that $r \in \mathbb{N}$ and $a \cdot d + r = a \cdot d + b - a \cdot d = b$. To finish the proof of the existence in remains to prove that $r \leq a$. If r was larger than $|a|$ this would mean that $|b| - |a \cdot d| > |a|$ which would be equivalent to say that $|b| > |a| \cdot |d + 1|$, contradicting the choice of d.

To prove the uniqueness of d and r assume, by reduction to absurd, that there are $d' \neq d$ e $r' \neq r$ such that $b = a \cdot d' + r'$ and $0 \leq r' < |a|$. Then, one would have that $a \cdot d + r = a \cdot d' + r'$, i.e., $a \cdot (d - d') = r' - r$ and $|r - r'| < |a|$. Hence, since $d \neq d'$, $|d - d'|$ is at least 1, one would get that $|a| \leq |a| \cdot |d' - d| = |r - r'| < |a|$ which is absurd. Thus $d' = d$ and $r = r'$. ■

Given two integers a and b, one says that b divides a, or that b is a multiple of a, or yet that b is a divisor of a, and it is usually written by $b|a$ is in the decomposition described in the last theorem $r = 0$, i.e., if there are integers d (necessarily unique) such that $a = b \cdot d$. In the next proposition some of the basic properties of the divisibility relation are stated.

Proposition 2.2.1. *Let $a, b, c, d, e \in \mathbb{Z}$. The relation $\cdot|\cdot$ has the following properties:*

1. *it is reflexive, i.e., $a|a$;*
2. *if $a|b$ then $a \leq b$;*
3. *it is transitive, i.e., if $b|a$ and $c|b$ then $c|a$;*
4. *it is antisymmetric, i.e., if $b|a$ and $a|b$ then $a = b$;*
5. *if $a|b$ e $a|c$ then $a|(b \cdot d \pm c \cdot e)$;*
6. *if $a \not| b \cdot c$ then $a \not| b$ and $a \not| c$;*
7. *if $a|b$ e $a \not| (b \cdot d + c)$ then $a \not| c$.*

The proof of these properties is left to the interested reader.

Definition 2.2.1

Let a and b integers. The natural number d is said the grater common divisor between a and b, if:

$$\begin{cases} d|a \wedge d|b \\ \forall c \ (c|a \wedge c|b \implies c|d) \end{cases}.$$

It is usual to denote by $gdc(a,b)$ the grater common divisor between a and b. It is also common, in particular, if $gcd(a,b) = 1$ then a and b are said to be co-prime or relatively prime.

Notice that the great common divisor is unique since if d and d' were such that $d = gcd(a,b)$ and $d' = gcd(a,b)$, then, in particular, one would have that $d|d'$ and $d'|d$, whence by the last properties one would have $d = d'$.

Exercise 2.2.2. *Let a, b and c be non null integers. Show that if $a|bc$ and a and b are co-primes, then $a|c$.*

Theorem 2.2.2

Let a and b be non null integers. Then

$$gcd(a,b) = \min\{z : z > 0 \ \wedge \ \exists \, x, y \in \mathbb{Z} \ (z = a \cdot x + b \cdot y)\}.$$

Proof: Consider the set $C = \{z : z > 0 \ \wedge \ \exists \, x, y \in \mathbb{N} \ (z = a \cdot x + b \cdot y)\}$. Notice that $C \neq \emptyset$, since a and b being non null, $z = a^2 + b^2 = a \, . \, a + b \, . \, b$ belongs to C. Thus, since $\mathbb{Z}$ is well ordered, C has a first element.

Let d be the first element and let $x, y \in \mathbb{Z}$ such that $d = a \cdot x + b \cdot y$ and $0 \leq y < |a|$.

To show that $d|a$ assume that q and r are such that $a = d \, . \, q + r$ e $0 \leq r \leq |d|$ as stated in Theorem 2.2.1. Hence $r = a - d \, . \, q = a - (a \cdot x + b \cdot y) \, . \, q = a \, . \, (1 - q \, . \, x) + b(-yq)$ and thus $r \in C$. Thus, if $r > 0$ then, in particular on would have that r would be at least the minimum of C, i.e., $r \geq d$, which is not possible. It follows that $r = 0$ and thus $d|a$. In an analogous way one can show that $d|b$.

With this reasoning one showed that d is a common divisor of a and b. Assume that c is another common divisor of a and b. Hence, in particular c divides $a \cdot x$ and $b \cdot y$ and, hence, by previous

proposition $c|(a \cdot x + b \cdot y)$, i.e $c|d$, which implies that d is the greater common divisor. ■

As an example consider the $gcd(12, 20) = 4$. In fact one can write 4 as the following combination of 12 and 20: $4 = 2 \,.\, 12 - 1 \,.\, 20$.

The next exercise asks the reader to prove some of basic properties of the greater common divisor.

Exercise 2.2.3. *Let a, b, c and d be non null integers. Show that:*

1. $1 = gcd(a, b)$ *if and only if there are* $x, y \in \mathbb{N}$ *such that* $1 = a \,.\, x + b \,.\, y$;
2. $gcd\left(\frac{a}{gcd(a,b)}, \frac{b}{gcd(a,b)}\right) = 1$;
3. $gcd(a \,.\, d, b \,.\, d) = d \cdot gcd(a, b)$;
4. *if* $a|b \,.\, c$ *then* $\left.\frac{a}{gcd(a,b)}\right| c$.

Definition 2.2.2

An number $p \in \mathbb{N}$ is called prime if $p > 1$ and its unique divisors are 1 and p.

A number $n \in \mathbb{N}$ that is not prime, i.e., a number that has at least one non trivial divisor (i.e., different from 1 and itself) is called composed.

It is widely known any composed natural number n can be written as a product of prime numbers.

Theorem 2.2.3 (Fundamental of arithmetic)

let n be a natural number such that $n > 1$. Then there are prime numbers $p_1 < \cdots < p_k$ and naturals $a_1, \cdots, a_k$ such that $n = \prod p_i^{a_i}$.

Proof:[Sketch of the proof] If n is already a prime the result is trivially true. Assume without loss of generality that n is not prime, i.e., there are $m \in \mathbb{N}$ such that $m > 1$ and $m|n$. Consider the set

$$D_1 = \{m : m|n \ \wedge m > 1\}$$

of natural divisors of n. Since D_1 is a subset of $\mathbb{N}$ it admits minimum element, say p.

Claim 2.2.4. *p is a prime number.*

In fact, if p was not prime then itself would admit a nontrivial divisor $q < p$ and thus, since $p|n$ and $q|p$ then by Proposition 2.2.1 one would have $q|n$, contradicting the choice of q.

Take $p_1 = p$ and consider now the set

$$D_2 = \left\{ m : m \left| \frac{n}{p_1} \ \wedge m > 1 \right. \right\}.$$

Again the minimum of D_2 is a prime number p. Take $p_2 = p$ and repeat the process considering, while possible, the set:

$$D_i = \left\{ m : m \left| \frac{n}{\prod_j p_{j<i}} \ \wedge m > 1 \right. \right\}.$$

Notice that this process must finish. It is easy to see that with at most $n/2$ steps the set D_i will be empty since each time one considers a new subset of divisors the upper bound of the set is divided by two (the least prime number candidate to be in D).

Notice also that, by construction, $p_i \leq p_{i+1}$. Hence the thesis follows by associating the similar primes. ■

Theorem 2.2.4

The set of prime numbers is infinite and the difference between two consecutive prime numbers can be arbitrarily large.

Proof: Assume by contradiction that the set of prime numbers is finite, i.e., that the set of prime numbers is:

$$P = \{p_1, \cdots, p_k\}.$$

Let $n = p_1 . \cdots . p_k + 1$. Notice that n is larger than any number in P since it is the product of all of elements of P which are prime numbers and thus, in particular, larger than 1. Thus n is not a prime and thus must be compose. Thus, it must to exist $p_i \in P$ such that $p_i|n$. Consequently by Proposition 2.2.1, since $p_i|p_1 . \cdots . p_k$ one would conclude that $p_i|n - p_1 . \cdots . p_k$ which would imply that $p_i|1$ and thus $p_i = 1$ contradicting the primality of p_i. The contradiction followed from the supposition that P was finite.

To show that the difference between two consecutive prime numbers can be arbitrarily large it is enough to observe that the sequence of the form $a_i = (n+1)! + i$ with $2 \leq i \leq n+1$ does not contain any prime number. In fact, for all i such that $2 \leq i \leq n+1$, $i|(n+1)!$ and, thus, again by Proposition 2.2.1, $i|(n+1)! + i$. Notice that the sequence a_i can be arbitrarily large. ■

The last proposition establishes the existence of infinitely many prime numbers but it does not say states anything about its density, as a subset of $\mathbb{N}$. The next theorem gives an idea of the prime numbers are distributed giving an asymptotic behavior.

Theorem 2.2.5 (*Prime numbers theorem*)

Let $\pi(n)$ be the number of primes between 2 and n. Thus:

$$\pi(n) = \lim_{n} \frac{n}{\ln n}$$

where ln *is used to denote, as usual, the logarithm computed in the natural base e.*

Proof: to be done ■

For any positive integer n define $\phi(n)$, the Euler function[1] as the number of positive integers a less than n such that a is relatively prime to n. By definition of prime number it is easy to see that $\phi(p) = p - 1$. Also, $\phi(p^n) = p^n - p^{n-1}$, since among the $p^n - 1$ positive integers less than p^n there are $p^{n-1} - 1$ numbers that are not co-prime to p^n.

Exercise 2.2.5. *Show that if m and n are co-primes then $\phi(m \cdot n) = \phi(m) \cdot \phi(n)$ and conclude that, in particular, if m and n are different prime numbers, $\phi(m \cdot n) = (m-1)(n-1)$.*

Exercise 2.2.6. *Show that if n is a non null natural number, then $\sum_{d|n} \phi(d)$.*

[1] By a function f from A to B, written $f : A \to B$, we mean that $\text{dom}\, f = A$ and the $\text{img}\, f \subseteq B$ since we only work in this course with total functions.

SECTION 2.3

Congruences and modular algebras

In this section it is studied the theme of congruences modulo a natural number and it is explored some of its basic properties that well be used in due course in the subsequent chapters of this work.

Definition 2.3.1 (*congruence modulo n*)

Let a, b be integers and n a natural number non null. It is said that a is congruent with b modulo n, or that a and b are congruents modulo n, and one writes $a \equiv b \pmod{n}$, if $n|b-a$.

In the previous definition one may have $n = 1$, but this case is not very interesting from the mathematical properties point of view since any two integers are divisible by 1 and thus no useful properties can be explored. The following theorem gives an equivalent formulation that is commonly used.

Theorem 2.3.1

Let a, b be integers and n a non null natural number. a and b are congruents modulo n if and only if the remainder of the quotient of each of them by n are equal.

Proof:

($\leftarrow$)

Let d_1, d_2, r_1 and r_2 be integers such that $a = d_1 \,.\, n + r_1$, $b = d_2 \,.\, n + r_2$ and $0 < r_1, r_2 < n$. Since $a \equiv b \pmod{n}$, on has $n|(b-a)$ and thus $n|(d_1 \,.\, n + r_1 - d_2 \,.\, n - r_2)$. Hence, by Proposition 2.2.1 one can conclude that $n|(r_1 - r_2)$. Since $0 < r_1, r_2 < n$ then $r_1 = r_2$, as envisaged.

($\rightarrow$)

Assume that $a = d_1 \,.\, n + r$ and that $b = d_2 \,.\, n + r$ with $0 < r < n$, for some integers d_1, d_2 and r. Hence $a - b = (d_1 - d_2) \,.\, n$ and, so, $n|a-b$, i.e., $a \equiv b \pmod{n}$. ■

Theorem 2.3.2

The congruence modulo a natural number relationship $\equiv$ is an equivalence relationship in $\mathbb{Z}$, i.e., given $a, b, c \in \mathbb{Z}$ and $n \in \mathbb{N}$ non null:

- *$\equiv$ is reflexive: $a \equiv a \pmod{n}$;*
- *$\equiv$ is symmetric: if $a \equiv b \pmod{n}$ then $b \equiv a \pmod{n}$;*
- *$\equiv$ is transitive: if $a \equiv b \pmod{n}$ and $b \equiv c \pmod{n}$ then $a \equiv c \pmod{n}$;*

Proof: To prove the reflexivity just observe that $n|a - a = 0$.

For the symmetry notice that if $n|b - a$ then $n|a - b = -(b - a)$.

To show the transitivity notice that if $n|b - a$ and $n|c - b$ then $n|c - a = (c - b) + (b - a)$. ■

In the next exercise the reader is invited to show some of the basic properties of the congruences.

Exercise 2.3.1. *Let a, b, c and d be integers and $n > 1$. Show that:*

1. *If $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$ then $a + c \equiv b + d \pmod{n}$;*
2. *If $a \equiv b \pmod{n}$ then $-a \equiv -b \pmod{n}$;*
3. *If $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$ then $a \cdot c \equiv b \cdot d \pmod{n}$. In particular, for all d, if $a \equiv b \pmod{n}$ then $a^d \equiv b^d \pmod{n}$;*
4. *$a + b \equiv b + a \pmod{n}$;*
5. *$(a + b) + c \equiv a + (b + c) \equiv a + b + c \pmod{n}$;*
6. *$(a + b) \cdot c \equiv c \cdot (a + b) \equiv a \cdot c + b \cdot c \pmod{n}$.*

It is well known that an equivalence relationship like congruence modulo n over a set partitions $\mathbb{Z}$ into equivalence classes. Since there are at most n possible different remainders considering the quotient by n, $\mathbb{Z}$ is portioned into n classes each of them containing exactly the numbers that are congruent modulo n. Denoting by $\bar{a}$ the class of equivalence of $a \pmod{n}$, i.e.,

$$\bar{a} = \{x \in \mathbb{Z} : a \equiv x \pmod{n}\} = a\mathbb{Z}$$

then one can consider the quotient set $\mathbb{Z}_n = \mathbb{Z}/\equiv_n$ formed by the equivalent classes modulo n i.e.,

$$\mathbb{Z}_n = \left\{\overline{0}, \overline{1}, \cdots, \overline{n-1}\right\}.$$

Using the properties established in Exercise 2.3.1 it is easy to define $+$ and $\times$ as follows:

- $\overline{a} + \overline{b} = \overline{a+b}$
- $\overline{a} \times \overline{b} = \overline{a \cdot b}$

It is also easy to derive the following theorem:

Theorem 2.3.3

$(\mathbb{Z}_n, +)$ is a group.

Exercise 2.3.2. *Show or refute the following statement. "For all $n \in \mathbb{N}$, $(\mathbb{Z}_n, \times)$ is a group."*

Clearly the statement of the last exercise is false since for example, when $n = p \times q$ where p and q are primes, then $\overline{p}$ does not have a symmetric element, i.e., the equation $p \times x \equiv 1 \pmod{n}$ does not have any solution. In fact, in general, the equation "$ax \equiv b \pmod{n}$" is solvable if and only if $gcd(a, n)|b$.

Exercise 2.3.3. *Show that for a and b integers and $n > 1$, the equation $ax \equiv b \pmod{n}$ is solvable if and only if $gcd(a, n)|b$.*

It follows from the previous exercise that $(\mathbb{Z}_n^*, \times)$, where $\mathbb{Z}_n^* = \{\bar{a} : gdc(a, n) = 1\}$ is also a group. In fact, using the properties established in Exercise 2.3.1 one can derive the following theorem.

Proposition 2.3.4. *If n is prime, $(\mathbb{Z}_n^*, +, \times)$ is a ring with the property that $(\mathbb{Z}_n^*, +)$ is a cyclic group.*

The Chinese remainder theorem is very useful tool that will be used in due course and states that if $m_1, m_2, \ldots, m_r$ are natural numbers greater than or equal to 2 that are pairwise co-primes, and if $0 \leq a_i < m_i$ for $1 \leq i \leq r$, then there is a unique integer a such that $0 \leq a < m = m_1 m_2 \cdots m_r$ and $a \equiv a_i \pmod{m}_1$ for $1 \leq i \leq r$.

Theorem 2.3.4 (Chinese remainder theorem)

Let $m_1, m_2, \ldots, m_r$ be positive integers that are pairwise co-primes. Let $a_1, a_2, \ldots, a_r$ be arbitrary integers. Then there is an integer a such that

$$\begin{cases} a \equiv a_1 \pmod{m_1} \\ a \equiv a_2 \pmod{m_2} \\ \quad \vdots \\ a \equiv a_r \pmod{m_r} \end{cases}.$$

Furthermore, a is unique modulo $M = m_1 \times m_2 \times \cdots m_r$.

Proof: The proof is carried out by finding an algorithm for constructing a. For each $1 \leq i \leq r$, define M_i by

$$M_i = (M/m_i)^{\phi(m_i)}.$$

Since M/m_i is relatively prime to m_i and divisible by m_j for every j not equal to i, one has that

$$\begin{aligned} M_i &\equiv 1 \pmod{m_i} \\ M_j &\equiv 0 \pmod{m_i} \quad \text{if} \quad j \neq i. \end{aligned}$$

Also consider a as

$$a = a_1 \times M_1 + a_2 \times M_2 + \cdots + a_r \times M_r.$$

To see that a is unique modulo M, let b be any other integer satisfying all the congruences. Then for each m_i, a and b are congruent modulo m_i. In other words, m_i divides $b - a$. Since this is true for every i, M divides $b - a$ which means that

$$a \equiv b \pmod{M}.$$

■

Theorem 2.3.5 (Fermat's little theorem)

Let p and a be two natural numbers. If p is prime and $p \nmid a$, then $a^{p-1} \equiv 1 \pmod{p}$.

From the fact that p is prime and for that reason $\phi(p) = p - 1$, Fermat's little theorem is a special case of the following theorem.

Theorem 2.3.6 (Euler's theorem)

Let n and a be two natural numbers. If $gcd(a,n) = 1$ then $a^{\phi(n)} \equiv 1 \pmod{n}$.

Proof: Since $gcd(a,n) = 1$ then $a \pmod{n} \in \mathbb{Z}_n^*$. Thus, by Theorem 2.1.6 $ord(a)|ord(\mathbb{Z}_n^*) \Leftrightarrow ord(a)|\phi(n)$, which implies that, there is $d \in \mathbb{N}$ such that $\phi(n) = d \cdot ord(a)$. On the other hand, by definition of $ord(a)$ in $\mathbb{Z}_n$, $a^{ord(a)} \equiv 1 \pmod{n}$. So by Exercise 2.3.1.3

$$1 \equiv 1^d \equiv (a^{ord(a)})^d \equiv a^{\phi(n)} \pmod{n}.$$

■

Theorem 2.3.7

Let p and a be two natural numbers. If p is prime then $a^p \equiv a \pmod{p}$.

The proof of this theorem is left to the interested reader.

Theorem 2.3.8

Let p and a be two natural numbers such that p is prime. Then a is a generator of $\mathbb{Z}_p^*$ if and only if $a^{(p-1)/q} \not\equiv 1 \pmod{p}$ for all primes q such that $q|(p-1)$.

Proof:

($\leftarrow$)

If a is a generator of $\mathbb{Z}_p*$ then, for all $1 \leq i \leq p-2$, $a^i \not\equiv 1 \pmod{p}$ and thus, in particular, the result follows.

($\rightarrow$)

Assume that a is not a generator of $\mathbb{Z}_p^*$. Let d be the order of a, i.e., $a^d \equiv 1 \pmod{p}$. Then, $d < p-1$ and by Lagrange's Theorem 2.1.1, one has $d|(p-1)$. Hence $(p-1)/d$ is an integer exceeding 1. Let q be a prime divisor of $(p-1)/d$. Clearly, $d|((p-1)/q)$. Since $a^d \equiv 1 \pmod{p}$ and $d|((p-1)/q)$, it follows that $a^{(p-1)/q} \equiv a^{dc} \pmod{p} \equiv 1 \pmod{p}$. ■

...

2.3.1 Finite Fields

In this sub subsection it is discussed other kind of finite fields other done $\mathbb{Z}_p^*$ for p is prime. In particular it will be discuss field with $q = p^n$ elements where p is prime and $n >1$ is an integer.

Definition 2.3.2

Let p be a prime number. The set $\mathbb{Z}_p[x]$ is the set of all polynomials with coefficients over $\mathbb{Z}_p$. With the usual operations of sum and multiplication over $\mathbb{Z}_p$ then $(\mathbb{Z}_p[x], +, \times)$ is a finite ring.

Definition 2.3.3

Let $(\mathbb{Z}_p[x], +, \times)$ be a finite ring and let $f(x)$, $g(x)$ be two polynomials in $\mathbb{Z}_p[x]$. One says that $f(x)$ divides $g(x)$ and denotes it by $f(x)|g(x)$ if there exists $q(x) \in \mathbb{Z}_p[x]$ such that $g(x) = q(x) \times f(x)$.

Definition 2.3.4

Let $(\mathbb{Z}_p[x], +, \times)$ be a finite ring and let $f(x)$ be a polynomial in $\mathbb{Z}_p[x]$. One says that f has degree n, and denotes by $deg(f) = n$, if the term ax^n is the highest non null term.

Definition 2.3.5

Let $(\mathbb{Z}_p[x], +, \times)$ be a finite ring and let $f(x)$, $g(x)$, $h(x)$ be polynomials in $\mathbb{Z}_p[x]$. Assume that $deg(f) > 1$. Then one defines $g(x) \equiv h(x) \pmod{f(x)}$ if $f(x)|(g(x) - h(x))$.

Let now $\mathbb{Z}_p[x]$ be a ring and let $f(x)$ be a polynomial in $\mathbb{Z}_p[x]$ of degree at least 1. The ring of polynomials "modulo $f(x)$", denoted by $\mathbb{Z}_p[x]/(f(x))$ is obtained by considering all possible remainders modulo $f(x)$. Notice that, again by the Euclides algorithm, given $g(x)$ there are unique polynomials $q(x)$ (the quotient) and $r(x)$ (the remainder) such that $g(x) = q(x)f(x) + r(x)$ and $deg(r) < n$, and hence any polynomial in $\mathbb{Z}_p[x]$ is congruent modulo $f(x)$ to a polynomial of degree $n - 1$, which implies that $\mathbb{Z}_p[x]/(f(x))$ is the set of all p^n polynomial of degree at most $n - 1$.

Definition 2.3.6

Let now $\mathbb{Z}_p[x]$ be a ring and let $f(x)$ be a polynomial in $\mathbb{Z}_p[x]$. f is said to be irreducible is there are no polynomials $p(x)$ and $q(x)$ in $\mathbb{Z}_p[x]$ of degree at least 1 such that $f(x) = p(x)q(x)$.

Theorem 2.3.9

Let now $\mathbb{Z}_p[x]$ be a ring and let $f(x)$ be a polynomial in $\mathbb{Z}_p[x]$. $\mathbb{Z}_p[x]/(f(x))$ is a field if and only if $f(x)$ is irreducible. Moreover, $ord(\mathbb{Z}_p[x]/(f(x))) = p^{deg(f)}$.

Proof: It is left to the interested reader. ■

It is usual to denote this kind of fields by $\mathbb{GF}(p^k)$ and called them Galois fields named after the mathematician Évariste Galois who had studied them deeply.

Example 2.3.5. *Consider $\mathbb{Z}_2[x]$ and $g(x) = x^2 + 1$ and $f(x) = x^2$.*

Notice that $g(x)$ is not irreducible since $g(x) = (x+1)(x+1)$. One can easily verify that the polynomial $f(x)$ is irreducible. Thus the $\mathbb{Z}_p[x]/(f(x))$ if formed with polynomials of the form $a_0 + a_1 x$ where $a_0, a_1 \in \{0, 1\}$, i.e, 0, 1, x, $x+1$.

The operations of field are:

$+$	0	1	x	$x+1$
0	0	1	x	$x+1$
1	1	0	$x+1$	x
x	x	$x+1$	0	1
$x+1$	$x+1$	x	1	0

$\times$	0	1	x	$x+1$
0	0	0	0	0
1	0	1	x	$x+1$
x	0	x	$x+1$	1
$x+1$	0	$x+1$	1	x

The interesting aspect about this fields is the fact that they are cyclic, an important feature that will be used to create a public cryptographic system in a subsequent section. The proofs of the following results are easy and can be found in any classical algebra book. Hence, is an excellent exercise for the reader for the reader.

Theorem 2.3.10

Let $\mathbb{Z}_p[x]$ be a ring and k an integer. There exists an irreducible polynomial $f(x)$ of degree k in $\mathbb{Z}_p[x]$. Moreover, if $h(x)$ is another irreducible polynomial of degree k in $\mathbb{Z}_p[x]$ then $\mathbb{Z}_p[x]/(f(x))$ and $\mathbb{Z}_p[x]/(h(x))$ are isomorphic.

Theorem 2.3.11

Let $\mathbb{Z}_p[x]$ be a ring and $f(x)$ an irreducible polynomial in $\mathbb{Z}_p[x]$. Then $\mathbb{Z}_p[x]/(f(x))$ is cyclic.

• •

SECTION 2.4

Exercises

Exercise 2.4.1. *Assume that G be a finite group, and let g be an element of G of order a. Show that $g^n = g^m$ if and only if $n \equiv m \pmod{a}$.*

Exercise 2.4.2. *Show that if $x^{n-1} \equiv 1 \pmod{n}$ for all integers x that are not multiples of n, then n is prime.*

Notice that a slightly weaker statement that $x^{n-1} \equiv 1 \pmod{n}$ for all x such that $gcd(x, n) = 1$ is not enough not imply that n is prime. There are numbers, called Carmichael numbers that are counterexamples for this statement.

Exercise 2.4.3. *Let p be prime number. Show that $(p-1)! \equiv -1 \pmod{p}$.*

This exercise is known as Wilson's Theorem.

The next exercise will be important for the understanding the work of public key cryptographic systems.

Exercise 2.4.4. *Find the least non-negative residue of $3^{15} \pmod{1}7$ and $15^{81} \pmod{1}3$, without carrying long multiplications.*

Exercise 2.4.5. *Use Fermat's theorem 2.3.5 to compute $3^{47} \pmod{2}3$.*

The next exercise intends to provide the reader a mechanism to compute the inverses modulo n.

Exercise 2.4.6. *Using the Euclides algorithm for division, compute the inverses of:*

1. 2 *in* $\mathbb{Z}_{11}$*;*
2. 7 *in* $\mathbb{Z}_{15}$*;*
3. 7 *in* $\mathbb{Z}_{16}$*;*
4. 5 *in* $\mathbb{Z}_{13}$*;*

Exercise 2.4.7. *Use the Lagrange theorem 2.1.1 to determine the inverses of the previous exercise.*

Exercise 2.4.8. *Compute all the solutions with* $0 \leq x \leq 34$ *of the following system of equations*

$$\begin{cases} x \equiv 4 \pmod 5 \\ x \equiv 5 \pmod 7 \end{cases}$$

Exercise 2.4.9. *Compute all the solutions with* $0 \leq x \leq 104$ *of the following system of equations*

$$\begin{cases} x \equiv 2 \pmod 3 \\ x \equiv 4 \pmod 5 \\ x \equiv 5 \pmod 7 \end{cases}$$

Exercise 2.4.10. *Show that every irreducible polynomial in* $\mathbb{Z}_p[x]$ *is a divisor of* $x^{p^n} - x$ *for some n.*

Exercise 2.4.11. *Construct a finite field with 8 elements. hint: find an irreducible polynomial of degree 3 in* $\mathbb{Z}_2$*.*

Exercise 2.4.12. *Show thatab* $= ac$*, with* $a \neq 0$*, does not necessarily imply* $b = c$ *in a ring. Envisaged the validity of this statement in a ?eld?*

Exercise 2.4.13. *Show that* $\mathbb{Z}/\mathbb{Z}_n$ *is a ?eld if and only if n is a prime number.*

CHAPTER 3

Probabilities and Shannon entropy

In this chapter is introduced all the concepts regarding probabilities that will be used in due course on these notes. It is a central topic one talks about cryptography and this relation will become clear later on the text. Related with probabilities and cryptography is also the concept of Shannon entropy that is presented in Section 3.2.

.....

SECTION 3.1

Probability theory

The reader is referred, for example, to the book [Kal02] for further and details development on this issue. Here it is presented the necessary concepts and main results for the understanding of the rest of the material presented in this book.

The word probability, in mathematics, is used to refer a measure of the weight of empirical evidence that an event will occur among several ones.

Let X be an experiment that can produce several possible results. The collection of all results is called the sample space of the experiment. One possible collection of allowed results is called an event. For example, rolling a die can produce six possible results and the subset $\{1, 3, 5\}$ is an event that corresponds to "obtaining an odd number on the die".

A probability is an assignment of a value between zero and one to every event, with the requirement that the event made up of all possible results (in the example of the die, the event $\{1,2,3,4,5,6\}$) is assigned with the value one. The function that assigns to every elementary event a probability is called a Probability distribution over the power set of possible results.

Definition 3.1.1

A discrete random variable $\mathbf{X}$ *is a finite set* X *together with a probability distribution defined on* X*. The probability that the random variable* $\mathbf{X}$ *takes on the value* x *is denoted by* $\Pr[\mathbf{X} = x]$ *or by* $\Pr[x]$ *if the random variable* $\mathbf{X}$ *is clear from the context. It must be the case that the three following conditions are satisfied:*

$$\begin{cases} \Pr(\mathbf{X} = x) \geq 0, \forall x \in X; \\ \Pr(\mathbf{X}) = 1; \\ \Pr(\mathbf{X} \in A \cup B) = \Pr(\mathbf{X} \in A) + \Pr(\mathbf{X} \in B) \text{ if } A \cap B = \emptyset \end{cases}$$

where, if A *is a collection of events then* $\Pr[\mathbf{X} \in A] = \Pr[x \in A] = \sum_{x \in A} \Pr[x]$.

Example 3.1.1. *Consider a random throw of three coins which can be modeled by the following random variable* $\mathbf{Z}$ *defined on the set*

$$Z = \{Heads, Tail\} \times \{Heads, Tail\} \times \{Heads, Tail\}$$

where the probability of each of possible triplets x *is* $\Pr[x] = 1/8$*. Consider also the number of Heads on the triplet. Each possible number* $1, 2, 3$ *defines an event, and the probabilities of these events are:*

$$\Pr(H_1) = \frac{3}{8} \qquad \Pr(H_2) = \frac{1}{2}; \qquad \Pr(H_3) = \frac{1}{8}.$$

Since the events H_1*,* H_2 *and* H_3 *are disjoint and their union is the entire set of possibilities, i.e., they form a partition of* Z*, it follows that the value of the number of Heads is also a random variable in its own right.*

It will be also very useful the concepts of joint and conditional probabilities introduced in the next definition

Definition 3.1.2

Let $\mathbf{X}$ *and* $\mathbf{Y}$ *be random variables defined over the finite sets* X *and* Y*, respectively.*

Joint probability: *The joint probability* $\Pr[\mathbf{X} = x, \mathbf{Y} = y]$ *is the probability that* $\mathbf{X}$ *takes on the value* x *and* $\mathbf{Y}$ *takes on the value* y*.*

Conditional probability: *The conditional probability* $\Pr[\mathbf{X} = x | \mathbf{Y} = y]$ *denotes the probability that* $\mathbf{X}$ *takes on the value* x *given that* $\mathbf{Y}$ *takes on the value* y*.*

Independence: *The random variables* $\mathbf{X}$ *and* $\mathbf{Y}$ *are called independent random variables if for all* $x \in \mathbf{X}$ *and* $y \in \mathbf{Y}$, $\Pr[\mathbf{X} = x, \mathbf{Y} = y] = \Pr[\mathbf{X} = x] \Pr[\mathbf{Y} = y]$.

The conditional probability of events A and B such that $\Pr[B] \neq 0$ can be computed by the formula $\Pr[A|B] = \dfrac{\Pr[A \cap B]}{\Pr[B]}$.

The joint probability and the conditional probability are related since $\Pr[\mathbf{X} = x, \mathbf{Y} = y] = \Pr[\mathbf{X} = x] \cdot \Pr[\mathbf{Y} = y | \mathbf{X} = x]$.

Theorem 3.1.1 (Bayes' theorem)

If $\Pr[\mathbf{Y} = y] > 0$ *then* $\Pr[\mathbf{X} = x | \mathbf{Y} = y] = \dfrac{\Pr[\mathbf{X} = x] \Pr[\mathbf{Y} = y | \mathbf{X} = x]}{\Pr[\mathbf{Y} = y]}$.

Proof: Just use the facts that for all $x \in \mathbf{X}$ and $y \in \mathbf{Y}$, $\Pr[\mathbf{X} = x, \mathbf{Y} = y] = \Pr[\mathbf{X} = x] \Pr[\mathbf{Y} = y | \mathbf{X} = x]$ and $\Pr[\mathbf{X} = x, \mathbf{Y} = y] = \Pr[\mathbf{Y} = y] \Pr[\mathbf{X} = x | \mathbf{Y} = y]$. ■

Corollary 3.1.2. *The random variables* $\mathbf{X}$ *and* $\mathbf{Y}$ *are independent if and only if for all* $x \in \mathbf{X}$ *and* $y \in \mathbf{Y}$, $\Pr[\mathbf{X} = x | \mathbf{Y} = y] = \Pr[\mathbf{X} = x]$.

The concept of the expected value or expectation, or mean of a random variable is the weighted average of all possible values that the random variable can take on it will also be of great use in meantime. In the case of discrete random variables, the weights used in computing this average are the probabilities.

Definition 3.1.3

Let $\mathbf{X}$ *be a random variable defined over the finite set* X*. The expectation of this random variable, denoted by* $E(\mathbf{X})$ *is defined as* $E(\mathbf{X}) = \sum_{x \in X} x \Pr[\mathbf{X} = x]$.

Exercise 3.1.3. *Prove that E is a linear operator.*

• • •

3.1.1 The notion of bias of a distribution

In many applications of Cryptography and specially in the cryptanalysis of a system the notion of bias of a distribution is crucial, in the sense that it might give some insight on the choices that were made and, for example, determine roughly the security of the scheme, since the less bias the more random look the distributions has and harder to attack. Intuitively, given a random variable $\mathbf{X}$ the bias of the distribution measures how far that distribution is from being uniform. Formally:

Definition 3.1.4 (*bias of a random variable*)

Let $\mathbf{X}$ *be a random variable over* $\{0,1\}$*. The bias of* $\mathbf{X}$*, denoted b* ε *is*

$$\varepsilon = \Pr[\mathbf{X_i} = 0] - \frac{1}{2}.$$

Notice that, in these cases, the bias of a random variable $\mathbf{X}$ is a value between $-\frac{1}{2}$ and $\frac{1}{2}$ and, in particular, one can express the probability distribution based on the bias as follows:

$$\Pr[\mathbf{X_i} = 0] = \varepsilon + \frac{1}{2} \qquad \Pr[\mathbf{X_i} = 1] = \varepsilon - \frac{1}{2}.$$

Now, let $\mathbf{X_1}, \mathbf{X_2}$ be two independent random variables over $\{0,1\}$ and p_1 and p_2 be such that $\Pr[\mathbf{X_i} = 0] = p_i$, for $i = 1, 2$. Hence, by independence of the two random variables, one has:

$$\begin{aligned} &\Pr[\mathbf{X_1} = 0, \mathbf{X_2} = 0] = p_1 p_2, && \Pr[\mathbf{X_1} = 0, \mathbf{X_2} = 1] = p_1(1-p_2) \\ &\Pr[\mathbf{X_1} = 1, \mathbf{X_2} = 0] = (1-p_1)p_2 && \Pr[\mathbf{X_1} = 1, \mathbf{X_2} = 1] = (1-p_1)(1-p_2) \end{aligned}$$

and thus, the random variable $\mathbf{X_1} \oplus \mathbf{X_2}$ has probability distribution

$$\begin{aligned} \Pr[\mathbf{X_1} \oplus \mathbf{X_2} = 0] &= \Pr[\mathbf{X_1} = 0, \mathbf{X_2} = 0] + \Pr[\mathbf{X_1} = 1, \mathbf{X_2} = 1] = p_1 p_2 + (1-p_1)(1-p_2) \\ \Pr[\mathbf{X_2} \oplus \mathbf{X_2} = 1] &= \Pr[\mathbf{X_1} = 1, \mathbf{X_2} = 0] + \Pr[\mathbf{X_1} = 0, \mathbf{X_2} = 1] = p_1(1-p_2) + p_1(1-p_2) \end{aligned}$$

One can now derive the bias of this random variable from the bias ε_1 and ε_2 of $\mathbf{X_1}$ and X_2 respectively, namely:

$$\begin{aligned}
\varepsilon_{\mathbf{X_1}\oplus\mathbf{X_2}} &= \Pr[\mathbf{X_1}\oplus\mathbf{X_2}=0]-\frac{1}{2}\\
&= p_1p_2+(1-p_1)(1-p_2)-\frac{1}{2}\\
&= \left(\varepsilon_1+\frac{1}{2}\right)\left(\varepsilon_2+\frac{1}{2}\right)+\left(\varepsilon_1-\frac{1}{2}\right)\left(\varepsilon_2-\frac{1}{2}\right)\\
&= \varepsilon_1\varepsilon_2+\frac{1}{2}(\varepsilon_1+\varepsilon_2)+\frac{1}{4}+\varepsilon_1\varepsilon_2-\frac{1}{2}(\varepsilon_1+\varepsilon_2)+\frac{1}{4}-\frac{1}{2}\\
&= 2\varepsilon_1\varepsilon_2.
\end{aligned}$$

In the next lemma it is generalized this result for a finite number of independent random variables.

Lemma 3.1.4 (Piling-up lemma)**.** *Let* $\mathbf{X_1},\cdots,\mathbf{X_k}$ *be independent random variables over* $\{0,1\}$*. The bias of the random variable* $\mathbf{X}=\mathbf{X_1}\oplus\cdots\oplus\mathbf{X_k}$ *can be computed by the following formula:*

$$\varepsilon_{\mathbf{X}}=2^{k-1}\prod_{i=1}^{k}\varepsilon_{\mathbf{X_i}}.$$

where $\varepsilon_{\mathbf{X_i}}$ *for* $1\leq i\leq k$ *is the bias of the random variable* $\mathbf{X_i}$*.*

Proof: The proof of the lemma is done by induction on the number of independent random variables considered.

(basis) The result is trivially verified when $k=1$. Notice that the discussion above proves the case $k=2$.

(Step) Assume that for all $l\leq k$ the bias of $\mathbf{X}$ can be computed by the formula $\varepsilon_{\mathbf{X}}=2^{k-1}\prod_{i=1}^{k}\varepsilon_{\mathbf{X_i}}$ and assume that Let $\mathbf{X_1},\cdots,\mathbf{X_{k+1}}$ are independent random variables over $\{0,1\}$. Then

$$\mathbf{X}=\mathbf{X_1}\oplus\cdots\oplus\mathbf{X_k}\oplus\mathbf{X_{k+1}}=(\mathbf{X_1}\oplus\cdots\oplus\mathbf{X_k})\oplus\mathbf{X_{k+1}}$$

Now it is used twice the hypothesis of induction. In the first case, the bias of $\mathbf{X_1}\oplus\cdots\oplus\mathbf{X_k}$ is $2^{k-1}\prod_{i=1}^{k}\varepsilon_{\mathbf{X_i}}$ and thus applying the hypothesis of induction for $k=2$ with the two random distributions being $\mathbf{X_1}\oplus\cdots\oplus\mathbf{X_k}$ and X_{k+1} one has:

$$\varepsilon_{\mathbf{X}}=2\cdot\varepsilon_{\mathbf{X_1}\oplus\cdots\oplus\mathbf{X_k}}\cdot\varepsilon_{X_{k+1}}=2\cdot2^{k-1}\left(\prod_{i=1}^{k}\varepsilon_{\mathbf{X_i}}\right)\cdot\varepsilon_{X_{k+1}}=2^{k}\prod_{i=1}^{k+1}\varepsilon_{\mathbf{X_i}}.$$

as claimed. ■

Corollary 3.1.5. *Let* $\mathbf{X_1}, \cdots, \mathbf{X_k}$ *be independent random variables over* $\{0,1\}$*. If the bias of some the random variable* $\mathbf{X} = \mathbf{X_i}$ *then the bias of* $\mathbf{X_1} \oplus \cdots \oplus \mathbf{X_k}$ *cis also zero.*

Notice that this results require the independence condition. The reader is invited to present examples of non independent random variables in which the results above do not hold.

• •

SECTION 3.2

Entropy and information theory

The reader is referred to the book [CT91] for a full comprehensive study of this thematic.

The concept of entropy is central in information theory and thus in cryptography. Heuristically speaking, entropy is a measure to evaluate the uncertainty associated with a random variable by the expected value of the information, usually counted in number of bits, and can be realized as a function of the probability distribution inherent. The concept of entropy was introduced by Claude E. Shannon in his seminal paper [Sha48] of 1948 entitled "A Mathematical Theory of Communication".

One can formally present the concept of entropy in an axiomatic way, as follows:

Definition 3.2.1

Let f *be a function and let* $\mathbf{X}$, $\mathbf{Y}$ *be finite random variables. In order to have* f *has a measure of uncertainty then must satisfy:*

Axiom 1: $H(\mathbf{X})$ *is non negative, that is,* $H(\mathbf{X}) \geq 0$*;*

Axiom 2: $H(\mathbf{X})$ *must be continuos function;*

Axiom 3: $H(\mathbf{X})$ *is maximum if* $\mathbf{X}$ *is the uniform distribution;*

Axiom 4: $H(\mathbf{X}, \mathbf{Y}) \leq H(\mathbf{X}) + H(\mathbf{Y})$ *i.e., must be sub additive with the property that the equality holds if and only if the variables are independent.*

The first result that the interested reader can show is the following:

Theorem 3.2.1

The only function that have the four properties mentioned above is of the form:

$$H(\mathbf{X}) = -k \sum_{x \in \mathbf{X}} \Pr[\mathbf{X} = x] \log \Pr[\mathbf{X} = x]$$

where k is some positive constant.

The most common used measure of entropy is the so called Shannon entropy. Given a random variable $\mathbf{X}$ taking values from a finite set X accordingly to a fixed probability distribution. The Shannon entropy measures the number of bits of information one gains by the outcome of an experiment which takes place according to this probability distribution. For sake of understanding, assume that one is given a random variable $\mathbf{X}$ representing the toss of a coin with the following probability distribution:

$$\Pr[\mathit{Heads}] = \frac{1}{2} \qquad \Pr[\mathit{Tails}] = \frac{1}{2}.$$

Notice that every time a coin is tossed the information, or entropy, of this toss is just one bit since one can express the result of the experiment of tossing the coin using only one bit by encoding the result "Heads" by 1 and the result "Tails" by 0. Similarly, the entropy of n independent coin tosses is n, since the results of any experiment of this type can be encoded with a binary string of length n.

In general, if one is given a probability distribution with values $p_1, \cdots, p_n$ for the random variable $\mathbf{X}$, the number of bits necessary to encode each experiment is $-\log_2 p_1, \cdots, -\log_2 p_n$ which leads to the following formal definition. Due to its relevance for the remain of the text, it is now formally presented the concept of Shannon entropy as a special case of Definition 3.2.1.

Definition 3.2.2 (*Shannon Entropy*)

Let $\mathbf{X}$ be a random variable over the finite set X. The entropy of $\mathbf{X}$, denoted by $H(\mathbf{X})$ is defined by

$$H(\mathbf{X}) = -\sum_{\mathbf{x} \in \mathbf{X}} \Pr[\mathbf{X} = \mathbf{x}] \log_{\mathbf{2}}(\Pr[\mathbf{X} = \mathbf{x}])$$

with the convection that $\Pr[\mathbf{X} = x] \log_2(\Pr[\mathbf{X} = x]) = 0$ if $\Pr[\mathbf{X} = x] = 0$.

For example, if $\mathbf{X}$ is a uniform random variable over a set X with n elements, i.e., $\mathbf{X}$ is such that, for all $x \in X$, $\Pr[\mathbf{X} = x] = 1/n$, then $H(\mathbf{X}) = \mathbf{n}$, which is the maximum value that the entropy can take, since this is the most unpredictable possible distribution.

Lemma 3.2.1. *Let X is a finite set and $\mathbf{X}$ a random variable. Then $H(\mathbf{X}) \geq \mathbf{0}$.*

Proof: The thesis follows directly from the fact that $0 \leq \Pr[\mathbf{X} = x] \leq 1$ for all $x \in X$, and also by the convention assumed in Definition 3.2.2, which implies that $\Pr[\mathbf{X} = x] \log_2(\Pr[\mathbf{X} = x]) \leq 0$.
■

Exercise 3.2.2. *Let X be a finite set and $\mathbf{X}$ a random variable of X. Show that $H(\mathbf{X}) = \mathbf{0}$ if and only if the support of the probability distribution of $\mathbf{X}$ has only one element.*

Exercise 3.2.3. *Show that the function $f(x) = \log x$ in the interval $[1, \infty[$ is concave, i.e., for all $x, y \in [1, \infty[$, $f(\lambda x + (1-\lambda)y) \geq \lambda f(x) + (1-\lambda) f(y)$. Hint: Prove that the second derivative is negative in the interval*

Theorem 3.2.2 (Jensen's inequality)

Let f be a continuous strictly concave function on an interval I. Let also $a_i > 0$ be n values such that

$$\sum_{i=1}^{n} a_i = 1.$$

Then, for all $x_1, \cdots x_n \in I$:

$$\sum_{i=1}^{n} a_i f(x_i) \leq f\left(\sum_{i=1}^{n} a_i x_i\right)$$

and the equality holds if and only if $x_1 = \cdots = x_n$.

Proof: The proof is done by induction on the number n.

(Base:) For the case where $n = 2$, the inequality becomes $a_1 f(x_1) + (1 - a_1) f({}_x 2) \leq f(a_1 x_1 + (1 - a_1) x_2)$ which is true directly from the definition of concave function. The reader should observe that in a concave function all points of the interval I are evaluated with a value that is above the line defined by the points $(x_1, f(x_1))$ and $(x_2, f(x_2))$.

(Step:) Assume that the inequality is true for any $n-1$ values, i.e., for any set of $b_i > 0$ such that $\sum_{i=1}^{n-1} b_i = 1$, $\sum_{i=1}^{n-1} b_i f(x_i) \leq f\left(\sum_{i=1}^{n} b_i x_i\right)$.

Let $a_i > 0$ be n values such that $\sum_{i=1}^{n-1} a_i = 1$ and consider $a'_i = \frac{a_i}{1-a_n}$ for $i = 1, \cdots n-1$. Then:

$$\sum_{i=1}^{n} a_i f(x_i) = a_n f(x_n) + (1-a_n) \sum_{i=1}^{n-1} a'_i f(x_i).$$

Thus, applying the induction hypothesis on the second term one concludes that:

$$\sum_{i=1}^{n} a_i f(x_i) \leq a_n f(x_n) + (1-a_n) f\left(\sum_{i=1}^{n-1} a'_i x_i\right).$$

Now, from the fact that f is concave it follows that

$$\sum_{i=1}^{n} a_i f(x_i) \leq f\left(a_n x_n + (1-a_n) \sum_{i=1}^{n-1} a'_i x_i\right) = f\left(\sum_{i=1}^{n} a_i x_i\right).$$

■

Theorem 3.2.3

Let $\mathbf{X}$ *be a random variable over a finite set* $X = \{x_1, \cdots, x_n\}$ *such that the probability distribution is* $\Pr[x_1] = p_1, \cdots, \Pr[x_n] = p_n$ *such that* $p_i > 0$ *for all* $1 \leq i \leq n$. *Then* $H(\mathbf{X}) \leq \log_{\mathbf{2}} \mathbf{n}$ *and the equality holds, if and only if all* $p_i = 1/n$.

Proof: The proof follows directly from Jensen's Inequality above applied to the function $f(x) = -\log_2(x)$ and with $a_i = p_i$. In fact:

$$\begin{aligned} H(\mathbf{X}) &= -\sum_{i=1}^{n} p_i \log_2(p_i) \\ &= \sum_{i=1}^{n} p_i \log_2\left(\frac{1}{p_i}\right) \\ &\leq \log_2\left(\sum_{i=1}^{n} p_i \frac{1}{p_i}\right) \\ &= \log_2 n. \end{aligned}$$

The equality holds, also by the Jensen's Inequality, if and only if p_i are all equal and thus equal to $1/n$. ■

Definition 3.2.3 (*Joint entropy and conditional entropy*)

Let X and Y be finite sets and $\mathbf{X}$ and $\mathbf{Y}$ be the two random variables associated with X and Y respectively.

- *The joint entropy of $(\mathbf{X}, \mathbf{Y})$, denoted by $H(\mathbf{X}, \mathbf{Y})$ is the entropy of the joint probability distribution of $\mathbf{X}$ and $\mathbf{Y}$ i.e., defined by*

$$H(\mathbf{X}, \mathbf{Y}) = -\sum_{\mathbf{x} \in \mathbf{X}} \sum_{\mathbf{y} \in \mathbf{y}} \Pr[\mathbf{x}, \mathbf{y}] \log_{\mathbf{2}}(\Pr[\mathbf{x}, \mathbf{y}]).$$

- *The conditional entropy of $\mathbf{Y}$ given $\mathbf{X}$, denoted by $H(\mathbf{Y}|\mathbf{X})$, is defined as the entropy of the conditional distribution of $\mathbf{Y}$ given $\mathbf{X}$ i.e., it is defined by*

$$\begin{aligned} H(\mathbf{Y}|\mathbf{X}) &= \sum_{x \in X} \Pr[x] H(\mathbf{Y}|\mathbf{X} = x) \\ &= -\sum_{x \in X} \Pr[x] \sum_{y \in y} \Pr[y|x] \log_2(\Pr[y|x]) \\ &= -\sum_{x \in X} \sum_{y \in y} \Pr[x, y] \log_2(\Pr[y|x]). \end{aligned}$$

Theorem 3.2.4 (Chain rule)

Let $\mathbf{X}$ and $\mathbf{Y}$ be two random variables. Then:

$$H(\mathbf{X}, \mathbf{Y}) = \mathbf{H}(\mathbf{X}) + \mathbf{H}(\mathbf{Y}|\mathbf{X})$$

Proof:

$$\begin{aligned} H(\mathbf{X}, \mathbf{Y}) &= -\sum_{x \in X} \sum_{y \in y} \Pr[x, y] \log_2(\Pr[x, y]) \\ &= -\sum_{x \in X} \sum_{y \in y} \Pr[x, y] \log_2(\Pr[x] \Pr[y|x]) \\ &= -\sum_{x \in X} \sum_{y \in y} \Pr[x, y] \log_2(\Pr[x]) - \sum_{x \in X} \sum_{y \in y} \Pr[x, y] \log_2(\Pr[y|x]) \\ &= -\sum_{x \in X} \Pr[x] \log_2(\Pr[x]) - \sum_{x \in X} \sum_{y \in y} \Pr[x, y] \log_2(\Pr[y|x]) \\ &= H(\mathbf{X}) + \mathbf{H}(\mathbf{Y}|\mathbf{X}) \end{aligned}$$

■

An important concept related with entropy in the mutual information $I(\mathbf{X};\mathbf{Y})$ which can be seen as the reduction in the uncertainty of $\mathbf{X}$ due to the knowledge of $\mathbf{Y}$. Formally is defined as follows:

Definition 3.2.4

Let $\mathbf{X}$ and $\mathbf{Y}$ be two random variables. The mutual information between $\mathbf{X}$ and $\mathbf{Y}$ is defined by:

$$I(\mathbf{X};\mathbf{Y}) = \sum_{\mathbf{x}\in\mathbf{X}}\sum_{\mathbf{y}\in\mathbf{y}} \Pr[\mathbf{x},\mathbf{y}] \log_{\mathbf{2}} \left(\frac{\Pr[\mathbf{x},\mathbf{y}]}{\Pr[\mathbf{x}]\Pr[\mathbf{y}]}\right).$$

Notice that

$$\begin{aligned} I(\mathbf{X};\mathbf{Y}) &= \sum_{x\in X}\sum_{y\in y} \Pr[x,y]\log_2\left(\frac{\Pr[x,y]}{\Pr[x]\Pr[y]}\right) \\ &= \sum_{x\in X}\sum_{y\in y} \Pr[x|y]\log_2(\Pr[x]) \\ &= -\sum_{x\in X}\sum_{y\in y} \Pr[x,y]\log_2(\Pr[x]) - \left(-\sum_{x\in X}\sum_{y\in y} \Pr[x,y]\log_2(\Pr[x|y])\right) \\ &= H(\mathbf{X}) - \mathbf{H}(\mathbf{X}|\mathbf{Y}) \end{aligned}$$

Thus, mutual information can be expressed in terms of entropy. In the next exercise, the reader is invited to explore further properties of entropy and mutual information.

Exercise 3.2.4. *Let $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ be random variables. Show that:*

1. *If f is a function, then $H[f(\mathbf{X})] \leq H[\mathbf{X}]$;*
2. *$H[\mathbf{X},\mathbf{Y}] \geq H[\mathbf{X}]$ and $H[\mathbf{X},\mathbf{Y}] \geq H[Y]$;*
3. *$H[\mathbf{X}|\mathbf{Y}] \leq H[\mathbf{X}]$;*
4. *$H[\mathbf{X}|\mathbf{Y}] = 0$ if and only if $\mathbf{X} = f(\mathbf{Y})$ for some function f;*
5. *$H[\mathbf{X}|\mathbf{Y}] = H[\mathbf{X}]$ if and only if $\mathbf{X}$ is independent of $\mathbf{Y}$;*
6. *$H[f(\mathbf{X})|\mathbf{Y}] \leq H[\mathbf{X}|\mathbf{Y}]$;*
7. *$H[\mathbf{X}]|f(\mathbf{Y})] \leq H[\mathbf{X}|\mathbf{Y}]$;*
8. *$H[\mathbf{X},\mathbf{Y}|\mathbf{Z}] \geq H[\mathbf{X}|\mathbf{Z}]$;*

9. $H[\mathbf{X}|\mathbf{Y},\mathbf{Z}] \leq H[\mathbf{X}|\mathbf{Y}]$;

10. $H[\mathbf{X},\mathbf{Y}|\mathbf{Z}] = H[\mathbf{X}|\mathbf{Z}] + H[Y|\mathbf{X},\mathbf{Z}]$

11. $I(\mathbf{X};\mathbf{Y}) = \mathbf{H}(\mathbf{Y}) - \mathbf{H}(\mathbf{Y}|\mathbf{X})$;

12. $I(\mathbf{X};\mathbf{Y}) = \mathbf{H}(\mathbf{X}) + \mathbf{H}(\mathbf{Y}) - \mathbf{H}(\mathbf{X},\mathbf{Y})$;

13. $I(\mathbf{X};\mathbf{X}) = \mathbf{H}(\mathbf{X})$;

14. $I(\mathbf{Y};\mathbf{X}) = \mathbf{I}(\mathbf{X};\mathbf{Y})$.

CHAPTER 4

Notion of computational complexity

In this chapter it is settled the framework to deal with computational complexity issues. Nowadays modern cryptography is based on hard computational tasks that will be explained later in the text. Contrarily to perfect secrecy, where one gets a system that is secure against all possible attacks of an opponent in this approach the security that the parties rely on are hardness computational assumptions and for that reason is not completely secure. One of the reasons that cryptography is usually presented based on computational security and not on perfect secrecy is due to the fact that for example, perfect security can only be achieved if the length of the key is at least equal to the length of the message and the key cannot be used more than once, which it is a severe drawback regarding "practical" cryptography. All these comparisons and details will be clearer in due course. For the moment it is skipped further details regarding security and it is presented the concept of computational complexity and all the topics inherent to it that will be used in this work, like computational complexity classes and computational hard assumptions.

SECTION 4.1

Pseudo-code as base for computation

Several paradigms of computation can be used to introduce computational complexity. One of the most common is the so called Turing machine model, that was proposed by Alan Turing in 1936 to formally present the notion of "procedure" and algorithm. In this work it is necessary to present number theoretical algorithms, and Turing machines are not the most suitable model to deal with such algorithms. For a general introduction to computability and complexity it is referred to the reader the books [Sip12] and [**?**] (sernadas) and for the interested reader in Turing machines and computational complexity based on this model it recommend the reading, for example, of [AB09] or [Pap94].

The model used is almost irrelevant for the study of complexity since the models are equivalent. To define complexity one needs to specify an algorithm, which is the mechanism to solve computational task or problems. In this work instead of using any specific programming language and instead it will be used the notion of pseudocode, which is a very high level description of an algorithm. More or less it is a language very close to the human language that describes with a specific instructions actions that one wants to describe or intend the algorithm to do. For example, if one wants to do a conditional action on a number n (say divide it by two) if it is even and another action otherwise (say sum 1 and divide it by two) and keep it in a variable x one can represent it in the following way:

If n is even **then** $x = n/2$ **else** $x = (n+1)/2$.

Notice that just from the reading of the instruction given above the reader without knowing anything about programming can understand that the intention is that given a number do a certain operation if it is even or odd.

In the sequel it is presented the nomenclature that will be used in this book. If the reader thinks about an algorithm (as a recipe) there are several common components that are essential for its description. One needs:

Pseudo-code syntax

Input: The data that are given to the algorithm to perform actions;[1]

Output: The expected value that will be given by the algorithm at the end;[2]

Variable declaration: Usually are letters or simple expressions that are easy to remember their meaning and are used for intermediate calculations or instructions;

Body command: In this part the execution instructions are written as described below using terms and commands containing only variables and input and outputs..

Terms

Arithmetical terms: All basic operations are inbuilt in the code, $+$, $-$, $/$; Another important operations that will play a central role in the rest of the book are the (mod) operation that returns the remainder of a integer division and the **div** that returns the integer division.

Boolean terms: This term is used to designate operations of comparison, such as, $=$, $\neq$, $\leq$, $<$, $\geq$ and $>$. It is included also the disjunction and the conjunction operations **and** and **or** respectively.

Commands

Attributions: It is the action of given to a variable a certain value;

Random sample: Given a finite set, it chooses randomly (with uniform distribution) an element of that set;

Sequential composition: It has the form of

(command 1) **;**

(command 2)**;**

with the intend interpretation that command 1 is executed and then command 2 is executed.

[1] In the example given above, the input could be for example 4 which should be kept in n;

[2] In the example given above, the output is the value of x;

Alternative composition: It has the form

If (boolean term) **then** (command) **else** (command)

with the canonical interpretation, i.e., one test if the expression is true or false. In the former case it is performed the commands in **then** part and the latter one it is performed the commands on the **else** part;

Bounded iterative composition (For): This an element that allows to perform a set of instructions that are meant to be repeated a fixed number of times, like the name suggests. The structure involves a variable, say i, and is as follows:

For i **from** 1 **to** (number) **do** (command).

In the command part it is mandatory to increment the variable i in order to be able to rich the end of the cycle;

Iterative composition (While): This an element that allows to perform repeatedly a set of instructions while a certain condition is true. Contrarily to the bounded iterative composition, the number of time that the cycle is executed is not fixed a priori. The structure is:

While (boolean term) **do** (command).

In the command part it is mandatory to include a command that might change the condition in order to possibly break the cycle.

Some considerations are also needed. For example, given a variable x, the expression $x = x + 1$, is used to designate an actualization of the variable, i.e., to the variable x it is given the value that was already in that variable plus one. Notice that once the variable has been actualized than the previous value of x is lost.

Now it is presented an example of a pseudo code. It is described a basic operation for which one has an inbuilt function. Given two non zero natural numbers n and m as input it determines the integer division of n by m.

Example 4.1.1. *(Pseudo code for integer division on n by m)*

Input *Natural numbers n and m;*

Output *d the integer division of n by m;*

Variables *No extra variables are needed;*

Step 1 **If** $m > n$ **then** $d = 0$ **else**

Step 1.1 **While** $n < m$ **do**

Step 1.1.1 $d = d + 1$*;*

Step 1.1.2 $n = n - m$*;*

Step 2 *Output d;*

Clearly this is not the most efficient way of performing the division since one could simply use /*, but the idea is to present the use of the several structures that were described.*

Usually, as in the case of the example, if no extra variable are used or their use is clear from the context one may not declare them.

One of the basic transformations that one can operate it to modify the cycle **For** into a cycle **While** and vice versa. In the next exercise, the reader is invited to realize that transformation.

Exercise 4.1.2. *In pseudo code, transform a cycle* **For** *into a cycle while and vice versa.*

• •

SECTION 4.2

Measuring complexity

Given the pseudo code above as a tool to develop algorithms and processes one can now define the notion of complexity of an algorithm.

Essentially the complexity of an algorithm can be measured, among other, by the time required or by the quantify of space needed to perform all the tasks. In fact, this measure must be carefully treated since for example some tasks, like the arithmetic operations may require different times depending on the size of the data.

For example summing two number in binary with 10 digits each is not the same thing that summing two numbers with 1000 digits each. In fact this operation requires a linear number of steps when compared with the size of the inputs. In the next table it is expressed the cost of the basic actions expressed in units of time on the length on the input (that for example can be the machine cycles):

Operation	Cost in units of time on the length of the input
Attribution	Linear number of units
Comparisons	Linear number of units
Sum operation	Linear number of units
Difference operation	Linear number of units
Product	Quadratic number of units
Division	Quadratic number of units
(mod)	Quadratic number of units
Outputting a result	Linear number of units

Example 4.2.1. *Recall the pseudo code of Example 4.1.1 of dividing two integers and lets determine the cost this computation compared with n and m and lets count the time needed by that algorithm to perform this task.*

Input *Natural numbers n and m;*

Output *d the integer division of n by m;*

Variables *No extra variables are needed;*

Step 1 **If** $m > n$ **then** $d = 0$ **else** *(cost linear time units)*

Step 1.1 **While** $n < m$ **do** *(cost linear units each time is run)*

Step 1.1.1 $d = d + 1$; *(cost linear time units)*

Step 1.1.2 $n = n - m$; *(cost linear time units on m and n)*

Step 2 *Output d; (cost constant time units each time is run)*

Since each time the cycle is run the value of n decreases by m and thus the cycle is run at most $n|m < n$ *times and clearly* $d < n$*. Hence the entire algorithm requires at most* $c_1(n+m) + 1 + n(c_2(n+m) +$

$c_3 d + c_4 n$) where c_i are some constants. Hence asymptotically computing the division of two integers n by m costs, in terms of time units $n^2 + nm$.

The discussion above suggests the following definition of running time of an algorithm.

Definition 4.2.1

Given an algorithm A, its running time, denoted by $Time(A)$, is defined as a function of the number of units of time that the algorithm requires to compute its result on the length of the input.

As seen above in Example 4.1.1, the algorithm A that computes the division of two numbers satisfies $Time(A) \in O(n^2)$, i.e., the algorithm requires at most quadratic time on the size of the inputs.

• • •

4.2.1 The big-Oh notation

In the previous example one determined the number of operations that a certain procedure used to compute the division. Notice that the calculations are not very precise in the sense that no information about the exact number of units of time that each operation requires is provided. Notice that instead of being concerned with particular instances it is more relevant to compare the computational resources needed for instances of the same size since, for example, when running some processes on a computer one is concerned specially if the operations required by the algorithm are affordable by a bit string of length n rather than the structure of the bit string itself. Another way of saying this is that often no one is particularly interested in the precise time that the procedure will take to finish (if necessary one could probably wait a little longer, or use a faster computer) but instead it is better to know how this execution time changes as the input size increases in size. Another obstacle to the precise analyses of time is that cone is more concerned with the asymptotic behavior as it models better the running time of the procedure described since one does may not have access to the velocity of the processor which is something that depends on the machine where the algorithm is implemented, which is somehow independent of the algorithm itself. This asymptotic behavior allows comparing the running time with the size of the input data, meaning that it is defined a function, say $T(n)$,

that expresses how the time requirements depends on inputs of size n.

All the above arguments motivate the introduction of the big-Oh, which formally expresses the idea that, rather than describing exactly the function $T(n)$ of number of cycle units for the execution time, one can instead describe the so called rate of growth of the time function. Accordingly to all the arguments above, this notation provides a description of resource requirements of an algorithm that is (usually) independent of computational model and of implementation platform and describes exactly how one must increases the resources available as the input size increases. This is frequently the most relevant and informative information about an algorithm's behavior.

Lets consider the general example presented in the Example 4.2.1. The calculations determined that the total cost is approximately equal to $T(n,m) \approx c_1(n+m)+1+n(c_2(n+m)+c_3d+c_4n)$, which will be of the order of $nm+n^2$, and will be denoted by $O(nm+n^2)$ which avoids overloaded notation and a more concise representation of the idea of the time that the algorithm will take to finish.

Definition 4.2.2 (*The big-Oh notation*)

Let $f, g : \mathbb{N} \to [0,\infty)$ be two functions. One says that:

- *$f \in O(g)$ if there is a constant $c > 0$ such that $f(n) \leq c \cdot g(n)$, for almost all $n \in \mathbb{N}$.*
- *$f \in \Omega(g)$ if there is a constant $c > 0$ such that $f(n) \geq c \cdot g(n)$, for almost all $n \in \mathbb{N}$.*
- *$f \in \Theta(g)$ if and only if $f \in O(g)$ and $f \in \Omega(g)$.*
- *$f \in o(g)$ if for any constant $c > 0$ and for sufficiently large $n \in \mathbb{N}$, $f(n) \leq c \cdot g(n)$.*
- *$f \in \omega(g)$ if for any constant $c > 0$ and for sufficiently large $n \in \mathbb{N}$, $f(n) \geq c \cdot g(n)$.*

Some of the most used bounds are: $O(1)$ called constant, $O(\log n)$ called logarithmic, $n^{O(1)}$ called polynomial, $2^{n^{o(1)}}$ called sub-exponential and $2^{n^{O(1)}}$ called exponential.

In the next table it is referred in the Big-oh notation the cost of the basic operations of the Pseudo codes in order to establish the bounds of algorithms:

Operation	Time required assuming $n \geq \lvert x, \lvert y \rvert$
$x + y$	$O(n)$
$x \times y$	$O(n^2)$
x **div** y	$O(n^2)$
$x \pmod{y}$	$O(n^2)$
$x \leq y$	$O(n)$
$x = y$	$O(n)$

Expressions	Time required
$x = expr$	$O(Time(expr))$
$comm1; comm2$	$O(Time(comm1) + Time(comm2))$
If $expr$ **then** $comm1$ **else** $comm2$	$O(Time(expr) \max\{Time(comm1), Time(comm2)\})$
While $expr$ **do** $comm$	$O(Time(expr)(k+1) + Time(comm))$
For $i = 1$ **to**B **do** $comm$	$O(B \cdot Time(comm))$

Table 4.1: Costs of operation, evaulation of expressions and execution of commands.

In the unbounded iterative process k is an upper bound estimation on the number of iteration that will be performed. This knowledge as to be acquired before hand.

Exercise 4.2.2. *Show that, according to the definition of big-Oh notation, if $f(n) \in O(2^n)$ then also $f(n) \in O(n^{k+1})$ then also $f(n) \in O(n^k)$ and $f(n) \in O(2^n)$.*

Exercise 4.2.3. *Show that, according to the definition of big-Oh notation, if $f(n) \in \omega(2^n)$ then also $f(n) \in \omega(n^k)$ for all k.*

Exercise 4.2.4. *Show that, according to the definition of big-Oh notation, if $f(n) = n^2 + n$ then $f(n) \in \Theta(n^2)$.*

Exercise 4.2.5. *Show that, according to the definition of big-Oh notation, if $a, b > 0$ and $f(n) = \log_a n$ then $f(n) \in \Theta(\log_b n)$.*

Exercise 4.2.6. *Assume that $|x|, |y| \leq n$. Using the fact that $Time(z + z') \in O(Time(z) + Time(z'))$, show that:*

1. $x \times y \in O(n^2)$;
2. $x \pmod{y} \in O(n^2)$;
3. $x \ \mathbf{div} y \ \in O(n^2)$;

as claimed and expressed in the Table 4.1. As a curiosity, one can perform, using Fourier transformations, the multiplication in time of $O(n \log n)$.

• • •

4.2.2 Complexity classes

In this section it is introduced the notion of complexity classes. They are important as they catalogue the problems in families for which similar resources are needed to computer their solutions. Three of the most important ones that will be used in this text are the classes $\mathbb{P}$, $\mathbb{NP}$ and $\mathbb{BPP}$. The former one characterizes the family of problems for which one can find in polynomial time (on the length of the input) a solution. The second one characterizes the class of problems for which, given a solution, one can verify in polynomial time that it is really a solution, and the latter one, nowadays classified of feasible problems, is the class of problems that, using some randomness, is able to find a solution for a problem with a small probability of error in polynomial time.

It might seem to the reader that all this classes are similar, and in fact, they are, but they are also fonts of problems that are in the border line of human knowledge, in the sense that, many relationships between them are unknown and, for example, are part of the millennium problems. The differences between these two classes are fundamental. Without resources restrictions, checking if a given solution is in fact a solution and searching for a solutions are equivalent but, in the bounded resource models it is unknown if they are equivalent or not. This equivalence is known as the famous problem $\mathbb{P}$ versus $\mathbb{NP}$.

The difference of searching and checking is crucial for the modern cryptography that will be studied in detail later on this course. Since 1970, all basic cryptographic constructions and applications are founded in computational hardness assumptions similar to the one discussed above. For example, as it will be discussed in detail in due course, under reasonable assumption of complexity, there are functions for which one can compute its value efficiently given the

input, but there are no feasible ways of inverting it, or simply given an element in the range of the function to find one of it pre-image. A candidate for such function is the multiplication of two prime numbers. It is very easy to compute the product of two primes, but given a number that is composed of product of two primes, finding those two primes is believed to be infeasible. It is worthwhile to mention that even if $\mathbb{P} = \mathbb{NP}$, i.e. , in a scenario where checking is equivalent to searching, it is uncertain the existence of such functions. On the other hand if such functions do exist then, as it will be proved, $\mathbb{P} = \mathbb{NP}$.

To formally present all these computational complexity classes one need some technical definitions. After the proper definitions of these complexity classes some relationships are established and explained.

Definition 4.2.3 (*Input size*)

Given a program P with input $x_1 \dots x_k$, the input size $|(x_1 \dots x_k)|$ corresponds to $\sum_{i=1}^{k} |x_i|$ where $|x_i|$ is the number of bits required to represent the input x_i. In most cases the inputs are non-negative integers and so, $|x_i|$ is the number of bits in binary representation, that is

$$|x_i| = \lceil \log(x_i + 1) \rceil.$$

Definition 4.2.4 (*Polynomial-time algorithm*)

Given a program P with input $x_1 \dots x_k$, one says that P is polynomial-time on (part of the) inputs $x_{i_1}, \dots, x_{i_\ell}$, with $x_{i_j} \in \{x_1 \dots x_k\}$, if $Time(P) \in O(p(|x_{i_1}|, \dots, |x_{i_\ell}|))$ for some polynomial p.

An algorithm is said to be polynomial-time if it is polynomial time on all its inputs.

A function is said to be polynomial-time (on some inputs) if there is an polynomial-time (on some inputs) algorithm that computes it.

Definition 4.2.5 (*Polynomial-time set and the class $\mathbb{P}$*)

Let $A \subseteq \mathbb{N}^k$, A is said to be polynomial time if there is a polynomial-time algorithm to compute its indicator function $\chi_A : \mathbb{N}^k \to 2$.

The class of all polynomial-time sets is denoted by $\mathbb{P}$.

Example 4.2.7. *In this example, one proves that the set of even numbers is in $\mathbb{P}$.*

Consider $\mathcal{E} = \{x \in \mathbb{N} : x \text{ is even } \}$ and consider the following algorithm to compute $\mathcal{E}$:

Algorithm 4.2.8 ($\mathcal{A}$)

Input: x

Output: *0 if x is odd and 1 is even*

Step 1: $r = 1 - x \pmod 2$;

Step 2: *Output r;*

Notice that Time$(\mathcal{A}) \in O(n^2)$, where $|x| = n$ and taking $f_{\mathcal{A}}$ as the function computed by $\mathcal{A}$, $f_{\mathcal{A}} = \chi_{\mathcal{E}}$. Thus, accordingly to Definition 4.2.5 $\mathcal{E} \in \mathbb{P}$.

Exercise 4.2.9. *Show that:*

1. **Psquare** $= \{x \in \mathbb{N} : x \text{ is a perfect square}\} \in \mathbb{P}$;
2. **Ppower** $= \{x \in \mathbb{N} : x \text{ is a perfect power}\} \in \mathbb{P}$;
3. **Primes** $= \{x \in \mathbb{N} : x \text{ is a prime number}\} \in \mathbb{P}$

Definition 4.2.6 (*Non-deterministic polynomial-time set*)

Let $A \subseteq \mathbb{N}^k$, A is said to be non-deterministic polynomial time, if there is $f_A : \mathbb{N}^{k+1} \to 2$ which is polynomial-time in the first k inputs and fulfills the following conditions:

- *there exists $w \in \mathbb{N}$ such that $f_A(x_1, \ldots, x_k, w) = 1$ if $(x_1 \ldots, x_k) \in A$;*
- *for all $w \in \mathbb{N}$ $f_A(x_1, \ldots, x_k, w) = 0$ if $(x_1 \ldots, x_k) \notin A$.*

In these conditions, f_A is said to be a certification function for A. The class of all non-deterministic polynomial-time sets is denoted by **NP**.

Exercise 4.2.10 ($\mathbb{P} \subseteq \mathbb{NP}$)**.** *Show that if A is in $\mathbb{P}$ then A is in $\mathbb{NP}$.*

Example 4.2.11. *It will be proved that the following set is in $\mathbb{NP}$:*

$$A = \{(x, y) \in \mathbb{N}^2 : \exists k > 1, k < y \wedge k|x\}.$$

To prove that $A \in \mathbb{NP}$ one has to describe the certification function f_A. Consider the following definition of f_A:

$$f_A(x,y,w) = \begin{cases} 1 & \text{if } w|x \wedge 1 < w < y \\ 0 & \text{otherwise} \end{cases}$$

Notice that computing f implies only testing either $w|x$ which is quadratic on x (if $w > x$ one can reject it automatically) and comparisons involving w and y which, since $w \leq y$ is linear on w. Thus, f_A is in fact computable in polynomial time. Also notice that $f_A(x,y,w) = 1$ if and only if $x, y \in A$ and the witness, in the affirmative case, is, in fact, k defined in A.

From the previous exercise the reader can easily derive that the problem of factorization is also $\mathbb{NP}$. In fact, if A was computable in polynomial time then with a bijection argument querying f_A adaptively, $f_A(x, x/2)$, $f(x, x/4)$, $f(x, x/8)$ or $f(x/3x/8)$ and so on... The total number of queries is $log(x)$ which makes the algorithm for factoring polynomial. The interested reader can in fact also design directly a certification function for factoring and deduce that function can be used to compute A.

One of the most important problem in Mathematics and Theoretical Computer Science is to determine whether $\mathbb{P} = \mathbb{NP}$. This problem is classified by the Clay Institute as one of the millennium problem worthing a 1 million dollar price. All the modern cryptography that is used in practice since the 70's is based on the fact that some specific functions exist which, if $\mathbb{P} = \mathbb{NP}$, then they cannot exist. These function will be presented in next section. The question $\mathbb{P} = \mathbb{NP}$ basically wonders if having an easy, i.e. efficient, process of verifying if a solution is indeed a solution is equivalent of finding that solution also efficiently. This formalism is now explained in the next definitions and proposition.

Definition 4.2.7 (*Decision version of* $\mathbb{P} = \mathbb{NP}$)

For each set A in $\mathbb{NP}$ there is a polynomial-time algorithm that computes χ_A.

Definition 4.2.8 (*Search version of* $\mathbb{P} = \mathbb{NP}$)

For each set $A \subseteq \mathbb{N}^k$ in $\mathbb{NP}$ with certification polynomial-time function f_A, there exists a polynomial-time function $g_A : \mathbb{N}^k \to \mathbb{N}$ such that:

- $f_A(x_1, \ldots x_k, g_A(x_1, \ldots x_k)) = 1$ *if* $(x_1, \ldots x_k) \in A$;
- $f_A(x_1, \ldots x_k, g_A(x_1, \ldots x_k)) = 0$ *if* $(x_1, \ldots x_k) \notin A$.

Theorem 4.2.1 (*Equivalence between decision and search version of* $\mathbb{P} = \mathbb{NP}$)

The two definitions of $\mathbb{P} = \mathbb{NP}$ given in definitions 4.2.7 and 4.2.8 above are equivalent.

This theorem in fundamental from the cryptographic point of view as it states in two different ways the same problems. In some cases, the use of a particular definitions gives a more natural intuition on the result that one wants to prove.

Proof:

(Definition 4.2.8 $\Rightarrow$ Definition 4.2.7 i.e., Search implies Decision)

Assume that A is a set in $\mathbb{NP}$ for which there exists a certification function f_A and a polynomial time function g such that $f(x_1, \ldots x_k, g(x_1, \ldots x_k)) = 1$ if and only if $(x_1, \ldots x_k) \in A$.

In order to prove that A fulfills the Definition 4.2.7, one needs to show that there exists a polynomial time algorithm that computes its characteristic sequence. Take the following algorithm:

Algorithm 4.2.12 (*B to compute* χ_A)

Input: $(x_1, \ldots, x_k) \in \mathbb{N}^k$, f_A *and* g_A;

Output: $\chi_A(x_1, \ldots, x_k)$;

Step 1: $x = g_A(x_1, \ldots, x_k)$;

Step 2: $y = f_A(x_1, \ldots, x_k, x)$;

Step 2: *Output* y

Since $f_A(x_1, \ldots x_k, g_A(x_1, \ldots, x_k)) = 1$ if and only if $(x_1, \ldots x_k) \in A$ then B, on an input $(x_1, \ldots, x_k)$ computes exactly χ_A.

Notice also that B is also polynomial time on the input since g_A and f_A are polynomial-time computable and the output operation is also polynomial, thus $Time(B) \in O(p)$ for some polynomial p.

(Definition 4.2.7 $\Rightarrow$ Definition 4.2.8 i.e., Decision implies Search)

Assume without loss of generality that A is a subset of $\mathbb{N}$ that is in $\mathbb{NP}$ accordingly to Definition 4.2.6. Assume also that there is a polynomial-time algorithm that computes χ_A, i.e., A satisfies the conditions of Definition 4.2.7.

Consider a set $\tilde{A}$ such that $(x, y, z) \in \tilde{A}$ if and only if there is a w written in binary with z digits such that $f_A(x, e^{-1}(e(y).w)) = 1$ where f_A is the certification function of A, e is the polynomial time bijection with polynomial time inverse [3] between

$$\{a : \exists n \in \mathbb{N} \text{ such that } a = a_1 \ldots a_n \wedge a_i \in \{0, 1\}\}$$

and $\mathbb{N}$ and $\cdot$ is used to describe the binary concatenation of two strings.

Claim 4.2.13. $\tilde{A} \in \mathbb{NP}$.

To prove the claim just apply the Definition 4.2.6 of $\mathbb{NP}$ with the function $f_{\tilde{A}}$ defined by $f_{\tilde{A}}(x, y, (w, z)) = f_A(x, e^{-1}(e(y).w))$ as constructed above. Notice that f has 2 extra arguments but, since e and its inverse, and also f_A are polynomial time computable on x, y, the function $f_{\tilde{A}}$ is a certification function.[4]

From the fact that $\tilde{A} \in \mathbb{NP}$ one can conclude, accordingly to Definition 4.2.7 that $\chi_{\tilde{A}}$ is computable in polynomial time.

To prove that A is in the conditions of Definition 4.2.8 one needs to define a certification function f for which there exists a polynomial-time function g such that:

[3] The reader is invited to give the details of this bijection and prove that, in fact, can be computed in polynomial time and its inverse is also polynomial.

[4] In Definition 4.2.6 the function must have only one argument. It is left to the interested reader to verify that if one defines $\mathbb{NP}$ with $k + l$ arguments as in Definition 4.2.6, where l is some fixed value, then the two definitions of $\mathbb{NP}$ coincide.

$$f(x, g(x)) = 1 \text{ if and only if } x \in A.$$

One defines g as the function computed by the following algorithm:

Algorithm 4.2.14 (*to compute* $g(x)$)

Input: $x \in \mathbb{N}$, χ_A, f_A *and* χ_{tildeA};

Output: $g(x)$;

Step 1: **If** $\chi_A(x) = 1$ **then**

Step 1.1: $w = \varepsilon$;

Step 1.2: **While** $f_A(x, w) = 0$ **do**

Step 1.2.1: **If** $\chi_{\tilde{A}}(x, w.1) = 1$ **then**

Step 1.2.1.1: $w = w.1$;

Step 1.2.1.2: **else** $w = w.0$;

Step 2: $w = \varepsilon$;

Step 3: *Output* w

Notice that this algorithm terminates in polynomial time as the operations involved are all polynomial and from the fact that, since $A \in \mathbb{NP}$, there exist a witness w of polynomial size. ■

. . . . . .

SECTION 4.3

One-way functions

In this section some formal definitions of different one-way functions are presented and then it is briefly discussed some candidates that are widely believed to be one-way functions. Heuristically speaking a function is said to be one way if it is easy to compute on every input, but

relatively hard to invert given an element on the range of a random input. Accordingly to previous section the terms *"easy"* and *"hard"* are understood under computational complexity theory, namely the theory of polynomial time problems.

For the reader that is not familiar with the concept of one way functions it might be strange the fact that one can talk about one way functions that are not one-to-one since one requires that it has inverse. In fact, this constrain is not mandatory and almost all candidates are not one-to-one. It also clear that one-to-one requirement is not sufficient to be one way, since the identity is clearly an example of it.

Definition 4.3.1

A function $f : \mathbb{N} \to \mathbb{N}$ is honest *if $|f(x)|$ and $|x|$ are polynomially related, i.e., for some $k > 0$ and for every $x \in \Sigma^\star$,*

$$(|f(x)| \leq |x|^k + k) \wedge (|x| \leq |f(x)|^k + k).$$

Now it is presented the notion of one-way function:

Definition 4.3.2 (*One-way function*)

A function $f : \mathbb{N} \to \mathbb{N}$ is said to be a one-way function if:

- *f is honest;*
- *f is computable in polynomial time;*
- *f^{-1} is not computable in polynomial time;*

The first result one might prove is that, in fact, if one-way functions exist, then, $\mathbb{P} \neq \mathbb{NP}$. The reciprocate of this statement is not true. Even if $\mathbb{P} \neq \mathbb{NP}$ it might be case that one-way function do not exist.

Theorem 4.3.1

If one-way functions as presented in Definition 4.3.2, then $\mathbb{P} \neq \mathbb{NP}$.

Proof : The proof is carried over by contraposition.

Assume that $\mathbb{P} = \mathbb{NP}$ and let f be a honest bijection that is computable in polynomial time. The goal is

to show that f^{-1} is also computable in polynomial time. Consider the following set:

$$A = \{(y, x) \in \mathbb{N}^2 : \exists z \text{ such that } f(e^{-1}(e(x) \cdot z)) = y\}$$

where e is again the polynomial time bijection between $\mathbb{N}$ and $\{0,1\}^*$ and $e(x) \cdot z$ is again an abbreviation for the concatenation os binary strings.

First one needs to prove that $A \in \mathbb{NP}$. To do that, one needs to compute a certification function $h : \mathbb{N}^3 \to \{0,1\}$ for A, i.e., that is computable in polynomial time in the first two arguments and $h(y, x, w) = 1$ if and only if $(y, x) \in A$.

Consider

$$h(y, x, w) = \begin{cases} 1 & \text{if } f(e^{-1}(e(x) \cdot z)) = y \\ 0 & \text{otherwise} \end{cases}$$

Then h is polynomial in (y, x) since f is polynomial time computable and f is honest. In fact, the absolute value of $|e^{-1}(e(x)) \cdot z| - |y|$, since f is honest, is polynomial bounded by $|y|$. It is then easy to verify that h is a certification function for A.

Since $A \in \mathbb{NP}$ there exists a polynomial time computable function g with $g(y, x) = w$ satisfying $h(y, x, g(y, x)) = 1$ if and only if $(y, x) \in A$. By the Search version of the problem $\mathbb{P} = \mathbb{NP}$ of Definition 4.2.8 g is computable in polynomial time. If one considers $f^{-1}(y) = g(y, e^{-1}(\varepsilon))$ then f^{-1} is, in fact, computable in polynomial time and is the inverse of f. Notice that if $g(y, e^{-1}(\varepsilon)) = w$ then $f(w) = y$. In fact, $g(y, e^{-1}(\varepsilon)) = w$ means that $h(y, e^{-1}(\varepsilon), w) = 1$ which is equivalent to say that:

$$y = f(e^{-1}(e^{-1}(e^{-1}(\varepsilon) \cdot e(w)))) = f(e^{-1}(\varepsilon \cdot e(w)))) = f(e^{-1}(e(w))) = f(w)$$

which means that g is in fact the inverse of f. ■

As it will be seen later, the existence of one-way functions is not enough to guarantee that one can design a public key cryptographic systems as they, bey themselves, do not ensure the existence of an easy way of decrypting a message that was ciphered with a one-way function. In order to achieve that goal one will need the notion of a trapdoor one way function. Intuitively, these functions are one-way function but with a "backdoor" that allows the decryption efficiently in the presence of a secret.

Definition 4.3.3 (*trapdoor one-way function is a one-way function*)

A function $f : \mathbb{N} \to \mathbb{N}$ is said to be a trapdoor one-way function if it is a one-way function with the additional property that given some extra information, usually called the trapdoor information, it becomes feasible to find for any given y in the range of f one its pre images, i.e., exists a function $g : \mathbb{N}^2 \to \mathbb{N}$ computable in polynomial time for which there exists a sequence of secrets $(s_n) - n \in \mathbb{N}$ such that $g(y, s_{|y|}) = x$ and $y = f(x)$.

Notice that if s_n was computational easy to obtain then f would not be a one-way function. This result is expressed in the next proposition which formal proof is left for the interested reader.

Proposition 4.3.1. *Let f be a trapdoor one-way function and $(s_n)_{n\in\mathbb{N}}$ as in the previous definition. Then $s_{|y|}$ cannot be computed in polynomial even with the knowledge of f.*

There are other several definitions that one can adopt as one-way functions, based on probabilistic algorithms that are essentially similar to the one presented above. Here they are presented with the intuit to be a more complete reference for the reader. The different between the definitions presented bellow concerns the security requirements made.

In all definitions of the rest of the section it is assumed that f is honest. As it will be seen later on, the requirement of honesty of f cannot be avoided in order to define properly one-way functions that are not trivially one-way due to the fact that the function compress to much.

The weakest version of one-way functions considered in the literature is based on opponents that only have deterministic polynomial time resources to try to invert the function.

Definition 4.3.4 (*Deterministic one-way function*)

An honest function $f : \Sigma^* \to \Sigma^*$ is a deterministic one-way function *if the following two conditions hold:*

Easy to compute: *there is a (deterministic) polynomial time algorithm A such that on every input x, the algorithm A outputs $f(x)$ (i.e., $A(x) = f(x)$).*

Slightly hard to invert: *for any deterministic polynomial time algorithm B, for some polynomial $q(\cdot)$, for every sufficiently large n,*

$$\Pr_{x \in \Sigma^n}[f(B(f(x), n)) \neq f(x)] > \frac{1}{q(n)}.$$

If one allows some access to randomness to the opponents it is possible to define two other variants of one-way functions.

In the case of a strong one-way function, it is required that the inversion happens with low probability and in the weak version the non inversion must happen with non-negligible probability.

Definition 4.3.5 (*Weak one-way function*)

An honest function $f : \Sigma^* \to \Sigma^*$ is a weak one-way function *if the following two conditions hold:*

Easy to compute: *there is a (deterministic) polynomial time algorithm A such that on every input x, the algorithm A outputs $f(x)$ (i.e., $A(x) = f(x)$).*

Slightly hard to invert: *for any polynomial $t(\cdot)$, there is a polynomial $q(\cdot)$ such that for every probabilistic t-time-bounded algorithm B and for every sufficiently large n,*

$$\Pr_{(x,r) \in \Sigma^n \times \Sigma^{t(n)}}[f(B(f(x), r, n)) \neq f(x)] > \frac{1}{q(n)}.$$

Definition 4.3.6 (*Strong one-way function*)

An honest function $f : \Sigma^* \to \Sigma^*$ is a strong one-way function *if the following two conditions hold:*

Easy to compute: *there is a (deterministic) polynomial time algorithm A such that on every input x, the algorithm A outputs $f(x)$ (i.e., $A(x) = f(x)$).*

Hard to invert: *for any polynomial $t(\cdot)$, for every probabilistic t-time-bounded algorithm B, for every positive polynomial $q(\cdot)$, and for every sufficiently large n,*

$$\Pr_{(x,r)\in\Sigma^n\times\Sigma^{t(n)}}[f(B(f(x),r,n)) = f(x)] < \frac{1}{q(n)}.$$

In the previous definitions, r denotes the randomness used by the algorithm B and its length is bounded by its running time.

It is easy to see that any weak one-way function is a deterministic one-way function and that any strong one-way function is a weak one-way function and is left as an exercise for the interested reader.

Exercise 4.3.2. *Let f be an honest function. Show the following statements:*

1. *If f is a strong one-way function then f is also a weak one-way function.*
2. *If f is a weak one-way function then f is also a deterministic one-way function.*
3. *Conclude that if f is a strong one-way function then f is also a deterministic one-way function.*

Exercise 4.3.3. *Show that if f is an one-way function then f cannot have a polynomial sized range, i.e., for every polynomial p there exists an infinite many n for which $\#\{f(x) : x \in \{0,1\}^n\} > p(n)$.*

Exercise 4.3.4. *Show that if f is a one-way function then $g = f \circ f$ is also a one way function.*

Exercise 4.3.5. *Show that if f and g are two weak one-way functions then their composition may not be a one way function.*

The reader may be puzzled by the requirement of honesty of f in the definitions of one-way functions. To make clear the importance of this requirement it is presented the next example.

Example 4.3.6. *Consider the following function f:*

$$\begin{aligned} f: \quad \{0,1\}^* &\to \{0,1\}^* \\ x = x_{[1:n]} &\to y = x_{[1:\log n]} \end{aligned}$$

I.e., the function that given an input x only outputs the first logarithmic of its bits. Clearly, this function is not honest, as it exponentially compressed the input. To understand that f could in fact be a one-way function just observe that given y of length $\log n$, printing for example $x = y0...0$ of length n (that is a pre image of y by f) any algorithm requires at least $O(n)$ computational steps, but $O(n)$ is not polynomial on the size of y which is $\log(n)$.

A fundamental question regarding one-way functions is their relationships. For instance, in the previous exercise, one showed that strong one-way functions are in fact stronger than the weak one-way functions, in the sense that the existence of the former implies the existence of the latter, but nothing is said about the opposite. In the next example it is shown that several weak one-way functions might not be a strings one, but later on, one proves that the existence of weak one-way functions imply the existence of weak one-way functions by a constructive argument.

Example 4.3.7. *Assume that $f : \{0,1\}^* \to \{0,1\}^*$ is a strong one-way function and consider the following function g:*

$$\begin{aligned} g: \quad \{0,1\}^* &\to \{0,1\}^* \\ x &\to \begin{cases} 0f(x) & \text{if } x \text{ starts with } 0 \\ x & \text{if } x \text{ starts with } 1 \end{cases} \end{aligned}$$

Clearly g is not a strong one-way function since in the case that one has as an image y which the first bit is 1 then it is easy to compute a pre image of it, since in that case, g is just the identity. On the other hand, g is weak one-way function since, intuitively, every time one tries to invert y which the first bit is 0, determining a pre image of y requires the inversion of f which it is assume to be a very difficult task.

Formally, to prove that g is a weak one-way function, one must prove the existence of a polynomial such that for all probabilistic polynomial time algorithm B that tries to invert the function, its probability of

error is at least the inverse of that polynomial.

$$\begin{aligned}\Pr_{(x,r)\in\Sigma^n\times\Sigma^{t(n)}}[f(B(f(x),r,n))\neq f(x)] &\geq \Pr_{(0x,r)\in\Sigma^n\times\Sigma^{t(n)}}[f(B(f(x),r,n))\neq f(x)] \\ &> 1-\frac{1}{2q(n)}.\end{aligned}$$

The result follows by observing that q is any possible polynomial and thus, $1-\frac{1}{2q(n)} \geq \frac{1}{2}$ for all sufficiently large n.

Theorem 4.3.2

The existence of weak one-way functions is equivalent to the existence of strong one way functions.

This theorem shows that despite the fact that not all weak one-way functions are string one way functions, if they exist then strong one way functions also exist. The core of the proof of this theorem is made by constructing from a weak-one way function a strong one-way function.

Proof: The proof written here is based on the arguments presented in the text book [Gol04] of Goldreich for the same theorem.

(From strong to weak one-way functions)

This part of the theorem is trivially true since in the former one it is required that the probability of inversion is smaller than any polynomial and in the latter one it is only required that the inversion happens with non negligible probability, i.e., with probability that is at least larger than some polynomial.

(From weak to stung one-way functions)

This part is the non trivial one. As mentioned the proof is constructive in the sense that from a weak one-way function, a strong one will be derived.

Let f be a weak one-way function accordingly to definition 4.3.5 and let p be such that no probabilistic algorithm fail the inversion of f with probability at least $1/p(n)$ and consider the function g:

$$g(x_1,....,x_{s(n)}) = (f(x_1),....,f(x_{s(n)}))$$

where $|x_1| = \cdots = |x_{s(n)}| = n$ and $s(n) = n \cdot p(n)$. It will be proved that g is a strong one-way function. Clearly, g is easy to compute since all that one needs to do is to run a polynomial number of times the polynomial time algorithm that computes f. In order to derive the fact that g is a strong one way function it is necessary to prove that for every for any polynomial time $t(\cdot)$, for every probabilistic t-time-bounded algorithm B, for every positive polynomial $d(\cdot)$, and for every sufficiently large n,

$$\Pr_{(x,r)\in\Sigma^n\times\Sigma^{t(n)}}[g(B(g(x),r,s(n))) = g(x)] < \frac{1}{d(n)}.$$

If one assumes that B tries to invert g by inverting each of the components of g then the probability of success of B can be computed as follows:

$$\Pr_{(x,r)\in\Sigma^n\times\Sigma^{t(n)}}[g(B(g(x),r,s(n))) = g(x)] =$$

$$\Pr_{(x,r)\in\Sigma^n\times\Sigma^{t(n)}}[f(B(f(x_1),r,n)) = f(x_1),\cdots,f(B(f(x_{s(n)}),r,n)) = f(x_n)]$$

$$= \prod_{i=1}^{s(n)} \Pr_{(x,r)\in\Sigma^n\times\Sigma^{t(n)}}[f(B(f(x_i),r,n)) = f(x_i)]$$

$$= \prod_{i=1}^{s(n)} (1 - \Pr_{(x,r)\in\Sigma^n\times\Sigma^{t(n)}}[f(B(f(x_i),r,n)) = f(x_i)])$$

$$\leq \left(1 - \frac{1}{q(n)}\right)^{n\times p(n)} \leq 2^{-p(n)}$$

Notice that the last expression is exponential on the size of the input ($n^2 \times p(n)$) and thus larger than the inverse of any polynomial, in particular $d(n)$.

Notice that the assumption that B inverts each of g's components is not reasonable since the opponent might attack all the inputs all together.

The general argument is done by contradiction, i.e., it is assumed that g is not strong one-way function and then a contradiction is derived, namely that f is not a weak one-way function.

If g would not be a strong one-way then it would exist a probabilistic polynomial-time algorithm B and a polynomial $q(\cdot)$ such that for infinitely input sizes m,

$$\Pr_{(x,r)\in\Sigma^m\times\Sigma^{t(m)}}[g(B(g(x),r,s(n))) = g(x)] > \frac{1}{q(n)}.$$

Denote by M the infinite set of integers for which the last inequality holds and let N denote the infinite set of integer of the form $n^2 \cdot p(n) \in M$

The next procedure it is described a process to invert f.

Algorithm 4.3.8 (*Called I for inverting f.*)

Input: *The values, n, and y of length n and the value $t(n)$;*

Output: *A string x' such that $f(x) = f(x)$;*

Step 1: $i = 1$;

Step 2: **While** $i < t(n)$ **do**

Step 2.1: *Select uniformly and independently $x_1, \cdots, x_{t(n)} \in \{0,1\}^n$:*

Step 2.2: *Compute*

$$(z_1, \cdots, z_{t(n)}) = B(f(x_1), \cdots, f(x_{i-1}), y, f(x_{i+1}), \cdots, f(x_{t(n)}));$$

Step 2.2: **If** $f(z_i)y$ **then** $x' = z$ *and* $i = t(n)$ **else** $i = i + 1$;

Step 3: *Output x'.*

Consider the set $S_n = \{x : \Pr[f(I(f(x)) = f(x)] \geq n/a(n)\}$, where n is some polynomial.

Claim 4.3.9. $\Pr_{(x,r)\in S_n \times \Sigma^{t(m)}}[f(A(f(x))) = f(x)] \geq 1 - 2^n$.

Notice that since I inverts $f(x)$ with probability $n/a(n)$ A runs it $a(n)$ times and then the bound follows trivially, i.e.,

$$\Pr[f(A(f(x))) \neq f(x)] < (1 - n/a(n))^{a(n)} \leq 2^{-n}.$$

Claim 4.3.10. *For every $n \in N$, $\#S_n > (1 - 1/(2p(n))) \cdot 2^n$.*

The proof of this fact is left for the interested reader.

Combining the two claims, one gets:

$$\begin{aligned}\Pr_{(x,r)\in\Sigma^n\times\Sigma^{t(n)}}[f(A(f(x)))=x] &\geq \Pr[x\in S_n]\cdot\Pr_{(x,r)\in S_n\times\Sigma^{t(n)}}[f(A(f(x)))=x|x\in S_n]\\ &\geq \left(1-\frac{1}{2p(n)}\right)\cdot(1-2^{-n})\\ &> 1-\frac{1}{p(n)}\end{aligned}$$

Thus, there is an probabilistic polynomial time algorithm A inverting f with probability larger than $1-1/p(n)$ which contradict the fact that f is a weak one-way function. ■

One can have a similar result to Theorem 4.3.1 for these versions of one-way functions. The result is proved bellow.

Theorem 4.3.3

If $\mathbb{P}=\mathbb{NP}$ then no one-way functions exist.

The proof is informally explained as follows. Notice that if $\mathbb{P}=\mathbb{NP}$ then for every $\mathbb{NP}$ procedure there exists a polynomial time algorithm that is equivalent to the $\mathbb{NP}$ procedure, i.e., one can simulate the $\mathbb{NP}$ hardness in polynomial time. Notice that since if f is a one-way function, then is, in particular an honest function which means that the input size and the output size are polynomially related. Thus, inverting an one way function is clearly an $\mathbb{NP}$ problem. In fact, notice that checking that a given x is in fact a pre image of y it is only necessary to compute $f(x)$ and check it is equal to y, which can be done in polynomial time on the size of x and thus on polynomial time on the size of y. Hence, there exists a polynomial time algorithm that for all given the input y determines its pre image and thus f cannot be a one-way function as defined above.

Proof: Consider the language $\mathcal{L}=\bigcup_{i=1}^{\infty}L_i$, where $L_i=\{(1^{|x|},f(x),x),x\in\{0,1\}^i\}$. It is easy to check that $\mathcal{L}$ is in $\mathbb{NP}$ as the the corresponding x can be taken as a certificate for any element, which is smaller than then length of the input. Notice that verifying the certificate amounts to checking its length and computing $f(x)$ which are clearly polynomial time task as f is assumed to be polynomial time computable.

Since $\mathbb{P}=\mathbb{NP}$, then there exists an algorithm A running in polynomial time on the length of

$(1^{|x|}, f(x), x)$ to check the membership in $\mathcal{L}$. Let $q(n)$ be the polynomial time bound for that algorithm A.

Now A is used to create an algorithm B that inverts f. The construction done here is creating the x bit by bit. On the input $(1^n, f(x))$, B produces $(1^n, f(x), 0)$ and runs it in A. Define x_1 to be 0 if A returns true, and 1 if it returns false and repeat the process to define the value of x_{i+1} with $i \leq n$, by 0 if A evaluates $(1^n, f(x), x_1 \dots x_i 0)$ to true and 1 otherwise. After defining x_n, output the string $x = x_1 x_2 \dots x_n$.

The procedure is describe in the following algorithm.

Algorithm 4.3.11

Input: *The values, n, 1^n and $f(x)$ and the algorithm A*

Output: *A string x' such that $f(x) = f(x)$;*

Step 1: $i = 0$*;*

Step 2: **While** $i < n + 1$ **do**

Step 2.1: **If** $A(1^n, f(x), 0) = TRUE$ **then** $x = x0$ **else** $x = x1$*;*

Step 2.2: *i=i+1;*

Step 3: *Output x.*

Clearly, the output is actually an x' such that $f(x') = f(x)$, since at each step it only accepts a digit that can be perpetuated to some x that evaluates correctly.

It remains to show that B is a polynomial time algorithm. Assume that m is the length of $(1^n, f(x))$ and just compute the cost of the above operations. The creation each of the $x_1 ... x_i$ can be performed in less than linear amount of time on twice of the length of the input string and each call of A costs $q(cm)$ for some c. This process is repeated $n < m$ times, and then a string of length $n < m$ is outputted. This last process requires also $\leq c'm$ steps for some c'. Thus the total cost of B is bounded by $m(c'm + q(cm)) + cm$ which is polynomial on m.

Hence B is an algorithm that inverts f in polynomial time on all inputs $(1^n, f(x))$. So

$$\Pr_{x\in\{0,1\}^n : y=f(x)} \left[A(1^n, y) = x' \text{ and } f(x') = y\right] = 1.$$

Thus f cannot be a one-way function. ■

• • •

4.3.1 The candidates to one-way functions

Now several examples of functions candidate to one-way functions are discussed. Notice that it cannot be claimed that each of them is in fact a one-way function by the arguments presented above, but extensive research has so far failed to produce an efficient inverting algorithm for any of them and thus, the assumption of hardness of inversion is merely widely believe to be true or follow from conjectures of hardness also widely believed to hold.

- **Multiplication**

  Consider the function:

  $$\begin{aligned} f : \mathbb{P}\mathrm{rimes}^2 &\rightarrow \mathbb{N} \\ (p, q) &\rightarrow n = p \cdot q \end{aligned}$$

  where $\mathbb{P}\mathrm{rimes}$ is the set of all prime numbers. From what it was seen previously, this function can be computed in time $O(m^2)$ where m is the total length in the number of binary digits of the inputs (p, q). The inverting this function requires that given an integer n to find the factors of it. At this date, the best factoring algorithms known requires $O(2^{\sqrt[3]{\log n}\sqrt[3]{(\log\log n)^2}})$, which is only pseudo-polynomial in the length of the binary representation of n.

- **Modular squaring**

  Let $n = p \cdot q$ where $p, q \in \mathbb{P}\mathrm{rimes}$. Consider the function:

  $$\begin{aligned} f : \mathbb{Z}_n &\rightarrow \mathbb{Z}_n \\ x &\rightarrow x^2 \pmod{n} \end{aligned}$$

Inverting this function requires computing square roots modulo n, i.e, given y and n, find some x such that $x^2 \pmod{n} = y$. This problem is equivalent to factoring described in the previous item as the inverse of the multiplication.

- **Modular exponential**

  Let p be a prime number. Consider the function:

$$\begin{aligned} f : \mathbb{Z}_p &\rightarrow \mathbb{Z}_p \\ x &\rightarrow 2^x \pmod{p} \end{aligned}$$

  The computation of this function requires $O(n^3)$ many step where $n = \lceil \log p + 1 \rceil$. Inverting this function requires computing the function called the discrete logarithm modulo p, i.e, given a prime p and an integer y between 0 and $p-1$, find x such that $2^x = y$.

Example 4.3.12. *Show that the function*

$$\begin{aligned} f : \mathbb{N}^2 &\rightarrow \mathbb{Z} \\ (p, q) &\rightarrow n = p \cdot q \end{aligned}$$

is not one-way function.

Pseudo-examples for trapdoor one way functions include all the above functions described above with some extra information. For example, the knowledge of one of the factor in the case of multiplication allows the reader to compute the inverse very easily and efficiently. These functions are important in the design of cryptographic systems with the use of public keys that will be discussed in due course.

Part II

Classical Cryptography

CHAPTER 4

CHAPTER 5

Classical cryptographic systems

Since ancestor times the Man is interested in "hiding" information and from the invention of writing it became indispensable to share secret messages. Soon it had been realized that Cryptography is all about mathematics and how one can design and break codes.

Nowadays cryptography is not only used for military purposes and in fact it is widely used in most of daily routines: purchase over the Internet, on the use of Automatic Teller Machines (ATM), satellite TV, where cryptography is crucial to keep financial and other confidential information secure.

Cryptography is the science that studies the encoding and the decoding "secret messages" processes. Etymologically, the word "cryptography" is derived from two Greek words: *kryptos* which means hidden or secret, and *graphia*, which means writing. The goal of this science is to enable two parties to communicate secretly some piece of information. By secretly it is meant, for example, without a third party, to whom the information was not meant to be told, is able to see what the other two parties were communicating.

In this chapter it is introduced some of basic concepts of symmetric cryptography and cryptanalysis. It is also presented some classical examples of cryptographic systems and the respective ways of "breaking it". The term symmetric is used to characterize the encryption and decryption that are similar and the fact that the same "key" is used in both processes.

SECTION 5.1

Cryptographic systems

As it was already mentioned, the holly grail of cryptography is to enable two parties, say A and B, to communicate over an insecure channel in such a way that someone else, say O (an observer or an opponent), cannot get or understand the message that A and B wanted to share. This channel could be a telephone line or computer network, for example. The original information that A wants to send to B, which is a readable message for B is called *plaintext* or *clear text*. It can be, for example, a text written in English or using numerical data that is understandable by both of the intervinients. If A would simply send the *plaintext* without any kind of precaution, the message that A and B meant to kept in secret from anyone else would be compromised. So, in order to prevent this problem, A must "encode" the message into a so called *ciphertext* or *encrypted message* in such a way that B can "decode" the message to recover the plaintext of the original message. Also O is unable, upon seeing the ciphertext in the communication by eavesdropping, cannot determine what the plaintext was. The process of encoding the original message is also called *encryption* and the process of decoding the ciphertext is also called *decrypting*, or *deciphering*.

The common way of encrypt and decrypt the messages requires an additional piece of information, usually called the *key* or the *password* shared only by the two parties communicating. Sometimes there are different keys used to encrypt and decrypt messages. The idea of cryptography is to present methods such that the process of encrypting the messages and decrypting them with the right key in easy to do[1] and also that decrypting a message without knowing the corresponding key, i.e., trying to *crack* or *break* the message is unfeasible.

These ideas are described formally using the following mathematical notation.

[1] Here "easy" is used to mean that are feasible in terms of resources needed to implement it in a computer

Definition 5.1.1 (*Cryptographic system*)

A cryptographic system is a tuple of the form (P, C, K, E, D), with the following properties:

1. *P is a finite set of possible plaintexts;*
2. *C is a finite set of possible ciphertexts;*
3. *K is a finite set of possible keys;*
4. *For each $k \in K$, there is an encryption rule $e_k : P \to C$ in E (encryption function) and a corresponding decryption rule $d_k : C \to P$ (decryption function) in D that such that $d_k \circ e_k = id_P$ i.e., for every plaintext element $x \in P$, $d_k(e_k(x)) = x$.*

Notice that it is assumed the existence of a reliable channel of communication between A and B and in the last definition no requirements of secrecy are imposed, in the sense that, no assumptions are made about who besides B might decrypt the "secret" information, and, in particular what kind of information of the message (for examples, some meta information, like size or the language in which was originally written) is accessible to whom.[2] Hence, a simple example of a cryptographic system consists of (P, C, K, E, D) where $P = C = K$ and $E = D = \{id : P \to P\}$.

For sake of understanding, the ultimate goal of Cryptography is to allow the confidentially of the messages exchanged between the two parties i.e., making it unintelligible for a third party that eavesdrop the communication between A and B.[3] Thus if A and B want to communicate secretly they must agree on several things. Namely which specific cryptographic system to use and the the which key $k \in K$ that they will use to encrypt/decrypt the messages. This must be done beforehand and without O to see, either using a secure channel or personally.

Notice that the property 4 of the cryptographic systems, also known as consistency require-

[2]Historically it is known that this kind of information can be important to crack the encryption. For example, in the II world war, it was important for the alliance to know that the messages exchanged between the Germans where written in Germany and had information about the weather at the beginning of the message...

[3]There are some different goals that one may consider regarding the confidentiality of the message. Either no opponent can determine the message, or no adversary can determine any information about the message or no intruder can determine any meaningful information about the message.

ment, is the one that allows B to decrypt correctly the message. Hence the function e_k must be injective to avoid overlaps in the decryption and thus ambiguity.

There are other issues that are also considered to be "goals" of cryptography that deserve to be mentioned here:

Confidentiality or secrecy: the cryptographer must unsure that the content of information must be kept secret from all but those to which the message is meant to be keeping it data unintelligible for the others.

Data integrity: the contents of the message is compromised if an unauthorized alteration of data is made, i.e., one must ensure that the system allows the detection of data manipulation by unauthorized parties.

Authentication: in the communication process both entities should identify each other.

Non-repudiation: each message should have a fingerprint, in the sense that the system should not allow a party to deny commitments or actions that he made to the message contents.

A fundamental goal of cryptography is to adequately address these four areas in both theory and practice. Cryptography is about the prevention and detection of cheating and other malicious activities.

After that, if A needs to communicate the $x = x_1 \cdots x_n$ with Bob over an insecure channel, it encrypts each x_i using the encryption rule e_k specified by the predetermined key k and produces a encrypted message y where $y_i = e_k(x_i)$ for all $1 \leq i \leq n$, which is sent over the channel to B. On its hand, B decrypts y using the decryption function d_k, obtaining the original plaintext string.

In the next exercise the reader is asked to verify simple properties of the cryptographic systems

Exercise 5.1.1. *Let (P, C, K, e, d) and (P', C', K', e', d') be two cryptographic systems. Show that if $C \subset P'$ then (P, C', K'', e'', d'') where $K' = K \times K'$, $e''_{(k,k')} = e_{k'} \circ e_k$ and $d''_{(k,k')} = d_k \circ d_{k'}$ is a cryptographic system (called the composition or the product cryptographic system).*

Definition 5.1.2

Let (P, C, K, e, d) and (P', C', K', e', d') be two cryptographic systems. (P, C, K, e, d) is reducible to (P', C', K', e', d'), and is written $(P, C, K, e, d) \sqsubseteq (P', C', K', e', d')$ if the following properties hold:

1. *$P = P'$ and $C = C'$;*
2. *There is a relation $R \subset K \times K'$ such that for all $k \in K$, there is $k' \in K'$ such that $(k, k') \in R$ and $e_k = e_{k'}$.*

The two cryptographic systems are said to be equivalent and is written $(P, C, K, e, d) \equiv (P', C', K', e', d')$ if $(P, C, K, e, d) \sqsubseteq (P', C', K', e', d')$ and $(P', C', K', e', d') \sqsubseteq (P, C, K, e, d)$.

Exercise 5.1.2. *Show that if $C \subset P'$ then (P, C', K'', e'', d'') where $K' = K \times K'$, $e''_{(k,k')} = e_{k'} \circ e_k$ and $d''_{(k,k')} = d_k \circ d_{k'}$ is a cryptographic system (called the composition or the product cryptographic system).*

Given the notion of product cryptographic system in the last exercise, one is able to define what is understood by a product of a cryptographic system S to be idempotent. As expected S is idempotent if S^2 is equivalent to S. The drawback of these systems is the fact that there is no point in using the product system S^2, as it requires an extra key but provides no more "security" in the sense that will be introduced later on. Notice that if a cryptographic system is not idempotent, then there is a potential increase in security by iterating it several times. This idea is used in the Data Encryption Standard, which consists of 16 iterations. A simple non idempotent system can be obtained as a product of two different cryptographic systems.

• • •

5.1.1 Steganography

Although this type of cryptography is somehow far from fulfill the requirements presented in the definition of cryptographic system, it is presented for historical reasons. Steganography rather than being encryption, is more like, the process of hiding the fact that a message exists at all, basing the security of the message on obscurity. As suggested in Wikipedia, the first recorded use of steganography was reported in 440 BC when Herodotus mentions two examples of steganography in his reports. One, where Demaratus sent a warning about a

forthcoming attack to Greece by writing it directly on the wooden backing of a wax tablet before applying its beeswax surface. The other, tells the story of a message tattooed on the shaved head of a slave of Histiaeus with a warning about Persian invasion plans, hidden by the hair that afterwards grew over it.

Other examples of the use of steganography can be found in recent history. For example, in the World War I secret messages were sent hidden in other messages. For instance, a German spy had sent the message

> *Pershing sails from NY June I.*

by hiding in on the following text and choosing the second letter of each word to write the message

> *Apparently neutral's protest is thoroughly discounter and ignored. Isman hard hit. Blockade issue affects pretext for embargo on byproducts, ejecting suets and vegetable oils.*

Another example, happened during World War II, when the French Resistance sent some messages written on the backs of couriers using invisible ink[4].

In 1966 during a televised press conference Jeremiah Denton repeatedly blinked his eyes in Morse Code that he was forced into as an American POW by his North Vietnamese captors, spelling out the word, "T-O-R-T-U-R-E", confirming that U.S. soldiers were being tortured in North Vietnam.

• • •

5.1.2 The substitution cipher

From now on it is assumed that the alphabet in use is the English one in which the palintext and cipher texts are written.

The Substitution Cipher are simple ciphers that are based on the idea of replacing each letter in the plaintext with a different letter. Choosing P and C both to be the 26-letter English alphabet

[4]This technique is also used in Alexander Dumas' book "The count of Monte Cristo", if the reader is familiar with this piece of literature.

and the encryption and decryption processes are just permutations of alphabetic characters. This kind of system was used until the XVIII century.

It is adopt the convention to represent plaintext characters in lower case and ciphertext characters in upper case ones. Note that in this case, the key is rather long (as long as the alphabet, since it is necessary to indicate which character replaces the original one), and there are many possible encryptions (26! or $403291461126605635840000000 \approx 4 \times 10^{27}$ which is considered to be sufficiently large to prevent exhaustive search for key.).

For convenience and practical use, letters are first transformed into numbers, to which are applied numerical operations and then reconverted into letter.

For sake of example consider that A wants to send the message

$$\begin{aligned} x &= wewillmeetatmidnight \\ &= wewil\ lmeet\ atmid\ night \end{aligned}$$

First using the following table we transform the text into numbers.

A/a	B/b	C/c	D/d	E/e	F/f	G/g	H/h	I/i	J/j	K/k	L/l	M/m
0	1	2	3	4	5	6	7	8	9	10	11	12

N/n	O/o	P/p	Q/q	R/r	S/s	T/t	U/u	V/v	W/w	X/x	Y/y	Z/z
13	14	15	16	17	18	19	20	21	22	23	24	25

Table 5.1: Alphabetic correspondence to numbers

The message is then transformed into

w	e	w	i	l	l	m	e	e	t	a	t	m	i	d	n	i	g	h	t
22	4	22	8	11	11	12	4	4	19	0	19	12	8	3	13	8	6	7	19

Now, assume that the key to this particular application is the following permutation π:

0	1	2	3	4	5	6	7	8	9	10	11	12
23	13	24	0	7	15	14	6	25	16	22	1	19

13	14	15	16	17	18	19	10	21	22	23	24	25
18	5	11	17	2	21	12	20	4	10	9	3	8

i.e., in terms of alphabet permutations, π is:

a	b	c	d	e	f	g	h	i	j	k	l	m
X	N	Y	A	H	P	O	G	Z	Q	W	B	T

n	o	p	q	r	s	t	u	v	x	w	y	z
S	F	L	R	C	V	M	U	E	K	J	D	I

Applying π to each letter of the message we transform the plaintext into:

w	e	w	i	l	l	m	e	e	t	a	t	m	i	d	n	i	g	h	t
22	4	22	8	11	11	12	4	4	19	0	19	12	8	3	13	8	6	7	19
10	7	10	25	1	1	19	7	7	12	23	12	19	25	0	18	25	14	6	12
H	P	H	T	W	W	X	P	P	E	L	E	X	T	O	Y	T	R	S	E

and the resulting ciphertext message is:

$$\begin{aligned} y &= HPHTW\ WXPPE\ LEXTO\ YTRSE \\ &= HPHTWWXPPELEXTOYTRSE \end{aligned}$$

To decrypt the message B repeats this process but instead of using π uses is inverse π^{-1}.

Summing up, a substitution cipher system can be formalized as follows:

Cryptographic system 5.1.3 (*Substitution cipher system*)

A substitution cipher system is a tuple $\mathcal{S} = (P, C, K, e_k, d_k)$ where:

1. $P = C = \mathbb{Z}_{26}$;

2. *K consists of all possible permutations of the 26 symbols;*

3. *For each permutation* $\pi \in K$, $e_\pi(x) = \pi(x)$ *and* $d_\pi(x) = \pi^{-1}(x)$.

In the case that one want to use this type of encryption, it is not very convenient to use purely random permutation since it is not very practical to decorate the permeation. So usually one fixes a phrase without repetitions of letters to be the key. For example one can use the phrase:

"X-night try blew Jack Zvq of Drumps."

Exercise 5.1.4. *Show that the above cipher is in fact a cryptographic system.*

Other examples of cryptographic systems are the shift ciphers (which a particular case with $c = 3$, for the letter C, is also known as Ceasar's ciphers which was described in "De Vita Caesarum, Divus Iulius" , i.e, "The Lives of the Caesars, The Deified Julius" , written in approximately 110 A.C) and the affine cipher which are presented in the next exercises.

Exercise 5.1.5 (shift or transposition cipher system)**.**

Consider the following tuple $\mathcal{T} = (P, C, K, e, d)$ *where:*

1. $P = C = K = \mathbb{Z}_{26}$;

2. *for all* $k \in K$ *let* $e_k(x) = (x + k) \pmod 2 6$ *and* $d_k(y) = (y - k) \pmod{26}$.

Show that (P, C, K, e, d) *is a cryptographic system.*

Exercise 5.1.6 (affine cipher system)**.** *Consider the following tuple* $\mathcal{A} = (P, C, K, e, d)$ *where:*

1. $P = C = \mathbb{Z}_{26}$;

2. $K = \mathbb{Z}_{26}^* \times \mathbb{Z}_{26}$;

3. *for all* $(k_1; k_2) \in K$ *let* $e_{(k_1,k_2)}(x) = (k_1 \cdot x + k_2) \pmod{26}$.

First, envisigate the need of the restriction to $\mathbb{Z}_{26}^* \times \mathbb{Z}_{26}$ *of the keys. After, for each* $e_{(k_1,k_2)}$ *define the respective deconding function* $d_{(k_1,k_2)}$. *Conclude by proving that the defined cipher system is, in fact, a cryptographic system.*

Exercise 5.1.7 (composition of shift cipher system). *Let* $\mathcal{T} = (\mathbb{Z}_{26}, \mathbb{Z}_{26}, \mathbb{Z}_{26}, e, d)$ *and* $\mathcal{T}' = (\mathbb{Z}_{26}, \mathbb{Z}_{26}, \mathbb{Z}_{26}, e', d')$ *be two shift cipher systems defined in Exercise 5.1.6. Show that they are equivalent accordingly to Definition 5.1.2. Study the structure of a composition of any two of these systems.*

• • •

5.1.3 The Verman cipher system

This cipher system was created and patented in 1917, by Gilbert Vernam and is also called the one-time pad cryptographic system. It is one of the simplest cryptographic systems and is described as follows:

The key is random, has the same length of the plaintext and it is used once justifying the alternative name. It is a particular case of the substitution cipher that later on will be proved to be perfectly secure.

Cryptographic system 5.1.8 (*Verman cipher system*)

The Verman cipher system also called the One time pad cipher system is the tuple $\mathcal{O} = (P, C, K, e, d)$ *such that:*

1. $P = C = K = \{0,1\}^m$;
2. *For a key* $k = k_1 \cdots k_m \in K$ *and for* $x = x_1 \cdots x_m \in P$ *and* $y = y_1 \cdots y_m \in C$, *define*

$$e_k(x) = x \oplus k = x_1 \oplus k_1 \oplus x_m \oplus k_m$$

and

$$d_k(y) = y \oplus k = y_1 \oplus k_1 \cdots y_m \oplus k_m$$

where the operation $\oplus$ *is the exclusive OR defined by*

$\oplus$	0	1
0	0	1
1	1	0

...

5.1.4 The Vigenère cipher

Although the cryptographic cipher system that is presented in the subsection ha the name of Blaise de Vigenère, who lived in the sixteenth century, the authority of this kind of ciphers was first reported by Giovan Battista Bellaso in his 1553 book *La cifra del. Sig. Giovan Battista Bellaso*.

The major difference between the substitution cipher and the Vigenère cipher is the fact that in the latter one instead of encrypting character by character the massage to be sent, A encrypts m alphabetic characters at a time by the use of a key of length m. For sake of presentation consider for example that A wants to encrypt the message

$$\begin{aligned} x &= attackatdawn \\ &= attac\ katda\ wn \end{aligned}$$

and A and B had chosen the key to be

$$k = LEMON$$

First using the Table 7.3 one gets:

a	t	t	a	c	k	a	t	d	a	w	n
0	19	19	0	2	10	0	19	3	0	22	13

and

l	e	m	o	n
11	4	12	14	13

Now, organizing the numbers of the plaintext elements in groups of 5 elements, it is summed each element of that group with the correspondent value of the key modulo 26, as follows:

plain text	a	t	t	a	c	k	a	t	d	a	w	n
code of plaintext	0	19	19	0	2	10	0	19	3	0	22	13
key	l	e	m	o	n	l	e	m	o	n	l	e
key code	11	4	12	14	13	11	4	12	14	13	11	4
ciphertext code	11	23	5	14	15	21	4	5	17	13	7	17
ciphertext	L	X	F	O	P	V	E	F	R	N	H	R

Thus the corresponding chiphertext is

$$\begin{aligned} x &= LXFOPVEFRNHR \\ &= LXFOP\ VEFRN\ HR \end{aligned}$$

The decryption process is similar using the same key, but this time with subtraction operation modulo 26 from the ciphertext, instead of adding.

Cryptographic system 5.1.9 (*Vigenère cipher system*)

Let m be a non null integer. The Vigenère cipher system is the tuple $\mathcal{V} = (P, C, K, E, D)$ where:

1. *$P = C = K = (\mathbb{Z}_{26})^m$;*
2. *For a key $k = (k_1, \cdots, k_m) \in K$, consider:*

$$e_k(x_1, \cdots, x_m) = (x_1 + k_1 \pmod{26}, \cdots, x_m + k_m \pmod{26})$$

and

$$d_k(x_1, \cdots, x_m) = (x_1 - k_1 \pmod{26}, \cdots, x_m - k_m \pmod{26}).$$

Notice that the example given above one can consider shorter keys or longer plaintext but replicating the key one can assume that the plaintext and the key words have the same length.

• • •

5.1.5 The Hill cipher

In this section, it is described the Hill cipher, another polyalphabetic cryptographic system invented in 1929 by Lester Hill. The basic idea is to consider m linear combinations of the m alphabetic characters in one plaintext element and works on groups of m letters at once and thus the encryption can be achieved by matrix multiplication.

Cryptographic system 5.1.10 (*Hill cipher system*)

Let $m \geq 2$ be a natural number. The Hill cipher system is the tuple (P, C, K, e, d) such that:

1. $P = C = (\mathbb{Z}_{26})^m$;
2. $K = \{T : (\mathbb{Z}_{26})^m \to (\mathbb{Z}_{26})^m$ *such that T is a linear isomorphism in* $(\mathbb{Z}_{26})^m\}$;
3. *For a key $k \in K$ and for $x \in P$ and $y \in C$, define*

$$e_k(x) = x \cdot k$$

and

$$d_k(x) = y \cdot k^{-1}.$$

Recall that $T : (\mathbb{Z}_{26})^m \to (\mathbb{Z}_{26})^m$ is a linear isomorphism in $(\mathbb{Z}_{26})^m$ if for all $a \in \mathbb{Z}$ and $x, y \in (\mathbb{Z}_{26})^m$, $T(a \cdot x + y) = aT(x) + T(y)$ and also if it is invertible.

Proposition 5.1.11. *If T is a linear transformation in $(\mathbb{Z}_{26})^m$ then there is a 26×26 matrix M with entries in $(\mathbb{Z}_{26})^m$ such that $T(x) = Mx$ for all $x \in (\mathbb{Z}_{26})^m$.*

Proof: The element $M_{i,j}$ of M is equal to $T(e_i)_j$ where e_i is the vector of all zeros and 1 in the position i. ■

Proposition 5.1.12. *T is an isomorphism if and only if M is invertible.*

The proof is direct and is left as an exercise.

Definition 5.1.3

Let $(\mathbb{M}_{\mathbb{Z}_n}, +)$ be the group of all $m \times m$ matrix with coefficients in $\mathbb{Z}_n$. There is a function $det : \mathbb{M}_{\mathbb{Z}_n} \to \mathbb{Z}_n$ such that, for all $M = [M^1, \cdots, M^n]$ (where it is seen a matrix as a vector of column vectors of dimension n):

Linearity on the columns *for all $a, b \in \mathbb{Z}_n$,*

$$\det([M^1, \cdots, M^{i-1}, aM^i + N^i, M^{i+1}, \cdots, M^n]) =$$
$$= a \det([M^1, \cdots, M^{i-1}, M^i, M^{i+1}, \cdots, M^n]) + \det([M^1, \cdots, M^{i-1}, N^i, M^{i+1}, \cdots, M^n])$$

Countervariance on the columns *if there is i such that $M^{i+1} = M^i$, then*

$$\det([M^1, \cdots, M^i, M^{i+1}, \cdots, M^n]) = 0.$$

Determinant of identity

$$\det(I) = 1.$$

That function is called the Determinant.

Exercise 5.1.13. *Let $(\mathbb{M}_{\mathbb{Z}_n}, +)$ be the group of all $m \times m$ matrix with coefficients in $\mathbb{Z}_n$. Show that:*

1. $\det([M^1, \cdots, M^i, M^{i+1}, \cdots, M^n]) = -\det([M^1, \cdots, M^{i+1}, M^i, \cdots, M^n])$;
2. $\det([M^1, \cdots, M^i, \cdots, M^k, \cdots, M^n]) = -\det([M^1, \cdots, M^k, \cdots, M^i, \cdots, M^n])$;
3. $\det(MN) = \det(M)det(N)$;
4. $\det(A^t) = \det(A)$ *where* $(A^t)_{i,j} = A_{j,i}$.

The proof of next lemmas are left as exercises for the interested reader.

Lemma 5.1.14. *Let $(\mathbb{M}_{\mathbb{Z}_n}, +)$ be the group of all $m \times m$ matrix with coefficients in $\mathbb{Z}_n$. The determinant exists and is unique.*

Lemma 5.1.15. *Let $(\mathbb{M}_{\mathbb{Z}_n}, +)$ be the group of all $m \times m$ matrix with coefficients in $\mathbb{Z}_n$. The determinant can be computed in one of the following ways:*

1. $\det(M) = \sum_{k=1}^{m} (-1)^{i+k} M^{ik}$, *where M^{ik} is the matrix obtained from M ignoring the i^{th} line and k^{th} column.*

2. $\det(M) = \sum_{\sigma \in S_n} sign(\sigma) \prod_{i=1} nM_{i,\sigma_i}$, *where* S_n *is the set of all permutations* σ *over* $\{1, \cdots, n\}$ *and* $sign(\sigma) = (-1)^{N(\sigma)}$ *where* $N(\sigma)$ *is the number of transposition in the decomposition of* σ.

Theorem 5.1.1

Let $(\mathbb{M}_{\mathbb{Z}_n}, +)$ *be the group of all* $m \times m$ *matrix with coefficients in* $\mathbb{Z}_n$. *A matrix* $M \in \mathbb{M}_{\mathbb{Z}_n}$ *is invertible if and only if* $\det(M)$ *is an invertible number in* $\mathbb{Z}_n$.

Proof:

($\Leftarrow$) Simply observe that

$$1 = \det(I) = \det(MM^{-1}) = \det(M)\det(M^{-1})$$

$$1 = \det(I) = \det(M^{-1}M) = \det(M^{-1})\det(M).$$

This way, the inverse of $\det(M)$ is $\det(M^{-1})$.

($\Rightarrow$)

Assume that M is a matrix and assume that $\det(M) = d$ where $d \in \mathbb{Z}_{26}$ is invertible. The matrix N defined by $N_{i,j} = d^{-1}(-1)^{i+j} det(M^{ji})$ is the inverse of M.

In fact,

$$(MN)_{i,j} = \sum_{k=1}^{n} M_{i,k}N_{k,j} = d^{-1} \sum_{k=1}^{n} M_{i,k}(-1)^{k+j} \det(M^{jk})$$

In particular, if $j = i$

$$(MN)_{i,j} = d^{-1} \sum_{k=1}^{n} M_{i,k}(-1)^{k+j} \det(M^{jk}) = d^{-1} \cdot \det(M) = d^{-1} \cdot d = 1$$

and if $j \neq i$

$$(MN)_{i,j} = d^{-1} \sum_{k=1}^{n} M_{i,k}(-1)^{k+j} \det(M^{jk}) = d^{-1} \cdot 0 = 0.$$

■

For sake of presentation assume that $m = 2$, and the A wants to encrypt a message $x = (x_1, x_2)$. To produce $y = (y_1, y_2)$ the ciphetext, A uses an invertible linear transformation $\pmod{26}$, for example:

$$T = \begin{pmatrix} 2 & 3 \\ 5 & 6 \end{pmatrix}.$$

Notice that $\det(T) = 2 \cdot 6 - 5 \cdot 3 \pmod{26} \equiv -3 \pmod{26} = 23 \pmod{26}$, and thus T is invertible. To encode, the message

$$x = letsdoit$$

just form groups of two letter and and transform it into number accordingly to Table 7.3. For example taking $(l, e) = (12, 5)$, it is transformed to

$$T(11,5) = \begin{pmatrix} 11 & 4 \end{pmatrix} \begin{pmatrix} 2 & 3 \\ 5 & 6 \end{pmatrix} = \begin{pmatrix} 16 & 5 \end{pmatrix}$$

one end with the code $(16, 5) = (Q, F)$.

plaintext	le	ts	do	it
plaintext code	$(11,4)$	$(19,20)$	$(3,14)$	$(8,19)$
ciphertext code	$T(11,4)$ = $(16,5)$	$T(19,20)$ = $(8,21)$	$T(3,14)$ = $(24,15)$	$T(8,19)$ = $(7,8)$
ciphertext	(Q,F)	(I,V)	(Y,P)	(H,I)

So, the cipher text is

$$y = QFIVYPHI.$$

To decipher, one has to apply the inverse of T to the ciphertext:

$$T^{-1} = \begin{pmatrix} 24 & 2 \\ 19 & 8 \end{pmatrix}.$$

Theorem 5.1.2

Let $(\mathbb{M}_{\mathbb{Z}_n}, +)$ be the group of all $m \times m$ matrix with coefficients in $\mathbb{Z}_n$ and let M be an invertible matrix in $\mathbb{M}_{\mathbb{Z}_n}$. Then

$$M^{-1} = (\det M)^{-1} M^*$$

where $M^*_{i,j} = (-1)^{i+j} \det(M^{ij})$.

The proof of this result is left to the interested reader.

• • •

5.1.6 Stream ciphers

Until now all the cryptographic systems that were considered have the property that successive plaintext elements are encrypted using the same key and are also memoryless, in the sense that the same function is used to encrypt successive blocks and there is no need to use previous computed information to encrypt other elements. This means that in this kind of systems to encrypt, for example the second element the system will use some informations computed to the first element encrypted.

Another approach possible to design cryptographic schemes is to use what are called stream ciphers where individual characters, one at a time, are encrypted, with a cipher that varies with time and have memory since there are state functions that record the state of the function to be used in later stages. An example of this type of cryptographic system, called synchronous is one that constructs a keystream, i.e., a sequence of characters generated from the key and the state of the function, as input to the stream cipher and independently from the plaintext message and is presented bellow. Another example, called self-synchronizing or asynchronous stream cipher is a stream cipher in which the keystream is a function of the key and a fixed number of previous ciphertext characters.

Cryptographic system 5.1.16 (*synchronous stream cipher*)

A tupe (P, C, K, L, e, d) together with a function g is said to be a synchronous stream cipher if the following properties hold:

1. *P is the finite set of all possible plaintexts;*
2. *C is the finite set of all possible ciphertexts;*
3. *K is the finite set of possible keys;*
4. *L is a finite set called the keystream alphabet;*
5. *g is the keystream generator, i.e., g takes a key $k \in K$ as input, and generates an infinite string $z_1 z_2 \cdots$ called the keystream, such that for all $i > 1$, $z_i \in L$.*
6. *For each $z \in L$, there is an encryption function $e_z : P \to C$ and a decryption function $d_z : C \to P$ such that for all $x \in P$, $d_z(e_z(x)) = x$.*

A stream cipher is a periodic stream cipher with period d if $z_{i+d} = z_i$ for all integers $i > 1$. The Vigenère Cipher with keyword length m,is a periodic stream cipher with period m since it can be define using the keystream

$$z_i = \begin{cases} k_i & \text{if } 1 \leq i \leq m \\ z_{i-m} & \text{if } otherwise \end{cases},$$

where $k = (k_1, \cdots, k_m)$ is the key of the Vigenère cipher.

• • •

5.1.6.1 Linear Feedback Shift Registers

A linear feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state and hence is an elementary algorithm to generate a keystream, that among other properties, is able to produce sequences with large periods and is easy to implement in a computer.

In general, one can define a LFSR from the algorithmic point of view, using an n staged

algorithm, where each stage $R_{n-1}, \cdots, R_1, R_0$, store one bit, with one input and one output, and a timer which marks the evolution of the stages such that on the i^{th} cycle:

1. the contents of stage 0 is "output"
2. the contents of R_i moves to R_{i-1}, for $1 \leq i \leq n-1$ and
3. the content of stage R_{n-1} is a bynary linear combination of all the contents of the earlier stages $0, 1, \ldots, n-2$.

If one denotes the contents of stage R_j on cycle i by s_{i+j}, this allows to describe the stage R_{n-1} in terms of a recurrence relationship

$$s_{n+i} = \sum_{j=0}^{n-1} c_{n-j} s_{i+j},$$

where c_j, $1 \leq j \leq n$ are some constants 0 or 1 specifying the stages which contribute to the sum.

It is common to identify the constants c_k as coefficients of a polynomial

$$g(x) = \sum_{k=0}^{n} c_k x^k,$$

called connection polynomial of the LFSR.

For sake of understanding the LFSR it is presented next an example of this type of cryptographic cipher system to generate a synchronous keystream.

Example 5.1.17. *Start with a binary m-tuple $(k_1, \cdots, k_m)$ and define for all $1 \leq i \leq m$, $z_i = k_i$. The subsequent keystream is defined using a linear recurrence of degree m (i.e., each term depends on the previous m terms):*

$$z_{i+m} = \sum_{j=0}^{m-1} c_j z_{i+j} \pmod 2.$$

So the key Here is $k = k_0, \cdots, k_{m-1}, c_0, ..., cm_1$. Notice that if all $k_i = 0$ then the keystream is just all 0's leading to a ciphertext equal to the plaintext. The trick is to choose the constants $c_0, \cdots, c_{m_1}$ cleverly to produce a periodic keystream with period $2m-1$.

An example of non-synchronous stream cipher is the Autokey Cipher where the plaintext is used to construct the keystream with the initial priming key k.

Example 5.1.18 (Autokey). *Let the encryption be* $e_z(x) = (x+z) \pmod{26}$, *let the the initial key* $k = 8$ *and let the plaintext be* $x = comeonover$. *The strategy is to use* $z_1 = 8$ *and* $Z_i = x_i - 1$ *for all the other indexes.*

Plaintext	c	o	m	e	o	n	o	v	e	r
Plaintext code	2	14	12	4	14	12	14	21	4	17
Key	8									
Ciphertext code	10	24	10	14	2	14	2	23	1	18
Ciphertext	K	Y	K	O	C	O	C	X	B	S

And thus the resulting ciphertext is $y = KYKOCOCXBS$.

To decrypt just use $d_z(x) = (x-z) \pmod{26}$.

SECTION 5.2

Block Cipher Systems

This type of system is nowadays very common and are, in fact, largely used in practice to encode/decode messages since they provide confidentiality and are central part of many applications of cryptography other than data compression, namely authentication and digital signatures. The idea of these systems is to operate on n bits (called the block length) at the same time to perform the encoding of a message transforming the "block" into another block and look like a substitution cipher but with large character size (the block). The encryption function is again parameterized by a key k of a certain size, say l_k and usually are chosen at random. Also, in order to be a cryptographic system the encryption function must be a bijection.

Definition 5.2.1

Let P *and* C *be set of plaintexts and ciphertexts of size* n. *An* n*-bit block cipher is an invertible function* $e_k : P \to C$. *The inverse mapping is the decryption function and is denoted by* d_k.

The idea, similar to what happens in the previous example, if the plaintext messages exceeds one block in length then several techniques are applied to operate all the blocks individually.

One can evaluate the block ciphers with respect to the estimated security level (based on the general use by the community), the key size (that defines upper bound on the security but longer keys require, in general, more resources, memory, processing time, etc), the efficiency of the process, the block size (that affects the complexity of the system and its security), etc.

Recall that when a message has more than n bits, one can, for example, divide the message into n-bit blocks and encrypt each separately enabling the repetition and secure use of a block cipher under a single key. In what follows, it is described the 4 basic modes of operation of a block cipher system to address the encryption of several block. The four most common modes are ECB, CBC, CFB, and OFB. These are the first modes of operation and were proposed and specified in FIPS 81 in 1981.

• • •

5.2.1 ECB mode

The nomenclature ECB means electronic codebook and is the mode of operation where each block is ciphered using the same key, like the Vigenère cipher system. As it will be seen in the next section this type of encryption in vulnerable to attack that exploit statistical properties of the texts. Notice that identical blocks of plaintext are encrypted into identical ciphertext which means that is does not hide patterns.

Algorithm 5.2.1 (*ECB mode*)

Input: *A l_k bit string k as a key and $x_1, \cdots, x_t$ blocks of plaintexts of a certain length*

Output: *$y_1, \cdots, y_t$ blocks of ciphertexts;*

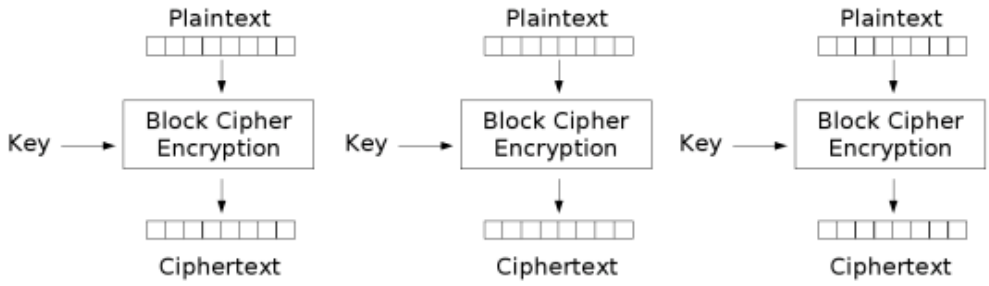
Step 1: $i = 0$*;*

Step 2: **While** $i \leq t$ **do**

Step 2.1: $y_i = e_k(x_i)$*;*

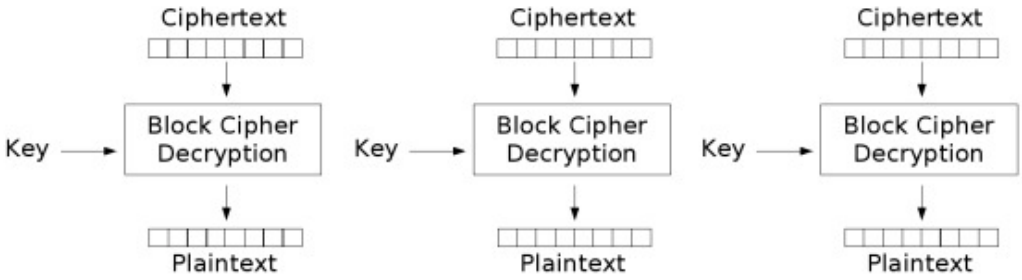
Step 2.2: $i = i + 1$*;*

Step : *Output* $y = y_1 \cdots y_t$*;*



Electronic Codebook (ECB) mode encryption

The decoding method is clear and consists basically by replacing the use of e_k by the use of d_k.



Electronic Codebook (ECB) mode decryption

• • •

5.2.2 CBC mode

The nomenclature CBC means cipher block chaining and is the mode of operation where each block is ciphered using the same key but at each block the plaintext ins mixed with the previous one. Notice that identical blocks of plaintext are only encrypted into identical ciphertext whenever the same part of the initial plain text and the same part of the key is used.

Notice also that since to encrypt a part of the plaintext one needs part of the previous block then the entire system is dependent on each other and the chaining mechanism causes ciphertext to be completely different and thus the decryption process cannot be executed separately like in the previous case.

Algorithm 5.2.2 (*CBC mode - encryption*)

Input: *A l_k bit string k as a key, $x_1, \cdots, x_t$ blocks of plaintexts of a certain length and an initial block x_0 of the same size of the others.*

Output: *$y_0 y_1, \cdots, y_t$ blocks of ciphertexts;*

Step 1: $y_0 = x_0$;

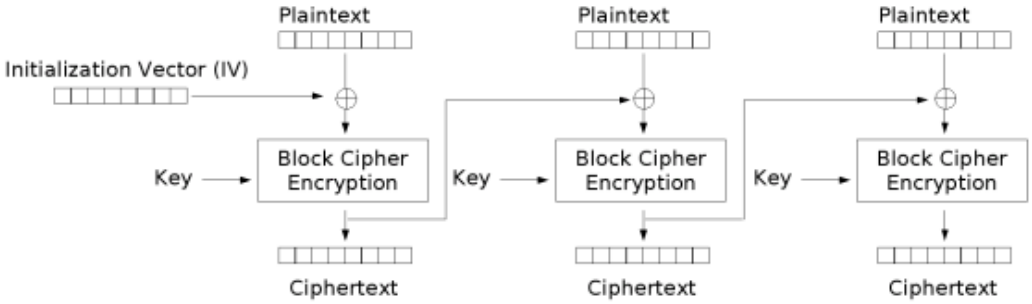
Step 2: $i = 1$;

Step 3: **While** $i \leq t$ **do**

Step 3.1: $y_i = e_k(y_{i-1} \oplus x_i)$;

Step 3.2: $i = i + 1$;

Step 4: *Output* $y = y_0 \cdots y_t$;



Cipher Block Chaining (CBC) mode encryption

The decoding method is:

Algorithm 5.2.3 (*CBC mode- decryption*)

Input: *A l_k bit string k as a key, $y_1, \cdots, y_t$ blocks of ciphertexts of a certain length.*

Output: *$x_1, \cdots, x_t$ blocks of plaintexts;*

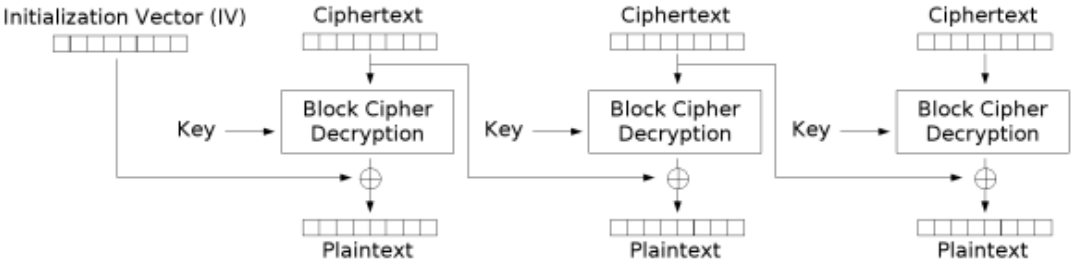
Step 2: $i = t$;

Step 3: **While** $i \geq 0$ **do**

Step 3.1: $x_i = y_{i-1} \oplus d_k(y_i)$;

Step 3.2: $i = i - 1$;

Step 4: *Output* $x = x_1 \cdots x_t$;



Cipher Block Chaining (CBC) mode decryption

• • •

5.2.3 CFB mode

The nomenclature CFB means cipher feedback and is closely related with the CBC mode, which makes a block cipher into a self-synchronizing stream cipher. It is assume that a shift register is available as described in the previous section.

Algorithm 5.2.4 (*CFB mode - encryption*)

Input: *A l_k bit string k as a key, $x_1, \cdots, x_u$ blocks of plaintexts of length r and an initial value for a shift register of length $n > r$ called x_0.*

Output: *$y_1, \cdots, y_t$ blocks of ciphertexts;*

Step 1: $I_1 = x_0$*;*

Step 2: $i = 1$*;*

Step 3: **While** $i \leq u$ **do**

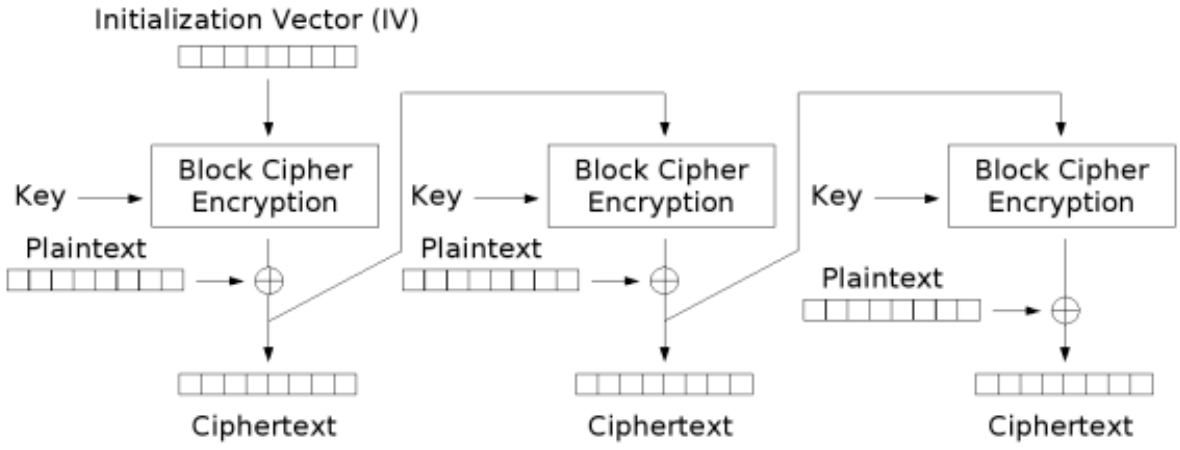
Step 3.1: $O_i = e_k(I_i)$*;*

Step 3.2: $t_i =$ *the r leftmost bits of* O_i*;*

Step 3.3: $y_i = x_i \oplus t_i$*;*

Step 3.4: $I_{i+1} = 2^r I_i + y_i \pmod{2^n}$

Step 4: *Output* $y = y_1 \cdots y_u$*;*



Cipher Feedback (CFB) mode encryption

The decoding method is:

Algorithm 5.2.5 (*CFB mode- decryption*)

Input: *A l_k bit string k as a key, $y_1 \cdots, y_u$ blocks of ciphertexts of size r and a initial vale y_0 of length $n > r$.*

Output: *$x_1, \cdots, x_u$ blocks of plaintexts;*

Step 1: $I_1 = y_0$*;*

Step 2: $i = 1$*;*

Step 3: **While** $i \leq u$ **do**

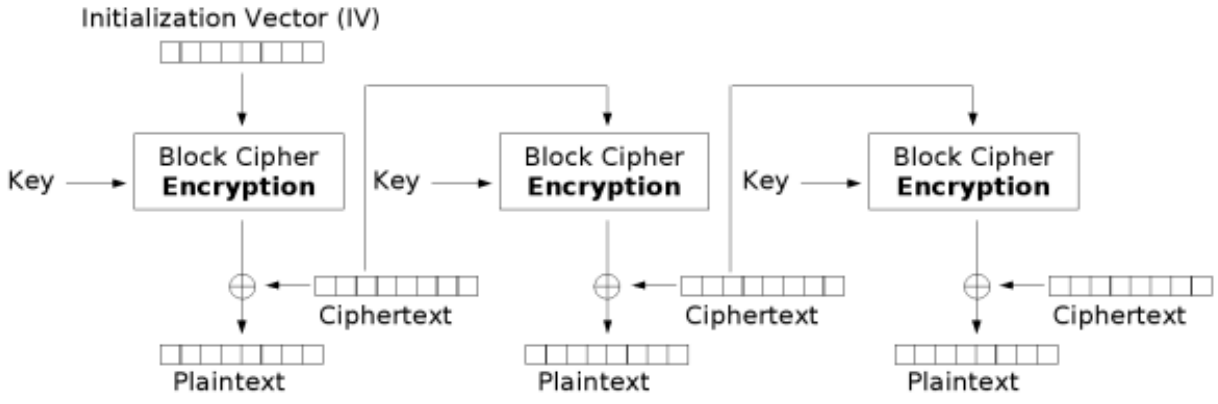
Step 3.1: $O_i = e_k(I_i)$*;*

Step 3.2: $t_i =$ *the r leftmost bits of* O_i*;*

Step 3.3: $x_i = y_i \oplus t_i$*;*

Step 3.4: $I_{i+1} = 2^r I_i + y_i \pmod{2^n}$

Step 4: *Output* $x = x_1 \cdots x_u$*;*



Cipher Feedback (CFB) mode decryption

...

5.2.4 OFB mode

The nomenclature (OFB) means output feedback mode and is similar to CFB, and allows encryption of various block sizes (characters), but differs in that the output of the encryption block function e_k (rather than the ciphertext) serves as the feedback. This means that it uses a synchronous stream cipher. It generates keystream blocks, which are then XORed with the plaintext blocks to get the ciphertext. Notice that, flipping a bit in the ciphertext produces a flipped bit in the plaintext at the same location. This property allows many error correcting codes to function normally even when applied before encryption.

Algorithm 5.2.6 (*OFB mode - encryption*)

Input: *A l_k bit string k as a key, $x_1, \cdots, x_u$ blocks of plaintexts of length r and an initial value for a shift register of length $n > r$ called x_0.*

Output: *$y_1, \cdots, y_t$ blocks of ciphertexts;*

Step 1: $I_1 = x_0$;

Step 2: $i = 1$;

Step 3: **While** $i \leq u$ **do**

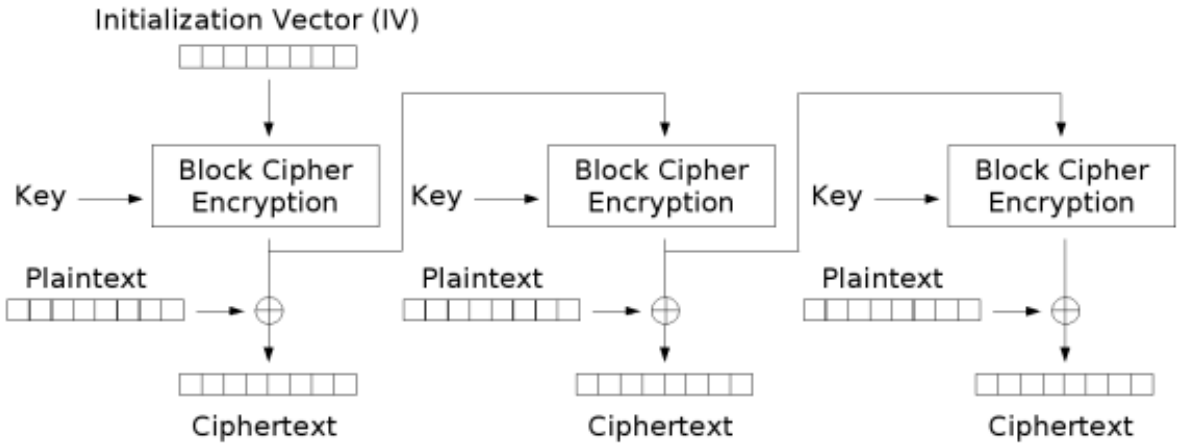
Step 3.1: $O_i = e_k(I_i)$;

Step 3.2: $t_i =$ *the r leftmost bits of* O_i;

Step 3.3: $y_i = x_i \oplus t_i$;

Step 3.4: $I_{i+1} = O_i$

Step 4: *Output* $y = y_1 \cdots y_u$;



Output Feedback (OFB) mode encryption

The decoding method is:

Algorithm 5.2.7 (*CFB mode- decryption*)

Input: *A l_k bit string k as a key, $\cdots, y_u$ blocks of ciphertexts of size r and a initial vale y_0 of length $n > r$.*

Output: *$x_1, \cdots, x_u$ blocks of plaintexts;*

Step 1: $I_1 = y_0$;

Step 2: $i = 1$;

Step 3: **While** $i \leq u$ **do**

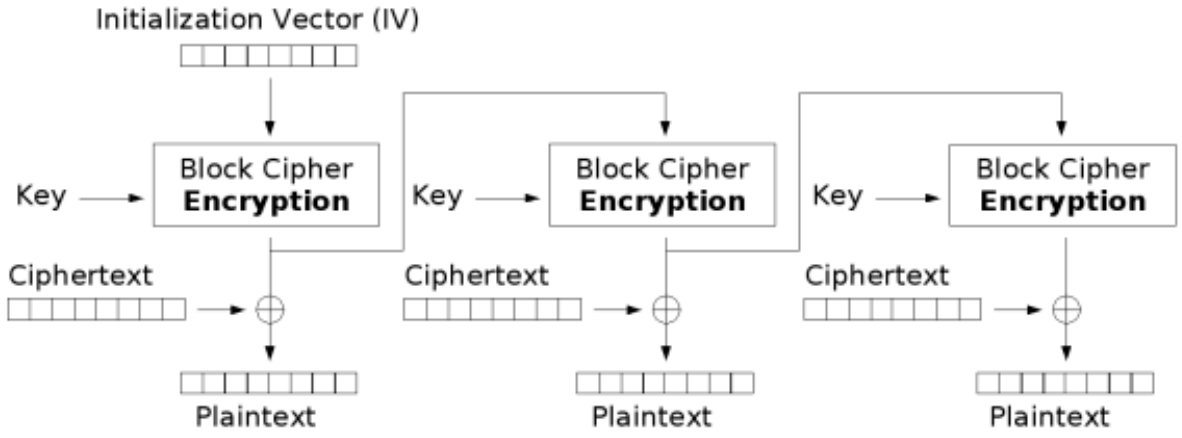
Step 3.1: $O_i = e_k(I_i)$;

Step 3.2: $t_i =$ *the r leftmost bits of* O_i;

Step 3.3: $x_i = y_i \oplus t_i$;

Step 3.4: $I_{i+1} = O_i$

Step 4: *Output* $x = x_1 \cdots x_u$;



Output Feedback (OFB) mode decryption

SECTION 5.3

Breaking down the cryptographic system

Since someone hd the need of ”hiding” information there is always someone on the other side trying to break it down without the authorization and the consent of the rightful parties. There are several types of attacks that an opponent can preform to access the secret information.

Before explaining the techniques there some comments have to me made. The first one is related with the following statement:

> The security of a cipher text should not be based on the assumption that an opponent does not which cipher A and B were using.

In fact, if one bases the security of “hiding” secret information on the fact that the adversaries do not know a priori the cipher system since it can try in parallel several attacks presuming, at each time, that the system is one of the known systems. Also, rather that putting all the eggs in the same basket, it is better to ensure security by the design presuming that it “secure” rather than using some techniques of encrypting that are known to be insecure and by obscurity.

This principle is known as the "Kerckhoffs' principle". From the point of view of A and B they rather have a system that is secure even if someone else knowing the type of encryption that was used, than risk the hypothesis of their secret message being discovered by trial and error.

The name of the principle is due to Auguste Kerckhoffs that in the19th century,published his opinion about the security of a cryptographic system outlining important design principles of such systems.

> *The cipher method must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvenience. A cryptographic system should be secure even if everything about the system, except the key, is public knowledge.*
>
> *Auguste Kerckhoffs*

Stated differently, Kerckhoffs' principle only requires that security rely only on the secrecy of the key, which seems dangerous. Although, one can argue in favor of this principle. First, for convenience, is that it is much easier for A and B to keep secret only a short key rather than keeping secret all the encryption/decryption method, since the first one is a string of some bits or bits and store this string securely, and the latter case also one needs to share and securely store the entire scheme that is thousands of times larger. Also, it might happen that details of scheme can be leaked (perhaps by an insider) or learned through reverse engineering. This situation is very improvable in the case of the secret key is a randomly generated string. Another point in favor to Kerckhoffs' principle is that in case the key is exposed, it is simpler to A and B agree to use another key that to replace the algorithm that is being-used. In terms of practically of use, in case many pairs of people, for example in a company, that encrypt their communication, one can imagine to be easier for all parties to use the same scheme, but different keys, than for everyone to use a different schemes. A discussion about the practically of shared string will be done in due course.

Nowadays, Kerckhoffs' principle is interpreted with the sense that security can not depend on secrecy of the scheme and, in fact, they should be public in contrast to the notion of "security by obscurity " which is the idea that improved security can be achieved by keeping a cryptographic algorithm hidden. Notice that being public enables the establishment of standards

used worldwide and also, such designs have been widely studied and for that reason one can easily believe that it is stronger.

The different possible attack models on cryptographic systems can be specified in terms of the information available to the adversary when he mounts his attack and can be:

Ciphertext only attack: the cryptanalyst has access only to a collection of ciphertexts or codetexts. This type of attack is also known as Eavesdropping.

Known plaintext attack: the attacker has access to some ciphertexts with the corresponding plaintext, which might result from, for example, a temporary access to the encryption method so it can choose a plaintext string and construct the corresponding encryption. It might also result from some informations that the attacker might know to be present in certain parts of the ciphertext.

Chosen plaintext (resp. ciphertext) attack: the opponent has access to some plaintexts (resp. ciphertext) corresponding to an arbitrary set of plaintexts (resp. ciphertext) of his own choice. Notice that in the case of ciphertext attacks, if the opponent only wants to know the corresponding plaintext he can get it without any effort, but in the ultimate case, he wants to be able to decrypt anything. So the idea is to gain knowledge not only about the text but also about the decryption process (namely the key).

Adaptive chosen plaintext (resp. ciphertext) attacks: This type of attacks are similar to the chosen plaintext (resp. ciphertext) attacks, but in this particular type the attacker can choose subsequent plaintexts (resp. ciphertext) based on information learned from previous encryptions (resp. decryptions).

Physical modification of messages: In this type of attacks the opponent forges some messages trying to convince the parties to be a legal message.

In all these cases, the goal of the opponent is to determine the ”key” that is used, allowing the decryption not only a specific ciphertext but also the decryption of any ciphertext encrypted using the same key.

The most basic attack is the cipher text only attack and it is normally the only information that an opponent has access to. In the sequel it is assumed that the plaintext is written in English,

without punctuation or ”spaces”. The reader is invited to think why that if punctuations and spaces where allowed the breaking process would become easier.

The main technique for breaking the ciphers is the statistic analysis. Each language, and in particular the English one, has a unique statistical signature. It is well known that letters in a natural language have stable frequencies and the knowledge of this frequency distribution provides a clue for cryptanalysis.

In fact, the frequency of each letter, the frequency of each digrams and other combination of letters and the frequency of certain words in common texts can be seen as “fingerprint” of the language and it can be used to cryptographic analyze cipher texts.

Several studies of this statistical properties have been done and they conclude that, for example, the letter E is the most frequent one, the digram TH is the most common and the combination ING appears quite a lot.

In the next figure and table it is summarized the frequency of each letter:

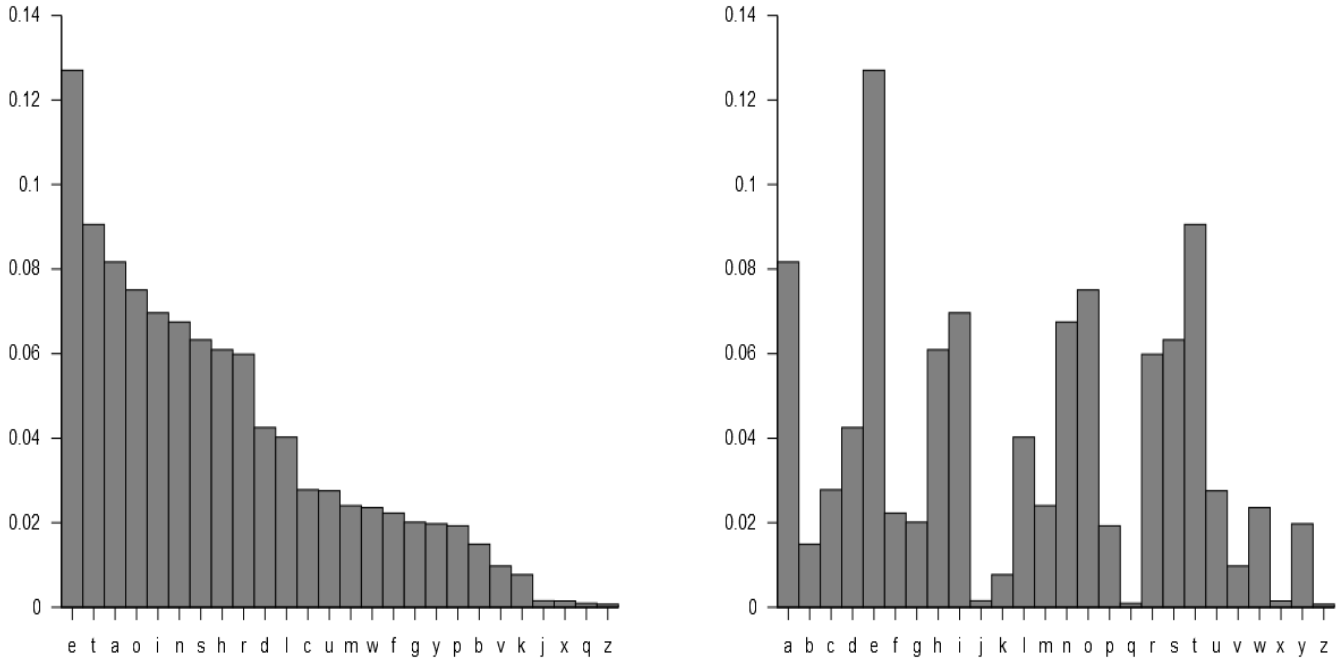


Figure 5.1: English letter frequencies, sorted by frequency and alphabetic, from Wikipedia

	Letters	Frequencies		Letters	Frequencies
I	E	12.7%	IV	M	2.4%
II	T	9.1%	IV	W	2.3%
II	A	8.2%	IV	F	2.2%
II	O	7.5%	IV	G	2.0%
II	I	7.0%	IV	Y	2.0%
II	N	6.7%	IV	P	1.9%
II	S	6.3%	IV	B	1.5%
II	H	6.1%	V	V	1.0%
II	R	6.0%	V	K	0.8%
III	D	4.3%	V	J	0.2%
III	L	4.0%	V	X	0.1%
IV	C	2.8%	V	Q	0.1%
IV	U	2.8%	V	Z	0.1%

Table 5.2: Frequencies of letters in English

In Table 5.2 the letters are grouped in 5 groups with the particularity that the frequencies are very similar. This way of grouping the letters is crucial for the statistical analysis when one is trying to crack the cryptographic system since normally, the cipher texts are not too long and thus fluctuations of frequencies might happen and thus some correspondences might not be the correct ones and require adjustments, i.e., for example, if in a ciphertext the letter R occurs with frequency 0.061 it would be natural to conjecture that it would correspond to h in the plaintext as it has the same frequency, but sometimes, depending on the text (and specially on short ones) it might be the case that it would correspond to some other letter in the Group II.

As it was mentioned previously it is also important the analysis of the combination of letters. The 30 most common digrams are:

TH HE IN ER AN RE ED ON ES ST EN AT TO NT HA
ND OU EA NG AS OR TI IS ET IT AR TE SE HI OF.

and the twelve most common trigrams are:

THE ING AND HER ERE ENT THA NTH WAS ETH FOR DTH.

...

5.3.1 Breaking down the substitution cipher

Recall that in the substitution cipher system each letter is replaced by another one according to a permutation. This kind of system has the property of preserving the properties of frequency of the permuted letter allowing the opponent to do a statistical analysis and infer the permutation used.

Assume that the following ciphertext is given

```
JWWEJ TYNUM FDFYJ TYJVU TQFVE FFCSE VURVF TNFJO FTHEU QUOZU NFYVE
FQRFH NZRJT SVUVE FCRRF NZFQV CXFNV JVCUT NJVVE FNZJT AFRMR JCWNJ
TYUKV EJKWV UZNJC WNEFF VNJTY EJWDJ RYNVE FPCMY UHTEJ KWJTY VEFVU
ZNJCW QWFHW CTFNJ TYMKT VWCTF NVEFD UKTSN JCWUR SJXFJ WUUAV UNFFV
EJVEC NURYF RNHFR FZRUO ZVWDJ TYJQQ KRJVF WDUMF DFYJT YVEFT VKRTF
YJSJC TVUVE FUHTF R
```

and one wants to decode the "secret" message hidden.

Notice that the text has 291 characters. The first step is to count the characters and perform a statistical analysis:

F	J	V	T	U	N	E	R	Y	W	C	Z
38	30	29	23	22	20	18	16	15	15	13	8
13.4%	10.3%	9.9%	7.9%	7.6%	6.8%	6.2%	5.5%	5.2%	5.2%	4.5%	2,7%

K	Q	D	H	M	S	O	X	A	P	L	B	I	G
7	7	6	6	5	5	3	2	2	1	0	0	0	0
2.4%	2.4%	2.1%	2.1%	1.7%	1.7%	1.0%	0.7%	0.7%	0.3%	0%	0%	0%	0%

Since F occurs more frequently than others it is reasonable to conjecture that $d(F) = e$.

The other 8 most frequent characters that happen in the text are J, V, T, U, N, E, R, Y and for the same reason it is reasonable to expect that these letters are encryptions of (a subset of) t, a,

o, i, n, s, h, r, but the frequencies really do not vary enough to tell us what the correspondence might be. The digram EF occurs 10 times, the trigram VEF occurs 9 times and FE does not occur it is reasonable to assume $d(E) = h$ and $d(V) = t$. Also FT occurs 3 times and FTN occurs once one may assume that $d(T) = n$ and $D(Y) = d$. A similar argument can be used to conjecture that $d(R) = r$ and $d(U) = o$ since "er" and "to" are also common digrams.

Since "an" is also a common digram one may guess that $d(J) = a$.


```
JWWEJ TYNUM FDFYJ TYJVU TQFVE FFCSE VURVF TNFJO FTHEU QUOZU NFYVE
a--ha nd-o- e-eda ndato n-eth ee--- torte n-ea- en--o -o--o -edth

FQRFH NZRJT SVUVE FCRRF NZFQV CXFNV JVCUT NJVVE FNZJT AFRMR JCWNJ
e-re- --ran -toth e-rre --e-t --e-t at-on -atth e--an -er-r a---a

TYUKV EJKWV UZNJC WNEFF VNJTY EJWDJ RYNVE FPCMY UHTEJ KWJTY VEFVU
ndo-t ha--t o--a- --hee t-and h---a rd-th e---d o-nha --and the-o

ZNJCW QWFHW CTFNJ TYMKT VWCTF NVEFD UKTSN JCWUR SJXFJ WUUAV UNFFV
--a-- --e-- -ne-a nd--n t--ne -the- o-n-- a--or -a-ea -oo-- o-eet

EJVEC NURYF RNHFR FZRUO ZVWDJ TYJQQ KRJVF WDUMF DFYJT YVEFT VKRTF
hath- -orde r--er e-ro- -t--a nda-- -rate --o-e -edan dthen t-rne

YJSJC TVUVE FUHTF R
d--a- ntoth eo-ne r
```

If the guesses are right and since aWW is followed by two equal letter one may conjuncture that $d(W) = l$. Since "se", "es" and "st" are common digrams and "NF", "FN" and "NV" appear also in the chiphertext it is reasonable to assume that $d(N) = s$, leading to:

```
JWWEJ TYNUM FDFYJ TYJVU TQFVE FFCSE VURVF TNFJO FTHEU QUOZU NFYVE
allha ndso- e-eda ndato n-eth ee--- torte nsea- en--o -o--o sedth
```

```
FQRFH NZRJT SVUVE FCRRF NZFQV CXFNV JVCUT NJVVE FNZJT AFRMR JCWNJ
e-re- s-ran -toth e-rre s-e-t --est at-on satth es-an -er-r a-lsa

TYUKV EJKWV UZNJC WNEFF VNJTY EJWDJ RYNVE FPCMY UHTEJ KWJTY VEFVU
ndo-t ha-lt o-sa- lshee tsand h-l-a rdsth e---d o-nha -land the-o

ZNJCW QWFHW CTFNJ TYMKT VWCTF NVEFD UKTSN JCWUR SJXFJ WUUAV UNFFV
-sa-l -le-l -nesa nd--n tl-ne sthe- o-n-s a-lor -a-ea loo-- oseet

EJVEC NURYF RNHFR FZRUO ZVWDJ TYJQQ KRJVF WDUMF DFYJT YVEFT VKRTF
hath- sorde rs-er e-ro- -tl-a nda-- -rate l-o-e -edan dthen t-rne

YJSJC TVUVE FUHTF R
d--a- ntoth eo-ne r
```

Now one can infer the code of the rest of the consonants. For example, $stat - on$ which is encoded by $NVJVCUT$ suggests that $d(C) = i$. Similarly, $VKRTFY$ which is $t - rned$ decoded suggests that $d(K) = u$.

```
JWWEJ TYNUM FDFYJ TYJVU TQFVE FFCSE VURVF TNFJO FTHEU QUOZU NFYVE
allha ndso- e-eda ndato n-eth eei-- torte nsea- en--o -o--o sedth

FQRFH NZRJT SVUVE FCRRF NZFQV CXFNV JVCUT NJVVE FNZJT AFRMR JCWNJ
e-re- s-ran -toth eirre s-e-t i-est ation satth es-an -er-r ailsa

TYUKV EJKWV UZNJC WNEFF VNJTY EJWDJ RYNVE FPCMY UHTEJ KWJTY VEFVU
ndout hault o-sai lshee tsand h-l-a rdsth e-i-d o-nha uland the-o


ZNJCW QWFHW CTFNJ TYMKT VWCTF NVEFD UKTSN JCWUR SJXFJ WUUAV UNFFV
-sail -le-l inesa nd-un tline sthe- oun-s ailor -a-ea loo-- oseet
```

```
EJVEC NURYF RNHFR FZRUO ZVWDJ TYJQQ KRJVF WDUMF DFYJT YVEFT VKRTF
hathi sorde rs-er e-ro- -tl-a nda-- urate l-o-e -edan dthen turne

YJSJC TVUVE FUHTF R
d--ai ntoth eo-ne r
```


Now for example one could infer that $d(M) = f$ has for example the word "of" is common. But in the text one would get words like "frails" which is a non sense. Using similar reasoning it is now easy to decode the rest of the text to get:

> *all hands obeyed and at once the eight or ten seamen who composed the crew sprang to their respective stations at the spanker brails and outhaul topsail sheets and halyards the jib downhaul and the topsail clewlines and buntlines the young sailor gave a look to see that his orders were promptly and accurately obeyed and then turned again to the owner*
>
> "The count of Monte Cristo"
>
> Alexandre Dumas.

Exercise 5.3.1. *Envisigate on a possible attack to a shift cipher system and affine cipher system presented in Exercises 5.1.5 and 5.1.6 respectively.*

The reader might not find any difficulty breaking down a Ceasar chipper of any size and in particular the smallest one since there are only a few keys that one can use to encrypt. The same issue happens for the substitution ciphers (specially if one uses the computer for a brute force attack). The vulnerability is the number of possible keys available.

This consideration leads to what it is called the "sufficient key space principle":

> *Any secure encryption scheme must have a key space that is not vulnerable to exhaustive search.*

• • •

5.3.2 Breaking the Vigenère cipher

Recall that the Vigenère cipher system requires the use of a key to encrypt that is repeated several number of times to cover the length of the entire plaintext.

At first site it seems to be unbreakable, since it uses of up to 26 different cipher alphabets. During two centuries this encryption method seemed incredibly difficult to decipher, because of their resistance to letter frequency analysis. Indeed, over time, the Vigenère cipher became known as "Le Chiffre Undechiffrable", i.e. "The Unbreakable Cipher".

In 1854, over two hundred years later, that the Vigenère Cipher was finally broke down by the British cryptographer Charles Babbage. Babbage employed a mix of cryptographic genius, intuition and sheer cunning to break the Vigenère Cipher. Amazingly, his work was never published in his lifetime, and it was over a hundred years later, in the 1970's, that his technique was finally made public.

Notice that each at time the key is repeated the same unique key is used to encrypt, i.e., if m is the length of the key of the Vigenère cipher, dividing the ciphertext into m block consisting of the letters that are encrypted with the same letter of the key, they form a shift cipher introduced in exercise 5.1.5 which are easy to break down (simply find the most frequent letter that in principal is the encode of "e"). Concretely,

$$\begin{aligned} y \;=\; & y_1 y_{1+m} y_{1+2m} \cdots \\ & y_2 y_{2+m} y_{2+2m} \cdots \\ & \vdots \\ & y_m y_{2m} y_{3m} \cdots \end{aligned}$$

i.e., writing the ciphertext in columns, each line above is a shift cipher. Hence, the trick to break down this particular type of ciphers is to infer correctly the length of the key used. In what follows it is explained another approach of determining the key length and the key directly.

There are several techniques that can be used to find the length of the key used. In the sequel it is presented the Kasiski test that was described by Friedrich Kasiski in 1863, although it was already described by Charles Babbage. The trick is to observe that two identical segments

of plaintext will be encrypted to the same ciphertext whenever their occurrence are δ positions apart from each other, where δ is such that $\delta \equiv 0 \pmod{m}$. On the other hand, if two segments of ciphertext (of length at least 3) is observed, then the probability that they correspond to identical segments of plaintext is high and can be used to infer the factors of the length of the key. The Kasiski test can be described in the following way: First one searches the ciphertext for pairs of identical segments of length at least 3, and write down the respective distances between the starting positions of the segments. In fact, if several of such distances occur, then one may infer that, probably, the length of the key is a divisor of the greatest common divisor of all those distances.

Example 5.3.2. *Assume that an opponent intersects the following block of ciphertext from a communication between A and B that was ciphered with a Vigenère cipher.*

```
ISETI LLSUS BFCJX DCECH UZRQR TEHKR VLNFW DXUEL UTNGL TLLVL IZBGT JWLGH
SZWPS JEOHX WPYQY CRBTI PEHIM KTNIE XCIDS JRHVE SZZGR KZLWQ TDOPF PYKKR
VLNFG GPDKX PYDKR KPSVQ TYTUI RFRKX XPSCR SEHGC HEOQH DYMAW WPLHM CCEFE
COGQP SWIMI CPWOS CPYHV DXTJI BTNVT GZMKW XYGVS JYFQP SEHGW WTNKR VDEEV
TESVL PEOPP NXIFE HLNFQ DCGCR PYDOE TNEPE HVNGA
```

The trigrams "KRV" and "LNF" occur 3 times each in the ciphertext on the positions 32, 122 and 257 and 29, 119 and 234 leading to the distances 90, 115 and 135 which have 5 as greatest common divisor. So, with high probability, 5 in the length of the key.

The index of coincidence is another useful tool that one can use to crack the Vigenère cipher. This tool was proposed and studied by William Friedman in 1920, as follows.

Definition 5.3.1 (*Index of coincidence*)

Let $x = x_1x_2\cdots x_n$ be a string of n alphabetic characters. The index of coincidence of x, denoted by $Ic(x)$ is defined to be the probability that two random elements of x are identical. If $f_0, \cdots f_{25}$ are the frequencies of the letter of the alphabet in x then

$$Ic(x) = \frac{\sum_{i=0}^{25} \binom{f_i}{2}}{\binom{n}{2}} = \sum_{i=0}^{25} \frac{f_i(f_i-1)}{n(n-1)}.$$

In an English written text the index of coincidence is near 0.065 and in a random written text, it is 0.038. The utility of this concept is based on the fact that if one consider blocks as described above for the ciphertext with the correct key length one gets, with high probability $Ic(x) = 0.065$, since in that case the cipher in each block is just a shift, and if the blocks are divided with the wrong key length the index of coincidence will be far from that value.

Example 5.3.3. *Recall the cipher text on the Example 5.3.2. In the next table is presented the average of the indexes of coincidence for the length of the key consider:*

length of the key	Average of indexes of coincidence of the blocks
$m = 1$	$Ic = 0.039$
$m = 2$	$Ic = 0.050$
$m = 3$	$Ic = 0.032$
$m = 4$	$Ic = 0.054$
$m = 5$	$Ic = 0.058$
$m = 6$	$Ic = 0.032$
$\vdots$	

Table 5.3: Average indexes of coincidence of the cipher text

Although 0.058 *is a bit faraway from* 0.065 *it is the closest one and thus,*[5] *one may conjecture that* $m = 5$.

A similar approach to the Index of coincidence can be implemented to infer, in each block the value of the shift that was used to encrypt it. Assume that m is the length of the key and for each y_i, where $1 \leq i \leq m$, let $f_0, \cdots, f_{25}$ be the frequencies of all letters of the alphabet in y_i. Then the probability of the letter j is f_j/l where $l = n/m$. Since each block is a shift of the plaintext of k_i one would expect that the probabilities on the block y_i would be $f_{k_i}/l, \cdots, f_{25+k_i}/l$ would be similar to the probability $p_0, \cdots, p_{25}$ of the occurrence of the letters in English expressed in Table 5.2. Thus, if one defines, for each block y_i and all $0 \leq g \leq 25$ the quantities:

[5] the difference might be because of the length of the message or the type of writing of the author.

$$M_{y_i,g} = \sum_{i=0}^{25} \frac{p_i \cdot f_{i+g}}{l}$$

one expects to get $M_{y_i,g} \approx 0.065$ if $g = k_i$ and $M_{y_i,g} \not\approx 0.065$ if $g \neq k_i$.

Example 5.3.4. *Continuing the Example 5.3.2, and producing the calculus for all of the* 5 *blocks one would conclude that the key used was* $K = (15, 11, 0, 2, 4)$ *leading to the key* $K =$ *place and to the plaintext:*

> *there was so much to read for one thing and so much fine health to be pulled down out of the young breath giving air i bought a dozen volumes on banking and credit and investment securities and they stood on my shelf in red and gold like new money from the mint promising to unfold the shining secrets that only midas and morgan and maecenas knew*
>
> *"The great Gatsby"*
>
> *F. Scott Fitzgerald*

• • •

5.3.3 Breaking the Hill cipher

This cipher it is more difficult to break down[6] using a ciphertext only attack than the previous one, but it is easily breakable to chosen plaintext attacks.

Recall that in this cipher system there is a key that can be seen as a squared matrix that has the particular property: it is invertible modulo 26. This is a vulnerability that an opponent can take advantage of if it has access to the encryption process. Assume that the opponent already know the length of the key used, say m. All that the opponent has to do is to choose m plaintexts and solve a system of m independent equations. An example will make this ideas clear.

Example 5.3.5. *Assume that the plaintext "brainwash" is encrypted using a Hill Cipher with a key of length* 3*, and the ciphertext obtained is "ETT BMV CPG". Thus one have:*

[6] This does not mean it is "secure", in fact, when the length of the key is two, this cipher system still be vulnerable to the digram frequencies.

$$e(1,17,0) = (4,19,19)$$

$$e(8,13,22) = (1,12,21)$$

$$e(0,18,7) = (2,15,6)$$

leading to the equation:

$$\begin{pmatrix} 4 & 19 & 19 \\ 1 & 12 & 21 \\ 2 & 15 & 6 \end{pmatrix} = \begin{pmatrix} 1 & 17 & 0 \\ 8 & 13 & 22 \\ 0 & 18 & 7 \end{pmatrix} \cdot k$$

Thus:

$$\begin{aligned} k &= \begin{pmatrix} 1 & 17 & 0 \\ 8 & 13 & 22 \\ 0 & 18 & 7 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 4 & 19 & 19 \\ 1 & 12 & 21 \\ 2 & 15 & 24 \end{pmatrix} \\ &= \begin{pmatrix} 5 & 19 & 22 \\ 12 & 5 & 14 \\ 10 & 2 & 7 \end{pmatrix} \cdot \begin{pmatrix} 4 & 19 & 19 \\ 1 & 12 & 21 \\ 2 & 15 & 6 \end{pmatrix} \\ &= \begin{pmatrix} 5 & 3 & 2 \\ 3 & 4 & 1 \\ 0 & 3 & 2 \end{pmatrix} \end{aligned}$$

If an opponent does not know the length of the key, it can try successively the length $m = 1,2,3,4,\cdots$ until the right length is found. One can be sure if the right length is found if the matrix found in the way described above will agree with other plaintext-ciphertext.

If the opponent does not have access to the encryption method used and tries to attack the Hill system with a ciphertext only attack it can break the ciphertext into blocks of length of the key, say for example 2, and picking the most frequent ciphertext digram, which with high probability will corresponds to the encryption of a common digram of the message (for example, TH or ST) and for each such guess, proceed as in the known-plaintext attack, until the correct encryption matrix is found.

• • •

5.3.4 Breaking the LFCS

Recall that the ciphertext is obtained from the plaintext and the keystream by summing them up modulo 2, i.e., $y_i = (x_i + z_i) \pmod 2$ and the keystream is produced from an initial key, i.e., $(z_1, \cdots, z_m) = (k_1, \cdots, k_m)$, using a linear combination $z_{m+i} = \sum_{j=0}^{m-1} c_j z_{i+j} \pmod 2$. This can again be used to show that this cipher can be break with a known-plaintext attack.

Assume that an opponent has a plaintext $x = x_1 \cdots x_n$ and the corresponding ciphertext $Y = y_1 \cdots y_n$. Notice that, since $y_i = x_i + z_i \pmod 2$ then $z_i = y_i + x_i$, so it is easy, in O knows the length of the initial key of the keystream it is easy to recover it. All that is left to do is to determine the keystream bits $c_0, \cdots, c_{m_1}$ in order to be able to reconstruct the entire keystream.

In other words, all that O has to determine the values of m unknowns. In fact, for any $i > 1$, by the knowledge of the plaintext and the ciphertext one gets a linear equation in the m unknowns. In the case of $n > 2m$ the system

$$z_{m+i} = \sum_{j=0}^{m-1} c_j z_{i+j} \pmod 2$$

can be determined which can be rewritten in matrix form as follows:

$$(z_{m+1}, \cdots, z_{2m}) = (c_0, \cdots, c_{m_1}) \begin{pmatrix} z_1 & z_2 & \cdots & z_m \\ z_2 & z_3 & \cdots & z_{m+1} \\ \vdots & \vdots & \vdots & \vdots \\ z_m & z_{m+1} & \cdots & z_{2m} \end{pmatrix}$$

and if the matrix is invertible modulo 2 one can determine the coefficients $c_0, \cdots, c_{m-1}$. Notice that the matrix has an inverse if m is the degree of the recurrence used to generate the keystream.

SECTION 5.4

Exercises

Exercise 5.4.1. *Show that:*

1. $\mathcal{T} \circ \mathcal{T} \equiv \mathcal{T}$;
2. $\mathcal{A} \circ \mathcal{A} \equiv \mathcal{A}$;
3. $\mathcal{V} \circ \mathcal{V} \equiv \mathcal{V}$;
4. $\mathcal{H} \not\equiv \mathcal{H} \circ \mathcal{V}$;
5. $H \circ \mathcal{V} \equiv \mathcal{V} \circ \mathcal{H}$;

where $\mathcal{T}$, $\mathcal{A}$, $\mathcal{V}$ and $\mathcal{H}$ are, respectively a transposition, a afire, a Vigenère and a Hill cryptographic systems.

Exercise 5.4.2. *Show that if $\mathcal{S} = (X, X, K, E, D)$ is a cryptographic system such that $(K, \circ)$ is a monoid then $\mathcal{S} \circ \mathcal{S} \equiv \mathcal{S}$.*

Exercise 5.4.3. *Show that if $\mathcal{S} = (X, X, K, E, D)$ is a cryptographic system and the structure over the keys $(\{e_k : X \to X : k \in K\}, \circ)$ is a monoid then $\mathcal{S} \circ \mathcal{S} \equiv \mathcal{S}$.*

Exercise 5.4.4. *Let p be a prime umber. Prove that the number of 2×2 matrices that are invertible over $\mathbb{Z}_p$ is $(p^2 - 1)(p^2 - p)$.*

Exercise 5.4.5. *Envisage if the following matrices over $\mathbb{Z}_2 6$ have inverse and in affirmative case determine their inverse:*

1. $\begin{pmatrix} 3 & 5 \\ 2 & 12 \end{pmatrix}$;
2. $\begin{pmatrix} 2 & 5 \\ 9 & 5 \end{pmatrix}$;

3. $\begin{pmatrix} 3 & 12 & 25 \\ 6 & 24 & -2 \\ 0 & 1 & 3 \end{pmatrix}$;

4. $\begin{pmatrix} 15 & 9 & 17 \\ 23 & 2 & 4 \\ 11 & 12 & 1 \end{pmatrix}$.

Exercise 5.4.6. *Encrypt the message*

$$all\ in\ all\ all\ that\ meters\ is\ you$$

using a Vigenère with keys

- $k_1 = LIFE;$
- $k_2 = PASSION$

Exercise 5.4.7. *Assume that following message*

$$SSDTTRRNNRICNAWCOILOATHKIUSGYITATOAAEUN$$

was encrypted using double transposition of Vigenère. Find the keys as well as the plaintext message knowing that the keys where chosen in the set

$$\{SYSTEMS; ENGINEERING; UNIVERSITY; ESSEN; DUISBURG; CRYPTOGRAPHY\}$$

Exercise 5.4.8 (exercise 1.16 on [Sti06])**.** *Consider the following permutation of over* $\{1, \cdots, 8\}$*:*

x	1	2	3	4	5	6	7	8
$\pi(x)$	4	1	6	2	7	3	8	5

1. *Compute the cipher text of the following plaintext message:*

$$friendshipislikeabookittakesfewsecondstoburnbutittakesyearstowrite$$

2. *Compute the permutation* π^{-1}.

3. *Decrypt the following ciphertext:*

$$TGEEMNELNNTDROEOAAHDOETCSHAEIRLM$$

Exercise 5.4.9. *Show that the Shift cipher system presented in Exercise 5.1.5, the Substitution cipher system presented in Cryptographic System 5.1.3 and the Vigenère cipher system presented in Cryptographic System 5.1.9 are trivial to attack with a chosen plain text attack. Envisaged on least number of characters needed to attack these cipher systems.*

Exercise 5.4.10. *Decode the following text knowing that was ciphered with a shift cipher system.*

```
GUIOM UERJM MOWJQ OUEDF COQXD PRKBX QOXKV MCDLF COMJI IOQDA POKQD
EGUIO MOERO RMJER UBOER ORTVF CUEDF COQDP BILVW XDPRI ORUPA DQFCO
KOMFK UMNOF KUPPL PUBOQ VEFCO TDQPR UERCV MOPOY OEFOU IIUFO MOERO
RTVFC GUIOM GDVEV EXIVW CUOPG DQRUE VEDEO DAKUM NOFKU PPMID MFOSW
PJMVY OWPJK MFCOJ EVFOR MFUFO MDJFP UMFOR MLUVE KBMOY OELDV EFMAD
QVFMM OWDER MFQUV XCFXD PRIOR UPTVF CFCVQ FBLDV EFMAQ DINOY VERJQ
UEFUE RPUFO COQDV WMAQD IPOKQ DEGUI OMVEU EJEOS LOWFO RPBWP DMOXU
IO
```

Infer the title, and punctuation of the text.

Exercise 5.4.11. *Decode the following text knowing that was ciphered with a substitution cipher system.*

```
VGXYJ ZSAZY LYQVY FFSQJ YJSRU UWJJR UPZYV QZRAS ROYQJ ZSJZR ALWXX
RBSQW UQZSQ SUJVS JWSVY UNSFD WFFSP SRUDY VIARL PSYJX WNAJS SUGSY
AQWFL OZSAS RASQR LSLJZ ASSGS YAQYU LYCCF RSLVG QSFXD FWQSJ WVGQJ
NLRSQ INJJZ SDZYA PSWXV YRUJY RURUP VSYFJ ZWNPZ RZYLY BSAGQ DYUJG
YFFWO YUDSI SRUPJ WWPAS YJXWA YUYAA WOXWA JNUSR OYQIW NULYC CASUJ
RDSJW VATYV SQIYJ SQYUS VRUSU JQNAP SWURU FWULW UORJZ OZWVR DWUJR
UNSLX WNAGS YAQYU LVGXY JZSAU WOYUL JZSUQ SULRU PVSQV YFFQN VQWXV
WUSGR FYRLJ ZSVWN JRUFS YAURU PUYBR PYJRW UYULW JZSAC YAJQW XJZSV
YJZSV YJRDQ NQSXN FJWJZ WQSOZ WRUJS ULJWJ AYBSF YQRYF OYGQI SFRSB
SLRJO WNFLI SQWVS JRVSW AWJZS AVGXW AJNUS JWLW
```

Infer the title, and punctuation of the text.

Exercise 5.4.12. *Decode the following text knowing that was ciphered with a Vigenère cipher system. Hint: the text is written in Portuguese.*

```
QWEAR EMUIW URWSI PMMIO EXUFQ KOJUG RWFIL EHSLV AABIM BFQWD NSAVG
WJYRB RSUYH GWVFS SIRPV QATUR IMWRY HLIEQ STUVC RRWIZ VIEQA IWSLV
DSKFE MVALE XSWSO AHLSL RGBQH ITRGN ULHMJ DWBUG IJOVE LEURZ ZUNIW
LGACJ PMJAW DNIWI OEQFL MKSSF YWIIR WININ ISKSI RWDSV UPMDV BVQWE
UGSKJ LIWIO EQFLM KSKEG FZZCU QMEZR CVUJE ZVRWI MIDVF SIWSU LAJUH
UCVRW UMXZV WLQMZ IIOFT UWLVT WHLSV FDJYG IQICS DXEZV DGHWM URIEQ
NMUZR SVCPI USBQH ITZBZ QMEJI WYQXE ARPWY LELFH WBBEL FHADB EWRGH
UWXWK FAINS VYCVU LIAZR WDWMI VQDUM MIJHA SUUCV QGCJI BZOSK GEMUW
XYWEK RCVEL IQEOV EXEAV BZELE LDOJY UMKFA MCUWQ ESLQY GWDIE QWVCQ
BGJIT WRGKU GITYO JIYMI RIESI PMXWG TYNMJ IAJUW
```

Infer the title, and punctuation of the text.

Exercise 5.4.13. *(exercise 1.23 in [Sti06].) Suppose that you are told that the plaintext*

$$\textit{breathtaking}$$

yields the ciphertext

$$RUPOTENTOIFV.$$

Determine the encryption matrix knowing that the Hill Cipher is used (but m is not specified).

Exercise 5.4.14. *(exercise 1.25 in [Sti06].) Decrypt the following text knowing that was ciphered with a Hill cypher:*

```
LMQET XYEAG TXCTU IEWNC TXLZE WUAIS PZYVA PEWLM GQWYA XFTCJ
MSQCA DAGTX LMDXN XSNPJ QSYVA PRIQS MHNOC VAXFV
```

CHAPTER 6

Perfect secrecy

In this chapter it is studied the influence of Shannon's seminal paper on the notion of security of a cryptographic systems. But what security means, i.e., what does it mean for a cryptographic system to be secure?

In the previous chapter, some historical encryption schemes (ciphers) were presented and showed how one can, with little computational effort, break them down. In this chapter, it is studied conditions of the encryptions that are on the other extreme, i.e., it is studied encryption schemes and conditions required by those systems that make them provably secure even against an adversary who has unbounded computational power. Such schemes are called perfectly secret.

Security is a term that is not easy to define clearly since it depends on the context where it is considered and it is intrinsic to the kind of information one wants/needs to preserve. So, security is a characteristic of the information and the communication process and, of course, is definable depending on the goals the parties. For example if the goal in a communication is to preserve "privacy" of the message, then security is achieved if the information is kept secret from everyone but the authorized people to see it. Some other examples of most used "security" requirements are the following:

Data integrity: Make sure that the information of the message has not been altered by unauthorized or unknown means.

Identification: Corroboration of the identity of an user.

Message authentication: Corroboration of the source of information.

Signature: The original owner of the message should be bind information to an entity.

Access control: The information of the message or segments of it should be confidential to a restrictive access to resources to privileged entities.

Timestamping: The record of the time of creation or existence of information should be kept invulnerable.

There are several approaches to measure informally the "security" of a system:

Unconditional Security: This method is the ultimate goal of cryptography and thus excellent in theory. It means that there are no assumptions about the computing power and resources available to an adversary and the system still be unbreakable leaking no information whatsoever.

Provable Security: It is said that a system has provable security if its security requirements can be stated formally in an adversarial model. Usually the proof of security of such systems is the creation of a reduction from the assumptions about the adversary's access to the system to meet the security requirements stated for the system.

It is crucial to observe that this approach only ensures security relative to some other problem, not an absolute proof of security. As an analogy think about a theorem: it is only true if the assumptions are confirmed to hold, otherwise it might not be true.

Computational Security: In many cases, provable security cannot be realistically achieved, and in such cases one must relax the security to computational security. Roughly speaking this means that these systems are secure assuming that any adversaries are computationally limited. Formally this measure is related with the computational effort required to break down the system i.e., the resources (time or space) that the best algorithm cracking the system requires.

The security of a cryptographic system is related with the kind of attack that the opponent is considering, as mentioned in the previous chapter. In the sequel it will be proved that certain

cryptographic systems are unconditionally secure against a ciphertext-only attack. In particular, a Shift Cipher or a Substitution Cipher or even the one time pad are both unconditionally secure if a single element of plaintext is encrypted with a random key and the Vigenère Cipher with keyword length m is unconditionally secure if the key is used to encrypt only one block of the plaintext.

For that purpose, some mathematical definitions will be given and the unconditional security with be rigorously proved, which gives some conditions os unconditional security without relying on any unproven assumptions. Although the obvious advantages of this approach this approach has inherent limitations.

SECTION 6.1

Definition of perfect security and results

To explain the concept of perfect secrecy and study in which conditions one says that a particular system has perfect secrecy several assumptions are made:

- The existence of a fixed cryptographic system (P, C, K, e, d) with the particularity that a key $k \in K$ is used for only one encryption.
- A fixed probability distribution over the plaintext space, P and thus the induced a so called, a priori probability distribution of the plaintext elements, which is a random variable, denoted by $\mathbf{x}$.
- The key k is chosen using some fixed probability distribution which is often the uniform one, and thus that one also has a random variable over the keys, denoted by $\mathbf{k}$.
- The keys and the plain texts are independent random variables.

Since any ciphertext y is obtained from a plaintext and a key, i.e., $y = d_k(x)$ then the probability distribution over K and the probability distribution $\mathbf{X}$ induce a probability distribution on C and thus one can see the ciphertext elements as a random variable, which for convenience it is denoted by $\mathbf{y}$. Notice that, denoting by $C(k)$ the set of possible ciphertexts that can be obtained

using the key k, i.e.,

$$C(k) = \{y \in C : \exists x \in P, d_k(x) = y\}$$

then the probability distribution related with $\mathbf{y}$ is defined, for every $y \in C$, by

$$\Pr[\mathbf{y} = y] = \sum_{\{k:y \in C(k)\}} \Pr[\mathbf{k} = k] \Pr[\mathbf{x} = d_k(y)].$$

It is also easy to compute the conditional probability distribution of the ciphertexts given the plaintexts:

$$\Pr[\mathbf{y} = y | \mathbf{x} = x] = \sum_{\{k:x = d_k(y)\}} \Pr[\mathbf{k} = k].$$

From Baye's Theorem 3.1.1 stated in page 31 one can know compute the a posteriori probability distribution of the plaintexts given the ciphertexts in the following way:

$$\Pr[\mathbf{x} = x | \mathbf{y} = y] = \Pr[\mathbf{x} = x] \cdot \frac{\sum_{\{k:x = d_k(y)\}} \Pr[\mathbf{k} = k]}{\sum_{\{k:y \in C(k)\}} \Pr[\mathbf{k} = k] \Pr[\mathbf{x} = d_k(y)]}.$$

Putting all together, and heuristically speaking, one system is said to have perfect secrecy if the a priori and the a posteriori probability distributions are independent. Formally one reaches the following definition

Definition 6.1.1 (*Perfect secrecy*)

Let $\mathcal{S}$ be a cryptographic system and let $\Pr[\mathbf{x} = x | \mathbf{y} = y]$ and $\Pr[\mathbf{x} = x]$ be, respectively the a posteriori and a priori probability distributions defined in the discussion above.
One says that $\mathcal{S}$ has perfect secrecy if the a posteriori probability that the plaintext is x, given that the ciphertext y is observed, is identical to the a priori probability that the plaintext is x i.e., $\Pr[\mathbf{x} = x | \mathbf{y} = y] = \Pr[\mathbf{x} = x]$ for all $x \in P$ and $y \in C$.

Example 6.1.1 (Example 2.3 of [Sti06]). *Consider $P = \{a, b\}$ such that $\Pr[a] = \frac{1}{4}$ and $\Pr[b] = \frac{3}{4}$, $K = \{k_1, k_2, k_3\}$, such that $\Pr[k_1] = \frac{1}{2}$, $\Pr[k_2] = \frac{1}{4}$ and $\Pr[k_3] = \frac{1}{4}$ and $C = \{1, 2, 3, 4\}$ such that the encryption is described by:*

	a	b
k_1	1	2
k_2	2	3
k_3	3	4

Thus, for example, since $2 \in C(k_1) \cup C(k_2)$ *it follows that*

$$\begin{aligned}
\Pr[\mathbf{y} = 2] &= \Pr[\mathbf{k} = k_1] \Pr[\mathbf{x} = d_{k_1}(2)] + \Pr[\mathbf{k} = k_2] \Pr[\mathbf{x} = d_{k_2}(2)] \\
&= \Pr[\mathbf{k} = k_1] \Pr[\mathbf{x} = b] + \Pr[\mathbf{k} = k_2] \Pr[\mathbf{x} = a] \\
&= \frac{1}{2} \cdot \frac{3}{4} + \frac{1}{4} \cdot \frac{1}{4} \\
&= \frac{7}{16}
\end{aligned}$$

The complete probability distribution of the random variable is:

$$\Pr[\mathbf{y} = 1] = \frac{1}{8} \qquad \Pr[\mathbf{y} = 2] = \frac{7}{16} \qquad \Pr[\mathbf{y} = 3] = \frac{1}{4} \qquad \Pr[\mathbf{y} = 4] = \frac{1}{16}$$

For example the a posteriori probability of $\mathbf{x} = a$ *given* $\mathbf{y} = 3$ *can be computed in the following manner:*

$$\{k : a = d_k(y)\} = \{k_2\}$$

and

$$\{k : y \in C(k)\} = \{k_1, k_2\}$$

thus:

$$\begin{aligned}
\Pr[\mathbf{x} = x | \mathbf{y} = y] &= \Pr[\mathbf{x} = x] \cdot \frac{\sum_{\{k : x = d_k(y)\}} \Pr[\mathbf{k} = k]}{\sum_{\{k : y \in C(k)\}} \Pr[\mathbf{k} = k] \Pr[\mathbf{x} = d_k(y)]} \\
&= \Pr[\mathbf{x} = a] \cdot \frac{\Pr[\mathbf{k} = k_2]}{\Pr[\mathbf{k} = k_1] \Pr[\mathbf{x} = d_{k_1}(2)] + \Pr[\mathbf{k} = k_2] \Pr[\mathbf{x} = d_{k_2}(2)]} \\
&= \Pr[\mathbf{x} = a] \cdot \frac{\Pr[\mathbf{k} = k_2]}{\Pr[\mathbf{k} = k_1] \Pr[\mathbf{x} = b] + \Pr[\mathbf{k} = k_2] \Pr[\mathbf{x} = a]} \\
&= \frac{1}{4} \cdot \frac{\frac{1}{4}}{\frac{1}{2} \cdot \frac{3}{4} + \frac{1}{4} \cdot \frac{1}{4}} \\
&= \frac{1}{7}
\end{aligned}$$

The complete a posteriori probability distribution is:

$$\Pr[\mathbf{x}=a|\mathbf{y}=1]=1 \quad \Pr[\mathbf{x}=a|\mathbf{y}=2]=\frac{1}{7} \quad \Pr[\mathbf{x}=a|\mathbf{y}=3]=\frac{1}{4} \quad \Pr[\mathbf{x}=a|\mathbf{y}=4]=0$$

$$\Pr[\mathbf{x}=b|\mathbf{y}=1]=0 \quad \Pr[\mathbf{x}=a|\mathbf{y}=2]=\frac{6}{7} \quad \Pr[\mathbf{x}=a|\mathbf{y}=3]=\frac{3}{4} \quad \Pr[\mathbf{x}=a|\mathbf{y}=4]=1$$

Although that $\Pr[\mathbf{x}=a|\mathbf{y}=3]=\Pr[\mathbf{x}=a]$ *all the other values are different and thus one concludes that this system has no perfect secrecy.*

Theorem 6.1.1 (*Perfect secrecy of shift cipher system*)

Let $S=(\mathbb{Z}_{26},\mathbb{Z}_{26},\mathbb{Z}_{26},e_k,d_k)$ *be a shift cipher system. If* $\mathbf{k}$ *is the random variable associated with the uniform probability distribution over the set of the keys then for all probability distribution of the plaintexts,* S *has prefect secrecy.*

Proof: Assume that $\Pr[\mathbf{x}=x]$ is the probability distribution associated with the random variable over P. Recall that the encryption rule is defined by $e_k(x)=x+k \pmod{26}$ where $0\leq k\leq 25$. Since the keys are uniformly distributed then,$C(k)=\mathbb{Z}_{26}$ for all $k\in K$ and thus for all $y\in C$

$$\begin{aligned}
\Pr[\mathbf{y}=y] &= \sum_{\{k:y\in C(k)\}} \Pr[\mathbf{k}=k]\Pr[\mathbf{x}=d_k(y)] \\
&= \sum_{k\in\mathbb{Z}_{26}} \frac{1}{26}\Pr[\mathbf{x}=y-k] \\
&= \frac{1}{26}\sum_{k\in\mathbb{Z}_{26}} \Pr[\mathbf{x}=y-k] \\
&= \frac{1}{26}\sum_{x\in\mathbb{Z}_{26}} \Pr[\mathbf{x}=x] \\
&= \frac{1}{26}.
\end{aligned}$$

Notice that for a fixed y the values $y-k \pmod{26}$ for $0\leq k\leq 25$ is a permutation over $\mathbb{Z}_{26}$ which is a justification the penultimate equality.

Intuitively, the value $\Pr[\mathbf{x}=x|\mathbf{y}=y]=\Pr[\mathbf{x}=x]$ since there is a bijection induced by the keys between P and C. Formally, to compute $\Pr[\mathbf{x}=x|\mathbf{y}=y]$ first observe that given a key k and plaintext x they uniquely determine y and thus, for all $x,y\in\mathbb{Z}_{26}$, that for all $y\in C$, $\Pr[\mathbf{y}=y]>0$

$$\Pr[\mathbf{y} = y | \mathbf{x} = x] = \Pr[\mathbf{k} = k] = \frac{1}{26}.$$

Now applying Bayes' Theorem 3.1.1 presented in Page 31, it follows that:

$$\begin{aligned} \Pr[\mathbf{x} = x | \mathbf{y} = y] &= \frac{\Pr[\mathbf{x} = x] \cdot \Pr[\mathbf{y} = y | \mathbf{x} = x]}{\Pr[\mathbf{y} = y]} \\ &= \frac{\Pr[\mathbf{x} = x] \cdot 1/26}{1/26} \\ &= \Pr[\mathbf{x} = x] \end{aligned}$$

leading to the conclusion that the shift cipher has perfect secrecy. ■

Notice that the crucial observation in the last proof was the fact that every $|K| = |P| = |C|$ and the keys were randomly chosen.

Lemma 6.1.2. *Let $S = (P, C, K, e, d)$ be a cryptographic system. If S has perfect secrecy then $|K| \geq |P|$.*

Proof: The proof is made by contradiction, i.e., by assuming that $|KI < IPI$ then S is not perfectly secret.

Assume also that the uniform distribution over P and let $y \in C$ be such that $\Pr[\mathbf{y} = y] > 0$. Define the set

$$P(y) = \{x \in P : \exists k \in K, y = e_k(x)\}.$$

Clearly $|P(y)| \leq IKI$ since for each message $x \in P(y)$ there is at least one key $k \in K$ for which $x = d_k(y)$. Since $IKI < IMI$, this means that there is some $x' \in P - P(y)$ and thus

$$\Pr[\mathbf{x} = x' | \mathbf{y} = y] = 0 \neq Pr[\mathbf{x} = x'],$$

and so S would not have perfect secrecy. ■

This leads to the following nice characterization of prefect secrecy in this cases:

Theorem 6.1.2 (*Shannon's theorem*)

Let $S = (P, C, K, e, d)$ be a cryptographic system such that $|K| = |P| = |C|$ and such that for all $y \in C$, $\Pr[\mathbf{y} = y] > 0$. Then S has perfect secrecy if and only if every key is uniformly chosen with equal probability $1/|K|$, and for every $x \in P$ and every $y \in C$, there is a unique key k such that $e_k(x) = y$.

Notice that assuming that that for all $y \in C$, $\Pr[\mathbf{y} = y] > 0$ is a reasonable assumption since if for some $y \in C$, $\Pr[\mathbf{y} = y] = 0$ this would mean that y is never used as a ciphertext and thus one could consider $C' = C - \{y\}$ without loss of generality.

Proof:

($\Leftarrow$)

Since for all $y \in C$, $\Pr[\mathbf{y} = y] > 0$ and the distributions $\mathbf{x}$ and $\mathbf{y}$ are independent, then for all x and y, $\Pr[\mathbf{y} = y | \mathbf{x} = x] = \Pr[\mathbf{y} = y] > 0$ and hence there is at least one $k \in K$ such that $e_k(x) = y$ and thus $|C| = |\{e_k(x) : k \in K\}| \leq |K|$.

Notice that fixing an x, the function

$$\begin{aligned} \mathbb{key}_x : K &\rightarrow Y \\ k &\rightarrow e_k(x) \end{aligned}$$

is then surjective and since $|K| = |C|$ it follows that $|\{e_k(x) : k \in K\}| = |K|$ which implies that it is also injective and thus, it cannot exist two distinct keys k_1 and k_2 such that $e_{k_1}(x) = e_{k_2}(x) = y$. This leads to the conclusion that for any $x \in P$ and $y \in C$, there is exactly one key k such that $e_k(x) = y$.

Assume that $|K| = |P| = |C| = n$ and assume that the elements of P are ordered, i.e., $P = \{x_i : 1 \leq i \leq n\}$. Given y one can enumerate the keys in the following way: $e_{k_i}(x_i) = y$.

By Bayes' Theorem 3.1.1 presented in page 31, one has:

$$\begin{aligned} \Pr[\mathbf{x} = x_i | \mathbf{y} = y] &= \frac{\Pr[\mathbf{y} = y | \mathbf{x} = x_i] \Pr[\mathbf{x} = x_i]}{\Pr[\mathbf{y} = y]} \\ &= \frac{\Pr[\mathbf{k} = k_i] \Pr[\mathbf{x} = x_i]}{\Pr[\mathbf{y} = y]}. \end{aligned}$$

By the perfect secrecy requirement $\Pr[\mathbf{x} = x_i|\mathbf{y} = y] = \Pr[\mathbf{x} = x_i]$, from the last equality it follows, for all $1 \leq i \leq n$, $\Pr[\mathbf{k} = k_i] = \Pr[\mathbf{y} = y]$ which means that all the keys are used with equal probability $\Pr[\mathbf{y} = y]$ and thus $\Pr[\mathbf{k} = k_i] = \frac{1}{|K|}$ for all $k_i \in K$.

($\Rightarrow$)

Assume that $\Pr[\mathbf{x} = x]$ is the probability distribution associated with the random variable over P. Since the keys are uniformly distributed then, $C(k) = |C|$ for all $k \in K$ and thus for all $y \in C$

$$\begin{aligned}
\Pr[\mathbf{y} = y] &= \sum_{\{k:y\in C(k)\}} \Pr[\mathbf{k} = k] \Pr[\mathbf{x} = d_k(y)] \\
&= \sum_{k\in K} \frac{1}{|K|} \Pr[\mathbf{x} = d_k(y)] \\
&= \frac{1}{|K|} \sum_{k\in |K|} \Pr[\mathbf{x} = d_k(y)] \\
&= \frac{1}{|K|} \sum_{x\in |K|} \Pr[\mathbf{x} = x] \\
&= \frac{1}{|K|}.
\end{aligned}$$

Notice that since there is a unique key k such that $e_k(x) = y$ the penultimate equality holds.

Again, to compute $\Pr[\mathbf{x} = x|\mathbf{y} = y]$ first notice that given a key k and plaintext x they uniquely determine y and thus, for all $x \in P$ and $y \in C$,

$$\Pr[\mathbf{y} = y|\mathbf{x} = x] = \Pr[\mathbf{k} = k] = \frac{1}{|K|}.$$

Applying again the Bayes' Theorem 3.1.1 presented in page 31, it follows that:

$$\begin{aligned}
\Pr[\mathbf{x} = x|\mathbf{y} = y] &= \frac{\Pr[\mathbf{x} = x] \cdot \Pr[\mathbf{y} = y|\mathbf{x} = x]}{\Pr[\mathbf{y} = y]} \\
&= \frac{\Pr[\mathbf{x} = x] \cdot 1/|K|}{1/|K|} \\
&= \Pr[\mathbf{x} = x]
\end{aligned}$$

which implies that the cryptographic system S has perfect secrecy. ■

SECTION 6.2

Exercises

Exercise 6.2.1. *Let $\mathcal{S} = (P, C, K, D, E)$ be a cryptographic system such that:*

- $P = C = K = \{0, 1, 2, 3\}$;
- *For a key k, $e_k(x) = (x + k) \pmod 4$;*

Assume also the following probability distributions over P and K:

- $\Pr[k = 0] = 1/3$, $\Pr[k = 1] = 1/3$, $\Pr[k = 2] = 1/6$, $\Pr[k = 3] = 1/6$;
- $\Pr[p = 0] = 1/7$, $\Pr[p = 1] = 1/7$, $\Pr[k = 2] = 3/7$, $\Pr[k = 3] = 1/7$;

Envisage if this system has perfect secrecy or not.

Exercise 6.2.2. *Let $\mathcal{A}$ be the affine cryptographic system defined in Exercise 5.1.6. Show that this cryptographic system achieves perfect secrecy if every key is used with equal probability $1/312$.*

Exercise 6.2.3. *Let $\mathcal{S}$ be a cryptographic system and assume that the system is perfect secure for the uniform distribution over the plaintext set. Show that, this system also have perfect secrecy for any other plaintext probability distribution.*

Exercise 6.2.4. *$\mathcal{S} = (P, C, K, E, D)$ be a cryptographic system that is perfectly secure. Show that if $|P| = |C| = |K|$, not only the keys are equally probable but also every ciphertext is equally probable.*

Exercise 6.2.5 (exercise 2.4 in [KL07])**.** *Prove or refute the following statement:*

For every encryption scheme that is perfectly secret it holds that for every distribution over the plaintext, every $p_1, p_2 \in P$, and every $c \in C$:

$$\Pr[P = x_1 | C = c] = \Pr[P = x_2 | C = c.]$$

Exercise 6.2.6 (exercise 2.5 in [KL07])**.** *Prove or refute the following statement:*

Every encryption scheme for which the size of the key space equals the size of the message space and for which the key is chosen uniformly from the key space, is perfectly secret.

Exercise 6.2.7 (exercise 2.9 in [KL07])**.** *Consider the following definition of perfect secrecy for the encryption of two messages.*

An encryption scheme over a plaintext space P is perfectly-secret for two messages if for all distributions over P, all $p, p' \in P$ and all $c, c' \in C$ such that $\Pr[C = c \wedge C' = c'] > 0$

$$Pr[P = p \wedge P = p' | C = c \wedge C' = c'] = Pr[P = p \wedge P = p']$$

where p and p' are sampled independently from the same distribution over P.

Prove that no encryption scheme satisfies this definition. (Hint: Take $p \neq p'$ but $c = c'$.)

Exercise 6.2.8. *There are other ways of defining perfect secrecy. In [KL07] the authors proposed the following alternative notion of perfect secrecy:*

Consider the an experiment called $Priv_K^{eav}$ which considers the a private-key encryption and an eavesdropping opponent that only has access to a ciphertext c and then tries to determine something about the plaintext that originates c. The experiment is defined for any encryption scheme over plaintext P and for any adversary A and let $Priv_{K,\Pi}^{eav}$ denote an execution of the experiment for a given Π and A defined as follows:

1. The adversary A outputs a pair of messages $p_0, p_1 \in P$.
2. A uniformly random key k is generated, and a uniformly random bit $b \in \{0, 1\}$ is chosen. Then, the ciphertext $c = e_k(p_b)$ is computed and given to A.?
3. A outputs a bit b'.

The output of the experiment are: $Priv_{K,\Pi}^{eav} = 1$ if $b' = b$, and 0 otherwise. **Comment:** *One should think of A as trying to guess the value of b that is chosen in the experiment, and A succeeds when its guess b' is correctly. Observe that it is always possible for A to succeed in the experiment with probability one half by just guessing b' randomly. One says that the encryption is perfect secure if this is the best probability that the adversary has to infer the bit used.*

Prove that this definition is equivalent to the Definition 6.1.1 presented on Page 124.

CHAPTER 7

Block ciphers: DES and AES

In this chapter it will be used the notion of product cipher system to explain and construct the system and the system AES, respectively, Data Encryption Standard and Advanced Encryption Standard. From a block cipher one expects that it should behave like a random permutation to provide diffusion and confusion and in this way prevent the vast majority of known practical attacks. Notice that a truly random permutation would be desirable but its representation for an input of size n would require $\log(2^n!) \approx 2^n$ which is impractical for $n > 20$ and completely unfeasible for $n > 50$. Thus, one need to construct a concise function that behaves like a random one.

The idea of block ciphers is to consider a block of plaintext and transform it at once into a block of a ciphertext. To begin it is explained an iterated cipher that typically consists on an application of a function g, properly named round function, for a specific number of rounds, say Nr, and a key schedule $(K^1, \cdots, k^{Nr})$ that usually is formed from a initial key K and a public algorithm to construct each K^i, the subkey for each round from K. Notice that from the name of the schemes, iterated ciphers, the function g, each takes a code and a key and produces another code, must have the property that the set of the original code must contain the range set of the original code, since it works by taking a a subkey K^r and a current code w^{r-1} and produces the code $w^r = g(w^{r-1}, K^r)$. The initial code, w^0 is the plaintext and w^{Nr} is the ciphertext. Another important feature of g, in order to all decryption, it must be injective

for each key, i.e., when a key is fixed, one can compute $g_k^{-1}(g(w,k)) = w$ for all w.

In the next section it is presented the Substitution-Permutation Networks that are examples of iterated cipher systems that have most of the basic ideas to construct the DES and AES.

• •

SECTION 7.1

Substitution-Permutation Networks

In the last Chapter the notion of secrecy was studied as an important contribute of Shannon to the analyses of security of cipher system. In addition to this contribution, Shannon also introduced the confusion-diffusion paradigm which aims to construct concise random looking permutations with a large block length from many smaller random (or random looking) permutations with small block length. In this case, diffusion means a method of dissipate the statistical structure of the plaintext hiding the redundancy in the ciphertext. Also, confusion is used to make the relation between the simple statistics of ciphertext and the simple description of the key a very complex and involved one. One way of realizing Shannon's principles is the so called substitution permutation network (SPN) cryptographic system.

An SPN cryptographic system consists of Nr rounds, each of them made up of a substitution stage and a permutation stage. The idea of an SPN is that the repetition of confusion and diffusion in each round ensures that any small change in the input will be mixed throughout the entire process and will affect all or almost all the bits of the output. Thus, the effect of small changes in the input have a significant effect on the output similarly to what one would expect to happen with the use of a random permutation.

Letting, l and m be two integers. The plaintext and ciphertext are binary strings of length lm. In the substitution stage is a permutation $\pi_S : \{0,1\}^l \to \{0,1\}^l$ and is used to replace l bits with a different set of l bits. In the permutation stage, like the name suggests is a permutation $\pi_P : \{0,1\}^{lm} \to \{0,1\}^{lm}$ used to permute lm bits.

The idea is to consider a lm bit string corresponding to a plain text x and divide it into m blocks of l-bit strings denoted by $x_{\langle 1\rangle}, \cdots, x_{\langle lm\rangle}$ where $x_{\langle i\rangle} = x_{(i-1)l}, \cdots, x_{il}$.

In each of Nr rounds (except for the last round, which is slightly different), first the subkey k_i is

incorporated via and then perform m substitutions using π_S, followed by a permutation using π_P. The complete description of the cryptographic system is presented in the following.

Cryptographic system 7.1.1 (*Substitution-Permutation Network*)

Let L, m and Nr be three non null integers. Consider also $\pi_S : \{0,1\}^l \rightarrow \{0,1\}^l$ and $\pi_P : \{0,1\}^{lm} \rightarrow \{0,1\}^{lm}$ two permutation. The Substitution-Permutation Network cryptographic system is the tuple (P, C, K, E, D) where $P = C = \{0,l\}^{lm}$, and $K \subset (\{0,1\}^{lm})^{Nr+1}$ consists of all possible key schedules that could be derived from an initial key k using a public key scheduling algorithm. For a key schedule $(k^1, \cdots, k^{Nr+1})$, the encryption function is described as follows.

Algorithm 7.1.2 (SPN encryption function)

Input: *The plaintext x, the permutations π_S and π_S and the key scheduling $(k^1, \cdots, k^{Nr+1})$.*

Output: *The ciphertext y.*

Step 1: $w^0 = x$;

Step 2: *for $r = 1$ to $Nr - 1$ do:*

Step 2.1: $u^r = w^{r-1} \oplus K^r$;

Step 2.2: *for $i = 1$ to m do:*

Step 2.2.1: $v^r_{\langle i \rangle} = \pi_S(u^r_{\langle i \rangle})$

Step 2.3: $w^r = v^r_{\pi_P(1)} \cdots v^r_{\pi_P(lm)}$

Step 3: $u^{Nr} = w^{Nr-1} \oplus K^{Nr}$;

Step 4: *for $i = 1$ to m do:*

Step 4.1: $v^{Nr}_{\langle i \rangle} = \pi_S(u^{Nr}_{\langle i \rangle})$

Step 5: $y = v^{Nr} \oplus k^{Nr+1}$.

Notice that the very first and last operations performed in this SPN are x-ors with subkeys. This is called whitening, and is regarded as a useful way to prevent an attacker from even beginning to carry out an encryption or decryption operation if the key is not known.

Example 7.1.3 (Example 3.1 of [Sti06])**.** *This SPN cryptographic system is represented in schematically represented in next figure:*

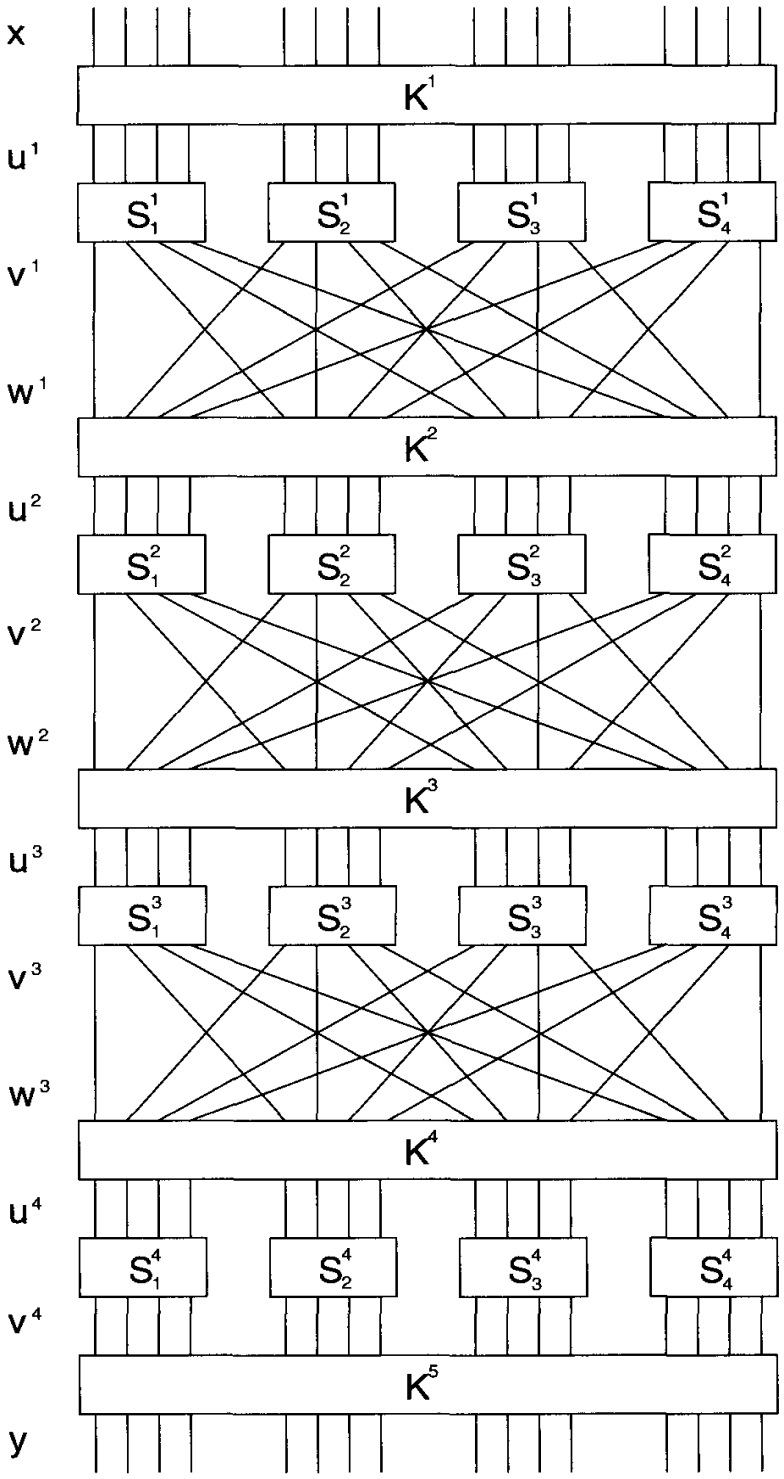


Figure 7.1: Scheme of the SPN of the example

The parameters for the SPN are:

m= *4*

l= *4*

Nr= *4*

Substitution permutation: *It is the following permutation over* $\{0,1\}^4$

x	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$\pi_S(x)$	1	0	1	0	0	1	1	1	0	1	0	1	0	1	0	0
	1	1	1	0	0	1	0	0	0	0	1	1	1	0	0	1
	1	0	0	0	1	1	1	0	1	1	1	0	0	0	0	1
	0	0	1	1	0	1	1	0	1	0	0	0	1	1	0	1

The other permutation: *It is the following permutation over* $\{1,\cdots,16\}$

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\pi_P(x)$	1	5	9	13	2	6	10	14	3	7	11	15	4	8	12	16

Key scheduling: *Given a 32-bit key* $k = 0011\ 1010\ 1001\ 0100\ 1101\ 0110\ 0011\ 1111$ *all the 5 subkeys are constructed by considering 16 consecutive bits of k, beginning with the bit* k_{4r-3}, *where* $1 \leq r \leq 4$ *i.e.,*

$$k^1 = 0011\ 1010\ 1001\ 01001$$

$$k^2 = 1010\ 1001\ 0100\ 1101$$

$$k^3 = 1001\ 0100\ 1101\ 01101$$

$$k^4 = 0100\ 1101\ 0110\ 0011$$

$$k^5 = 1101\ 0110\ 0011\ 1111$$

plaintext: $x = 0010\ 0110\ 1011\ 0111$.

Now it is described roughly the trace of the Algorithm 7.1.2:

Step 1: $w^0 = x = 0010\ 0110\ 1011\ 0111$*;*

Step 2-cicle 1:

$$\begin{aligned}
\text{xor with the subkey} \quad u^1 &= w^0 \oplus k^1 \\
&= 0010\ 0110\ 1011\ 0111 \oplus 0011\ 1010\ 1001\ 0100 \\
&= 0001\ 1100\ 0010\ 0011 \\
\text{applying } \pi_S \quad v^1 &= 0100\ 0101\ 1101\ 0001 \\
\text{applying } \pi_P \quad w^1 &= 0010\ 1110\ 0000\ 0111
\end{aligned}$$

Step 2-cicle 2:

$$\begin{aligned}
\text{xor with the subkey} \quad u^1 &= w^1 \oplus k^2 \\
&= 0001\ 1100\ 0010\ 0011 \oplus 1010\ 1001\ 0100\ 1101 \\
&= 1000\ 0111\ 0100\ 1010 \\
\text{applying } \pi_S \quad v^2 &= 0011\ 1000\ 0010\ 0110 \\
\text{applying } \pi_P \quad w^2 &= 0100\ 0001\ 1011\ 1000
\end{aligned}$$

Step 2-cicle 1:

$$\begin{aligned}
\text{xor with the subkey} \quad u^3 &= w^2 \oplus k^3 \\
&= 0100\ 0001\ 1011\ 1000 \oplus 1001\ 0100\ 1101\ 0110 \\
&= 1101\ 0101\ 0110\ 1110 \\
\text{applying } \pi_S \quad v^3 &= 1001\ 1111\ 1011\ 0000 \\
\text{applying } \pi_P \quad w^3 &= 1110\ 0100\ 0110\ 1110
\end{aligned}$$

Step 3:

$$\begin{aligned}\text{xor with the subkey} \quad u^4 &= w^3 \oplus k^4 \\ &= 1110\ 0100\ 0110\ 1110 \oplus 0100\ 1101\ 0110\ 0011 \\ &= 1010\ 1001\ 0000\ 1101\end{aligned}$$

Step 4:

$$\text{applying } \pi_S \quad v^4 = 0110\ 1010\ 1110\ 1001$$

Step 5:

$$\begin{aligned}\text{xor with the subkey} \quad y &= v^4 \oplus k^5 \\ &= 0110\ 1010\ 1110\ 1001 \oplus 1101\ 0110\ 0011\ 1111 \\ &= 1011\ 1100\ 1101\ 0110\end{aligned}$$

SECTION 7.2

DES – Data Encryption Standards

The Data Encryption Standard or simply DES is the result of efforts of IBM to the call on May 15, 1973, of the National Bureau of Standards, nowadays the National Institute of Standards and Technology (NIST) request in the Federal Register for an encryption algorithm with following properties:

- have a high security level related to a small key used for encryption and decryption;
- be easily understood;
- independent of the algorithm's confidentiality;
- be adaptable and economical;
- be efficient and exportable.

The DES was standardized by the ANSI (American National Standard Institute) under the name of ANSI X3.92, better known as DEA (Data Encryption Algorithm) and was published in the Federal Register of March 17, 1975. In 1976, after consultation of the National Security Agency (NSA) and public discussion was published as an official Federal Information Processing Standard (FIPS) for the United States in January 15, 1977 for "unclassified" applications. The publication of an NSA-approved encryption standard simultaneously resulted in its quick international adoption and widespread academic scrutiny. Also some criticism was presented since there were made statements that the NSA could break any code of DES using a backdoor.

This algorithm was initially expected to be used as a standard for 10-15 years, however, it proved to be much more durable and is reviewed approximately every five years. Its last renewal was in January 1999.

DES is now considered to be insecure mainly because the 56-bit key size being too small. In January, 1999, distributed.net and the Electronic Frontier Foundation collaborated to publicly break a DES key in 22 hours and 15 minutes. There are also some analytical results which demonstrate theoretical weaknesses in the cipher, although they are infeasible to mount in practice. Other than exhaustive key search, the two most important cryptanalytic attacks on DES are differential cryptanalysis and linear cryptanalysis. (For SPNs, theseThe algorithm is for now computationally secure in the form of Triple DES, although there are known theoretical attacks. In recent years, the cipher has been superseded by the Advanced Encryption Standard (AES) and has been withdrawn as a standard by the National Institute of Standards and Technology.

• • •

7.2.1 Description of DES

As it was already mentioned the complete description of the Data Encryption Standard was publish in January 15, 1977 in the Federal Information Processing Standards (FIPS) Publication 46 [oST99] and it can be roughly described by its central block consisting of four operations:

1. key generation;

2. key mixing;

3. substitution table lookup and

4. data mixing between the two halves of the code words.

This central block is iterated for sixteen with only one key generation parameter dependent on iteration number. This single parameter specifies one or two shifts of the circular registers from which the current key is derived.

More concretely the DES cipher is a 16-round of a Feistel network of a block length of 64 bits and a key length of 56 bits (from which a 48-bit sub-key is generated for each round). The Feistel network begins by dividing the state into two halves which are usually denoted by L and R. The mixing function on each state takes L^i, R^i and k^i and outputs the pair (L^{i+1}, R^{i+1}) where $L^{i+1} = R^i$ and $R^{i+1} = R^i \oplus f(R^i, k^i)$ for some specific function f. Before the 16 rounds the plaintext is prepared by applying to it a fixed initial permutation, called IP and then divide it into two halves. Also, at the end of 16 rounds of encryption, it is applied to $L^{16}R^{16}$ the inverse permutation IP^{-1}. From the point of view of security the use of the permutation IP is irrelevant. An illustrative description of DES is presented in the figure 7.2.

The description of function $f : \{0,1\}^{32} \times \{0,1\}^{48} \to \{0,1\}^{32}$, given R and k is:

- R is expanded to the following 48 bits string

$$\begin{aligned} E(R) = (\quad & R_{32}, R_1\ , R_2\ , R_3\ , R_4\ , R_5\ , R_4\ , R_5\ , R_6\ , R_7\ , R_8\ , R_9\ , \\ & R_8\ , R_9\ , R_{10}, R_{11}, R_{12}, R_{13}, R_{12}, R_{13}, R_{14}, R_{15}, R_{16}, R_{17}, \\ & R_{16}, R_{17}, R_{18}, R_{19}, R_{20}, R_{21}, R_{20}, R_{21}, R_{22}, R_{23}, R_{24}, R_{25}, \\ & R_{24}, R_{25}, R_{26}, R_{27}, R_{28}, R_{29}, R_{28}, R_{29}, R_{30}, R_{31}, R_{32}, R_1 \quad); \end{aligned}$$

- Compute $B = E(R) \oplus k$ and divide B into eight 6-bit strings, i.e., $B = B_1B_2B_3B_4B_5B_6B_7B_8$;

- Using the eight S-boxes presented in Table 7.1 where each S-box $S_i : \{0,1\}^6 \to \{0,1\}^4$ is traditionally depicted as a 4 x 16 array with entries between 0 and 15. Given the block $B_j = b_1b_2b_3b_4b_5b_6$ one compute the value $C_j = S_j(B_j)$ is the binary representation of the element in row with decimal number corresponding to the binary number b_1b_6 and column with decimal number corresponding to the binary number $b_2b_3b_4b_5$.

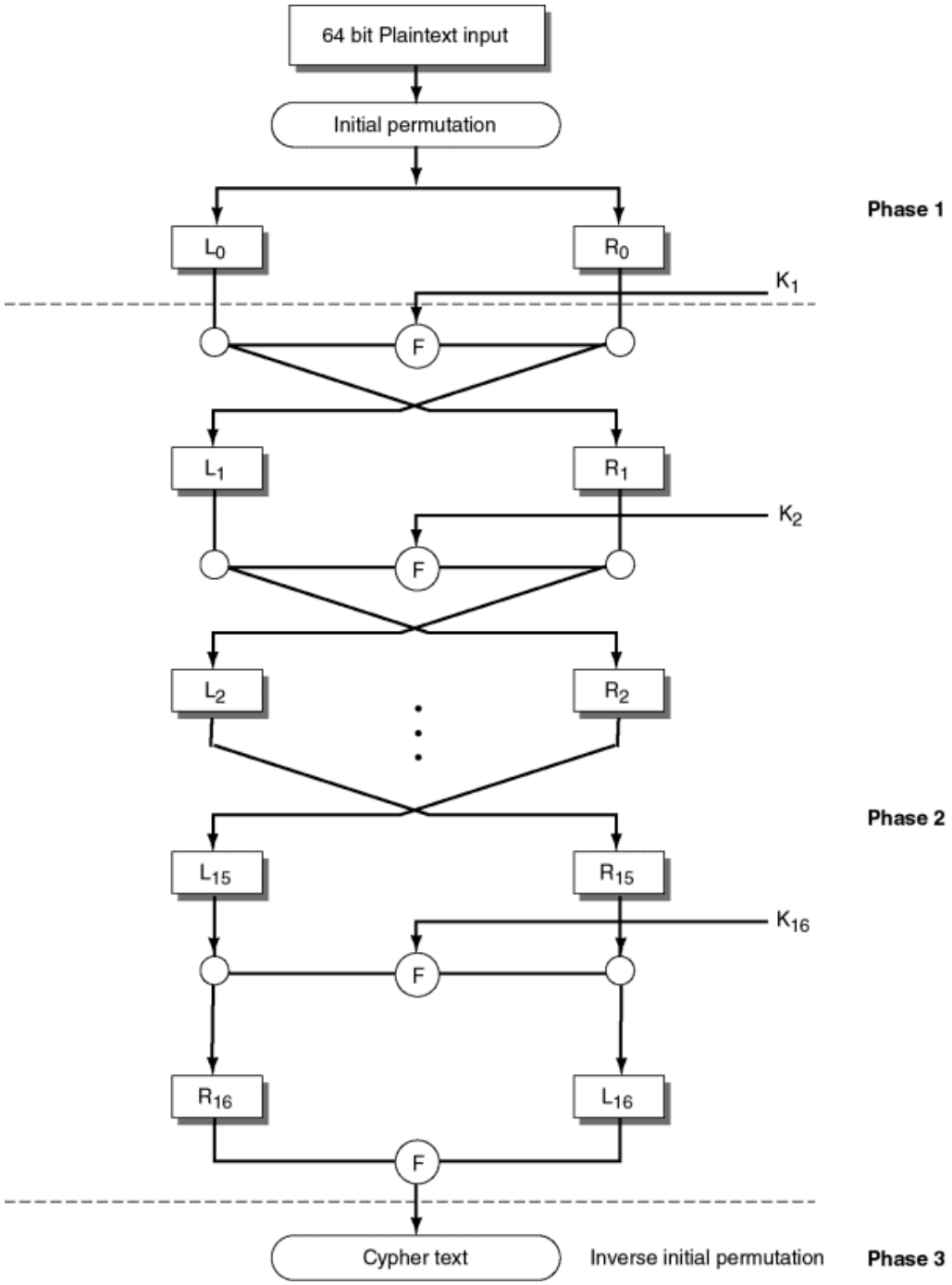


Figure 7.2: Description of DES

S_1

$b_2b_3b_4b_5$																b_1b_2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	00
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	01
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	10
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	11

S_2

$b_2b_3b_4b_5$																b_1b_2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	00
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	01
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	10
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	11

S_3

$b_2b_3b_4b_5$																b_1b_2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	00
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	01
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7	10
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	11

S_4

$b_2b_3b_4b_5$																b_1b_2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	00
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	01
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	10
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	11

S_5

$b_2b_3b_4b_5$																b_1b_2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	00
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	01
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	10
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3	11

S_6

$b_2b_3b_4b_5$																b_1b_2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11	00
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	01
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6	10
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13	11

S_7

$b_2b_3b_4b_5$																b_1b_2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	00
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	01
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	10
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12	11

S_8

$b_2b_3b_4b_5$																b_1b_2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	00
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	01
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	10
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11	11

Table 7.1: The eight DES S – boxes

- The value of $f(R,k)$ is then obtained by permuting the bit string $C = C_1C_2C_3C_4C_5C_6C_7C_8$ by the following permutation P:

$$\begin{aligned} P(C) = (\quad & c_{16}, c_7\ , c_{20}, c_{21}, c_{29}, c_{12}, c_{28}, c_{17}, \\ & c_1\ , c_{15}, c_{23}, c_{26}, c_5\ , c_{18}, c_{31}, c_{10}, \\ & c_2\ , c_8\ , c_{24}, c_{14}, c_{32}, c_{27}, c_3\ , c_9 \\ & c_{19}, c_{13}, c_{30}, c_6, c_{22}, c_{11}, c_4, c_{25} \qquad) \end{aligned}$$

• • •

7.2.2 Breaking down the DES

Nowadays the DES is no longer used as a standard of NIST as it was already mentioned earlier. After almost 30 years of intensive study, the best known practical attack on DES is still just an exhaustive search through its key space. Although some theoretical attacks are well known and explain/support the revocation of the its use. This attacks are subject of study in this section.

Typical block ciphers like DES are involve constructions that are not that simple, but in any case, complicated construction does not necessarily mean difficulty on the cryptographic analyzes and consequently unbreakable. On the contrary, it is very difficult to construct a secure block cipher and surprisingly easy to construct a trivially insecure one.

• • •

7.2.2.1 Linear Approximation of S-boxes

The idea of linear cryptanalysis is to find affine approximations to the action of the cipher. This type of attacks were discovered by Mitsuru Matsui, who first applied the technique to the FEAL cipher and subsequently mounted an attack to the Data Encryption Standard (DES), eventually leading to the first experimental cryptanalysis of the cipher reported in the open community. The attack on DES is not so practical since a fair number of plaintexts must be known.

In this attack first some linear equations relating plaintext, ciphertext and key bits with fairly large bias are constructed and then these linear equations are used together with known plaintext/ciphertext pairs to derive the key. The construction of the linear approximations varies

x	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$\pi_S(x)$	1	0	1	0	0	1	1	1	0	1	0	1	0	1	0	0
	1	1	1	0	0	1	0	0	0	0	1	1	1	0	0	1
	1	0	0	0	1	1	1	0	1	1	1	0	0	0	0	1
	0	0	1	1	0	1	1	0	1	0	0	0	1	1	0	1
	1	1	1	1	0	1	1	1	1	0	1	1	1	1	1	1
	$\mathbf{X_3} \oplus \mathbf{X_4} \oplus \mathbf{Y_1} \oplus \mathbf{Y_2}$															

Table 7.2: Computation of the random variable $\mathbf{X_3} \oplus \mathbf{X_4} \oplus \mathbf{Y_1} \oplus \mathbf{Y_2}$

with the cipher used but in the case of the substitution-permutation network, the analysis is concentrated primarily on the S-boxes, the only part of the system that is not linear (i.e. the involved operation in an S-box cannot be encoded in a linear equation) and for a small S-boxes, it is feasible to enumerate every possible linear equation which relate the input and output bits of the boxes and thus determining their bias. For sake of presentation and understanding this attack will be explained continuing the Example 7.1.3.

Example 7.2.1 (continuing over Example 7.1.3)**.** *The S-box used in the example is a permutation* $\pi_S : \{0,1\}^4 \to \{0,1\}^4$ *and choosing* $\mathbf{X_1}$, $\mathbf{X_2}$, $\mathbf{X_3}$ *and* $\mathbf{X_4}$ *independently and uniformly at random one can compute the the bias of the random variables* $\mathbf{X_3} \oplus \mathbf{X_4} \oplus \mathbf{Y_1} \oplus \mathbf{Y_2}$*. See the Table 7.2 for details.*

$$\varepsilon_{\mathbf{X_3} \oplus \mathbf{X_4} \oplus \mathbf{Y_1} \oplus \mathbf{Y_2}} = \Pr[\mathbf{X_3} \oplus \mathbf{X_4} \oplus \mathbf{Y_1} \oplus \mathbf{Y_2} = 0] - \frac{1}{2} = \frac{2}{16} - \frac{1}{2} = -\frac{3}{8}$$

To describe the attack method, i.e, of finding a set of linear approximations of S-boxes that can be used to derive a linear approximation of the entire SPN it will be used again Example 7.1.3.

In Figure 7.3 it is schematically presented the used approximation where the arrowed lines are the random variables which will be involved in linear approximations. The labeled S-boxes are the ones used in these approximations usually called the active S- boxes. There are four active S-boxes along the process giving access to the following random variables T:

1. In S_2^1, $\mathbf{T}_1 = \mathbf{U}_5^1 \oplus \mathbf{U}_7^1 \oplus \mathbf{U}_8^1 \oplus \mathbf{V}_6^1$, with bias $1/4$;
2. In S_2^2, $\mathbf{T}_2 = \mathbf{U}_6^2 \oplus \mathbf{V}_6^2 \oplus \mathbf{V}_8^2$, with bias $-1/4$;
3. In S_2^3, $\mathbf{T}_3 = \mathbf{U}_6^3 \oplus \mathbf{V}_6^3 \oplus \mathbf{V}_8^3$, with bias $-1/4$;
4. In S_2^4, $\mathbf{T}_4 = \mathbf{U}_{14}^3 \oplus \mathbf{V}_{14}^3 \oplus \mathbf{V}_{16}^3$, with bias $-1/4$;

Assuming that $\mathbf{T}_1; \mathbf{T}_2, \mathbf{T}_3, \mathbf{T}_4$ are independent then by the Piling-up Lemma 3.1.4 in page 33, $\varepsilon_{\mathbf{T}_\oplus \mathbf{T}_2 \oplus \mathbf{T}_3 \oplus \mathbf{T}_4} = -\frac{1}{32}$. Notice that:

$$\begin{aligned}
\mathbf{T}_1 = \mathbf{U}_5^1 \oplus \mathbf{U}_7^1 \oplus \mathbf{U}_8^1 \oplus \mathbf{V}_6^1 &= \mathbf{X}_5 \oplus \mathbf{k}_5^1 \oplus \mathbf{X}_7 \oplus \mathbf{k}_7^1 \oplus \mathbf{X}_8 \oplus \mathbf{k}_8^1 \oplus \mathbf{V}_6^1 \\
\mathbf{T}_2 = \mathbf{U}_6^2 \oplus \mathbf{V}_6^2 \oplus \mathbf{V}_8^2 &= \mathbf{V}_6^1 \oplus \mathbf{k}_6^2 \oplus \mathbf{V}_6^2 \oplus \mathbf{V}_8^2 \\
\mathbf{T}_3 = \mathbf{U}_6^3 \oplus \mathbf{V}_6^3 \oplus \mathbf{V}_8^3 &= \mathbf{V}_6^2 \oplus \mathbf{k}_6^3 \oplus \mathbf{V}_6^3 \oplus \mathbf{V}_8^3 \\
\mathbf{T}_4 = \mathbf{U}_{14}^3 \oplus \mathbf{V}_{14}^3 \oplus \mathbf{V}_{16}^3 &= \mathbf{V}_8^2 \oplus \mathbf{k}_{14}^3 \oplus \mathbf{V}_{14}^3 \oplus \mathbf{V}_{16}^3
\end{aligned}$$

Thus:

$$\begin{aligned}
\mathbf{T}_\oplus \mathbf{T}_2 \oplus \mathbf{T}_3 \oplus \mathbf{T}_4 = \; & \mathbf{X}_5 \oplus \mathbf{k}_5^1 \oplus \mathbf{X}_7 \oplus \mathbf{k}_7^1 \oplus \mathbf{X}_8 \oplus \mathbf{k}_8^1 \oplus \mathbf{V}_6^1 \oplus \mathbf{V}_6^1 \oplus \mathbf{k}_6^2 \oplus \mathbf{V}_6^2 \oplus \mathbf{V}_8^2 \\
& \oplus \mathbf{V}_6^2 \oplus \mathbf{k}_6^3 \oplus \mathbf{V}_6^3 \oplus \mathbf{V}_8^3 \oplus \mathbf{V}_8^2 \oplus \mathbf{k}_{14}^3 \\
= \; & \mathbf{X}_5 \oplus \mathbf{X}_7 \oplus \mathbf{X}_8 \oplus \mathbf{V}_6^3 \oplus \mathbf{V}_8^3 \oplus \mathbf{V}_{14}^3 \oplus \mathbf{V}_{16}^3 \oplus \mathbf{k}_5^1 \oplus \mathbf{k}_7^1 \oplus \mathbf{k}_8^1 \oplus \mathbf{k}_6^2 \oplus \mathbf{k}_6^3 \oplus \oplus \mathbf{k}_{14}^3 \\
= \; & \mathbf{X}_5 \oplus \mathbf{X}_7 \oplus \mathbf{X}_8 \oplus \mathbf{U}_6^3 \oplus \mathbf{k}_6^4 \oplus \mathbf{U}_{14}^4 \oplus \mathbf{k}_{16}^4 \oplus \mathbf{U}_8^4 \oplus \mathbf{k}_8^4 \oplus \mathbf{U}_{16}^4 \oplus \mathbf{k}_{16}^4 \\
& \oplus \mathbf{k}_5^1 \oplus \mathbf{k}_7^1 \oplus \mathbf{k}_8^1 \oplus \mathbf{k}_6^2 \oplus \mathbf{k}_6^3 \oplus \mathbf{k}_{14}^3 \\
= \; & \mathbf{X}_5 \oplus \mathbf{X}_7 \oplus \mathbf{X}_8 \oplus \mathbf{U}_6^3 \oplus \mathbf{U}_{14}^4 \oplus \mathbf{U}_8^4 \oplus \mathbf{U}_{16}^4 \\
& \oplus \mathbf{k}_5^1 \oplus \mathbf{k}_7^1 \oplus \mathbf{k}_8^1 \oplus \mathbf{k}_6^2 \oplus \mathbf{k}_6^3 \oplus \mathbf{k}_{14}^3 \oplus \mathbf{k}_8^4 \oplus \mathbf{k}_{14}^4 \oplus \mathbf{k}_{16}^4
\end{aligned}$$

Notice that if the key is fixed then the random variable $\mathbf{k}_5^1 \oplus \mathbf{k}_7^1 \oplus \mathbf{k}_8^1 \oplus \mathbf{k}_6^2 \oplus \mathbf{k}_6^3 \oplus \mathbf{k}_{14}^3 \oplus \mathbf{k}_8^4 \oplus \mathbf{k}_{16}^4 \oplus \mathbf{k}_{16}^4$ is zero or one and the bias of $\mathbf{X}_5 \oplus \mathbf{X}_7 \oplus \mathbf{X}_8 \oplus \mathbf{U}_6^3 \oplus \mathbf{U}_{14}^4 \oplus \mathbf{U}_8^4 \oplus \mathbf{U}_{16}^4$ is equal to the bias of $\mathbf{T}_\oplus \mathbf{T}_2 \oplus \mathbf{T}_3 \oplus \mathbf{T}_4$ in modulo which is far from 0 and hence allows an attack. With this attack one can recover the 8 key bits of $k_{\langle 2 \rangle}^5$ and $k_{\langle 4 \rangle}^5$ resulting from the x-ores with the outputs of the last active boxes, namely S_2^4 and S_4^4. Thus, if one has access to T plaintext/ciphertext pairs from

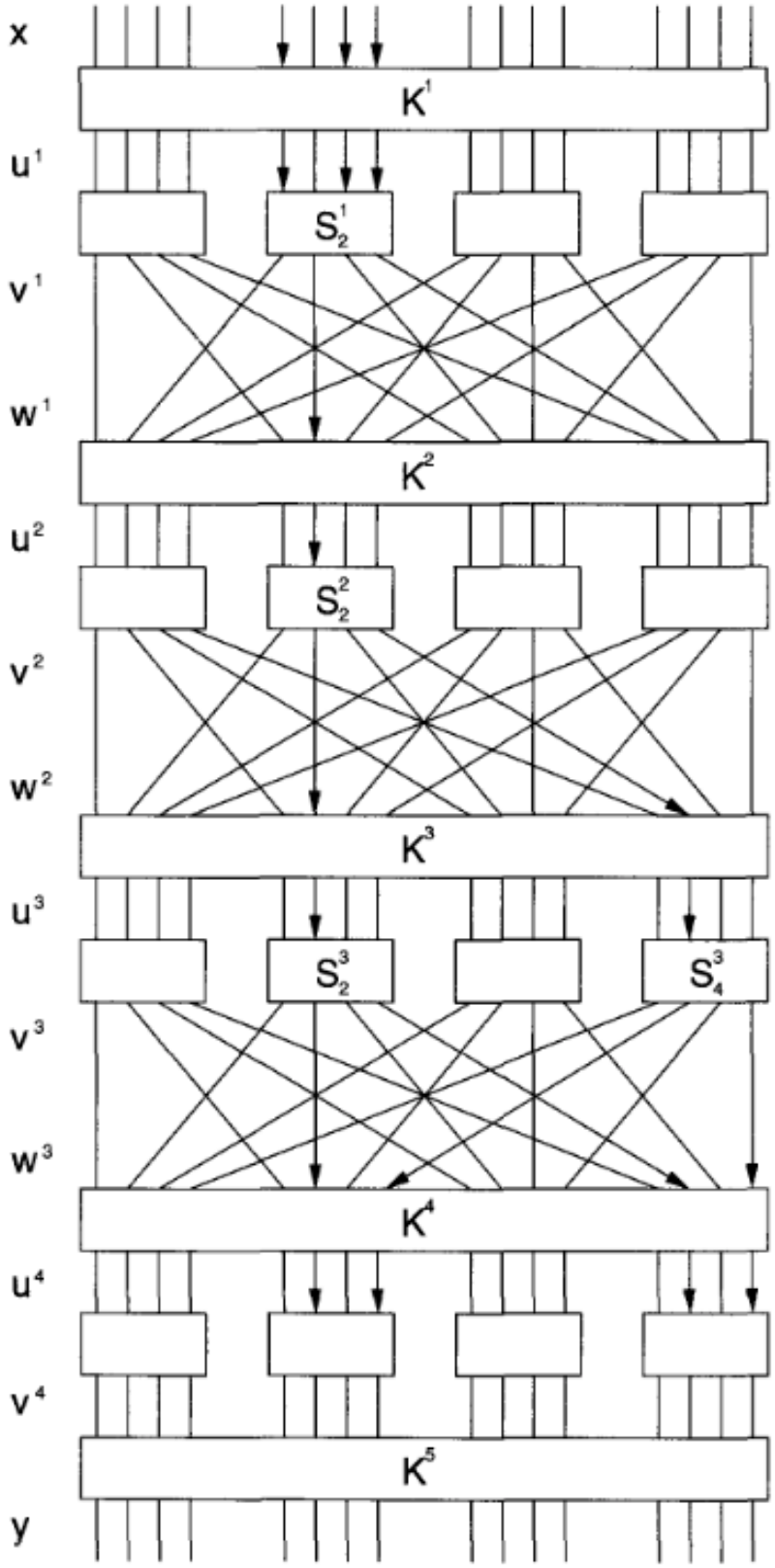


Figure 7.3: Linear approximation to a SPN

the partial decryption of the ciphertext y to obtain $u^4_{\langle 2\rangle}$ and $u^4_{\langle 4\rangle}$ and hence one can compute the value $x_5 \oplus x_7 \oplus x_8 \oplus u^4_6 \oplus u^4_8 \oplus u^4_{14} \oplus u^4_{16}$. Keeping an array of counters indexed by the 256 possible candidate subkeys, and increment the counter corresponding to a particular subkey has the value 0 and in the end it is expected that most counters will have a value close to $T/2$, but the counter for the correct candidate subkey will have a value that is close to $T/2 \pm T/32$ which allows the identification of eight subkey bits. These ideas are expressed in the next algorithm designed for this particular linear attack.

Algorithm 7.2.2 (*Linear attack*)

Input *A number T, a set $\mathcal{T}$ of T plaintext/ciphertexts and the inverse of the substitution permutation π^{-1}.*

Output *The eight bits of the key.*

Step 1 **For** (L_1, L_2) **from** $(0,0)$ **to**(F,F) *(the number are in hexadecimal base)* **do** *:*

Step 1.1 $Count(L_1, L_2) = 0$*;*

Step 2 *For all* $(x, y) \in \mathcal{T}$ *do:*

Step 2.1 **For** (L_1, L_2) **from** $(0,0)$ **to**(F,F) *do:*

Step 2.1.1 $v^4_{\langle 2\rangle} = L_1 \oplus y_{\langle 2\rangle}$*;*

Step 2.1.2 $v^4_{\langle 4\rangle} = L_2 \oplus y_{\langle 4\rangle}$*;*

Step 2.1.3 $u^4_{\langle 2\rangle} = \pi^{-1}(v^4_{\langle 2\rangle})$*;*

Step 2.1.4 $u^4_{\langle 4\rangle} = \pi^{-1}(v^4_{\langle 4\rangle})$*;*

Step 2.1.5 $z = x_5 \oplus x_7 \oplus x_8 \oplus u^4_6 \oplus u^4_8 \oplus u^4_{14} \oplus u^4_{16}$*;*

Step 2.1.6 **If** $z = 0$ **then** $Count(L_1, L_2) = Count(L_1, L_2) + 1$*;*

Step 3 $max = -1$*;*

Step 4 **For** (L_1, L_2) **from** $(0,0)$ **to**(F,F) **do** *:*

Step 4.1 $Count(L_1, L_2) = |Count(L_1, L_2) - T/2|$*;*

Step 4.2 **If** $Count(L_1, L_2) > max$ **then** $max = Count(L_1, L_2)$ *and* $maxkey = (L_1, L_2)$.

Step 5 *Output (maxkey)*

In general, a linear attack based on a linear approximation having bias equal to ε will be successful if the number of plaintext/ciphertext pairs is approximately $c\varepsilon^{-2}$, for some small constant c.

• • •

7.2.2.2 The differential attack

In this technique (also consisting of plaintext/ciphertext attacks) was developed by Biham and Shamir in 93 and the basic idea behind it is to keep track of specific differences in the input that lead to specific differences in the output with noticeable probability when compared with a random permutation. In a not looking random block cipher there may be differentials that appear with significantly higher probability and those are the ones that are attackable. So one will look for those differences via a careful analysis of the block cipher itself. The point of differential cryptanalysis is to use many differentials, to recover the secret key. Similar to the linear attack the ability to isolate parts of a key is crucial and enables the attacker to obtain the key in time less than a brute force search, however, since chosen plaintexts/ciphertexts are required the practicality of this attack is questionable (but real).

Differential cryptanalysis involves than the comparison of two xor of two inputs with the xor of their corresponding outputs. Assume that the attacker has access to a large number of tuples (x, x^*, y, y^*), where $x' = x \oplus x^*$ is a fixed value and all of then are encrypted using the same unknown key, k. The trick is for each of these tuples, one decrypt the respectively ciphertexts y and y^*, using all possible candidate keys for the last round of the cipher and analyze the values of certain state bits, and determine if their x-or has a certain value (namely, the most likely value for the given input x-or). Whenever it does, the counter for that particular key incremented. At the end of this process, the candidate key that has the highest frequency count contains the correct values for these key bits.

Definition 7.2.1

For any $x' \in \{0,1\}^m$ define the set $\Delta(x')$ of all 2^m ordered pairs (x, x^*) of input with x-or equal to x', i.e., $\Delta(x') = \{(x, x^*) \in \{0,1\}^{2m} : x \oplus x^* = x'\}$.

Given a pair in $\Delta(x')$, one can compute the output x-or of their encryption, i.e., $y \oplus y^*$ and store the resulting distribution. If that distribution is not uniform among the 2^m possible values then one can successfully implement a differential attack.

Example 7.2.3 (continuing over Example 7.1.3)**.** *Consider $x' = 1011$. The $\Delta(x')$ pairs and the respective ciphertexts and their x-or are described in the next table:*

x	x^*	y	y'	$y \oplus y^*$
0000	1011	1110	1100	0010
0001	1010	0100	0110	0010
0010	1001	1101	1010	0111
0011	1000	0001	0011	0010
0100	1111	0010	0111	0101
0101	1110	1111	0000	1111
0110	1101	1011	1001	0010
0111	1100	1000	0101	1101

x	x^*	y	y'	$y \oplus y^*$
1000	0011	0011	0001	0010
1001	0010	1010	1101	0111
1010	0001	0110	0100	0010
1011	0000	1100	1110	0010
1100	0111	0101	1000	1101
1101	0110	1001	1011	0010
1110	0101	0000	1111	1111
1111	0100	0111	0010	0101

and the distribution of $y \oplus y^*$ is:

$y \oplus y^*$	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Fr_i	0	0	8	0	0	2	0	2	0	0	0	0	0	2	0	2

Table 7.3: Distribution of $y \oplus y^*$

For each pair (x', y'), called the differential, let $N_D(x', y')$ be the set of pairs $(x, x^*) \in \Delta(x')$ such that $\pi_S(x) \oplus \pi_S(x^*) = y'$ and define $R_P = \dfrac{N_D(x', y')}{2^m}$ the ratio of propagation which basically is the

conditional probability $\Pr[z \oplus z^* = y' | w \oplus w^* = x'] = R_p(x', y')$. *For sake of understanding, consider the particular differential trail of Figure 7.2.3 using the propagation ratios of differentials:*

- S_2^1, $R_p(1011, 0010) = 1/2$;
- S_2^3, $R_p(0100, 0110) = 3/8$;
- S_3^2, $R_p(0010, 0101) = 3/8$;
- S_3^3, $R_p(0010, 0101) = 3/8$;

that combined lead to a propagation ratio for a differential trail of the first three rounds of the SPN (assuming that the ratios are independent):

$$R_p(0000\ 1011\ 0000\ 0000, 0000\ 0101\ 0101\ 0000) = \frac{1}{2} \cdot \left(\frac{3}{8}\right)^3 = \frac{27}{1024}$$

which means that $x' = 0000\ 1011\ 0000\ 0000$ *will lead to* $(v_3)' = 0000\ 0101\ 0101\ 0000$ *with probability* $\frac{27}{1024}$. *But from the fact that* $(v_3)' = 0000\ 0101\ 0101\ 0000$ *is transformed into the value* $(u_4)' = 0000\ 0110\ 0000\ 0110$ *and thus* $x' = 0000\ 1011\ 0000\ 0000$, *in fact, will lead to* $(u_4)' = 0000\ 0101\ 0101\ 0000$ *with probability* $\frac{27}{1024}$. *Note that* $(u_4)'$ *is the x-or of two inputs to the last round of S-boxes. So the following algorithm, for this particular example, based on the informal description at the beginning of this section, will lead to the key:*

Algorithm 7.2.4 (Differential attack)

Input *A number* T, *a set* $\mathcal{T}$ *of* T *plaintext/ciphertexts and the inverse of the substitution permutation* π^{-1}.

Output *The eight bits of the key.*

Step 1 **For** (L_1, L_2) **from** $(0, 0)$ *to* (F, F) *(the number are in hexadecimal base)* **do** *:*

Step 1.1 $Count(L_1, L_2) = 0$;

Step 2 *For all* $(x, y, x^*, y^*) \in \mathcal{T}$ *do:*

Step 2.1 **If** $y_{\langle 1 \rangle} = (y_{\langle 1 \rangle})^*$ *and* $y_{\langle 3 \rangle} = (y_{\langle 3 \rangle})^*$ **then**

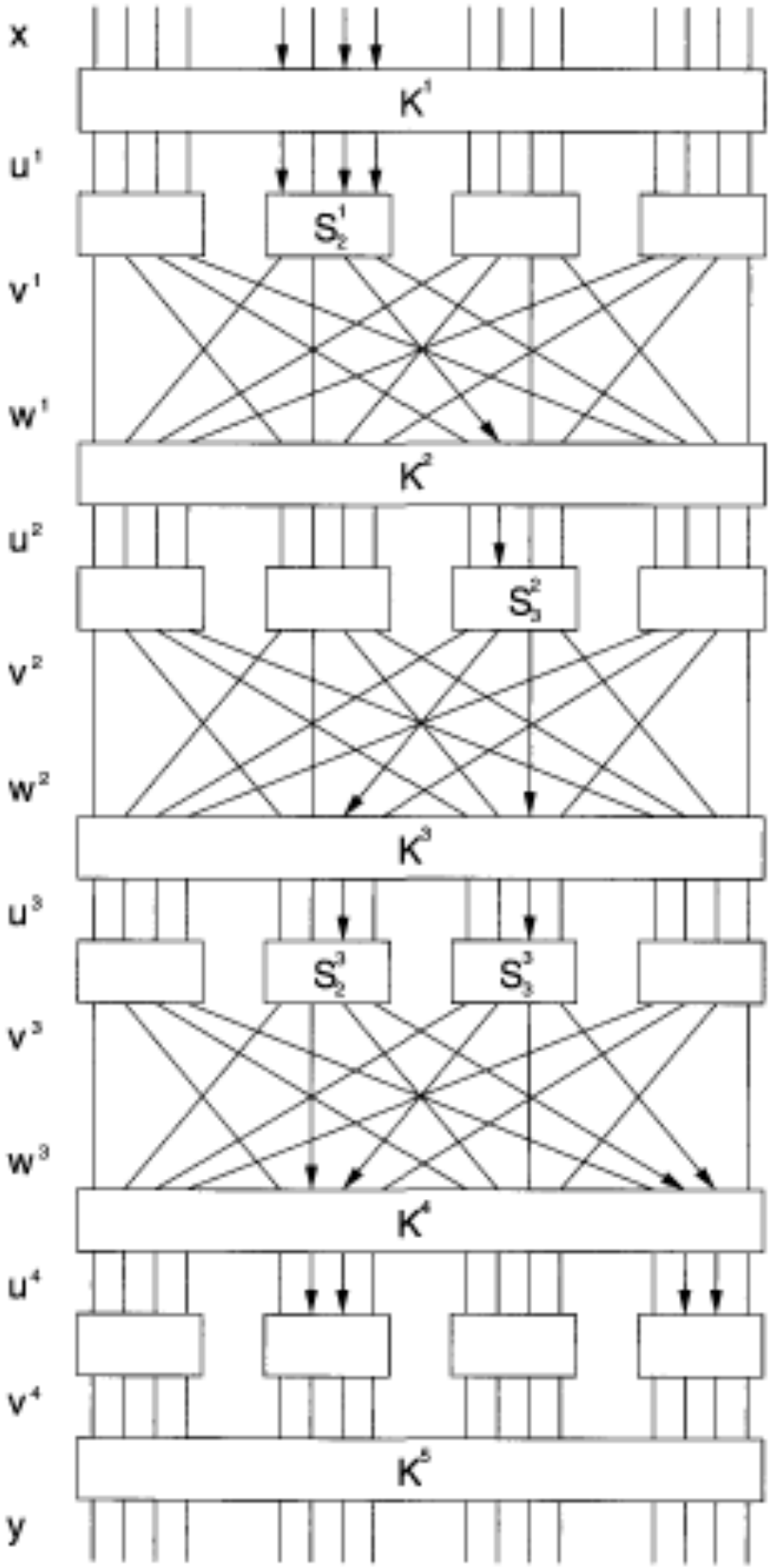


Figure 7.4: Differential attack for a SPN

Step 2.2.1 **For** (L_1, L_2) **from** $(0,0)$ **to**(F,F) **do** *:*

Step 2.2.1.1 $v^4_{\langle 2\rangle} = L_1 \oplus y_{\langle 2\rangle}$

Step 2.2.1.2 $v^4_{\langle 4\rangle} = L_2 \oplus y_{\langle 4\rangle}$

Step 2.2.1.3 $u^4_{\langle 2\rangle} = \pi^{-1}(v^4_{\langle 2\rangle})$

Step 2.2.1.4 $u^4_{\langle 4\rangle} = \pi^{-1}(v^4_{\langle 4\rangle})$

Step 2.2.1.5 $(v^4_{\langle 2\rangle})^* = L_1 \oplus (y_{\langle 2\rangle})^*$

Step 2.2.1.6 $(v^4_{\langle 4\rangle})^* = L_2 \oplus (y_{\langle 4\rangle})^*$

Step 2.2.1.7 $(u^4_{\langle 2\rangle})^* = \pi^{-1}(v^4_{\langle 2\rangle})^*$

Step 2.2.1.8 $(u^4_{\langle 4\rangle})^* = \pi^{-1}(v^4_{\langle 4\rangle})^*$*;*

Step 2.2.1.9 $u^4_{\langle 2\rangle})' = u^4_{\langle 2\rangle} \oplus (u^4_{\langle 2\rangle})^*$*;*

Step 2.2.1.10 $(u^4_{\langle 4\rangle})' = u^4_{\langle 4\rangle} \oplus (u^4_{\langle 4\rangle})^*$*;*

Step 2.2 **If** $(u^4_{\langle 4\rangle})' = 0110$ *and* $(u^4_{\langle 4\rangle})' = 0110$ **then** $Count(L_1, L_2) = Count(L_1, L_2) + 1$*;*

Step 3 $max = -1$*;*

Step 4 **For** (L_1, L_2) **from** $(0,0)$ **to**(F,F) **do** *:*

Step 4.1 **If** $Count(L_1, L_2) > max$ **then**

Step 4.1.1 $max = Count(L_1, L_2)$*;*

Step 4.1.2 $maxkey = (L_1, L_2)$*;*

Step 5 *Output (maxkey)*

• • •

7.2.2.3 Analytic attack

To be written

SECTION 7.3

AES – Advanced Encryption Standards

Since one can "possibly" break DES with sufficiently large pair of plaintext/ciphertext and the need of secretly encrypt messages were the main reasons for NIST to began in 1997 the substitution of DES and that would be called Advanced Encryption Standards (AES) that can offer a higher level of security. AES candidates were required to:

1. support a symmetric block cipher supporting multiple key lengths (128, 192, 256);

2. have the algorithm publicly available, free to use, and able to run efficiently in both hardware and software;

3. ensure that the only known approach to decrypt a message is for an intruder to try every possible key;

There were presented 21 proposals, 15 of them meeting all the necessary criteria that were announced at the "First AES Candidate Conference". In August 1999, five of the candidates were chosen as finalists: MARS, RC6, Rijndael, Serpent and Twofish. On October 2, 2000, Rijndael was selected to be the Advanced Encryption Standard and in February 2001, the NIST published a draft for the AES for public review and comment.

AES was adopted as a standard on November 2001, and it was published as FIPS 197 in the Federal Register on December 4, 2001.

The great thing about the all process for the AES was it openness and its international feedback which provided the opportunity for public discussion and analysis of the candidates. The selected AES, was invented by two Belgian researchers, Daemen and Rijmen.

...

7.3.1 Description of AES

AES operates on blocks of size 128 bits and uses a key size of 128, 192, or 256 bits, whereas Rijndael can be specified with block and key sizes in any multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits.

AES operates on a 4×4 array of bytes (i.e., 32×32 bits), termed the $\mathtt{state}$ and almost all calculations are done in a special finite field. For each key length the cipher is specified by a number of repetitions of transformation rounds that convert the input plaintext into the final output of ciphertext.

Each round consists of several processing steps, including one that depends on the encryption key. A high level description of the AES is presented in the next algorithm:

Algorithm 7.3.1

Step 1 $\mathbb{K}\mathrm{eyExpansion}$

Step 2 *Initial round:*

Step 2.1 $\mathbb{A}\mathrm{ddRoundKey}$*;*

Step 3 *Rounds:*

Step 3.1 $\mathbb{S}\mathrm{ubBytes}$*;*

Step 3.2 $\mathbb{S}\mathrm{hiftRows}$*;*

Step 3.3 $\mathbb{M}\mathrm{ixColumns}$*;*

Step 3.4 $\mathbb{A}\mathrm{ddRoundKey}$*;*

Step 4 *Final round:*

Step 4.1 $\mathbb{S}\mathrm{ubBytes}$*;*

Step 4.2 $\mathbb{S}\mathrm{hiftRows}$*;*

Step 4.3 $\mathbb{A}\mathrm{ddRoundKey}$*;*

Notice that all the operations are important for the security of AES. In fact, the operation $\mathbb{SubBytes}$ achieves non-linearity which is important to prevent differential cryptanalysis, the operations $\mathbb{ShiftRows}$ and $\mathbb{MixColumns}$ are used to mix up bits to achieve a more uniform distribution of ciphertexts and the operation $\mathbb{AddRoundKey}$ provides the necessary secret randomness.

Let x be the plaintext to be ciphered with the AES, consisting of 16 bytes. Usually the content of a byte will be represented in hexadecimal (in the case composed of two digits). The initial state S (a word of 16 bytes) is prepared as follows:

$$\begin{array}{|c|c|c|c|} \hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline x_0 & x_4 & x_8 & x_{12} \\ \hline x_1 & x_5 & x_9 & x_{13} \\ \hline x_2 & x_6 & x_{10} & x_{14} \\ \hline x_3 & x_7 & x_{11} & x_{15} \\ \hline \end{array}$$

Now it described the process $\mathbb{Subbyte}$ which consists of replace each byte by another according to the Table 7.4.

So for a byte XY the result is $\pi_S(XY)$ where x is looked in column and Y in the line. Notice that Π_S contrarily to what happens in DES, it can be represented algebraically. In fact each byte represents a polynomial in $\mathbb{Z}_2[x]/(x^8 + x^4 + x^3 + x + 1)$, where the byte $a_7a_6a_5a_4a_3a_2a_1a_0$ represents the polynomial $p(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$. The $\mathbb{Subbyte}$ corresponds to invert $p(x)$ in $\mathbb{Z}_2[x]/(x^8 + x^4 + x^3 + x + 1)$, say $q(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$ and the coefficients of each of the bits of the result of this operation say $r(x) = c_7x^7 + c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$ are obtained by the formula $c_i = b_i + b_{i+4} + b_{i+5} + b_{i+6} + b_{i+7} + d_i \pmod 2$ where $d = 01100011$.

The $\mathbb{ShiftRow}$ is the operation realized by the following diagram:

$$\begin{array}{|c|c|c|c|} \hline s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ \hline s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ \hline s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ \hline s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ \hline s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ \hline s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \\ \hline \end{array}$$

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Table 7.4: S-Box of the encryption substitution

where $s' = \mathbb{ShiftRow}(s)$.

The $\mathbb{MixColumns}$ takes each column of State and replaces it by a new column which is a product of that column by a certain matrix of elements of the field $\mathbb{F}_{2^8}$.

To this end it remains to discuss the key schedule. For the case of a 128 bits key is been used (for the10-round case), 11 round keys, each consisting of 16 bytes are needed and the key scheduling algorithm is word-oriented of 4 bytes each and their concatenation is called the expanded key. So it consists of 44 words that herein are denoted by $w[0], ..., w[43]$. The expanded key is constructed using the operation $\mathbb{KeyExpansion}$ that basically consists on the application of a cyclic shift and the application of a $S - Box$. The first is called $\mathbb{Rotword}$, and rotates the blocks, i.e., $\mathbb{Subword}(B_0, B_1, B_2, B_3) = (B_1, B_2, B_3, B_0)$ for the case of 128-bits being used. The second one applies the so called AES S-box described in Table 7.4.

Algorithm 7.3.2 ($\mathbb{KeyExpansion}$)

Input *a key k;*

Subroutines $\mathbb{Subword}$ *and* $\mathbb{Rotword}$*;*

Output *44 words composing the keys to all the rounds.*

Step 1 $RCon[1] = 01000000$

Step 2 $RCon[2] = 02000000$

Step 3 $RCon[3] = 04000000$

Step 4 $RCon[4] = 08000000$

Step 5 $RCon[5] = 10000000$

Step 6 $RCon[6] = 20000000$

Step 7 $RCon[7] = 40000000$

Step 8 $RCon[8] = 80000000$

Step 9 $RCon[9] = 1B000000$

Step 10 $RCon[10] = 36000000$

Step 11 **For** $i = 0$ **to***3* **do** *:*

Step 11.1 $w[i] = (k[4i], k[4i+1], k[4i+2], k[4i+3])$

Step 12 **For** $i = 4$ **to***53* **do** *:*

Step 12.1 $aux = w[i-1]$

Step 12.2 **If** $i \equiv 0 \pmod 4$ **then**

$$aux = \mathbb{S}\mathrm{ubword}(\mathbb{R}\mathrm{otword}(temp)) \oplus RCon[i/4];$$

Step 12.2 $w[i] = w[i-4] \oplus aux$

Step 13 *Output* $(w[0], \cdots, w[43])$

• •

SECTION 7.4

Exercises

Exercise 7.4.1 (exercise 5.5 of [KL07])**.** *Consider a Feistel network defined above. Envisage the output of an r-round when the input is* (L_0, R_0) *in each of the following two cases:*

1. *Each round function outputs all 0's, regardless of the input.*
2. *Each round function is the identity function.*

Exercise 7.4.2. *Consider a DES encryption scheme with the following modification.*

Let $g : \{0,1\}^{64} \to \{0,1\}^{32}$ and each round key k_i is defined by $g(k_i)$ and modify the round function f to $f(R_{i-1}; k_i) = R_{i-1} \oplus k_i$.

Describe the output L_16 *and* R_16 *of given input* L_0 *and* R_0*.*

Exercise 7.4.3. *Prove that if two vectors words* w_1 *and* w_2 *of four bytes differ in just one byte, then their images under* $\mathbb{M}\mathrm{ix}\mathbb{C}\mathrm{olumns}$ *differ in all four bytes.*

Exercise 7.4.4. *Assume that a cipher block chaining mode as defined in the Algorithm 5.2.2 on Page 95*

is use in DES. For $i = 1, 2, 3, \cdots$, the encryption rule for a plaintext P_i, key k and a ciphertext C_{i-1} is

$$C_i = DES(P_i \oplus C_{i-1}, k)$$

where C_0 is an initial block.

Compute the decryption rule. Suppose that an opponent changes C_i to a different C_i'. How many messages are then decrypted incorrectly?

Exercise 7.4.5 (exercise 3.5 of [Sti06])**.** *Assume that the following 128-bit AES key is given in hexadecimal notation:*

$$2B7E151628AED2A6ABF7158809CF4F3C$$

Construct the complete key schedule arising from this key.

Part III

The Public Key Cryptography

CHAPTER 7

CHAPTER 8

The story behind Public Key Cryptography

This part of cryptography was only considered very recently and for that reason its history is short but very fruitful on the impact on the real life.

In previous part of this redaction only cryptographic schemes that demand a private previous communication between the parties to agree on the key that would be used to encrypt and also to decrypt the messages exchanged were considered. This particular type of schemes can be perfectly secure as was previously proved but, fundamentally, these two constrains are severe drawbacks on the applicability of these schemes on daily use. The introduction of public-key encryption is one of the milestones that complete changed the course of cryptography as every one saw it before and how every one sees it now after the introduction of these techniques.

To give a general panorama of what public key cryptography is all about consider the following example:

Example 8.0.6. *Assume that it is impossible for A and B to meet, and that they are therefore unable to exchange the key in order to communicate privately using a symmetric cryptographic system. In particular, A and B cannot agree on the key to use since that would have to be plaintext and their supposedly secret key would be exposed. But A and B can do the following:*

B selects both an encryption function and a corresponding decryption function that are not symmetric

(i.e., knowing the encryption process does not convey any information about the decryption process, usually based on an one-way function defined in Section 4.3). Then B sends only the encryption function to A that uses it to encrypt the message. Notice that even A, once message is encrypted, cannot decrypt the it. After the encryption in done, A sends the ciphertext to B, who is the only one able to decrypt it since it has the decryption function. Even if O, the malicious opponent intercepted B message and learned the encryption function, it is in the same position as A, i.e., knowing both the encryption function and the encrypted message, it remains unable to decrypt it in feasible time.

Public-key cryptographic schemes were designed to achieve two practical issues: enable the parties to communicate privately without having to agree on any secret information in advance and without meeting personally and secondly enable every one to communicate with each other without to have to for each person that one wishes to communicate a different key. The latter one is clear for the reader to be unpractical in large scale as one would have to keep a huge database of keys that would be difficult to manage, specially for the ones that do not have the knowledge to do it. Notice that in practical terms and specially based on the reader use of internet, anyone can have a secret (only one) password, for example. For the former one, the reader that is not familiar with this theory it can be surprising that such thing can be possible but a simple example can help the reader to understand the basic idea of public key cryptographic schemes. Imagine that two people in different parts of the globe want to communicate over the internet and share no initial secret and all that travels around the internet is public but they want to keep the message secret in such a way that no one else can learn anything about what they are saying! One way of doing this is just A to send B encrypted message that used the public part of the key of B and then if the system is secure and the only way to recover the message is to know the private key of B which is assumed to be untraceable form the public key. This simple solution is only possible if some conditions are verified. Notice that in this cases one cannot use symmetric key schemes, since if the public was equal or easily computable from the private key than no secret would be possible to be shared in this scenario. It is then mandatory to have, even if somehow related, to have two different keys to encrypt and decrypt processes.

The public key cryptography, also known as modern cryptography, had born in the decade of 70s of the last century when, mainly the two issues discussed above, were realized and became

impossible to have for each pair of users a pair of strings. Although the main impulse for public key cryptography came from military and commercial use rather than particular parties A and B. Similar to the described issue discussed above, the task of military encryption key management had greatly expanded with the growth in electronic communications networks, and the number of parties that needed to communicate to organize a large-scale military operation such as World War II. In this practical experience communications showed that much of the messages had ended up unencrypted because there was no time to establish a secure connection to transmit the agreed key. In the commercial domain the problem impulsed the public key cryptography since the quickly growing, with increasing use of electronic networks to conduct sensitive business communications and financial transactions, often between parties from different organizations that had not previously communicated or had the chance to exchange encryption keys beforehand.

The advent of Cryptography is due to several authors, for example, Ellis and Markle that later joint Diffie and Hellman who were responsible for putting together the ideas underlying the public key cryptography and Rivest, Shamir and Adleman there were responsible for the key transport and digital signing schemes called the RSA system in 1977. Seven years later another name marked the public key cryptographic system, namely Elgammal, that designed another system based on elliptic curves and on the Diffie–Hellman key exchange.

All these schemes and many other that were developed insure its security on assumptions on the hardness of some problems like the discrete logarithm problem or the integer factorization problem since they, in fact, are not perfectly secure. But just the problems themselves are not sufficient to guarantee sufficient security and a careful protocol design and a methodology for defining precisely the security objective and proving that a protocol met that objective are required. The first step in that direction is to imagine the most powerful adversary possible and to assign to that adversary a goal or set of goals as weak as possible. In this scenario a protocol could be deemed secure if an ultra-potent adversary could not achieve his or her ultra-weak goals and here is where provable security appears in the form of computational assumptions. Only in middle of 90s provably secure protocols began to be efficient enough to be used in the real-world.

In the last decades since the appearing of public key cryptography many applications have

been developed to different purposes that will be discussed in detail in due course of this Part of this work.

. . . . . .

SECTION 8.1

Describing a public key cryptographic system

In the previous part where symmetric cryptographic systems were described. The term symmetric was used to describe the real symmetry of the encryption and the decryption processes. In fact, the two processes are very similar since the decryption process can be easily derived from the encryption process. For example, in the case of substitution scheme described in the Cryptographic System 5.1.3 the process of decryption is essentially equal to the encryption process but instead of using the permutation π one uses its inverse. Also, another name that is commonly use to these schemes are "shared keys" cryptographic systems, since the two parties involved use the same private key to both encrypt and decrypt information.

As was mentioned before, the symmetric cryptographic schemes have some advantageous and some disadvantages. The use of the same key for both decryption and encryption makes the process of encryption and decryption of the messages much faster, it is easier to implement and also, in general, requires less processing power than public key cryptography that will be discussed in next paragraphs ahead. The disadvantages of symmetric key cryptography is that the two parties sending messages to each other must agree to use the same private key before they start transmitting secure information which might be impossible depending on the circumstances. The advantage of public key cryptography is that the public key used for encryption does not need to be kept secret, and that the reason to call these schemes "public". Usually, when two parties want to communicate they take advantage of two worlds. They use public key cryptography to create a shared session key that they keep secret and then they communicate through symmetric key cryptography using the shared session key. This way they can get the best of both worlds – the performance/speed of shared key cryptography along with the convenience of public key cryptography to avoid meeting personally of expensive secure channels of communications.

From the discussion above one concludes that the public key cryptography is a system that

works with two separate keys: one of which is personal, called the private key and is kept secret and another one called public that is usually made available to everyone. Although different, the two parts of the key pair are mathematically linked and thus depend one on each other. The function of the public key is usually to locks or encrypts the plaintexts to be sent for the person from anyone, and the other one, the private key, unlocks or decrypts the ciphertext and a desirable property is that neither key can perform both processes.

An analogy can be made with lockers. The party A places an object, usually a message in a metal box, and then locks it with a locker left there by the other party B. Since B is the only one that has the key for that locker is the only one that can open it. The next figure illustrates the mechanism.

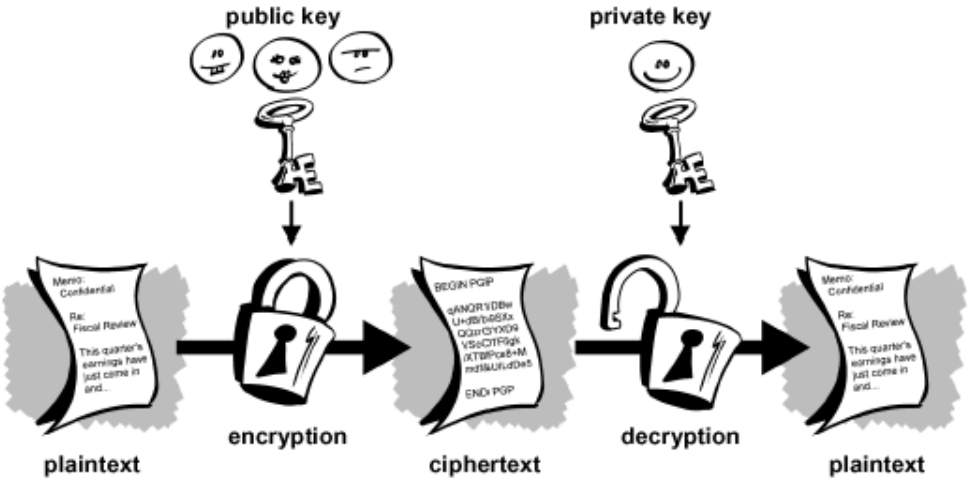


Figure 8.1: The public-key encryption.

To achieve this goals it is used asymmetric key algorithms like the case of RSA where the private key is just a pair of primes and the public key is just a number depending on the primes but from which is very difficult to recover the information about those primes. This justifies why, sometime,s these systems are also called by the more generic term "asymmetric key cryptography". Usually, the algorithms used for public key cryptography are based on mathematical relationships which are unknown to have efficient solution as mentioned in the Chapter of Notion of computational complexity 4 and in particular in the Sections of Complexity classes 4.2.2 of One-way function 4.3. These schemes need to have some properties that are essential for practical purpose:

- The production of public and private keys is computationally easy;
- Decrypting a message using the private key is feasible;
- Encrypting a message using the public key is also feasible;
- Anyone trying to derive the private key using the information that is publicly available, in the case the public key, for example is unfeasible and computational hard.

Notice that item 2 and item 4 above remind the concept of trapdoor one-way function presented in Definition 4.3.3 on page 4.3.3 since without the knowledge of the private key, computing the plaintext of a message is hard, but with the use of the private key that computation becomes trivial to achieve.

Unlike symmetric key algorithms, the public key algorithms do not require a secure initial exchange of one (or more) secret keys between the sender and receiver and, as an extra feature, they can also be used to authenticate a message by creating a digital signature of the message using the private key, which can then be verified by using the public key. The most common uses of the public key systems are the public key distribution systems, the digital signature systems, and the public key cryptographic systems, which can perform both public key distribution and digital signature services. In wikipedia some clever and very elucidative parallel situations explaining these concepts are presented (complementary to the analogy made above):

> *An analogy to public-key encryption is that of a locked mail box with a mail slot. The mail slot is exposed and accessible to the public – its location (the street address) is, in essence, the public key. Anyone knowing the street address can go to the door and drop a written message through the slot. However, only the person who possesses the key can open the mailbox and read the message.*
>
> *An analogy for digital signatures is the sealing of an envelope with a personal wax seal. The message can be opened by anyone, but the presence of the unique seal authenticates the sender.*
>
> *in* wikipedia

The basic work of a public key cryptographic system is the following. The party A widely disseminates its public key that will work as the encrypting-key and keeps private the decrypting-key. B, to encrypt a message to A encrypts with A public key. After receiving the message A uses the private key to decode the message sent by B.

Next it is presented formally the concept of public key Cryptographic system which will be use in the rest of these notes.

Definition 8.1.1 (*Public key cryptographic system*)

A public key cryptographic system is a tuple $\mathcal{PKC} = (P, C, K, E, D, R, r)$ where:

1. *P is the set of plaintexts;*
2. *C is the set of ciphertexts;*
3. *K is the set of keys;*
4. *R is the part of the key that is made public (which is revealed);*
5. *$r : K \to R$ is called the revealing or the publication function and is a polynomial time computable function.*
6. *E consists of encrypting one-way functions with trapdoor k (accordingly to Definition 4.3.3) using the part of the key that is public, i.e., consists of functions $e_r : P \to C$ that are trapdoor one way functions and where $r \in R$, which, of course depends on a key $k \in K$ such that $u(k) = r$ (k is trapdoor).*
7. *D consists of decrypting functions $d_k : C \to P$, such that $d_k(e_{r(k)}) = x$.*

CHAPTER 9

The RSA cryptographic system

The RSA is, as the reader might know or at least have an idea, an algorithm of public-key cryptography that is based on the presumed difficulty of factoring problem. The name RSA is due to its own designers, Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described it in 1977. It is worth mentioning that in 1973, Clifford Cocks, an English mathematician, working for the Communication Electronics Security Group (CESG), a special section of the British Government Communications Headquarters (GCHQ) had developed an equivalent system in 1973, but its design was kept secret and classified until 1997.

An A user of RSA creates and then publishes the product of two large prime numbers, along with an auxiliary value, as their public key. The prime factors must be kept secret, otherwise one could use them to perform the decryption process with acknowledgment of the interested party. Anyone can use the public key to encrypt a message, but with currently published methods, if the public key is large enough, only someone with knowledge of factorization of the number can feasibly decode the message. Whether breaking RSA encryption is as hard as factoring is an open question known as the RSA problem.

To properly present the RSA cryptographic system one needs to explain several issues concerning the practical implementation of RSA. For example, one needs to justify why one can perform modular exponentiation of find the inverse of a umber modulo an integer in polynomial time.

In Section 2.3 of Chapter 2 it was already proved that $(\mathbb{Z}_n, +, \times)$ is a ring and also that $b \in \mathbb{Z}_n$ admits inverse if and only if $gcd(n, b) = 1$.

• •

SECTION 9.1

The RSA cryptographic system

Now it is possible to describe the RSA Cryptographic system formally. The RSA is performed over $\mathbb{Z}_n$ where n is the product of two distinct odd primes p and q. Notice that, in this case one has $\phi(n) = (p-1)(q-1)$, where $\phi(n)$ in the value of Euler's function on n, i.e., the number of natural numbers less of equal to n that are co-prime with n.

Cryptographic system 9.1.1 (*RSA cipher system*)

The RSA cipher system is a tuple $\mathcal{RSA} = (P, C, K, E, D, R, r)$ where:

1. *$P = C = \mathbb{Z}_n$, where $n = p \times q$ for primes p and q;*
2. *K consists of pairs $k = (a, b) \in (\mathbb{Z}_n)^2$ such that $a \cdot b \equiv 1 \pmod{\phi(n)}$, i.e., a and b are inverse of each order modulo $\phi(n)$ where $\phi(n)$ is the n's value of Euler's function;*
3. *$R = \mathbb{Z}_n$;*
4. *Given a pair (a, b), $r(a, b) = b$;*
5. *For each $k = (a, b) \in K$ and all $x, y \in \mathbb{Z}_n$, $e_k(x) = x^b \pmod{n}$ and $d_k(y) = y^a \pmod{n}$.*
6. *The values n and b form the public key, and p, q, a are the private key.*

The first step is to prove that in fact the cryptographic system described above is indeed in conditions of Definition of public cryptographic system presented in 8.1.1 on Page 171 in the sense that decrypting an encrypted text leads to the original text.

Theorem 9.1.1

Let k, n, p, q, a and b be as defined above in the RSA cipher system. Then for all $x \in \mathbb{Z}_n$, $d_k(e_k(x)) = x$.

Proof: From the fact that $a \cdot b \equiv 1 \pmod{\phi(n)}$, it follows that $\phi(n)|(ab-1)$ and thus, there exists $c \in \mathbb{Z}$ such that $c\phi(n) = ab - 1 \Leftrightarrow ab = c\phi(n) + 1$. Then:

$$\begin{aligned} d_k(e_k(x)) &= (x^b)^a \pmod{n} \\ &\equiv x^{ba} \pmod{n} \\ &\equiv x^{c\phi(n)+1} \pmod{n} \end{aligned}$$

If $x \in \mathbb{Z}_n^*$ then $x^{c\phi(n)+1} \pmod{n} = (x^{\phi(n)})^c x \pmod{n} \equiv x \pmod{n}$ directly by Euler's Theorem 2.3.6 on Page 23.

If $x \in \mathbb{Z}_n - \mathbb{Z}_n^*$ then one needs to use the Chinese Remainder Theorem 2.3.4 on Page 22.

Since $x \in \mathbb{Z}_n - \mathbb{Z}_n^*$, then either p or q is a common divisor of x and n. Assume without loss generality that $q|x$. Then

$$x \equiv 0 \pmod{q} \Leftrightarrow$$

$$x^{c\phi(n)+1} \equiv 0 \pmod{q} \Leftrightarrow$$

$$x^{c\phi(n)+1} \equiv x \pmod{q}$$

Also, since $x < n$ and $q|x$ then $p \not| x$ and thus

$$\begin{aligned} x^{c\phi(n)+1} \pmod{p} &\equiv (x^{c(q-1)})^{(p-1)} \cdot x \pmod{p} \\ &\equiv 1 \cdot x \pmod{p} \end{aligned}$$

by Fermat's Little Theorem 2.3.5 on page 22. So, by the Chinese Remainder Theorem 2.3.4, one concludes that x is unique such that

$$x^{c\phi(n)+1} \pmod{n} \equiv x \pmod{n}.$$

■

Example 9.1.2. *To encrypt a message with the encryption scheme of RSA, each letter of the alphabet is transformed into a number.*

In order to avoid misinterpretation on the decryption process one can either choose to encrypt each letter assuming a numeration of letters that starts with $A = 10$, $B = 11$ *and so on or one can choose to do it in small groups, say of 2 letters and write them in base 26 with weights as in next table:*

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

So, to encrypt the message "math", one considers "ma" and "th" and represents it in the following form:

$$P_1(ma) = 12 \cdot 26 + 0 = 312$$

$$P_2(th) = 19 \cdot 26 + 7 = 501$$

With such groups, the largest number one can obtain corresponds to "zz" is 675, so one must have a number n that is larger than 675.

Consider for example $p = 29$, *and* $q = 31$ *and thus* $n = pq = 29 \cdot 31 = 899$. *Notice that*

$$\begin{aligned} \phi(n) &= (29-1)\cdot(31-1) \\ &= 840 = 2^3 \cdot 3 \cdot 5 \cdot 7 \end{aligned}$$

Thus, one has to choose b if and only if b is not divisible by 2, 3, 5 *or* 7. *Take for example,* $b = 11$. *Thus the public key* (n, b), *is in this case* $(899, 11)$.

So the encryptions of each of the blocks are:

$$e_k(P_1) = 312^{11} \pmod{899} = 64$$

$$e_k(P_2) = 501^{11} \pmod{899} = 583$$

To finish one concatenates the values obtained, i.e., $e_k(312501) = 064\ 583 = 064583$.

To decrypt one considers the inverse of b modulo $\phi(899)$ *i.e.,* $a = b^{\phi(\phi(n))-1} \pmod{840} = 11^{288-1} \pmod{840} = 611$. *In particular, to decode* 064583 *one brakes again into blocks*

$$T_1 = 064 \qquad P_2 = 583$$

$$d_k(T_1) = 64^{611} \pmod{899} = 312$$

$$d_k(T_2) = 191^{257} \pmod{899} = 501$$

Notice that all together the message needs to have a even number of elements. So, the deciphered message is $d_k(064583) = 312\ 501 = 312501$ *which is decoded by "math" reserving the calculations in base* 26.

In practice the numbers used in the RSA cryptographic system are very large, usually with size in the order of 1024 bits, i.e., with more than 300 decimal digits. This is a precaution against the fastest computers that with these kind of numbers are useless simple they would require more than universe time to factor it. One example is

$$\begin{aligned} n = {} & 11929413484016950905552721133125564964460656966152763801206748195494305685115033380631595703771562029730500011862877084668996911289221224545711806057499598951708004210526342737632227426639311619351783957077350563223159668112192733747397322031251259906123132225094550626006655755382385175753906212629403839139631 \end{aligned}$$

This is composed of the two primes

$$\begin{aligned} p = {} & 09337661836325758176115170347306682871557999846322234541387456711212734562876700082908433028755212749702453145932229461290645383585810186155398284791464691 \end{aligned}$$

and

$$\begin{aligned} q = {} & 10910616967349110231723734078614922645337060882141748968209834225138976011179993394299810159736904468554021708289824396553412180514827996444845438176099727 \end{aligned}$$

Notice that, in theory, the encryption scheme of RSA described in the Cryptographic system 9.1.1 on Page 174 works perfectly but in order to be a cipher scheme that can be used in practice one must so that its operations are computable in polynomial time (like the modular exponentiation and the determination of the greatest common divisor) and preferably has some security properties. Notice that, one needs the Euclidean algorithm to run in polynomial time to establish the public key and the modular exponentiation to determine the inverse of the public key modulo $\phi(n)$ to establish a private key.

The practical implementation is discussed in the next section and the security issues in the subsequent ones.

• •

SECTION 9.2

Euclidean Algorithm for the gcd and the modular exponentiation

In Chapter 2 it was already presented the modular algebras and some classical results concerning the elements of $\mathbb{Z}_n$ and $\mathbb{Z}_n^*$. One fundamental property that is used in the RSA is the fact that $(\mathbb{Z}_n^*, +, \times)$ is a field, and in particular, any element of $\mathbb{Z}_n^*$ has inverse. For practical purposes of implementation of RSA, it is crucial the Euclidean algorithm to ensure, for example, that one can determine the public key as mentioned above since, as it will be seen in this section, one can determine the greatest common divisor of two numbers efficiently and thus establish a public key properly in a reasonable time.

Algorithm 9.2.1 (*Euclidean Algorithm*)

For simplicity call it $\mathbb{Euclides}(a, b)$

Input: *Two natural numbers* a *and* b.

Output: $gcd(a, b)$ *the greatest common divisor between* a *and* b.

Step 1: If $b = 0$ **then** *Output* a **else**

Step 1.2: *Output* $\mathbb{Euclides}(b, a \pmod b)$.

Theorem 9.2.1 (*Lamé Theorem*)

If a and b be natural numbers such that $a > b \geq 0$ and $\mathbb{E}\mathrm{uclides}(a, b)$ has $k \geq 1$ recursive calls then $a \geq \mathcal{F}_{k+2}$ and $b \geq \mathcal{F}_{k+1}$, where $\mathcal{F}$ is the Fibonacci sequence.

Proof: The proof is carried out by induction over k.

(Base $k = 1$)

Assume by reduction to absurd that $a < \mathcal{F}_3 = 2$ or $b < \mathcal{F}_2 = 1$.

In the latter case if $b < 1$ then $b = 0$ and thus $\mathbb{E}\mathrm{uclides}(a, b)$ does not perform any recursive call and thus $k = 0$ which contradicts $k = 1$.

In the former case, since $a < 2$ and $b < a$ then $a = 1$ and $b = 0$ and by the same reason one concludes that is absurd.

Thus $a \geq \mathcal{F}_3$ and $b \geq \mathcal{F}_2$.

(Step $k + 1$)

Assume that the algorithm $\mathbb{E}\mathrm{uclides}(a, b)$ performs $k + 1$ recursive calls. It is easy to see that $\mathbb{E}\mathrm{uclides}(a, b) = \mathbb{E}\mathrm{uclides}(b, a \pmod b)$ and $b + a \pmod b \leq a$. Thus, the number of recursive calls of the algorithm $\mathbb{E}\mathrm{uclides}(a, b)$ is equal to the recursive calls of $1 + \mathbb{E}\mathrm{uclides}(b, a \pmod b)$, i.e, $\mathbb{E}\mathrm{uclides}(b, a \pmod b)$ uses k recursive calls. Hence, by induction hypothesis:

$$\begin{cases} b \geq \mathcal{F}_{k+2} \\ a \pmod b \geq \mathcal{F}_{k+1} \end{cases}$$

Since $b \geq \mathcal{F}_{k+2}$, to this end, it remains to show that $a \geq \mathcal{F}_{k+3}$. Since $a \geq b + a \pmod b \geq \mathcal{F}_{k+2} + \mathcal{F}_{k+1} = \mathcal{F}_{k+3}$. ■

Corollary 9.2.2. *If a and b be natural numbers such that $a > b \geq 0$ and $b \leq \mathcal{F}_{k+1}$, then there are at mot k recursive calls in $\mathbb{E}\mathrm{uclides}(a, b)$*

Notice that $\mathcal{F}_k < \dfrac{\Phi^k}{\sqrt{5}} \leq \Phi^k$, where $\Phi = \dfrac{1 + \sqrt{5}}{2}$ is the golden ration. Thus, if $b \geq \mathcal{F}_k$, then the number of recursive calls of $\mathbb{E}\mathrm{uclides}(a, b)$ is at most $\log_\Phi(b) \in O(\log(k)$. This observations lead to the following result:

Theorem 9.2.2

Let a, b and n be natural numbers such $b \leq a \leq n$. Thus, Time$(\mathbb{Euclides}(a,b)) \leq O(\log^3 n)$.

Proof: By Lamné Theorem 9.2.1 proved above on Page 179, there are at most $O(\log n)$ recursive calls of $\mathbb{Euclides}(a,b)$. Notice that the last call costs $c \log n$, (just one comparison), the penultimate costs $2c \log n$ (one comparison and the last recursive call) and so on. Thus:

$$Time(\mathbb{Euclides}(a,b)) \leq \sum_{i=1}^{c \log n} i \cdot c \log n \cdot c \log n \cdot (\log n)^2 \in O(\log^3 n).$$

■

One can generalize the Euclidean algorithm to the one that retrieves also the inverse modulo n. This algorithm is called the Extended Euclidean Algorithm.

Algorithm 9.2.3 (*Extended Euclidean Algorithm*)

For simplicity call it $\mathbb{ExtentedEuclides}(a,b)$

Input: *Two natural numbers a and b greater or equal to 1.*

Output: *$gcd(a,b)$ the greatest common divisor between a and b and the coefficients of the linear combination witnessing the $gcd(a,b)$.*

Step 1: **If** $b = 0$ **then** *Output* $(a,1,0)$ **else**

Step 2.1: $(r', s', t') = \mathbb{ExtentedEuclides}(b, a \pmod b)$;

Step 2.2: *Output* $(r,s,t) = (r', t', s' - (a \textbf{ div } b)t')$, *such that* $sa + tb = r$.

The next exercise ask the reader to choose the correctness of the Extended Euclidean algorithm.

Example 9.2.4. *Show that the outputs of* $\mathbb{ExtendedEuclidean}(a,b)$ *are in indeed such that:*

$$\begin{cases} r = gcd(a,b); \\ r = as + bt. \end{cases}$$

Example 9.2.5. *Show that* $\mathbb{ExtendedEuclidean}(a, b)$ *runs also polynomial time, in fact* $O(\log^3 n)$.

Notice that $gcd(a, b) = 1$, then from the output of the $\mathbb{ExtendedEuclidean}(a, b)$ one can determine the inverse of $a \pmod{b}$.

Corollary 9.2.6. *Let a and b be natural numbers such* $gcd(a, b) = 1$. *Then* $a^{-1} \equiv s \pmod{b}$ *where s is the output of the* $\mathbb{ExtendedEuclidean}(a, b)$.

Proof: Since $gcd(a, b) = 1$, the outputs of $\mathbb{ExtendedEuclidean}(a, b)$ satisfies:

$$\begin{cases} 1 = gcd(a, b); \\ 1 = as + bt. \end{cases}$$

Thus $as \equiv 1 \pmod{b}$, i.e., $a^{-1} \equiv s \pmod{b}$. ■

Now lets focus on the second ingredient of the RSA cryptographic system: the modular exponentiation, i.e., computation of a function of the form $x^c \pmod{n}$.

As stated above, both encryption and decryption operations of RSA use modular exponentiations, and thus, one needs to prove that it can be performed in polynomial time.

Naively, one could compute $x^c \pmod{n}$ using can be done using $c - 1$ consecutive multiplications, but this way is very inefficient if c is near $\phi(n) - 1$ since in that case is exponentially large compared to the length of n.

The solution is to use the algorithm $\mathbb{Square} - \mathbb{and} - \mathbb{multiply}$ algorithm that reduces the number of modular multiplications required to compute the value of $x^c \pmod{n}$ to at most $2|c|$, where $|c|$ is the length of the binary representation of c, i.e., the number of bits in the binary representation of c. Hence the modular exponentiation is computable in time $O(|c||x|^2)$, where $|x|$ is also the number of bits in the binary representation of x. In particular, since in the RSA $c < n$, the computation of the encoding and the decoding processes in the RSA can both be done in time $O(\log^3 n)$, which is a polynomial function of the number of bits in one plaintext (or ciphertext) character.

Algorithm 9.2.7 ($\mathbb{S}\mathtt{quare}-\mathtt{and}-\mathtt{multiply}$)

Input: *x, c and n, where c is given in its binary representation, i.e, $c=\sum_{i=1}^{l} c_i 2^i$;*

Output: $z = x^c \pmod{n}$;

Step 1: $z = 1$;

Step 2: **For** $i = l-1$ **to**0 **do**

Step 2.1: $z = z^2 \pmod{n}$;

Step 2.2: **If** $c_i = 1$ **then** $z = z \times x \pmod{n}$;

Step 2.3: $i = i-1$;

Step 3: *Output* z

It is left to the reader to prove that this algorithm computes in fact this algorithm computes $x^c \pmod{n}$. In the next proposition it is proved that the modular exponentiation is, in fact, computable in polynomial time.

Proposition 9.2.8. $\mathbb{S}\mathtt{quare}-\mathtt{and}-\mathtt{multiply}(x,c,n) \in \mathbb{P}$.

Proof: The first step requires a constant number of steps on the size of n.

The **Step 2.1**, requires $O(\log^2(z)+\log^2(n))$ computational steps, $O(\log^2(z))$ to compute z^2 and $O(\log^2(n))$ to compute the remainder, which is polynomial on the size of n.

The **Step 2.2** requires also $O(\log^2(n))$, since involves a comparison (which requires a constant number of computational steps on the size of n), a multiplication and a remainder.

Thus, the thesis follows since the number of iterations is $l = \log(c) \leq \log n$ which is polynomial in size of n. ■

SECTION 9.3

Checking fast primality of numbers

The primality test, i.e., checking if a given number is prime or not is also a crucial for the RSA process since this cryptographic system relies on the fact that one can produce relatively fast prime numbers to perform the operations to set the public and the private keys. In the previous section it was proved that if one have access to prime numbers easily then the RSA cryptographic system works perfectly under the assumption that one-way functions with trapdoors exist, more concretely that multiplication of two primes p and q is a one-way function with trapdoor with p and q.

In 2002, Agrawal, Kayal and Saxena proved that the problem $\mathbb{P}\mathrm{rimes}$, i.e., checking is a given number is prime, it is a problem that can be solved with a deterministic polynomial-time algorithm, by proving that $\mathbb{P}\mathrm{rimes} \in \mathbb{P}$. This problem was opened for long time and this result was a major breakthrough in the last years in computational complexity field. The AKS algorithm will be discussed in detail in one of the subsequent subsection. One of the issues concerning this major breakthrough is the practical implementation of the algorithm for testing primality, since the degree of the polynomial involved in the proof is high and then, although polynomial, it might require times of computation that seem unfeasible in practice. Contrarily to this deterministic algorithm and until the appearance of AKS, primality was tested with randomized polynomial-time algorithms that are allowed to error with a small probability claiming that a given number is prime when, in fact, it is not. An example of this algorithm is the so called Solovay–Strassen algorithm that can test primality of a number in quadratic time on the length of its binary representation.

To motivate the presentation of the probabilistic test of primality to generate parameter for the RSA cryptographic system consider the following argument. In Theorem 2.2.5 on Page 18, one already proved that the number of primes less or equal to n is near $n/\ln n$, and thus, if one randomly choses a number between 1 and n, then the probability that it is prime is roughly $1/\ln n$ and thus, on average picking $\ln n$ random numbers, one of them is prime. Just for sake

of presentation and understanding consider a more concrete example: pick a number n that is chosen to be a 1024 bit number which is a multiplication of two primes p and q. Then one can chose its prime factors with 512 bits. Among the numbers with 512 bits, the probability of choosing a prime number is approximately $1/\ln 2512 \approx 1/355$ and thus, on average, given choosing 355 random 512 bit integers one of them will be prime. This argument support the idea that one can generate sufficiently large random numbers that are almost for sure prime, and hence parameter generation for the RSA cryptographic system is doable in practice.

In the following it is presented and discussed this two processes for the primality testing but a few definitions are needed for the sequel, namely the concept of a probabilistic or randomized algorithm. Like the name suggests, randomized algorithms are algorithms that have access to some random numbers while a deterministic one does not have access to randomness. This kind of algorithms define other computational complexity classes other than $\mathbb{P}$ or $\mathbb{NP}$. They are called randomized classes and include $\mathbb{RP}$, $co-\mathbb{RP}$, $\mathbb{ZP} = \mathbb{RP} \cap (co-\mathbb{RP})$ and $\mathbb{BBP}$. In the first case, when a decision problem A is given, the algorithm is always correct on instances x if $x \notin A$ and is allowed to error with probability at most $1/2$ on instances $x \in A$, the second one is similar to the first one but with roles inverted concerning the instances, and the latter one, the algorithm is allowed to give wrong answers on both times of instances. The belonging relationships among these classes and $\mathbb{P}$ and $\mathbb{NP}$ are unestablished and are also major open questions in complexity.

Definition 9.3.1 (*Monte Carlo algorithm*)

Let A be a decision problem. A yes-biased Monte Carlo algorithm $\mathcal{A}$ for A (or an $co-\mathbb{RP}$ algorithm) is a randomized algorithm such that:

- *if $\mathcal{A}(x) = 1$ then $x \in A$;*
- *if $\mathcal{A}(x) = 0$ then there is some probability or error, i.e., there is a probability of cataloging wrongly instances of A.*

A no-biased Monte Carlo algorithm $\mathcal{A}$ for A (or an $\mathbb{RP}$ algorithm) is a randomized algorithm such that:

- *if $\mathcal{A}(x) = 0$ then $x \notin A$;*
- *if $\mathcal{A}(x) = 1$ then there is some probability or error, i.e., there is a probability of cataloging wrongly non-instances of A.*

Definition 9.3.2 (*Monte Carlo algorithm with error*)

Let A be a decision problem.
A yes-biased Monte Carlo algorithm $\mathcal{A}$ with error probability $\varepsilon > 0$ (or a $\mathbb{BPP}$ algorithm with error $\varepsilon > 0$) for A is a randomized algorithm such that:

- *if $\mathcal{A}(x) = 1$ then $x \in A$;*
- *if $x \in A$, $\mathcal{A}(x) = 0$ with probability at most ε, i.e., the probability of cataloging wrongly instances of A is at most ε.*

A no-biased Monte Carlo algorithm $\mathcal{A}$ with error probability $\varepsilon > 0$ (or a $\mathbb{BPP}$ algorithm with error $\varepsilon > 0$) for A is a randomized algorithm such that:

- *if $\mathcal{A}(x) = 0$ then $x \notin A$;*
- *if $x \notin A$, $\mathcal{A}(x) = 1$ with probability at most ε;*

The probability of the last two cases is computed over all possible random choices made by the algorithm $\mathcal{A}$ when it is run with a given input.

There are other type of algorithms, called Las Vegas, that contrarily to Monte Carlo ones, is

allowed to not give an answer but one it does, it is always correct. In this section the focus is to know whether or not a given number is prime or composite. The Monte Carlo algorithm are named this way because to the grand casino in the Principality of Monaco at Monte Carlo, which is well-known around the world as an icon of gambling. In the following it is formally describe the problems:

Problem 9.3.1 ($\mathbb{C}\mathrm{omposite}$)

Instance *An integer $n \geq 2$;*

Question *Is n composite?*

Problem 9.3.2 ($\mathbb{P}\mathrm{rimes}$)

Instance *An integer $n \geq 2$;*

Question *Is n prime?*

Notice that these two problems are complementary, i.e., if $n \notin \mathbb{C}\mathrm{omposite}$ then $n \in \mathbb{P}\mathrm{rimes}$.

Recall that a decision algorithm is only required to answer 1 or 0 and thus, for example in the case of $\mathbb{C}\mathrm{omposite}$ it will not give a factorization in case n is composite, which is a search problem.

The Solovay–Strassen algorithm, is a yes-biased Monte Carlo algorithm for $\mathbb{C}\mathrm{omposite}$ with error probability $1/2$, which means that if the algorithm answers 1 then n is composite for sure while if n is composite, then the algorithm answers 1 with probability at least $1/2$.

Before the formal presentation of the Solovay–Strassen algorithm for primality testing it is necessary some more background on number theory, namely Legengre's and Jacobi's symbol, that is presented in the next subsection.

• • •

9.3.1 The quadratic residue problem and the Legendre and Jacobi symbols

The quadratic residues is a very important problem in Cryptography that is believed unfeasible in general but easy to determine for the case where the modulo is taken over a prime or a prime power. The difficulty arises when the factorization of the modulo is unknown.

Definition 9.3.3 (*Quadratic residue modulo p*)

Let p be an odd prime and let a be an integer.
One says that a is a quadratic residue modulo p, if $a \not\equiv 0 \pmod{p}$ and the congruence $y^2 \equiv a \pmod{p}$ has a solution $y \in \mathbb{Z}_p$.
One says that a is a quadratic non-residue modulo p, if $a \not\equiv 0 \pmod{p}$ and it is not a quadratic residue modulo p, i.e, the congruence $y^2 \equiv a \pmod{p}$ has no solutions in $\mathbb{Z}_p$.

Notice that if p is an odd prime and a is such that $y^2 \equiv a \pmod{p}$, i.e., a is a quadratic residue modulo p, then also $(-y)^2 \equiv a \pmod{p}$. Since p is odd $y \not\equiv -y \pmod{p}$ and thus the quadratic congruence $x^2 \equiv a \pmod{p}$ is equivalent to the congruence $(x-y)(x+y) \equiv 0 \pmod{p}$. Thus, $p|(x-y)(x+y)$ which implies, since p is prime, that $p|(x-y)$ or $p|(x+y)$, i.e., $x = \pm y \pmod{p}$.

So, this arguments leads to the conclusion that the congruence $x^2 \equiv a \pmod{p}$ has exactly two solutions (modulo p) that are, in particular, negatives of each other modulo p.

One can now define the Quadratic Residues problem.

Problem 9.3.3 (Quadratic Residues *modulo primes*)

Instance *An odd prime p and an integer $a \in \mathbb{Z}_p$;*

Question *Is a a quadratic residue modulo p?*

The following Euler's theorem justifies the fact that Quadratic Residues modulo prime numbers can be solved in polynomial time.

Theorem 9.3.1 (*Euler's Criterion of residues*)

Let p be an odd prime and a and integer. The integer a is a quadratic residue modulo p if and only if

$$a^{(p-1)/2} \equiv 1 \pmod{p}$$

Proof:

($\Leftarrow$)

Assume that a is a quadratic residue modulo p, i.e., $a \not\equiv 0 \pmod{p}$ and there is $y \in \mathbb{Z}_p$ such that $y^2 \equiv a \pmod{p}$. By Fermat's little theorem 2.3.5 on Page 22, $y^{p-1} \equiv 1 \pmod{p}$. Thus:

$$\begin{aligned} a^{(p-1)/2} &\equiv (y^2)^{(p-1)/2} \pmod{p} \\ &\equiv y^{(p-1)} \pmod{p} \\ &\equiv 1 \pmod{p} \end{aligned}$$

($\Rightarrow$)

Assume that $a^{(p-1)/2} \equiv 1 \pmod{p}$ and let b be a generator of $\mathbb{Z}_p^*$. Then, by definition of generator, there exists $i \leq p-1$ such that $a \equiv b^i \pmod{p}$. Then:

$$\begin{aligned} (b^i)^{(p-1)/2} &\equiv a^{(p-1)/2} \pmod{p} \\ &\equiv 1 \pmod{p} \end{aligned}$$

since b is a generator of $\mathbb{Z}_p^*$ then $ord(b) = p-1$ and since $(b^i)^{(p-1)/2} \equiv 1 \pmod{p}$ then by Lagrange's theorem 2.1.1 on Page 11, $p-1 \left| \frac{i(p-1)}{2} \right.$, which implies that $i/2$ is an integer and hence is even, i.e, there exists $k \in \mathbb{N}$ such that $i = 2k$. thus, $a \equiv b^i \pmod{p} \equiv (b^k)^2 \pmod{p}$, and so a is a quadratic residue modulo p. ■

Notice that by the fact that the Algorithm 9.2.7 of $\mathtt{Square-and-multiply}$ runs in polynomial time (proved in Proposition 9.2.8), determining if $a^{(p-1)/2} \equiv 1 \pmod{p}$ can be done in $O(\log^3 p)$ and thus, the problem $\mathbb{Quadratic\ Residues}$ modulo primes is computable in polynomial time.

Definition 9.3.4 (*Legendre symbol*)

Let p be an odd prime and let a be an integer. The Legendre symbol is a function $\mathbb{Z} \to \{-1, 0, 1\}$, denoted by $\left(\frac{a}{p}\right)$ that is defined as follows:

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } a \equiv 0 \pmod{p} \\ 1 & \text{if } a \text{ is a quadratic residue modulo } p \\ -1 & \text{if } a \text{ is a quadratic non-residue modulo } p \end{cases}$$

In the next example it is realized the connection between Euler's criterion and the Legendre symbol for prime numbers.

Example 9.3.4. *From Euler's criterion for residues presented in Theorem 9.3.1 it is clear that $\left(\frac{a}{p}\right) = 1$ if and only if $a^{(p-1)/2} \equiv 1 \pmod{p}$. Also, if a is multiple of p then $\left(\frac{a}{p}\right) = 0$ and on the other cases $\left(\frac{a}{p}\right) = -1$.*

A more practical example is given now.

Example 9.3.5.

$$\left(\frac{2}{7}\right) = 1$$

since $4^2 \equiv 2 \pmod{7}$.

$$\left(\frac{5}{7}\right) = -1$$

as the reader can easily check, there is not $n < 7$ such that $n^2 \equiv 5 \pmod{7}$.

From the example above and again, by Theorem **??** it is easy to derive the following result:

Theorem 9.3.2

Let p be an odd prime and let a be an integer. Then the Legendre symbol can be computed by:

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \pmod{p}$$

It is possible to generalize the Legendre symbol for odd numbers that are not primes. This generalization is called the Jacobi symbol and is presented in the next definition.

Definition 9.3.5 (*Jacobi symbol*)

Let n be an odd number and let a be an integer. Assume that the decomposition in prime factor of n is:

$$n = \prod_{i^1}^{k} p_i^{l_i}.$$

The Jacobi symbol is a function $\mathbb{Z} \to \{-1, 0, 1\}$, denoted also by $\left(\frac{a}{n}\right)$ that is defined as follows:

$$\left(\frac{a}{n}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{l_i}.$$

To prove that the Solovay–Strassen is computable in polynomial-time it will be necessary to compute the Jacobi symbol. It is not necessary to factor n to compute the Jacobi symbol since one can use some results from number theory presented below:

Theorem 9.3.3 (*Congruent Integers*)

Let n be an odd number and let a and b two integers such that $a \equiv b \pmod{n}$. Then:

$$\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right).$$

Proof: Notice that if $a \equiv b \pmod{p}$ then the congruence $x^2 \equiv a \pmod{p}$ has a solution if and only if $x^2 \equiv b \pmod{p}$. So congruent integers are of the same quadratic character and thus has the same Jacobi symbol. ■

Theorem 9.3.4

Let n be an odd number. Then:

$$\left(\frac{2}{n}\right) = \begin{cases} 1 & \text{if } n \equiv \pm 1 \pmod 8 \\ -1 & \text{if } n \equiv \pm 3 \pmod 8 \end{cases}.$$

Proof: The statement of the Theorem, by the Euler's Criterion (Theorem 9.3.1), is equivalent to prove that:

$$2^{(n-1)/2} = (-1)^{(n^2-1)/8} \equiv \pm 1 \pmod n$$

depending on the case if $n \equiv \pm 1 \pmod 8$ or $n \equiv \pm 3 \pmod 8$. [1]

Now the thesis of the result follows from the fact that if $n \equiv \pm 1 \pmod 8$ then $(n^2-1)/8$ is even and if $n \equiv \pm 3 \pmod 8$ then $(n^2-1)/8$ is odd.

In fact, assume for now that $n \equiv 1 \pmod 8$. Then there exists k such that $8k - 1 = n$ and thus $n^2 = 64k^2 - 16k + 1$.

So, one has

$$\frac{n^2-1}{8} = 8k^2 - 2k$$

i.e., $\dfrac{n^2-1}{8}$ is a even number.

The other 3 cases for possible values of n are very similar to this one and are left as exercises for the interested reader. ■

[1] The proof of the equivalence uses a result due to Gauss that it is omitted. A complete proof of this equivalence can be found in any classical book of number theory.

Theorem 9.3.5 (*Multiplicative nature*)

Let n be an odd number and let a and b two integers. Then:

$$\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right).$$

In particular if $a = 2^k b$ with b an odd number, then

$$\left(\frac{a}{n}\right) = \left(\frac{2}{n}\right)^k \left(\frac{b}{n}\right).$$

Proof: It follows from the following identity $(ab)^{(p-1)/2} \equiv a^{(p-1)/2} b^{(p-1)/2} \pmod{p}$. ■

Theorem 9.3.6 (*The law of quadratic reciprocity*)

Let a and b be odd numbers. Then:

$$\left(\frac{a}{b}\right) = \begin{cases} -\left(\frac{b}{a}\right) & \text{if } a \equiv b \equiv 3 \pmod 4 \\ \left(\frac{b}{a}\right) & \text{otherwise} \end{cases}.$$

Proof: Notice that the statement of the theorem is equivalent to the equality

$$\left(\frac{a}{b}\right)\left(\frac{b}{a}\right) = (-1)^{\frac{(a-1)(b-1)}{4}}.$$

If $a \equiv b \equiv 3 \pmod 4$ then there are i and j such that $a - 3 = 4i \Leftrightarrow a - 1 = 4i + 2$ and $b - 3 = 4j \Leftrightarrow b - 1 = 4j + 2$ and thus:

$$(a-1)(b-1) = (4i+2)(4j+2) = 16ij + 8i + 8j + 4$$

and thus, dividing by 4 one has that

$$\frac{(a-1)(b-1)}{4} = 4ij + 2i + 2j + 1$$

which is odd, since by assumption b is odd.

On the other hand, assume without loss of generality that $a \equiv 1 \pmod 4$. In this case, $a - 1 = 4i$ for some i and thus $(a-1)(b-1) = 4i(b-1)$ and thus

$$\frac{(a-1)(b-1)}{4} = i(b-1)$$

which is even. ■

Corollary 9.3.6. *Let n be an odd number and let a and b two integers. Then:*

$$\left(\frac{n}{ab}\right) = \left(\frac{n}{a}\right)\left(\frac{n}{b}\right).$$

Next some examples are presented for sake of understanding.

Example 9.3.7. *First of all notice that contrarily to the Legendre symbol, if for some a and n, $\left(\frac{a}{n}\right) = 1$, when n is not prime does not necessarily mean that a is a quadratic residue modulo n. In fact, for example,*

$$\left(\frac{2}{9}\right) = \left(\frac{2}{3}\right)\left(\frac{2}{3}\right) = 1$$

but, as the reader can easily check, modulo 9, 2 is not a quadratic residue.

Example 9.3.8.

$$\begin{aligned}
\left(\frac{12345}{331}\right) &= \left(\frac{3}{331}\right)\left(\frac{5}{331}\right)\left(\frac{823}{331}\right) \\
&= \left(\frac{3}{331}\right)\left(\frac{5}{331}\right)\left(\frac{161}{331}\right) \\
&= \left(\frac{3}{331}\right)\left(\frac{5}{331}\right)\left(\frac{7}{331}\right)\left(\frac{23}{331}\right) \\
&= (-1)\left(\frac{331}{3}\right)\left(\frac{331}{5}\right)(-1)\left(\frac{331}{7}\right)(-1)\left(\frac{331}{23}\right) \\
&= -\left(\frac{1}{3}\right)\left(\frac{1}{5}\right)\left(\frac{2}{7}\right)\left(\frac{3}{23}\right)^2 \\
&= -(1)(1)(1)(1) = 1
\end{aligned}$$

• • •

9.3.2 Solovay–Strassen algorithm

Now it is presented the Solovay–Strassen algorithm. It is a yes-biased Monte-Carlo algorithm for the $\mathbb{Composite}$ with error probability at most $1/2$. This test is based on the Euler's criterion 9.3.1 and works by selecting at random an integer and computing the powers $(n-1)/2$ in $\mathbb{Z}_n$. In addition, it computes the Jacobi symbol for these numbers and if the results of both two calculations are different, then the number is composite.

Since this is a Monte-Carlo algorithm with probability of error $1/2$ it is not necessary for the two calculations to agree when n is composite. In fact, among those numbers which are relatively prime to n in the set $\{1, 2, \cdots, n-1\}$, at most 50% of them get a wrong answer. For any given integer n, if one wants to decresae the probability of error one can repeat it by picking k integers uniformly at random and run the basic algorithm. With this repetation one can decrease the probability to $1/2^k$ which is negligible (when compared with k and n) for sufficiently large k.

Algorithm 9.3.9 (*Solovay - Strassen* – $\mathbb{SS}$)

Input: *An integer* n.

Output: 1 *if* n *is prime and 0 otherwise*

Step 1: *Choose at random a number* a *between* 1 *and* $n-1$*;*

Step 2: $x = \left(\frac{a}{n}\right)$*;*

Step 3: **If** $x = 0$ **then** *output 0* **else**

Step 3.1: *Use the Algorithm* $\mathbb{Square} - \mathrm{and} - \mathbb{multiply}$ *define of Page 182 to compute the value* $y = a^{(n-1)/2} \pmod{n}$*;*

Step 3.2: **If** $x \equiv y \pmod{n}$ **then** *output 1* **else** *output 0;*

One can call a given composed number n a pseudo-prime if the Solovay–Strassen algorithm

output wrongly 1.

Proposition 9.3.10. *The Solovay–Strassen algorithm described above is polynomial.*

Proof: The only operation that needs a more concrete justification is the Jacobi symbol, since all the other operations are either comparisons, or application of subroutines that were already proved to be polynomial. Like it was already mentioned the idea is use the results presented above about number theory concerning the Jacobi Symbol.

Notice that by Theorem 9.3.5 the only arithmetic operations to compute the Jacobi symbol are modular reductions and factoring out powers of two. Thus, if an integer is represented in binary, then the operation of factoring out powers of two is just counting the number of zeros in that representation. This implies that the complexity of factoring out 2 is obviously polynomial on the length of the representation of the number. Thus, the complexity of $\mathbb{SS}$ is determined by the number of modular reductions that must be done. It is left as a simple exercise for the reader to check that the computation of the Jacobi symbol, at most $O(\log n)$ modular reductions are performed, each of which can is carried in time $O(\log^3 n)$. ■

To this end, one needs to prove that, in fact, the Solovay - Strassen algorithm is a Monte-Carlo algorithm with error probability $1/2$. To do so it is necessary the following lemma:

Lemma 9.3.11. *Let $n \geq 3$ be an odd number. n is prime if and only if*

$$\left\{a \in \mathbb{Z}_n^* : \left(\frac{a}{n}\right) = a^{(n-1)/2} \pmod{n}\right\} = \mathbb{Z}_n^*$$

Proof:

($\Leftarrow$)

It follows directly from Euler's criterion proved in Theorem 9.3.1.

($\Rightarrow$)

Assume by contradiction that $L = \left\{a \in \mathbb{Z}_n^* : \left(\frac{a}{n}\right) = a^{(n-1)/2} \pmod{n}\right\} = \mathbb{Z}_n^*$ but n is composite.[2]. Then n is square free i.e., it does not exist a prime p such that $p^2|n$. In fact if $p^2|n$ then, in particular, $p|b$ where b is the least integer such that for all $a \in \mathbb{Z}_n^*$, $a^b \equiv 1 \pmod{n}$, thus

[2] n is called a Carmichael number

by Fermat's little theorem 2.3.5, $p|n-1$. Since $p^2|n$, $p|n$ and thus $p|n-1-n$, i.e., $p|1$ which contradicts the fact that p was taken to be a prime number.

Since n is square free $n = pr$ for some prime p and $r > 1$ such that $gcd(p,r) = 1$. Let g be a non-quadratic residue modulo p and consider $a \equiv g \pmod{n}$ and $a \equiv 1 \pmod{r}$. Thus by Theorem 9.3.6 one has:

$$\begin{aligned}\left(\frac{a}{n}\right) &= \left(\frac{a}{p}\right)\left(\frac{a}{r}\right) \\ &= \left(\frac{g}{p}\right)\left(\frac{1}{r}\right) \\ &= (-1)\cdot 1 = -1\end{aligned}$$

Since $\left(\frac{a}{n}\right) = a^{(n-1)/2} \pmod{n}$ and since $a^{(p-1)/2} \equiv -1 \pmod{p}$ then by the Chinese remainder theorem 2.3.4, one concludes that $a^{(p-1)/2} \equiv -1 \pmod{r}$ contradiction the fact that a was chosen such that $a \equiv 1 \pmod{r}$. ■

Theorem 9.3.7

If n is an odd prime, and $a \in \{1, \cdots, n-1\}$, then

$$\Pr[\$\$(a) = 1] = 1.$$

If n is an odd composite, then

$$\Pr[\$\$(a) = 0] \geq \frac{1}{2}.$$

Proof: If n is an odd prime, then by the previous lemma the Jacobi symbol $\left(\frac{a}{n}\right)$ coincides always with $a^{(p-1)/2} \pmod{n}$ and thus, it is always the case that $\$\$(n) = 1$.

Assume know that n is an odd composite number. From the lemma above it follows that $L \neq \mathbb{Z}_n^*$. The reader will have no trouble to show that L is, in fact, a non trivial subgroup of $\mathbb{Z}_n^*$. Thus, by Lagrange theorem 2.1.1 on Page 11, $ord(L)|\phi(n)$, hence $\#L \leq \frac{n-1}{2}$. Thus

$$\Pr[\$\$(n) = 1] = \Pr[gcd(a,n) = 1 \wedge a \in L] = \frac{\#L}{n-1} \leq \frac{1}{2}$$

■

• • •

9.3.3 Miller-Rabin's algorithm for primality test

In this section it is briefly another algorithm that is also a Monte-Carlo algorithm with probability of error $1/2$.

Algorithm 9.3.12 (*Miller-Rabin* – $\mathbb{MR}$)

Input: *An integer* n.

Output: 1 *if* n *is prime and 0 otherwise*

Step 1: *Compute* $n-1 = 2^s d$ *with* d *odd.*

Step 2: *Choose at random a number* a *between* 1 *and* $n-1$*;*

Step 3: $a_0 = a^d \pmod{n}$*;*

Step 4: **If** $a_0 = 1$ **then** *output 1* **else**

Step 4.1: **For** $i = 0$ **to**s **do**

Step 4.1.1: **If** $a_0 = -1$ **then** *output 1* **else**

Step 4.1.1.1: $a_0 = a_0^2 \pmod{n}$

Step 5 *output* 0

This algorithm is clearly a polynomial-time algorithm and the details of the proof of that fact are left for the interested reader.

Notice again that if the answer is that n is compose then certainly n is not prime. But if the answer is n is prime, then n could still be composite, though the probability that we get the wrong answer is at most $1/2$.

Theorem 9.3.8

If n *is a prime number then* $\mathbb{MR}(n) = 1$.

Proof: Assume that n is prime and $\mathbb{MR}(n) = 0$. Notice that if $\mathbb{MR}(n) = 0$ then it must be the case that $a^d \equiv 1 \pmod{n}$. Let a be the sequence of values tested in the algorithm. Since a_i is squared in each iteration of the **For** loop this means that it is tested the values of the form a^d, $a^{2d}, \cdots, a^{2^{s-1}d}$. Hence, it must be case that $a^{2^i d} \not\equiv -1 \pmod{n}$ for all $0 \leq i < s-1$.

On the other hand, since n is is prime, by Fermat's little theorem 2.3.5 one concludes that $a^{2^s} \equiv 1 \pmod{n}$ since $n - 1 = 2^s d$ which means that $a^{2^{s-1}d}$ is a square root of 1 modulo n. Since n is prime, there are only two square roots of 1 modulo n, namely, $\pm 1 \pmod{n}$. Since $a^{2^{k-1}d} \not\equiv -1 \pmod{n}$, then $a^{2^{k-1}d} \equiv -1 \pmod{n}$. By repeating this process with $a^{2^{k-2}d}$ and so on obtains $a^m \equiv 1 \pmod{n}$, which is a contradiction, since the algorithm would have outputted 1 in that case. ■

• • •

9.3.4 The AKS algorithm proving that $\mathbb{P}\mathtt{rimes} \in \mathbb{P}$

In this section, it is presented the AKS primality test. The name AKS is due to the fact that it was purposed in 2002 by Agrawal, Kayal and Saxena in **[?]** and is was one of the major breakthrough on computational complexity problems with clear applications to cryptography. Notice that from the fact that one can test is a number is prime or not, does not any anything about, in the case that a number is not prime, of its prime factors.It is the first provably deterministic algorithm that one can use to determine whether a given number n is prime or not within a time which is guaranteed to be polynomial in the number of digits, in fact $O(\log^c n)$ for some constant c that will be determine later on in this section. In the previous sections, all primality testing algorithms were probabilistic but their advantage concerning this algorithm that is deterministic, is the fact that their running time is much slower, in fact a degree of the polynomials involved are at most cubic while this one is, a polynomial that for practical reasons is very high. This question is related with Factoring problem but, for the moment of the writing of these notes, they are not known to be equivalent from the computational complexity point of view.

The story of this algorithm starts in 1999 with Agarwal and Biswas when they proposed a probabilistic algorithm that picking a monic polynomial $Q \in \mathbb{Z}_n[x]$ of degree $poly(\log n)$ uniformly at random and test if $(x+a)^n \equiv x^n + a \pmod{Q}$ to derive thew conclusion of

primality of n. Since the degree of Q is small then this algorithm is an efficient one and one can use power method to compute $(x+a)^n \pmod{Q}$ in $poly \log n$ time. The correctness follows from the facts: 1) that with probability at least $\frac{1}{\deg Q}$, Q is irreducible over $\mathbb{Z}_n[x]$. Notice that if $d = \deg Q$, then $x^{q^d} - x = \prod_p p$ where in the product are considered all polynomials p that irreducible and have degree which divides d and since there are at least $\frac{q^d}{d}$ irreducible polynomials of degree dividing d, and since there are a total of q^d choices for monic polynomials of Q over $\mathbb{Z}_p[x]$, at least with probability $\frac{1}{d}$ we will get a polynomial Q that is irreducible over $\mathbb{Z}_p[x]$, and then of course Q is also irreducible module n. Also the probability that $(x+a)^n \equiv x^n + a \pmod{Q}$ if n is composite is very very small due to the Chinese Reminder Theorem. In fact, if $(x+a)^n \equiv x^n + a \pmod{Q})$ holds for many polynomials $Q_1, Q_2, \ldots, Q_t$'s, then the congruence also holds module with the least common multiplier due to Chinese Reminder Theorem, but when the degree of that polynomial exceeds n one has $(x+a)^n \equiv x^n + a \pmod{n}$ as well because the degree on both sides are only n, contradicting the fact that n is composite.

So the basic idea of AKS algorithm is to explore polynomial identities. In fact, it relies on polynomial identities that are known to be true modulo prime numbers and cannot hold for non-prime numbers since if they were true than a large number of additional polynomial identities could be generated, eventually violating the factor theorem that asserts that a polynomial identity of degree at most d can be obeyed by at most d values of the unknown variable. Consider again the Fermat's little theorem presented in Theorem 2.3.5 on page 22 that states that for any prime number p, $a^p \equiv a \pmod{p}$. On may think that this theorem can be used to primality test with the following argument: to test whether a number n is prime, pick a few values of a and see whether $a^n \equiv a \pmod{n}$ and determine that n is not prime if the equality fails for some a. Unfortunately, by the same reason that the Solovay–Strassen algorithm is Monte-Carlo one, the converse of this argument is also not true, i.e., there are non-prime numbers, for example, the Carmichael numbers, for which the stated equality holds for all numbers that are co-prime to n. So Fermat's little theorem cannot be used, by itself, to establish primality for general numbers, because it is too weak to eliminate all non-prime numbers.[3] However, there is a stronger version of Fermat's little theorem which does eliminate all non-prime numbers and is called *polynomial identity* that is stated below. The

[3]The situation improves though for more special types of number. In particular,, for the Mersenne numbers.

identity presented obviously implies the Fermat's little theorem, but the proof also relies on the Fermat's little theorem.

Theorem 9.3.9 (*polynomial identity*)

Let n and a be two numbers.

- *If n is prime then $(x-a)^n \equiv x^n - a \pmod{n}$.*
- *If n is composite and $gcd(a,n) = 1$, then $(x-a)^n \not\equiv x^n - a \pmod{n}$.*

Proof: By Newton's binomial expansion,

$$(x-a)^n = \sum_{i=0}^{n} \binom{n}{i} x^i a^{n-i}$$

Thus, if n is prime then $\binom{n}{i}$ then $n \left| \binom{n}{i} \right.$ for all $1 < i < n$

On the other hand if n is a composite number then there are other terms in the Newton's binomial expansion that is not divisible by n. It is left for the interested reader, to give an example of such a term.

Conversely, if $(x-a)^n \equiv x^n - a \pmod{n}$ for some a coprime to n, then by comparing coefficients using the binomial theorem we see that $\binom{n}{i}$ is divisible by n for all $1 \leq i \leq n$. But if n is divisible by some smaller prime p, then by setting i equal to the largest power of p that divides n, one sees that $\binom{n}{i}$ is not divisible by enough powers of p to be divisible by n, a contradiction. Thus one can use $(x-a)^n \equiv x^n - a \pmod{n}$ (for a single value of a coprime to n) to decide whether n is prime or not. ■

So the idea of AKS seems simple. Just by looking at all the elements of the n^{th} row of Pascal's triangle one can decide if n is prime or not. At this point the reader should be alerted that this simple idea does not work efficiently and the reason why is simply: the n^{th} row of Pascal's triangle has $n+1$ elements which is exponential on the size of n, which is $O(\log n)$. So in oder to design an algorithm that runs in polynomial time for testing primality one needs to "reduce" all these terms up to just the computation of a polynomial number of them.

This can be done by descending to a quotient ring of $\mathbb{Z}_n[x]$, such as $\mathbb{Z}_n[x]/(x^r - 1)$ for some r. Clearly, if the identity $(x-a)^n = x^n - a$ holds in $\mathbb{Z}_n[x]$, then it will also hold in $\mathbb{Z}_n[x]/(x^r-1)$, thus

$$(x-a)^n \equiv x^n - a \pmod{n, x^r - 1}$$

Notice that with this trick, if r is not too large (i.e., it is at most a logarithmic term of n, which means that the length of r is polynomial on the length of n), the left-hand side of equality presented above can now be computed efficiently by expanding n in binary and performing repeated squaring, because all polynomials can be reduced to be of degree less than r, rather than being as large as n as it will be explained below. Indeed, if $r \in O(\log^c n)$ for some constant c, then one can induce roots modulo n and thus test the validity of the equality $(x-a)^n \equiv x^n - a \pmod{x^r - 1}$ in time polynomial time on the size of n as stated in the next theorem:

Theorem 9.3.10

Let n, a and r be non-nulls integers. Assume that for some constant c, for all $r \leq O(\log^c n)$, $(x-a)^n \equiv x^n - a \pmod{n, x^r - 1}$ holds, and $gcd(a,n) = 1$. Then n is either a prime number, or a power of a prime number.

Notice that the computation of $gcd(a,n) = 1$ can be quickly tested using the $\mathbb{E}\mathrm{uclides}$ algorithm **??**, and if $gcd(a,n) \neq 1$ then n is of course composite. On the other hand, it is easy to quickly test for the property that n is a power of an integer. The reader is asked to design an algorithm to test this property in polynomial time. The idea is to compute the roots $n^{1/k}$ for $1 \leq k \leq \log^2 n$, and such powers are clearly composite. Thus, combining this result with the equality $(x-a)^n \equiv x^n - a \pmod{n}$, one can design a deterministic polynomial-time test for primality.

Theorem 9.3.11 (*AKS theorem, key step*)

Let n, a and r be non-nulls integers be such that $gcd(r,n) = 1$ and such that n has order greater than $\log^2 n$ in the multiplicative group $\mathbb{Z}_r^*$. Assume that for all $1 \leq r \leq O(\log^c n)$ for some c, one has $(x-a)^n \equiv x^n - a \pmod{n, x^r - 1}$, and $gcd(a,n) = 1$. Then n is either a prime number, or a power of a prime number.

To find an with the above properties we have

Lemma 9.3.13 (Existence of good r)**.** *Let n, a and r be non-nulls integers. There exists $1 \leq r \leq O(\log^c n)$ coprime to n, such that n has order greater than $\log^2 n$ in the multiplicative group $\mathbb{Z}_r^*$.*

Proof: For each $1 \leq k \leq \log^2 n$, the number $n^i - 1$ has, by the fundamental theorem of arithmetic, at most $O(\log^c n)$ for some c prime divisors. If one picks r to be the first prime not equal to any of these prime divisors, one obtains the claim. ■

It is clear that Theorem 9.3.10 follows from Theorem 9.3.11 and the Lemma 9.3.13. It is then sufficient to prove Theorem 9.3.11.

Proof:[of Theorem 9.3.11] If Theorem 9.3.11 fails then it means that n is divisible by some smaller prime p, but is not a power of p. Since n is coprime to all numbers of size $O(\log^c n)$ for all c, i.e. p is not of polylogarithmic size. Moreover, since r is coprime to n, it follows that r is not a multiple of p.

Let $\mathbb{F}$ be a field $\mathbb{F} = \mathbb{F}_p/h(x)$ where $h(x)$ is a factor in $\mathbb{F}_p$ of the r^{th} cyclotomic polynomial

$$\Phi_r(X) = \prod_{1 \leq k \leq n-1 \wedge gcd(n,k)=1} (x - e^{2i\pi k/n}).$$

From the hypothesis that $(x+a)^n = x^n + a \pmod{n, x^r - 1}$, one concludes that

$$(x+a)^n = x^n + a$$

in $\mathbb{F}$ for all $1 \leq a \leq c' r \log^c n$ for some c' and all c'. Notice that n is coprime to every integer less than $c' r \log^c n$, and thus $c' r \log^c n < p$.

Also $(x+a)^p = x^p + a$ also holds in $\mathbb{F}$ for all such a. These two equations give

$$(x+a)^{n/p} = (x^p)^{n/p} + a$$

Notice that the p^{th} power the r^{th} of a primitive root of unity x is again a primitive r^{th} root of unity [4] and hence we also have

[4] The converse also holds, in the sense that all primitive r^{th} root is of this form

$$(x+a)^{n/p} = x^{n/p} + a$$

in $\mathbb{F}$ for all $1 \leq a \leq c'r\log^c n$. Call these numbers introspective, i.e, a is introspective if $(x+a)^m = \Psi_m(x+a)$, where Ψ is the the ring homomorphism sending x to x^m.

Thus, one has that p, n and n/p are introspective.It is left, as an exercise for the reader, to prove that 1 is also trivially introspective and the introspective property is multiplicative, i.e., if m and m' are introspective, then mm' is also introspective.

Hence there are lot of introspective integers. In fact, any number of the form $p^i(n/p)^j$ for all $i, j \geq 1$ are introspective.

The conclusion of the proof of the Theorem follows from the fact that it is not possible to create so many different introspective numbers. Otherwise, many polynomial identities in the field would eventually violate the factor theorem. The incompatible bounds are prover in the next two lemmas. ■

Let $\mathbb{G} \subset \mathbb{F}[x]^*$ be the multiplicative group generated by $x+a$ for $1 \leq a \leq\leq c'r\log^c n$ as in the previous proof. Notice that $z^m = \Psi_m(z)$ for all $z \in \mathbb{G}$.

Lemma 9.3.14 (Lower bound on $ord(\mathbb{G})$)**.**

$$ord(\mathbb{G}) \geq 2^t.$$

Proof: Let $b = c'r\log^c n$ and let $p(x)$ be a product of less than t of the polynomials $x+1, \cdots, x+b$ with repetitions allowed, then $p(x) \in \mathbb{G}$.

Since $a \geq 2r \geq 2t$ there are certainly at least 2^t ways to pick such a product. So to establish the proposition it suffices to show that all these products are distinct.

Suppose for contradiction that $p(x) = q(x)$, where p and q are different products of less than t of the $x+1, \cdots, x+b$. Then, for every introspective m, $p(x^m) = q(x^m)$ as well since $p(x^m) = \Psi_m(p(x))$.

In particular, this shows that $x^{m_1}, \cdots, x^{m_t}$ are all roots of the polynomial $p-q$. But this polynomial has degree less than t, and the are distinct by hypothesis, and we obtain the desired contradiction by the factor theorem. ■

Lemma 9.3.15 (Upper bound on $ord(\mathbb{G})$)**.** *Suppose that there are exactly t residue classes modulo r of the form $p^i(n/p)^j \pmod r$ for $i, j \geq 0$. Then $ord(\mathbb{G}) \leq n^{\sqrt{t}}$.*

Proof: By the pigeonhole principle, it must exists a collision modulo r, i.e., $p^i(n/p)^j \equiv p^{i'}(n/p)^{j'} \pmod r$ for some $0 \leq i, j, i', j' \leq \sqrt{t}$ with $(i, j) \neq (i', j')$.

Setting $m = p^i(n/p)^j$ and $m' = p^{i'}(n/p)^{j'}$, thus there are two distinct introspective numbers m, m' of size most $n^{\sqrt{t}}$ which are equal modulo r. The distinctness of m and m' follows from the fact n is not a power of p. This implies that $\Psi_m = \Psi_{m'}$, and thus $z_m = z_{m'}$ for all $z \in \mathbb{G}$. But the polynomial $z^m - z^{m'}$ has degree at most $n^{\sqrt{t}}$, and the claim now follows from the factor theorem. ■

Now it is presented the original algorithm AKS and a subroutine use in it. It is worthwhile to mention that one can improve this version to incorporate some preliminary tests. For sake of understanding, in the algorithm $o_a(b)$, where $gcd(a, b) = 1$ denotes the order of b in $\mathbb{Z}_a^*$, i.e., the smallest number k such that $b^k \equiv 1 \pmod a$.

Algorithm 9.3.16 (𝔸𝕂𝕊 - *Primality test*)

Input: *An integer n;*

Output: *1 if n is prime and 0 otherwise;*

Step 1: **If** $n = a^b$ *for some* $a \in \mathbb{N}$ *and* $b > 1$ **then** *output* 0*;*

Step 2: *Use* $\mathbb{F}\mathrm{ind}(n)$ *to find the smallest r such that* $o_r(n) > \log^2(n)$.

Step 3: **If** $1 < gcd(a, n) < n$ *for some* $a = r$*, output* 0*;*

Step 4: **If** $n \geq r$*, output 1;*

Step 5: **For** $a = 1$ **to** $\sqrt{\phi(r)} \log n$ **do**

Step 5.1: **If** $(x + a)^n \neq x^n + a \pmod{X^r - 1, n}$*, output* 0*;*

Step 6: *Output 1;*

The routine $\mathbb{F}\mathrm{ind}(r, n)$ is:

Algorithm 9.3.17 ($\mathbb{F}\mathrm{ind}(n)$)

Input: *An integers n;*

Output: 0 *if the process fails, or r the smallest one such that $o_r(n) > \log^2(n)$;*

Step 1: $r = 1$.

Step 2: **While** $r < n$ **do**

Step 2.1: **If** $gcd(r, n) > 1$ *output* 0 **else**

Step 2.1.1: *Set q be the largest prime factor of r;*

Step 2.1.2: **If** $a \geq 4\sqrt{r}\log n$ *and* $n^{\frac{r-1}{q}} \not\equiv 1 \mod r$, *output r;*

Step 2.2: $r = r + 1$.

The complexity of $\mathbb{F}\mathrm{ind}(r, n)$ can be established using the following lemma:

Lemma 9.3.18. *Let n be a positive integer. Let r be an appropriated number for the $\mathbb{AKS}$ Algorithm on Step 2. Then $r \leq max\{3, \log^5 n\}$ such that $o_r(n) > \log^2 n$.*

Proof: The cases $r = 2$ and $r = 3$ are easy to see that verifies the conditions of the lemma and are left as exercises for the reader.

Assume that $n > 2$. Then, $\lceil \log^5 n \rceil > 10$ and thus, if $lcm(n)$ represents the least common multiplier of all number up to n, then $lcm(n) > 2^n$.[5]

Notice that the largest value of k such that for any number of the form $m^k = B = \lceil \log^5 n \rceil$ for $m \geq 2$, is $\lfloor \log B \rfloor$. Consider the smallest number r that does not divide

$$n^{\lfloor \log B \rfloor} \cdot \prod_{i=1}^{\lfloor \log^2 n \rfloor} (n^i - 1).$$

$gcd(r, n)$ is not divisible by all the prime divisors of r since otherwise, by the discussion above r would divide $n^{\lfloor \log B \rfloor}$. Therefore, $r/gcd(r, n)$ does not divide the above product. From the fact that r is the smallest number not dividing the product above, it follows that $gcd(r, n) = 1$.

[5]The reader should find no troubles proving this result. In fact, if $n > 7$, then result holds.

Furthermore, since r does not divide any of $n^i - 1$ for $1 \leq i \leq \lfloor \log^2 n \rfloor$, by Lagrange theorem **??**, $o_r(n) > \log^2 n$. Finally

$$\begin{aligned} n^{\lfloor \log B \rfloor} \cdot \prod_{i=1}^{\lfloor \log^2 n \rfloor} (n^i - 1) &\leq n^{\lfloor \log B \rfloor + \log^2 n \cdot (\log^2 n - 1) \cdot 1/2} \\ &\leq n^{\log^4 n} \leq 2^{\log^5 n} \leq 2^B \end{aligned}$$

■

The step 2.2.2 implies that $\lfloor \sqrt{\phi(n)} \log n \rfloor$ equations of the form $(x + a)^r \equiv x^r + a$, i.e., such that a are introspective numbers. The reader is invited to do the complexity analyses of the last algorithms based on the discussion above and conclude that, in fact, the $\mathbb{AKS}$ runs in polynomial time on the size of n.

• •

SECTION 9.4

Attacking the RSA - Factorizing n

Having a "secure" scheme like RSA does not really mean that its use is secure. As it will be seen in the exercises, a sloppy use of the schemes can lead to an insecure scheme.

Despite this fact, if the RSA is wisely used and the choice of the keys are made in such a way that an opponent can not infer any of its factor easily then, as the widely use of this encryption scheme suggests, the RSA seems to be, in practice, secure since no one knows any feasible attack that can be used.

Notice that the most obvious way to attack the RSA Cryptographic system is to try to factor n as it would reveal the primes and, thus, an opponent would also be able to invert the public key modulo n to get the private key that only the receiver should know and hence be able to decrypt the messages sent. Several theoretical attacks are known concerning this approach to break RSA and for that reason, theoretically speaking, RSA is far from being secure. But what about the practicality of these attacks? That is the point that enables the RSA to be used in practice. The point is that the bridge between theory and practice is not yet a reality. In this section several well known algorithms that One of this theoretical attacks is due to Peter Shor

that showed in 1994 that using a quantum computer and its respective power (that derives from quantum mechanic properties) it is possible to factorize a number in polynomial time. As the reader is aware, the price to pay to have a polynomial time factorization and thus an algorithm to brake RSA is just the realization of a quantum computer. At the time of writing these notes, this kind of technology is not yet affordable, since there is only one quantum computer available in the market for 10 million dollars. Another issue concerning the RSA, is the fact that one does not know if, in fact the problem of factorization is a problem to be in $\mathbb{P}$.

In this section it is analyzed several algorithms that one can use to try to factorize numbers including the Shor's one that will require a brief introduction to quantum computing.

Throughout this section it is assume that n is a large odd composite number. Notice that n always have a prime factor that is at most $\sqrt{n}$. Notice also that the case where n is even it is very easy to get a factor, in case 2, and as suggested in an initial exercise, checking parity is trivially a routine that one can perform in polynomial time. Also, the verb factoring has two interpretations. Either it is asked for a complete factorization of the number into primes or one can simply ask for a non trivial single factor. The algorithms that will be studied in this section are related with the second approach, i.e., their goal is to find a non trivial factor of n. It is worthwhile mention that one can use the latter case to infer if a complete factorization of n is accomplished, since, if one assumes that $n = rt$ for some r and t, one can, for example, use the $\mathbb{AKS}$ and test if r and t are primes in polynomial time.

• • •

9.4.1 Pollard's $p-1$ method

The Pollard algorithm was first presented in 1974 and is a simple algorithm that takes advantage from the assumption that if p is a large prime then it is probable that $p-1$ has a prime factorization where all its primes are small when compared with p. The inputs for $\mathbb{P}\mathtt{ollard}p-1$ is an odd number n and a fixed bound, B that can be seen as a guess for the size of the largest prime factor of $p-1$. The basic idea is to form the product over all primes $q \leq B$

$$k = \prod_{q \leq B} q^{\left\lfloor \log_q B \right\rfloor}.$$

Thus, it is expectable that k is highly divisible by all the small primes that one also expects to divide $p-1$, and hence one expects that $p-1$ to divide k. So, by Fermat's little theorem presented on page 22 it is known that $p|a^k-1$ for all $a \in \mathbb{Z}_p - \{0\}$ and hence clearly $p|d$ where d is the greatest common divisor of a^k-1 and n and so using the $\mathbb{Euclides}$ algorithm one can find efficiently on the size of n a nontrivial factor d of n.

Algorithm 9.4.1

Input: *n and a bound B;*

Output: *either a non trivial factor of n or failure;*

Step 1: $a = 2$*;*

Step 2: **For** $j = 2$ **to** B **do**

Step 2.1: $a = a^j \pmod{n}$*;*

Step 3: $d = \mathbb{Euclides}(a-1, n)$*;*

Step 4: **If** $1 < d < n$ **then** *output d* **else** *output failure;*

This algorithm, runs in $O(B \log B(\log^2 n) + \log^3 n)$ steps, since it requires only $B-1$ modular exponentiations each costing at most $2\log^2 B$ steps by using the $\mathbb{SQUARE} - \mathbb{AND} - \mathbb{MULTIPLY}$ algorithm presented in Page 182, and the computation of the *gcd* can be performed in time $O(\log^3 n)$. Thus if B is of order $\log^c n$ for some integer c, then the algorithm is indeed a polynomial-time algorithm as a function of the size of n. The drawback of this algorithm is that with such a choice of B, the probability of success will be very small. On the other hand, if the size of B is near $\sqrt{n}$, then the algorithm is guaranteed to be successful, but it will be no faster than trial division.

In the next example it is presented an use of the Polaroid method for factoring an integer n.

Example 9.4.2. *Let n be the number* 6380651 *and choose $B = 9$. Thus:*

$$k = 2^3 \cdot 3^2 \cdot 5 \cdot 7 = 2520 = 2^{12} + 2^8 + 2^7 + 2^6 + 2^4 + 2^3.$$

The computation of k can be easily done by repeated squaring modulo 6380651

$$\begin{array}{llll} 2^1 = 2 & 2^2 = 4 & 2^4 = 16 & 2^8 = 256 \\ 2^{16} = 65536 & 2^{32} = 789173 & 2^{64} = 4202423 & 2^{128} = 5994431 \\ 2^{256} = 5409973 & 2^{512} = 4188467 & 2^{1024} = 440743 & 2^{2048} = 1853005, \end{array}$$

so

$$2^k - 1 \equiv 256 \cdot 65536 \cdot 420243 \cdot 5994431 \cdot 5409973 \cdot 1853005 - 1 \equiv 6347878 \pmod{6380651}.$$

Using Euclides *algorithm to determine the gcd*$(6380651, 6347878)$ *one gets as output* 2521*, since*

$$6380651 = 6347878 + 32773$$

$$6347879 = 193 \cdot 32773 + 22689$$

$$32773 = 22689 + 10084$$

$$22689 = 2 \cdot 10084 + 2521$$

$$10084 = 4 \cdot 2521 + 0,$$

So $2521|6380651$*, giving the factorization of n as:*

$$6380651 = 2521 \cdot 2531.$$

There exists a derivation of this method called Pollard Rho method that takes advantage of some other properties of divisors of numbers to zoom in on a factor quite quickly. It is suggested to the reader the reading of [Sti06] for details on this method of factorization.

• • •

9.4.2 Dixon's random square algorithms

Dixon's algorithm is an improvement over Fermat's factorization method[6] that tries to find x and y efficiently by computing $x, y \in \mathbb{Z}_n$ such that $x^2 \equiv y^2 \pmod{n}$. Then with probability $\geq$

[6]Fermat's factorization method is basically a method to find integers x and y such that $n = x^2 - y^2 = (x + y)(x - y)$ and uses that fact to factor n.

$\frac{1}{2}$, $x \neq \pm y \pmod{n}$, and hence $\gcd(x-y,n)$ can be used to infer a nontrivial factor of n. For of understanding consider the following example: $10^2 \equiv 3^2 2 \pmod{77}$ and $gcd(10+32,77) = 7$, one find that 7 is a factor of 77.

The material presented in this subsection follows closely the Stinson's book.

The idea to get x and y with the Random Squares algorithm is to find a set B of the b smallest primes, for an appropriate value b. First several integers z are consider such that all the prime factors of $z^2 \mod n$ occur in B and then take the product of a subset of these z's in such a way that every prime in the factor base is used an even number of times which can be used to obtain a congruence of the type $x^2 \equiv y^2 \pmod{n}$.

Assume $B = \{p_1, \cdots, p_b\}$ and let $c = b + 4$. Assume also that the following c congruences are obtained:

$$z_j^2 \equiv p_1^{\alpha_{1_j}} \cdots p_b^{\alpha_{b_j}} \pmod{n}$$

where $1 \leq j \leq c$ and for each such j, consider the vector

$$a_j = (\alpha_{1_j} \pmod 2, \cdots \alpha_{b_j} \pmod 2) \in {\mathbb{Z}_2}^b.$$

Thus if one can find a subset of the α_j such that the above vector is zero then the product of the corresponding z_j's will use each factor in B an even number of times.

Notice that finding a subset of such c vectors corresponds exactly to finding a linear dependence of these vectors which is guaranteed to exist since $c > b$ and it can be found easily using the standard method of Gaussian elimination.

So all that is left to do is to explain how one can get the integers z such that the values $z^2 \pmod{n}$ factor completely over B. One way is simply to choose the z's at random which yields the so-called Random Squares algorithm. However, it is particularly useful to try integers of the form $j + \lceil \sqrt{kn} \rceil$, with $j = 0, 1, 2, \cdots$ and $k = 1, 2, \cdots$. These integers tend to be small when squared and reduced modulo n, and hence they have a higher than average probability of factoring over B.

Example 9.4.3. *Consider* $n = 15770708441$ *and select* $b = 6$. *Then* $B = \{2, 3, 5, 7, 11, 13\}$ *and one*

has the following three congruences:

$$\begin{aligned} 83409341562 &\equiv 3 \cdot 7 \pmod{n} \\ 120449429442 &\equiv 2 \cdot 7 \cdot 13 \pmod{n} \\ 27737000112 &\equiv 2 \cdot 3 \cdot 13 \pmod{n} \end{aligned}$$

Then, the product of these three congruences, one also has

$$\begin{aligned} (8340934156 \cdot 12044942944 \cdot 2773700011)^2 &\equiv (2 \cdot 3 \cdot 7 \cdot 13)^2 \pmod{n}. \Leftrightarrow \\ 95034357852 &\equiv 5462 \pmod{n} \end{aligned}$$

So $\mathbb{EUCLIDES}(9503435785 - 546, 15770708441) = 115759$*, and thus* $115759 | 15770708441$*.*

Now one can consider the following vectors:

$$a_1 = (0, 1, 0, 1, 0, 0)$$

$$a_2 = (1, 0, 0, 1, 0, 1)$$

$$a_3 = (1, 1, 0, 0, 0, 1)$$

It follows that

$$a_1 + a_2 + a_3 = (0, 0, 0, 0, 0, 0) \pmod{2}.$$

This gives rise to the congruence above that successfully allows a factorization of n.

How big should be b and thus what is the complexity of the algorithm that implements this idea? In general if b is large, then it is easier that an integer $z^2 \pmod{n}$ factors over B but in that case more congruences are needed to accumulate before one can find a dependence relation. A good choice for b is $m/\ln m$ and the details why this choice is a good one are omitted here and the reader is referred to Stinson's book [Sti06] for a detailed explanation. The resulting complexity for this choice of b is $O(e^{(1+o(n))\sqrt{\ln n \ln\ln n}})$.

...

9.4.3 Shor's algorithm for factorization

...

9.4.3.1 quantum mechanics

...

9.4.3.2 The algorithm and its explanation

.....

SECTION 9.5

Attacking the RSA - other attacks

In this section it is discussed the possibility of attacking the RSA cryptographic system other than just trying to factorizing n directly. One could try, for example, extract the information from other parts of the code. One could for example extract information about the factor of n by analyzing $\phi(n)$. In fact if n and $\phi(n)$ are known then one can factor n as explained below.

Assume that $n = pq$ and thus $\phi(n) = (p-1)(q-1)$ with p and q being unknown. Since $n = pq$ one can replace q by n/p on the computation of $\phi(n)$ to obtain the following quadratic equation in p:

$$p^2 - (n - \phi(n) + 1)p + n = 0$$

One can easily see that the two roots of this equation are p and q, the exact factor of n.

Another approach to attack RSA is an attack called Wiener's low decryption exponent attack that can be performed to determine the decryption exponent a if it satisfies the following conditions:

$$3a < \sqrt[4]{n} \qquad q < p < 2q$$

By the choices of a and b in the RSA, one knows that there exists $t < a$ such that $ab - t\phi(n) = 1$.

On the other hand, since $q < p$ then $q < \sqrt{n}$ and thus

$$\begin{aligned} n - \phi(n) &= p + q - 1 \\ &< 2q + q - 1 < 3q \\ &< 3\sqrt{n} \end{aligned}$$

and thus

$$\begin{aligned} \left|\frac{b}{n} - \frac{t}{a}\right| &= \left|\frac{ab - nt}{na}\right| = \left|\frac{1 + t(\phi(n) - n)}{na}\right| \\ &< \frac{3t\sqrt{n}}{an} < \frac{3t}{a\sqrt{n}} \\ &< \frac{1}{3a^2} \end{aligned}$$

i.e., the fraction t/a is very close to b/n and thus one can use the continued fraction expansion of t/a to obtain b/n.

A continued fraction expansion is an m tuple of the form

$$[q_1, \cdots, q_m] = q_1 + \cfrac{1}{q_2 + \cfrac{1}{q_3 + \cdots \cfrac{1}{q_m}}}$$

One says that $C_j = [q_1, \cdots, q_j$ for all $j < m$ is the j^{th}-convergent of $[q_1, \cdots, q_m]$. The convergent have the following property that is the base for the Wiener's algorithm.

Theorem 9.5.1

Let a, b, c and d be natural numbers such that $gcd(a, b) - gcd(c, d) = 1$ and $\left|\frac{a}{b} - \frac{c}{d}\right| \leq \frac{1}{2d^2}$. Then c/d is one of the convergents of the continued fraction expansion of a/b.

Notice that since b and n are public in the RSA cryptographic system then computing its convergent is easy and then all that is left is a method to test which is the right one. Notice that if t/a is a convergent of b/n then $\pi(n) = \frac{ab - 1}{t}$ and thus one can apply the method above to compute the factorization of n. The difficulty is that one cannot identify ahead which convergent will lead to the factorization of n and thus it is necessary to test all until the right one is found. In the next algorithm it is presented the Wiener's algorithm. the guarantee of success of this algorithm is precisely the condition expressed above.

The details are left for the interested reader as an exercise.

Algorithm 9.5.1 (*Wiener's algorithm*)

Input *The numbers b and n.*

Step 1: $r_0 = b$*;*

Step 2: $r_1 = n$*;*

Step 3: $m = 1$*;*

Step 4: **While** $r_m \neq 0$ **do**

Step 4.1: $q_m = (r_{m-1} - 1) \div r_m$

Step 4.2: $r_m = r_{m-1} - q_m r_m$*;*

Step 4.3: $m = m + 1$*;*

Step 5: $c_0 = 1$*;*

Step 6: $c_1 = q_1$*;*

Step 7: $d_0 = 0$*;*

Step 7: $d_1 = 1$*;*

Step 7: **For** $j = 2$ **to**m **do** *:*

Step 7.1: $c_j = q_j c_{j-1} + c_{j-2}$*;*

Step 7.2: $d_j = q_j d_{j-1} + d_{j-2}$*;*

Step 7.3: $n' = (d_j b - 1)$ *(notice that $n' = \phi(n)$ if c_j/d_j is the correct convergent);*

Step 7.4: **If** n' *is an integer* **then**

Step 7.5.1: *Determine p and q the square roots of $x^2 - (n - n' + 1)x + n = 0$;*

Step 7.5.2: **If** p *and* q *are positive integers less than* n *output* p *and* q*;*

Step 8: **else** *output failure*

SECTION 9.6

Attacking RSA based on partial information leaked

In every cryptographic system, one of the main concerns is the fact that leaking some partial information might be enough to mount an attack that reveals completely the secret information, namely about the plaintext. This concern also happens with the RSA cryptographic system.

In particular, in this section it is study the (im)possibility of computing the properties like $\mathbb{HALF}_k(y)$ or $\mathbb{PARITY}_k(y)$, since if one could compute such information for all plaintexts then one could recover the information of any plaintext. Notice that this information is related with a single bit (in fact, the least significant one) and one can decrypt the original message. As it will be explained in the next paragraphs, is for some reason the choices of n or b would lead to cipher text that have the same parity then this key is very weak for a secure RSA encryption.

Consider an instantiation of the RSA cryptographic system for some which n and b are the public part of the key. The properties $\mathbb{HALF}_b(y)$ or $\mathbb{PARITY}_b(y)$ are defined as follows:

$$\mathbb{HALF}_b(y) = \begin{cases} 1 & \text{if } y = e_b(x) \text{ and } x \text{ is odd} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbb{PARITY}_b(y) = \begin{cases} 0 & \text{if } y = e_b(x) \text{ and } 0 \leq x < \dfrac{n-1}{2} \\ 1 & \text{otherwise} \end{cases}$$

The first observation concerning these two properties is that they are inter drivable as the next theorem proves.

Theorem 9.6.1

In the RSA cryptographic system, $\mathbb{HALF}_b(y)$ and $\mathbb{PARITY}_b(y)$ are inter drivable.

Proof: First observe that the RSA cryptographic system satisfies the following property:

$$e_b(x_1x_2) = e_b(x_1)e_b(x_2)$$

Thus:

$$\begin{aligned}\mathbb{PARITY}_b(y) &= \mathbb{PARITY}_b(e_k(x)) \\ &= \mathbb{HALF}_b(e_k(x2^{-1})) \\ &= \mathbb{HALF}_b(e_k(x)e_k(2^{-1}))\end{aligned}$$

Notice that since it is assumed that $n = pq$ where p and q are two large primes, then n must be odd and thus, $gcd(n, 2) = 1$ which implies that $2 \in \mathbb{Z}_n^*$ and thus it must have an inverse. For convenience herein is called 2^{-1}.

On the other hand, it is easy to see that

$$\mathbb{HALF}_b(y) = \mathbb{PARITY}_b(e_k(x)e_k(2))$$

■

The result above prove that from parity one can determine the half property and vice versa within a polynomial time computation. This statement is easy to verify to be true from the derivations made in the above proof.

In the next algorithm it is shown that if one has oracle access to the computation of $\mathbb{HALF}$ then one can decrypt RSA.

The fundamental observation that allows this attack to work is that

$$\begin{cases} \mathbb{PARITY}_b(2x) = 0 & \text{if } x \in \left[0, \frac{n}{4}\right[\cup \left[\frac{n}{2}, \frac{3n}{4}\right[\\ \mathbb{PARITY}_b(4x) = 0 & \text{if } x \in \left[0, \frac{n}{8}\right[\cup \left[\frac{n}{4}, \frac{3n}{8}\right[\cup \left[\frac{n}{2}, \frac{5n}{8}\right[\cup \left[\frac{3n}{4}, \frac{5n}{8}\right[\\ & \ldots \\ \mathbb{PARITY}_b(2^ix) = 0 & \text{if } x \in \bigcup_{j=0}^{2^i-1} \left[\frac{jn}{2^i}, \frac{(2j+1)n}{2^{i+1}}\right[\end{cases}$$

Algorithm 9.6.1 (*Decrypting RSA based on an oracle for* $\mathbb{HALF}$)

Input *y, n and b such that $y = e_b(x)$, where e_b is the RSA encryption function with public part n and b;*

Output *x;*

Oracle $\mathbb{HALF}$*;*

Step 1 **For** $i = 0$ **to** $\lfloor \log_2 n \rfloor$ **do**

Step 1.1 $y_i = \mathbb{HALF}_b(y)$*;*

Step 1.2 $y = y \cdot e_b(2) \pmod{n} = y \cdot 2^b \pmod{n}$*;*

Step 2 $l = 0$*;*

Step 3 $h = n$*;*

Step 4 **For** $i = 0$ **to** $\lfloor \log_2 n \rfloor$ **do**

Step 4.1 $m = (h - l)/2$*;*

Step 4.2 **If** $y_i = 1$ **then** $l = m$ **else** $h = m$*;*

Step 5 *Output* $\lfloor h \rfloor$*;*

The read is invited to prove that if $\mathbb{HALF}$ is polynomial then, the algorithm presented above runs in polynomial time and, in fact, the value outputted is just the plaintext.

This example shows the fragility of RSA if the parity of the messages can be computed easily. It is worthwhile mention that there are also other kinds of information of the plaintexts that one opponent can use to attack the RSA cryptographic system and each of them require a separated proof of security.

SECTION 9.7

The Rabin's cryptographic system and its relationship with RSA

In this section, it described the Rabin's cryptographic system and explored its connection to RSA. This cryptographic system was presented in 1979 by Michael Rabin, in an effort to improve the RSA cryptographic system. This system is also an asymmetric cryptographic system which security relies on the difficulty of factorization. In fact, the remarkable point of this system is that it has the advantage that the problem on which it relies has been mathematically proved to be as hard as integer factorization while this is not known to be true for the RSA problem. I.e., even if factorization is not $\mathbb{P}$ then no one knows if RSA is not insecure. In fact, it is very well known that it possible, in theory, that breaking RSA does not necessarily mean that one has to determine p and q in polynomial time, i.e., one may have factorization problem not being a problem in $\mathbb{P}$ but the RSA being insecure. This issue was already realized in the original paper of Rivest, Shamir and Adleman that stated the following sentence:

> *It may be possible to prove that any general method of breaking our scheme yields an efficient factoring algorithm. This would establish that any way of breaking our scheme must be as difficult as factoring. We have not been able to prove this conjecture.*

The difficulties involved on proving this conjecture can be seen in the following way. If one could show that from a ciphertext y and its corresponding plaintext x one would be able to compute the factors p and q, then one could, in particular, factorize n by choosing any other message x' and its corresponding ciphertext $y = e_k(x)$ and then compute p and q using the presumed to exist procedure, giving a polynomial time algorithm for Factorization, which is not known to exist.

As it will be discussed on this section, the Rabin Cryptographic system, is an example of computable provably secure scheme against chosen-plaintext attacks provided that the modulus $n = pq$ cannot be factored efficiently. Although it is not used in practice for technical reasons

that will be explained soon, it is a very interesting theoretical system that illustrates some fundamental properties of public key cryptographic system in general and, in particular, of the RSA.

Similarly to RSA cryptographic system, in Rabin's cipher system it is assume that $n = pq$ is announced publicly, and the primes that compose it, i.e., p and q, are kept secret.

Formally the Rabin's cryptographic system can be defined as follows:

Cryptographic system 9.7.1 (*Rabin's cipher system*)

The Rabin cipher system is a tuple $\mathcal{R} = (P, C, K, E, D, R, r)$ *where:*

1. $P = C = \mathbb{Z}_n$, *where* $n = p \times q$ *for primes* p *and* q *such that* $p, q \equiv 3 \pmod 4$;
2. K *consists of tuples* $(n, p, q) \in (\mathbb{Z}_n)^3$;
3. $R = \mathbb{Z}_n$;
4. *Given a tuple* (n, p, q), $r(n, p, q) = n$;
5. *For each* $k = (n, a, b) \in K$ *and all* $x, y \in \mathbb{Z}_n$, $e_k(x) = x^2 \pmod n$ *and* $d_k(y) = \sqrt{y} \pmod n$.
6. *The values* p, q, *are the private key.*

It is an easy and good exercise for the reader to verify that in fact the encryption can be performed in polynomial time.

One of the issues of the Rabin Cryptographic system is that the encryption function e_k is not an injection over $\mathbb{Z}_N^*$, so decryption cannot be done in an unambiguous way even with knowledge of p and q. It is easy to see that there are 4 plaintexts that will be encrypted to the same ciphertext and thus one can only say that the number sent is one of the four x_i, without being able to identify which one is used. In fact, if y is a ciphertext of a plaintext, i.e., $y \equiv x^2 \pmod n$ then by the Chinese remainder theorem on page 22 there are four possible square roots modulo n, as $y \equiv x^2 \pmod p$ and $y \equiv x^2 \pmod q$ have two solutions each.

In practice, this is not serious, since it is very unlikely that more than one of the x_i would correspond to a real meaningful message. Another way to avoid this issue, and that will be explained below, is to consider only plaintexts that are themselves quadratic residues.

Notice that if p and q are known, it is easy to compute all the x_i such that $e_k(x_i) = y$ and the knowledge of n, y and all the x_i, one can compute p, q. On the other hand, one is not able to obtain p and q from just one of the x_i efficiently, so the approach based on x and $e_k(x)$ described above does not work in polynomial time. Moreover, n being composite implies that, as far as it is known in practice, an opponent cannot invert the encryption fast, i.e., without knowing the prime factors p and q then computing the square roots (to recover the original message) cannot be done in polynomial time. Hence the security of this system relies on the impossibility of fast computation of square roots modulo composite numbers.

From the decryption point of view, notice that the choices of the primes being congruents with 3 modulo 4 it is not really a requirement since the computation of the square roots of a number modulo primes can be done in polynomial time. Although this choice allows a faster way of decryption. Since $p, q \equiv 3 \pmod 4$, then there are integer k_1 and k_2 such that $4k_1 = p + 1$ and $4k_2 = q + 1$ and thus, in particular $(p + 1)/4$ and $(p + 1)/4$ are integers.

Thus, if $y_1 \equiv y \mod p$ and $y_2 \equiv y \mod q$ then $x \equiv {y_1}^{k_1} \pmod p$ and $x \equiv {y_2}^{k_2} \pmod q$. In fact if $x_1 \equiv {y_1}^{k_1} \pmod p$ and $x_2 \equiv {y_2}^{k_2} \pmod q$ by Euler's Theorem 2.3.6 on Page 2.3.6 one has:

$$\begin{aligned} {x_1}^2 &\equiv ({y_1}^{k_1})^2 \equiv (x^{2k_1})^2 \pmod p \\ &\equiv x^{4k_1} \equiv x^{p+1} \pmod p \\ &\equiv x^{p-1}x^2 \equiv x^2 \pmod p \end{aligned}$$

and similarly

$$\begin{aligned} {x_2}^2 &\equiv ({y_2}^{k_2})^2 \equiv (x^{2k_2})^2 \pmod p \\ &\equiv x^{4k_2} \equiv x^{q+1} \\ &\equiv x^{q-1}x^2 \equiv x^2 \pmod q. \end{aligned}$$

Then all the solutions of the equations above are

$$\begin{cases} z_1 \equiv x_1 \pmod{p} \\ z_1 \equiv x_2 \pmod{q} \end{cases} \qquad \begin{cases} z_2 \equiv x_1 \pmod{p} \\ z_2 \equiv -x_2 \pmod{q} \end{cases}$$

$$\begin{cases} z_3 \equiv -x_1 \pmod{p} \\ z_3 \equiv x_2 \pmod{q} \end{cases} \qquad \begin{cases} z_4 \equiv -x_1 \pmod{p} \\ z_4 \equiv -x_2 \pmod{q} \end{cases}$$

Thus, using the Chinese remainder theorem on page 22 one can recover all possible original messages very efficiently using the above equations. On the other hand, as stated above, if the original messages are themselves quadratic residues then the decryption process can be done unique. In fact if x is a quadratic residue then there exists $s \in \mathbb{Z}_n^*$ such that $s^2 \equiv x \pmod{n}$ and thus:

$$\begin{aligned} {y_1}^{k_1} &\equiv (x^2)^{k_1} \equiv (s^4)^{k_1} \pmod{p} \\ &\equiv s^{p+1} \equiv s^{p-1}s^2 \pmod{p} \\ &\equiv s^2 \equiv x \pmod{p} \end{aligned}$$

and similarly

$$\begin{aligned} {y_2}^{k_2} &\equiv (x^2)^{k_2} \equiv (s^4)^{k_2} \pmod{q} \\ &\equiv s^{p+1} \equiv s^{p-1}s^2 \pmod{p} \\ &\equiv s^2 \equiv x \pmod{q}. \end{aligned}$$

and again by the Chinese remainder theorem on page 22 one can recover $x \pmod{n}$.

In the discussion above it was mentioned that the security of Rabin's cryptographic scheme is equivalent to Factorization. This means that a probabilistic polynomial algorithm for breaking the Rabin's cryptographic scheme can be used to obtain a polynomial time algorithm for Factorization and vice versa.

Clearly, if one has access to an oracle for factoring numbers then, using the above description one can determine the square roots modulo n and thus one can break the Rabin's cryptographic

systems. In what follows it is proved the other direction, i.e., it is assumed that one has an oracle that is able to break Rabin's cryptographic scheme then one can factor n in probabilistic polynomial time.

Algorithm 9.7.2 (*for Factoring using* $\mathbb{R}\mathtt{abin's}\mathbb{D}\mathtt{ecrypting}$ *as an oracle*)

Input: *an integer* $n = p.q$*;*

Output: *the factors* p *and* q*;*

Step 1: *choose uniformly at random an integer* $r \in \mathbb{Z}_n^*$*;*

Step 2: *compute* $y = r^2 \pmod{n}$*;*

Step 3: *Let* $x = \mathbb{R}\mathtt{abin's}\mathbb{D}\mathtt{ecrypting}(y)$*;*

Step 4: **If** $x = \pm r \pmod{n}$ **then** *output "failure"* **else**

Step 4.1: $p = \mathbb{E}\mathtt{uclides}(x + r, n)$*;*

Step 4.2: $q = n/p$*;*

Step 5: *Output* q *and* p*;*

This kind of algorithms that might fail for some runs of the algorithm and for some inputs, but every time it gives an answer it is the correct one are called of Las Vegas type in counter position to the Monte Carlo ones that instead of failure with small probability it might produce a wrong answer also with small probability. Observe that if y is a valid ciphertext then $\mathbb{R}\mathtt{abin's}\mathbb{D}\mathtt{ecrypting}(y)$ will produce one of four possible plaintexts as the value of x, i.e., if $x' = \mathbb{R}\mathtt{abin's}\mathbb{D}\mathtt{ecrypting}(y)$ then $x' \equiv \pm r \pmod{n}$ or $x' \equiv \pm \omega r \pmod{n}$, where ω is one of the non-trivial square roots of 1 modulo n. The important observation is that if $x' \equiv \pm \omega r \pmod{n}$ then $x'^2 \equiv r^2 \pmod{n}$ and $x' \not\equiv r \pmod{n}$ i.e. n and $x + r$ have a common factor (either p or q). In fact, since $1 < x + r, x - r \pmod{n} < n - 1$, then $n \not| x + r$ or $n \not| x - r$ and hence $p|x + r$ and $q|x - r$ or $p|x - r$ and $q|x + r$, and thus it is enough to compute $gcd(x + r, n)$. Furthermore, the computation of $gcd(x + r, n)$ must yield either p or q, and thus the algorithm returns, as claimed, a factorization of n.

Notice that if $\mathbb{R}\mathtt{abin}'\mathtt{s}\mathbb{D}\mathtt{ecrypting}$ gives any of the four square roots of y with uniform probability then the overall probability of success of this algorithm, over all choices for the random value $r \in \mathbb{Z}_n^*$ that $x \equiv \pm\omega r \pmod{n}$ is $1/2$.

Notice that that the fact that $p \equiv q \equiv 3 \pmod{4}$ allows the process of decryption to be fast, i.e., can be performed in polynomial time. In fact, since $\frac{p-1}{4}$ is an integer and thus $\sqrt{y} \equiv \pm y^{(p+1)/4} \pmod{p}$ and $\sqrt{y} \equiv \pm y^{(q+1)/4} \pmod{q}$ and thus:

$$\begin{aligned}(y^{(p+1)/4})^2 &\equiv c^{(p+1)/2} \equiv c^{(p-1)/2}c \pmod{p} \\ &\equiv c \pmod{p}\end{aligned}$$

and

$$\begin{aligned}(y^{(q+1)/4})^2 &\equiv c^{(q+1)/2} \equiv c^{(q-1)/2}c \pmod{p} \\ &\equiv c \pmod{q}\end{aligned}$$

and thus, by the Chinese Remainder theorem on page **??** the property follows.

One can generalize the Rabin's Cryptographic system to the following variant:

Cryptographic system 9.7.3 (*Rabin's cipher system version 2*)

The other Rabin cipher system is a tuple $\mathbb{R}\mathtt{abin} = (P, C, K, E, D, R, r)$ *where:*

1. $P = C = \mathbb{Z}_n$, *where* $n = p \times q$ *for primes* p *and* q *such that* $p, q \equiv 3 \pmod{4}$;
2. K *consists of tuples* $(p, q, b) \in (\mathbb{Z}_n)^3$ *such that* $0 \leq b \leq n-1$;
3. $R = \mathbb{Z}_n$;
4. *Given a tuple* (p, q, b), $r(p, q, b) = b$;
5. *For each* $k = (n, a, b) \in K$ *and all* $x, y \in \mathbb{Z}_n$, $e_k(x) = x(x+b) \pmod{n}$ *and* $d_k(y) = \sqrt{\frac{b^2}{4} + y} - \frac{b}{2} \pmod{n}$.
6. *The values* p, q, *are the private key.*
7. *The values* $n = p \cdot q$ *and* b, *are the public key.*

The first observation concerning this scheme, similarly to the first one, is that is sound up to 4 solutions since the square root modulo $n = p \cdot q$ has four solutions, i.e., decrypting a original

message that was properly encrypted with Rabin's system then it leads to a original message that is one of the four possible messages.

In the next lemma it is formally proved this aspect of this system.

Lemma 9.7.4. *Let $n = p \times q$ be such that p and q. Then equation $x^2 \equiv 1 \pmod{n}$ has 4 solutions modulo n.*

Proof: Notice that, by the Chinese Remainder theorem 2.3.4:

$$\begin{aligned} x^2 \equiv 1 \pmod{n} &\Leftrightarrow x^2 - 1 \equiv 0 \pmod{n} \\ &\Leftrightarrow \begin{cases} x^2 \equiv 1 \pmod{p} \\ x^2 \equiv 1 \pmod{q} \end{cases} \\ &\Leftrightarrow \begin{cases} x \equiv \pm 1 \pmod{p} \\ x \equiv \pm 1 \pmod{q} \end{cases} \end{aligned}$$

Thus, reverting the Chinese Remainder theorem 2.3.4 $x^2 \equiv 1 \pmod{n}$ has the solutions 1, -1 and ω, a non trivial solution that in fact results from the above mentioned theorem for the case:

$$\begin{cases} x \equiv 1 \pmod{p} \\ x \equiv -1 \pmod{q} \end{cases}$$

■

Lemma 9.7.5. *Let e_k be the encryption function of Rabin's cipher system version 2 described in the Cryptographic system 9.7.3. Then e_k is not injective.*

Proof: Assume that $e_k(x) = x(x+b) \pmod{n}$. Then:

$$
\begin{aligned}
e_k(x) = e_k(z) \quad &\Leftrightarrow \quad x(x+b) \equiv z(z+b) \pmod{n} \\
&\Rightarrow \quad \begin{cases} x(x+b) \equiv z(z+b) \pmod{p} \\ \\ x(x+b) \equiv z(z+b) \pmod{q} \end{cases} \\
&\Rightarrow \quad \begin{cases} x^2 + xb + \dfrac{b^2}{4} \equiv z^2 + zb + \dfrac{b^2}{4} \pmod{p} \\ \\ x^2 + xb + \dfrac{b^2}{4} \equiv z^2 + zb + \dfrac{b^2}{4} \pmod{q} \end{cases} \\
&\Rightarrow \quad \begin{cases} \left(x + \frac{b}{4}\right)^2 \equiv \left(z + \dfrac{b}{4}\right)^2 \pmod{p} \\ \\ \left(x + \dfrac{b}{4}\right)^2 \equiv \left(z + \dfrac{b}{4}\right)^2 \pmod{q} \end{cases} \\
&\Rightarrow \quad \begin{cases} x + \dfrac{b}{4} \equiv \pm \left(z + \dfrac{b}{4}\right)^2 \pmod{p} \\ \\ x + \dfrac{b}{4} \equiv \pm \left(z + \dfrac{b}{4}\right) \pmod{q} \end{cases}
\end{aligned}
$$

Thus, modulo n, the solutions are:

$$
\begin{aligned}
x + \frac{b}{2} = z + \frac{b}{2} \qquad &\wedge \qquad x + \frac{b}{2} = -\left(z + \frac{b}{2}\right) \\
\wedge \quad x + \frac{b}{2} = \omega\left(z + \frac{b}{2}\right) \quad &\wedge \quad x + \frac{b}{2} = -\omega\left(z + \frac{b}{2}\right)
\end{aligned}
$$

and thus:

$$
\begin{aligned}
x = z \qquad &\wedge \qquad x = -z - \frac{b}{2} \\
\wedge \quad x = \omega\left(z + \frac{b}{2}\right) - \frac{b}{2} \quad &\wedge \quad x = -\omega\left(z + \frac{b}{2}\right) - \frac{b}{2}
\end{aligned}
$$

■

Proposition 9.7.6. *The decryption works up to finding the 4 square roots.*

Proof: Since $y \equiv x(x+b) \pmod{n}$ then considering $x_1 \equiv x + \frac{b}{2} \pmod{n}$ one has:

$$\begin{aligned} y &\equiv \left(x_1 - \frac{b}{2}\right)\left(x_1 - \frac{b}{2} + b\right) \pmod{n} \\ &\equiv \left(x_1 - \frac{b}{2}\right)\left(x_1 + \frac{b}{2}\right) \pmod{n} \\ &\equiv x_1^2 - \frac{b^2}{4} \pmod{n} \\ \Rightarrow x_1 &\equiv \sqrt{y + \frac{b^2}{4}} \end{aligned}$$

and thus $x \equiv \sqrt{y + \frac{b^2}{4}} - \frac{b}{2}$. ■

In the next theorem we show that both Rabin's cryptographic systems are vulnerable against ciphertexts attacks. In this type of attacks an opponent has access temporarily to the decryption machine but is able to ask for the decryption of some texts and its goal is to obtain p and q to be able to decrypt all other messages. This attack is similar to the factorization reduction studied above. In fact, the algorithm described is also a Monte-Carlo with negative bias for factorization. In the next proposition it is shown that if one has access to the decryption machine of Rabin's cipher system version 2 then one can break completely the system.

Theorem 9.7.1

There exists a Las Vegas algorithm that with probability of success of $1/2$ *to break Rabin's cryptographic system (by factorizing* n*).*

Proof: Consider the following algorithm

Algorithm 9.7.7

Input: *an integer* $n = p.q$ *and* b *as in the Rabin's cryptographic system;*

Output: *the factors* p *and* q*;*

Step 1: *choose uniformly at random an integer* $r \in \mathbb{Z}_n^*$*;*

Step 2: *compute* $y = r^2 - \frac{b^2}{4} \pmod{n}$*;*

Step 3: *Let* $x = \mathbb{R}\text{abin}'\mathbb{s}\mathbb{D}\text{ecrypting}(y)$*;*

Step 4: *Let* $x_1 = x + \dfrac{b}{2}$*;*

Step 5: **If** $x_1 = \pm r \pmod{n}$ **then** *output "failure"* **else**

Step 5.1: $p = \mathbb{E}\text{uclides}(x + r, n)$*;*

Step 5.2: $q = n/p$*;*

Step 6: *Output* q *and* p*;*

A similar argument to the one made in for the justification of Algorithm **??** proves the statement of the proposition. ■

• •

SECTION 9.8

Exercises

Exercise 9.8.1. *Show that there exists a constant* c *such that for all* $n > 1$*, there are at least* $c2^{n-1}/n$ *primes which binary representation is* n *bit long.*

Exercise 9.8.2. *Let* p *be an odd prime number. Show that in* $\mathbb{Z}_p^*$ *there are* $(p-1)/2$ *quadratic residues.*

Exercise 9.8.3. *Let* m *be a number. Show that if* m *is not prime, then at least* $\sqrt{m}$ *elements of* $\mathbb{Z}_m$ *do not have multiplicative inverses.*

Exercise 9.8.4. *Let* n *be a composite integer and consider* r *and* d *such that* $n - 1 = 2^r u$ *where* d *is odd. One call* $a \in \mathbb{Z}_n$ *a strong witness that* n *is composite if the following properties holds:*

1. $a^u \equiv \pm 1 \pmod{n}$*;*
2. *for all* $i \in \{1, \cdots r-1\}$*,* $a^{2^i u} \not\equiv -1 \pmod{n}$*.*

Show that:

1. *if n is odd that is not a prime power, then at least half the elements of $\mathbb{Z}_n^*$ that are strong witnesses that n is composite.*

2. *one can test in polynomial time if a is a strong witness that n is composite.*

Exercise 9.8.5. *Let p be an odd prime number. Show that there is only one element in $\mathbb{Z}_{p+1}$ that does not pass the primality test $x^{m-1} \equiv 1 \pmod{p+1}$, in the case, $m = p+1$.*

In the next exercise, the reader is invited to simulate the RSA encryption scheme for a simple case.

Exercise 9.8.6. *Assume that one choose $p = 11$, $q = 19$ for the RSA cryptographic system. Let (a, b) be key for this cryptographic system.*

1. *Prove that one can choose $b = 7$.*

2. *Compute the value of a.*

3. *Show how to encrypt the message 100.*

4. *Show how to decrypt the resulting message of the previous item.*

Exercise 9.8.7. *Assume that one choose $p = 11$, $q = 23$ for the RSA cryptographic system. Let (a, b) be key for this cryptographic system.*

1. *Prove that one can choose $b = 7$.*

2. *Compute the value of a.*

3. *Show how to encrypt the message 100.*

4. *Show how to decrypt the resulting message of the previous item.*

In the next exercise it is studied a variant of the RSA system where instead of assuming that a and b are chosen to be inverse of each other it is assumed that they are inverse modulo $\gamma(n) = \dfrac{\phi(n)}{gcd(\phi(p), \phi(q))}$.

Exercise 9.8.8. *One can use a simple variation of the RSA cryptographic system as follows:*

Cryptographic system 9.8.9 (Alternative RSA cipher system)

The cipher system consists of a tuple $\mathcal{C} = (P, C, K, E, D, R, r)$ *where:*

1. $P = C = \mathbb{Z}_n$, *where* $n = p \times q$ *for primes* p *and* q;
2. K *consists of pairs* $k = (a, b) \in (\mathbb{Z}_n)^2$ *such that* $a \cdot b \equiv 1 \pmod{\gamma(n)}$, *where* $\gamma(n) = \frac{\phi(n)}{gcd(\phi(p), \phi(q))}$, *i.e.,* a *and* b *are inverse of each order modulo* $\gamma(n)$;
3. $R = \mathbb{Z}_n$;
4. *Given a pair* (a, b), $r(a, b) = b$;
5. *For each* $k = (a, b) \in K$ *and all* $x, y \in \mathbb{Z}_n$, $e_k(x) = x^b \pmod{n}$ *and* $d_k(y) = y^a \pmod{n}$.
6. *The values* n *and* b *form the public key, and* p, q, a *are the private key.*

Show that this scheme is also sound, i.e., decrypting and encrypted message one gets the original message.

Exercise 9.8.10. *Let* n *be a composite integer that is product of the form* pq *where* p *and* q *are primes. Show that if* $\phi(n)$ *and* n *are known then it is possible to compute* p *and* q.

In the next exercise it is introduced the Carmichael function discussed in the section concerning the *AKS* algorithm.

Exercise 9.8.11. *Let* $n > 2$ *be an odd number and let* $\prod_{i=1}^{r} p_i^{e_i}$ *its prime decomposition. The* λ*-Carmichael function is defined by:*

$$\lambda(n) = lcm(\phi(p_1^{e_1}, \cdots, p_r^{e_r}))$$

where ϕ *is the Euler function. Prove that:*

1. $a^{\lambda(n)} \equiv 1 \pmod{n}$ *for all* $a \in \mathbb{Z}_n^*$;
2. $a^{n-1} \equiv 1 \pmod{n}$ *if and only if* $\lambda(n) | n - 1$;
3. *Consider* $C_n = \{a \in \mathbb{Z}_n^* : a^{\lambda(n)/2} \equiv \pm 1 \pmod{n}\}$. *Then,* C_n *is a multiplicative subgroup of* $\mathbb{Z}_n^*$;

4. *$C_n = \mathbb{Z}_n^*$ if and only if $n = p^e$ with p prime and $e > 1$.*

In the next exercise it is study a variant of the Solovay–Strassen algorithm for primality testing.

Exercise 9.8.12 (Lehmann's primality test)**.** *Let n be an odd integer larger than 3. Denote by $f : \mathbb{Z}_n^* \to \mathbb{Z}_n^*$ defined by $f(a) = a^{(n-1)/2}$. Let E be the range of f. Show that:*

1. *$E = \{-1, 1\}$ if and only if n is prime.*
2. *if n is not a prime power then $E \neq \{-1, 1\}$.*
3. *Show that if $n = p^e$ with p prime and $e \geq 2$ then $E \neq \{-1, 1\}$.*
4. *if $E = \{1\}$, n is a Carmichael number.*
5. *The Lehmann's algorithm for primality test is the following:*

Algorithm 9.8.13 (Lehmann's primality test – $\mathbb{L}$)

Input: *n an odd integer and an integer $t > 0$, the number of trials.*

Step 1: **For** $n = 1$ **to**t **do**

Step 1.1: *Choose uniformly at random $a_i \in \{1, \cdots, n-1\}$.*

Step 1.2: *Compute $b_i = a^{(n-1)/2} \pmod{n}$.*

Step 2: **If** $\{b_1, \cdots, b_t\} \neq \{-1, 1\}$ **then** *output probably composite* **else** *output probably prime.*

Prove the following properties:

(a) *if n is prime the probability that the algorithm returns probably prime is at least $1 - 2^{1-t}$.*

(b) *if n is composite, the probability that the algorithm returns probably composite is at least $1 - 2^{-t}$.*

(c) *the complexity of the algorithm is $O(t \log^2 n)$.*

(d) *for each of the numbers 343, 561, 667 and 841, compute E and confront with the results of the previous items.*

In the next exercise the reader is asked to redo the proof of Theorem 9.3.7

Exercise 9.8.14. *Let n be an odd integer and consider the following subset of $\mathbb{Z}_n^*$:*

$$E(n) = \left\{ a \in \mathbb{Z}_n^* : \left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \pmod{n} \right\}$$

1. *Prove that $E(n)$ is a subgroup of $\mathbb{Z}_n^*$.*
2. *Prove that if $n = p^k q$ where p is a prime number with $k \geq 2$ and $gcd(p, q) = 1$ then $G(n) \subsetneq \mathbb{Z}_n^*$, i.e, there is an $a \in \mathbb{Z}_n^*$ such that $\left(\frac{a}{n}\right) \not\equiv a^{(n-1)/2} \pmod{n}$. (Hint: consider the element $a = p^{k-1}q + 1$ and its binomial expansion)*
3. *Assume now that $n = pq$ where p and q are distinct primes.*
   (a) *Show that there exists $a \in \mathbb{Z}_n^*$ such that a is not a quadratic residue modulo p and $a \equiv 1 \pmod{q}$. (hint: use the Chinese remainder theorem).*
   (b) *Show that such a is not an element of $G(n)$ (hint: show that $\left(\frac{a}{n}\right) \not\equiv -1 \pmod{n}$ and $a^{(n-1)/2} \not\equiv -1 \pmod{n}$)*
4. *Prove that the same properties of the previous item hold if $n = p_1 \cdots p_l$ where p_i are all distinct odd primes.*
5. *Prove that if n is composite and odd then $ord(G(n)) \leq \frac{n-1}{2}$.*
6. *From the previous item conclude that the Solovay–Strassen Algorithm $\mathbb{SS}$ presented in Algorithm 9.3.9 is indeed a Monte-Carlo one with bias one half.*
7. *Discuss why in the previous items the question of n being even is ruled out of the discussion.*

Exercise 9.8.15. *Recall that one says that an encryption scheme is secure against chosen-ciphertext attacks if even one opponent having access to the decryption process on messages different from the original one it cannot recover the original message.*

Show that the RSA cryptographic system is not secure against this type of attacks by showing a way that an opponent can use to decrypt a message.

Exercise 9.8.16. *exercise 5.18 in [**?**] One says that a plaintext is fixed if $e_k(x) = x$. Show that the RSA cryptographic system has $gcd(b-1, p-1) \cdot gcd(b-1, q-1)$ fixed plain texts in $\mathbb{Z}_n^*$.*

In the next exercises it is shown that despite the fact one has a presumably secure encryption scheme a sloppy use of it can compromise its security.

Exercise 9.8.17. *Common modulus Assume that two users of the RSA cryptographic system A_1 and A_2 have as public keys, respectively (n, b_1) and (n, b_2) with $gcd(b_1, b_2) = 1$, i.e., they share the same modulus but with different encryption exponents. Suppose that a third party S wishes to send the same message x to both A_1 and A_2 using their encryption exponent.*

1. *Show how an opponent O one can easily obtain the original message if it intersects both encrypted messages. (hint: use the Euclides algorithm).*

2. *Another way of decrypting the message is to consider the following algorithm:*

   Algorithm 9.8.18

   Input: $n, b_1, b_2, y_1 = x^{b_1}$ *and* $y_2 = x^{b_2}$.

   Step 1: $c_1 = b_1^{-1} \pmod{b_2}$;

   Step 2: $c_2 = (c_1 b_1 - 1)\mathbf{div}\ b_2$;

   Step 3: $x_1 = y_1^{c_1}(y_2^{c_2})^{-1}$;

   Show that in fact $x_1 \equiv x \pmod{n}$;

Exercise 9.8.19 (small exponent attack)**.** *The public exponent b in the RSA Cryptographic system can be chosen arbitrarily as long as $gcd(b, \phi(n)) = 1$ but usually the users tend to choose small exponents to speed up the encryption process. Assume that three users A_1, A_2 and A_3 such that their public keys are, respectively, $(n_1, 3)$, $(n_2, 3)$ and $(n_3, 3)$.*

Show that it is reasonable to assume that n_1, n_2 and n_3 are all different and also that the pairwise greatest common divisor of them is equal to 1.

Assume that n_1, n_2 and n_3 are all different and they are pairwise co-primes and consider another party C that wants to send a message x to all of them using the RSA Cryptographic system with their public key. Show that an opponent that intercepts the three encrypted messages can decrypt them and recover the original message.

This attack takes advantage of the low exponent 3 *but can be performed with other small exponents. Explain how this attack can be performed in practice.*

In the next exercise the reader is asked to check that even if p and q are chosen to be congruent with 5 modulo 8 then one can perform square roots in polynomial time. This is an important observation concerning Rabin's cryptographic system.

Exercise 9.8.20 (square roots modulo $p \equiv 5 \pmod 8$)**.** *Let p be an odd prime number such that $p \equiv 5 \pmod 8$. Let a be a quadratic residue modulo p.*

1. *Show that $4|p-1$ and $8|p+3$;*
2. *Envisage the values of $a^{(p?1)/4}$;*
3. *Compute the value $4^{(p-1)/4}$.*
4. *Consider the following algorithm to compute the square root:*

   Algorithm 9.8.21

   Input *an odd prime p such that $p \equiv 5 \pmod 8$ and a quadratic residue a.*

   Output *a square root of a modulo p.*

   Step 1 *Compute the value $d = a^{(p-1)/4} \pmod p$.*

   Step 2 **If** $d = 1$ **then** *compute $r = a^{(p+3)/8} \pmod p$*
   else $r = 2a(4a)^{(p-5)/8} \pmod p$.

   Step 3 *Output r.*

   Prove that in fact the value retrieved by the algorithm is a square root of a.
5. *Prove that the algorithm runs in polynomial time.*

CHAPTER 10

Cryptographic schemes based on the discrete logarithmic problem

In this chapter it is studied the problem of discrete logarithm and some of its applications to cryptography. Heuristically speaking, this problem consists on finding the smallest exponent that one has to power a given number (usually a generator of a finite group) to obtain a target element. Similarly to the problem of determining the square roots, which is simple in $\mathbb{N}$ but presumably hard when moduli are considered, the discrete logarithm also shares similar properties. The hardness of this problem is the base of security of many cryptographic schemes such as Diffie–Hellman key agreement, ElGamal encryption and the ElGamal signature scheme. It will also be discussed the Pohling–Hellman algorithm.

.....

SECTION 10.1

The discrete logarithmic problem

As discussed in the basic material part of these notes one important question is the construction of easy to compute functions that are presumably hard to invert. One of those functions is constructed based on the following problem:

Problem 10.1.1 (*Discrete logarithm over a finite group* - $\mathbb{D}\text{iscrete }\mathbb{L}\text{ogarithm}$)

Let $(G, \cdot)$ be an Abelian group and consider two elements $g \in G$ and h such that g has order n and $h \in \langle g \rangle$, i.e., h is an element in the subgroup generated by g.

Instance: *G, g and h as described above and $n = ord(g)$.*

Question: *Find the unique integer $0 < a < n - 1$ such that and $g^n = h$?*

It is common to denote a by $\log_g h$, since by analogy to the logarithmic function in $\mathbb{R}$, the value a is defined to be the logarithm of h on base g since it is the least number that one has to power g to obtain h. Since the values are taken modulo n, the adopted name perfectly describes the problem. One common choice for G is some $(\mathbb{Z}_p^*, \times)$ where p is a prime and g is a generator of the group. Another one is to consider $g = \omega$ a nontrivial l^{th} roots of unity. Even another choice is to take g to be an element with prime order q, being in latter one the most fruitful choice.

In the next exercise the reader is asked to prove that this problem is a **NP** problem.

Example 10.1.2. *Let $(G, \cdot)$ be an Abelian group and consider two elements $g \in G$ and h such that g has order n and $h \in \langle g \rangle$. Prove that the discrete logarithm problem over G described above is* **NP** *i.e.,* $\mathbb{D}\text{iscrete }\mathbb{L}\text{ogarithm} \in$ **NP**.

Notice that computing the discrete logarithms is believed to be hard, since no one knows a polynomial time algorithm to compute it. The reader may think for a while about this question and how one could compute it in polynomial time. It is also an important open question/problem in Computer Science. Once the reader is convinced of the hardness of this problem in a group G then one can use it to give a strong evidence for the fact that the function

$$f : \mathbb{Z} \to G$$
$$n \mapsto n \cdot g$$

for $g \in G$ is likely to be a one-way function. In the discussion above it was suggested that one could consider a group $\mathbb{Z}_p$ with p prime, q to be a prime factor of $p - 1$ and g to be a nontrivial l^{th} root of the unity, say $g = \omega$ and in fact the difficulty of finding a such that $\omega_2 = \omega^a$ where ω is an element in the subgroup generated by ω relies crucially on the assumption that to solve a discrete logarithm in this subgroup is essentially equivalent to solve a discrete logarithm in $\mathbb{Z}_q^*$.

So, from the argument above one can conclude that, until the writing of this noted, the only know ways to attack the discrete logarithm directly in the cyclic subgroup is to use generic methods which all have exponential or subexponential complexity on the length of the order of the subgroup. For that reason, the standard for Digital Signature established by NIST recommends the choice of l to be of 160 bit and q to be of size about 1000 bit such that $l|q-1$. It is worthwhile mention that Shoup has proven that the use only of algorithms that use only properties of operations of the group then the computation of discrete logarithm in a group of prime order l can not be solved in less than $O(\sqrt{l})$ operations, and thus the most effective methods to solve the discrete logarithm problem in finite fields have to use of extra information about relations in the group.

This problem and some of its instances can be used in cryptography. For example this problem is used in the following situations:

- Considering a multiplicative group of a finite field $\mathbb{F}_q$ to establish keys agreement, first proposed by Diffie and Hellman.
- The discrete logarithm in the group of rational points of an elliptic curve over a finite field $\mathbb{F}_q$ to develop cryptographic systems, simultaneously and independently proposed by Koblitz and Miller in middle of 80's in two different and independent works.

SECTION 10.2

The ElGamal Public-key Cryptographic system

This cryptographic system was proposed by Taher Elgamal in 1984 as a scheme for a public-key cryptosystem based on the Discrete Logarithm problem in $(\mathbb{Z}_p^*, \times)$.

For sake of understanding it is now lightly explained this encryption scheme. Assume that α is the generator and $\beta = \alpha^a$ where a is the private key. The plaintext x is transformed into a ciphertext by multiplying it with β^k. The value k can be retrieved by the information of α^k that is also incorporated in the ciphertext. From the decryption point of view, since the private key a is known one can compute β^k using α^k and then recovering x by dividing by β^k the chipertext.

Formally the ElGamal Public-key Cryptographic System is defined as follows.

Cryptographic system 10.2.1 (*ElGamal Public-key Cryptographic System in $\mathbb{Z}_p^*$*)

Consider p to be a prime number such that, for some k, $|p| \leq n^k$ and let α be a generator of $\mathbb{Z}_p^*$. The ElGamal Public-key Cryptographic system is the tuple $\mathcal{G} = (P, C, K, E, D, R, r)$ where:

1. $P = \mathbb{Z}_p^*$;
2. $C = \mathbb{Z}_p^* \times \mathbb{Z}_p^*$;
3. *K consists of pairs $k = (a, \beta) \in (\mathbb{Z}_p^*)^2$ such that $\beta = \alpha^a \pmod{p}$;*
4. $R = \mathbb{Z}_p$;
5. *Given a pair (a, β), $r(a, \beta) = \beta$;*
6. *For each $k = (a, \beta) \in K$ and all $x \in \mathbb{Z}_p^*$, $e_{r(k)}(x) = (y_1, y_2)$ where $y_1 = \alpha^c \pmod{p}$, where $c \in \mathbb{Z}_{p-1}$ is chosen uniformly at random and $y_2 = x\beta^k \pmod{p}$;*
7. $d_{(a,\beta)}(y_1, y_2) = y_2(y_1)^{-a} \pmod{p}$.

Notice that the ElGamal cryptographic encryption function is probabilistic since a random c is generated to proceed with the encryption.

Before continuing several remarks are mandatory. It is a very interesting exercise for the reader to check that the process of establishing the key which requires two modular exponentiations can be performed in polynomial time. Also, the decryption process is also efficiently performed since it requires only a modular exponentiation. Another important aspect of the ElGamal system is that is sound. In the next proposition it is proved this fact. It is also easy to see that the generation primes that are suitable for running the cryptographic system can be done efficiently.

Proposition 10.2.2. *Let p to be a prime number, α a generator of $\mathbb{Z}_p^*$, β and a as in the ElGamal Public-key Cryptographic system in $\mathbb{Z}_p^*$ described in Cryptographic system 10.4.1.*

Let $x \in \mathbb{Z}_p^*$ be a plaintext and assume that $e_{r(k)}(x) = (y_1, y_2)$. Then $d_{(a,\beta)}(y_1, y_2) = x$, i.e., the decryption process of a legitimate message retrieves the original message.

Proof:

$$\begin{aligned} d_{(a,\beta)}(y_1, y_2) &= y_2(y_1)^{-a} \pmod{p} \\ &= x\beta^k((\alpha^k)^a)^{-1} \pmod{p} \\ &= x(\alpha^a)^k((\alpha^k)^a)^{-1} \pmod{p} \\ &= x \end{aligned}$$

■

One can generalize the use of the ElGamal cryptographic system to the case where the entire group is not considered but, instead, considering the discrete logarithm over a subgroup of G generated by and element α. The idea is that if $H = \langle \alpha \rangle$ has high order then computing the discrete logarithm over H is also hard as it was discussed previously when it was presented the problem of discrete logarithm. The Generalized ElGamal cryptographic system is then defined as follows:

Cryptographic system 10.2.3 (*Generalized ElGamal Public-key Cryptographic System*)

Consider G to be a group and let H be a cyclic subgroup of G. The Generalized ElGamal Public-key Cryptographic system is the tuple $\mathcal{GG} = (P, C, K, E, D, R, r)$ where:

1. *$P = G$;*
2. *$C = G \times G$;*
3. *K consists of tuples $k = (\alpha, a, \beta)$ such that $\beta = \alpha^a$ in G and α is a generator of H;*
4. *$R = H^2$;*
5. *Given a tuple (α, a, β), $r(\alpha, a, \beta) = (\alpha, \beta)$;*
6. *For each $k = (\alpha, a, \beta) \in K$ and all $x \in G$, $e_{r(k)}(x) = (y_1, y_2)$ where $y_1 = \alpha^c \pmod{p}$, where $c \in \mathbb{Z}$ is chosen at random and $y_2 = x\beta^k$ in G;*
7. *$d_{(\alpha,a,\beta)}(y_1, y_2) = y_2((y_1)^a)^{-1}$ in G.*

Notice that if one chooses a group where the operation is just the sum, as for example $G = (\mathbb{Z}_m, +)$ then $\alpha^a = a\alpha$ and thus if $\beta = a\alpha$ then $a = \beta\alpha^{-1}$ and thus the discrete logarithm in

this group is computable in polynomial time. Te same problems appear in the cases where the function $\Phi : \mathbb{Z}_p^* \to \mathbb{Z}_p$ is effectively computable. Notice that in such morphism $\Phi(1) = 0$ and $\Phi(ab) = \phi(a) + \Phi(b)$.

Proposition 10.2.4. *Let p be a prime number. If the morphism $\Phi : \mathbb{Z}_p^* \to \mathbb{Z}_p$ is effectively computable in polynomial time, then the discrete logarithmic problem over $(\mathbb{Z}_p^*, +)$ is in $\mathbb{P}$.*

The proof of this result is simple and uses the fact that in this case one can compute $\log_\alpha \beta$ in polynomial time by computing the value $\Phi(\beta)\Phi(\alpha)^{-1}$.

• •

SECTION 10.3

Attacking the Discrete Logarithm problem

Similarly to Factorization, if we use the ElGammal cryptographic system in a sloppy way than the system is not secure. In what follows, it is described some attacks that one can perform in order to compute the discrete logarithms. One of them is what in number theory is called the Pohlig–Hellman algorithm sometimes credited as the Silver–Pohlig–Hellman which is an algorithm for computing discrete logarithms in a multiplicative group whose order is a smooth integer, i.e., any of its prime factors are small.

This algorithm is used to compute logarithms and takes advantage of the factorization of the order of the group $\mathbb{Z}_p^*$, which is assumed to be

$$n = p - 1 = \prod_{i=1}^{k} p_i^{e_i}$$

with p_i being distinct primes. If $a = \log_\alpha \beta$, then this algorithm determines $a_i \equiv a \pmod{p_i^{e_i}}$ for $1 \leq i \leq r$, and then using the the Chinese remainder theorem presented in Page 22 to recover $a \pmod{n}$. The idea to compute each integer x_i is to determined the digits $l_0, l_1, \cdots, l_{e_i - 1}$ all less than $p_i - 1$ such that $x_i = l_0 + l_1 p_i + \cdots + l_{e_i-1} p_i^{e_i - 1}$.

Let then q be a prime number and let c be the maximal exponent of q dividing n i.e., such that $q^c | n$ and $q^{c+1} \not| n$. Thus, the crucial observation is that $a = x + sq^c$ for some integer s. In particular $a = \sum_{i=0}^{c-1} a_i q^i + sq^c$. The next lemma gives gives an idea how the algorithm can compute a_0.

Lemma 10.3.1. *Consider in $\mathbb{Z}_n$ discussed above and let c be an integer such that, $q^c | n$ and $q^{c+1} \not| n$ where q be a prime factor of n. Let also a, α and β be such that $a = \log_\alpha \beta$. If $a = \sum_{i=0}^{c-1} a_i q^i + sq^c$ then $\beta^{n/q} = \alpha^{a_0 n/q}$.*

Proof: Notice that by definition $\beta = \alpha^n$ and thus:

$$\begin{aligned} \beta^{n/q} &\equiv (\alpha^a)^{n/q} \pmod{p} \\ &= \alpha^{\left(\sum_{i=0}^{c-1} a_i q^i + sq^c\right) n/q} \pmod{p} \\ &= \alpha^{a_0 n/q} \pmod{p} \end{aligned}$$

Notice that the last equality is equivalent to prove, by Lagrange theorem, that

$$\left(\sum_{i=0}^{c-1} a_i q^i + sq^c\right) \frac{n}{q} - a_0 \frac{n}{q} \equiv 0 \pmod{n}$$

In fact

$$\begin{aligned} \left(\sum_{i=0}^{c-1} a_i q^i + sq^c\right) \frac{n}{q} - a_0 \frac{n}{q} &\equiv \left(\sum_{i=1}^{c-1} a_i q^i + sq^c\right) n/q \\ &\equiv \left(\sum_{i=1}^{c-1} a_i q^{i-1} + sq^{c-1}\right) n \\ &\equiv 0 \pmod{n} \end{aligned}$$

■

Thus computing a_0 is just calculate the powers $(\alpha^{n/q})^i$ until one finds $\beta^{n/q}$ and then $a_0 = i$.

So if $c = 1$ then the computation of a is done otherwise one uses the following generalization of the previous lemma:

Lemma 10.3.2. *Consider the preamble of Lemma 10.3.1. Assume also that $a_0, \cdots, a_{j-1}$ are determine. If $b_j = \beta \alpha^{-\sum_{i=0}^{j-1} a_i q^i}$ then $\beta_j^{n/q^{j+1}} = \alpha^{a_j n/q}$.*

Proof:

$$\begin{aligned} \beta_j^{n/q^{j+1}} &\equiv (\alpha^{a - \sum_{i=0}^{j-1} a_i q^i})^{n/q^{j+1}} \pmod{p} \\ &= \alpha^{(\sum_{i=j}^{c-1} a_i q^i + sq^c) n/q^{j+1}} \pmod{p} \\ &= \alpha^{a_j n/q} \pmod{p} \end{aligned}$$

Notice that the last equality is equivalent to prove that

$$\left(\sum_{i=j}^{c-1} a_i q^i + sq^c\right)\frac{n}{q} - a_j\frac{n}{q} \equiv 0 \pmod{n}$$

In fact

$$\begin{aligned}\left(\sum_{i=j}^{c-1} a_i q^i + sq^c\right)\frac{n}{q} - a_j\frac{n}{q} &\equiv \left(\sum_{i=j+1}^{c-1} a_i q^i + sq^c\right) n/q \\ &\equiv \left(\sum_{i=j+1}^{c-1} a_i q^{i-1} + sq^{c-1}\right) n \\ &\equiv 0 \pmod{n}\end{aligned}$$

■

Thus, given b_j one can again easily compute a_j. For sake of efficiency of the algorithm it is worthwhile to observe that β_j can be computed from β_{j-1} using the following relation:

$$\beta_j = \beta_{j-1}\alpha^{-\alpha_{j-1}q^{j-1}}.$$

Algorithm 10.3.3 (*The* $\mathbb{P}\mathrm{ohlig} - \mathbb{H}\mathrm{ellman}$ *algorithm*)

Input *Primes p and q such that $q|p-1$, α a generator of $\mathbb{Z}_p^*$ and $\beta \in \mathbb{Z}_p^*$.*

Output $a = \log_\alpha \beta \pmod{q}^c$*, where c is such that $q^c|p-1$ and $q^{c+1} \not| p-1$*

Step 1 **For** $i = 0$ **to**c **do**

Step 1.1 **While** $k < q$ **do**

Step 1.1.1 **If** $\alpha^{kn/q} \equiv \beta^{n/q^{i+1}}$ **then** $a_i = k$;

Step 1.1.1 $k = q$;

Step 1.2 $\beta = \beta\alpha^{-a_i q^i}$

Step 2 *Output β.*

Lemma 10.3.4. *The* $\mathbb{P}\mathrm{ohlig} - \mathbb{H}\mathrm{ellman}$ *algorithm runs in $O(c\sqrt{q})$.*

The proof is straightforward and it is left as an exercise for the interested reader. Notice that this means that this algorithm is only efficiently if $n = p - 1$, the order of $\mathbb{Z}_p^*$ admits only small factors. As a consequence of these results, we have the following corollary:

Theorem 10.3.1 (*Pohlig-Helman*)

Let p be a prime. If all prime factors of $p - 1$ are polynomial on the size of p, i.e., of order of $O(poly \log p) = O(|p|^e)$ for some e, then the discrete logarithm problem can be solved in polynomial time.

In order to make the Discrete Logarithm resistant to these types of attacks one has to choose primes such that $p - 1$ is not smooth. For example, this observation rules out the use of primes of the form $2^n + 1$, known as the Mersenne primes in the Elgamal cryptographic system. One particular common choice of p is of the form $2q + 1$ where q is also a prime. These kind of primes are called Sophie Germain prime numbers and it is conjectured that the number of such primes is infinite and not negligible on the size of those primes similarly to what happens to the distribution of prime numbers proved in the Prime numbers theorem on Page 18.

In the case of choosing these type of primes it is easy to construct a generator of $\mathbb{Z}_p^*$ as the next lemma proves.

Lemma 10.3.5. *Let q be a prime number. If $p = 2q + 1$ is also a prime number then $g \in \mathbb{Z}_p^*$ is a generator if and only if $g^2 \not\equiv 1 \pmod{p}$ and $g^q \not\equiv 1 \pmod{p}$.*

Proof:

($\Rightarrow$) If $g^2 \equiv 1 \pmod{p}$ or $g^q \equiv 1 \pmod{p}$ then $ord(g) < p - 1$ and then g could not be a generator.

($\Leftarrow$) Notice that since p is prime then $\phi(p) = 2q$. Then, by Euler's criterion on Page 188,

$$g^{(p-1)/2} \equiv g^q \equiv -1 \pmod{p}$$

Notice that by Lagrange theorem on Page 11 one has:

$$g^a \equiv 1 \pmod{p} \text{ if and only if } a \in \{1, 2, q, 2q\}$$

So a as to be equal to $2q$. Notice that since $g^2 \not\equiv 1 \pmod{p}$ then a cannot be 2 and thus it also cannot be one. Also since$g^q \not\equiv 1 \pmod{p}$ then the only case left is $a = 2q$. ■

As a corollary one has:

Corollary 10.3.6. *Let q be a prime number. If $p = 2q + 1$ is also a prime number and $g \in \mathbb{Z}_p^*$ is a generator and for odd i, $q \not| i$ then g^i is also a generator.*

Notice that by Fermat's little theorem presented in Page 22, one has:

$$(g^{2i})^q \equiv 1 \pmod{p}$$

and

$$(g^q)^2 \equiv 1 \pmod{p}$$

Then among the $2q$ elements of $\mathbb{Z}_p^*$, $q + 1$ are not generators of the group. So the following algorithm can determine a generator of $\mathbb{Z}_p^*$.

Algorithm 10.3.7

Input *a group $\mathbb{Z}_p^*$ with p prime of the form $p = 2q + 1$ where q is another prime.*

Output *a generator of $\mathbb{Z}_p^*$.*

Step 1 *Choose uniformly at random $g \in \mathbb{Z}_p^*$;*

Step 2 **If** $g^2 \equiv 1 \pmod{p}$ *and* $g^q \not\equiv 1 \pmod{p}$ **then** *output g* **else** *go to* **Step 1**;

Notice that the probability of the algorithm has to repeat k times is roughly $1/2^k$ and thus, on average, this algorithm will find a generator after a small number of steps.

Another way of attacking the Discrete Logarithm is similar to attacks of the RSA cryptographic system. In the sequel it is discussed a problem that is equivalent to the Discrete Logarithm problem, namely the computation of the second least significant bit. Formally, consider the following problem:

Problem 10.3.8 (*Finding the i^{th}-bit of discrete logarithm*)

Let p be a prime, α a generator of $\mathbb{Z}_p^*$.

Instance $\beta \in \mathbb{Z}_p^*$;

Question *Find $\mathbb{L}_\alpha^i(\beta)$, the i^{th} less significance bit of $\log_\alpha \beta$.*

Proposition 10.3.9. *The problem $\mathbb{L}_\alpha^1(\beta)$ is in $\mathbb{P}$.*

Proof: Consider the function:

$$\begin{aligned} f : \mathbb{Z}_p^* &\rightarrow \mathbb{Z}_p^* \\ x &\rightarrow x^2 \pmod p \end{aligned}$$

Notice that f if $y^2 \equiv x^2 \pmod p$ then $y \equiv \pm x \pmod p$. In fact

$$\begin{aligned} y^2 \equiv x^2 \pmod p &\Leftrightarrow y^2 - x^2 \equiv 0 \pmod p \\ &\Leftrightarrow (y-x)(y+x) \equiv 0 \pmod p \\ &\Leftrightarrow p|(y-x)(y+x) \\ &\Leftrightarrow p|(y-x) \wedge p|(y+x) \\ &\Leftrightarrow y \equiv \pm x \pmod p \end{aligned}$$

Thus, if y is a quadratic residue modulo p then $|f^{-1}(y)| = 2$. Notice that f is computable in polynomial time. Since p is prime then $|\mathbb{QR}(p)| = \dfrac{p-1}{2}$ and thus:

$$\mathbb{QR}(p) = \left\{ \alpha^{2i} : 0 \leq i \leq \frac{p-3}{2} \right\}$$

then

$$\mathbb{L}_\alpha^1(\beta) = \begin{cases} 0 & \text{if } \beta^{(p-1)/2} \equiv 1 \pmod p \\ 1 & \text{otherwise} \end{cases}$$

■

Proposition 10.3.10. *Let p be a prime number such that $p \equiv 3 \pmod 4$. The problem* $\mathbb{D}\mathrm{iscrete}\ \mathbb{L}\mathrm{ogarithm}$ *is in $\mathbb{P}$ if and only if the problem $\mathbb{L}^2_\alpha(\beta)$ is in $\mathbb{P}$.*

Before presenting the proof of this result one needs the following lemma:

Lemma 10.3.11. *Let p be a prime number such that $p \equiv 3 \pmod 4$. $\mathbb{L}^1_\alpha(\beta) \neq \mathbb{L}^1_\alpha(-\beta)$.*

Proof: Notice that $\beta \equiv \alpha^a \pmod p$ and $-1 \equiv \alpha^{(p-1)/2} \pmod p$, since α is a generator. Thus $-\beta \equiv \alpha^{a+(p-1)/2} \pmod p$.

Since $p \equiv 3 \pmod 4$ then $(p-1)/2$ is odd and thus a and $a+(p-1)/2$ have different parities. Thus, $\mathbb{L}^1_\alpha(\beta) \neq \mathbb{L}^1_\alpha(-\beta)$. ■

Now it is presented the proof of proposition 10.3.10.

Proof:

($\Rightarrow$) This direction is easy to prove and the details are left for the interested reader.

($\Leftarrow$) Since $\log_\alpha \beta = \sum_{i=0}^{|p-1|-1} a_i 2^i \pmod p$ then $\beta = \alpha^a$ with a odd and thus $\beta^{(p+1)/4} = \pm\alpha^{a/2}$. So the last lemma allows us to compute it. In fact, if $\mathbb{L}^2_\alpha(\beta) = \mathbb{L}^1_\alpha(\beta^{(p+1)/4})$, then $\beta^{(p+1)/4} = \alpha^{a/2}$.

On the other hand, if $\mathbb{L}^2_\alpha(\beta) \neq \mathbb{L}^1_\alpha(\beta^{(p+1)/4})$, then $-\beta^{(p+1)/4} = \alpha^{a/2}$.

Hence the polynomial time algorithm to compute $\log_\alpha \beta$ can be the following one:

Algorithm 10.3.12

Input α, β;

Oracle *The value of* $\mathbb{L}^2_\alpha(\cdot)$;

Ouput $a_0, \cdots, a_{|p-1|-1}$, *where* $\log_\alpha \beta = \sum_{i=0}^{|p-1|-1} a_i 2^i$.

Step 1 $a_0 = \mathbb{L}^1_\alpha(\beta)$;

Step 2 $\beta = \beta \cdot \alpha^{-a_0}$;

Step 3 $i = 1$;

Step 4 **While** $(\beta \neq 1)$ **do** *:*

Step 4.1 $a_i = \mathbb{L}_\alpha^2(\beta)$*;*

Step 4.2 $\beta = \beta^{(p+1)/4} \pmod{p}$*;*

Step 4.3 **If** $a_i \neq \mathbb{L}_\alpha^1(\beta)$ **then** $\beta = -\beta \pmod{p}$*;*

Step 4.4 $\beta = \beta\alpha^{-a_i}$*;*

Step 4.5 $i = i + 1$*;*

Step 5 *Output* $a_0, \cdots, a_{|p-1|-1}$*;*

■

• •

SECTION 10.4

The elliptic curves

The use of elliptic curve in the public key cryptography was proposed independently by Neal Koblitz and Victor S. Miller in the middle of 80's and explores the algebraic properties and structure of elliptic curves over finite fields to design public key cryptographic systems that are believed to be secure, at least in computational term, i.e., secure in practice. These structures are also used in several integer factorization algorithms that have applications in cryptography, such as, for instance, Lenstra elliptic curve factorization, but this use of elliptic curves is "not" usually referred to as "elliptic curve cryptography."

Informally, an elliptic curve over the real plane is a set of points (x, y) that are solutions of the equations of the form $y^2 = x^3 + ax + b$. These equations, known as Weierstrass equations, can also be realized as a zero of a cubic function of two variables $f(x, y) = y^2 - x^3 - ax - b$ and for that reason they are sometimes also called cubic curves.

As it will be since in this section, the restriction of having elliptic curves over $\mathbb{R}^2$ is merely artificial and used for sake of readability and understanding. In fact, ti will be considered elliptic curves over finite fields in general.

The basic idea behind the use of these curves is the fact that one can form an abelian group with the set of points on such curve together with a special point, called a point at infinity which plays the role of the identity element of the group. Thus, if one choses (x, y) in a large finite field, the solutions form a finite abelian group and thus it is possible to used similar techniques too the ones used in the previous Section.

The resilience of elliptic curve based protocols against attacks is based on the hardness of discrete logarithm over a random elliptic curve elements with respect to a publicly known base point. The size of the elliptic curve determines the difficulty of the problem as it determines the base where the logarithm is performed. For the moment, the greatest advantage of the use of these structures when compared with their counterparts in $\mathbb{Z}_p$ is that they seem to be harder to break. Thus the keys keys for the protocols based on elliptic curve can be much smaller when compared with the cases of the last Section, to achieve a similar level of security confidence. It is also worthwhile mentioning that all this security is virtual in the sense, that similar to what happens with RSA foundations, the problem that elliptic curves rely on is not to know to be feasible. If one find a polynomial time algorithm to compute the discrete logarithm then all these theory can not be used in practice as a secure encryption scheme. Despite this fact, and like RSA, the U.S. National Security Agency has endorsed the cryptographic system based on elliptic curves as one of the recommended algorithms that everyone should use.

Due to its relevance that they will play in this section, it is worthwhile emphasizing the definition of an elliptic curve. Here it is already presented the most general definition of it. In particular one can define an elliptic curve over any finite field in a similar way that one would do over the field of real numbers.

Definition 10.4.1

Let $\mathbb{F}$ a field and let a, b two elements of $\mathbb{F}$. An elliptic curve E over over $\mathbb{F}[x, y]$ is defined as the following set:

$$E = \{(x, y) \in \mathbb{F} \times \mathbb{F} : y^2 - x^3 - ax + b = 0\}.$$

In the next figure it is presented the most common curves that one can use when $\mathbb{F}$ is the set of real numbers with classical operations.

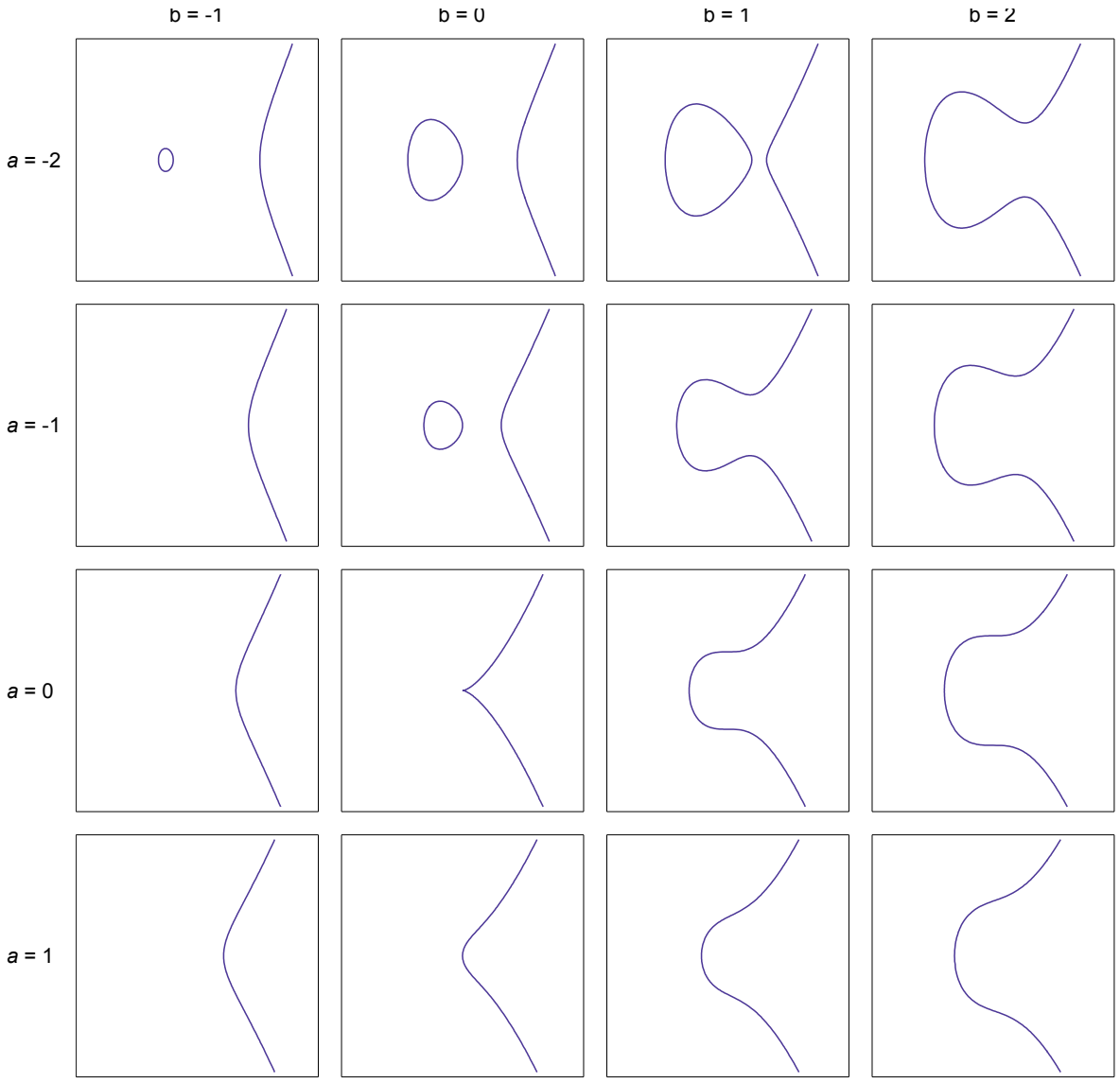


Figure 10.1: Examples of some Elliptic curves

The particular interest of this curves is when they are non-singlular, i.e., when they have three three distinct solutions which is equivalent to impose the condition that the partial derivatives do not vanish simultaneously for any point of the curve simultaneously. It can be shown, and is left to the interested reader to prove that the following condition is sufficient to guarantee the non-sigularity of the curve:

Theorem 10.4.1

Let $\mathbb{K}$ a field and E an elliptic curve over $\mathbb{K}[x,y]$, i.e., E is the set of solution of the function $f(x,y) = 0$ where $f(x,y) = y^2 - x^3 - ax + b$.
E is non singular if and only if $4a^3 + 27b^2$ is not zero in $\mathbb{F}$.

Usually the field that are used to consider in cryptography based on elliptic curves are of two special types: fields of odd characteristic i.e., $\mathbb{F}_p$, where $p > 3$ is a large prime number and fields of characteristic two, i.e., of the form $\mathbb{F}_{2^m}$. In the former case one uses simple modular algebra and in the latter one, also known as Galois fields, one uses polynomials and operations between them to represent the structure, that it will be explained in detail in due course. For this case, and for technical reasons, the set E is defined as the pairs (x,y) such that $y^2 - xy - x^3 - ax + b = 0$.

So, in order to use the this algebraic structures in cryptography it is necessary to define a binary operation over E, normally denoted by addition, that enriches E to an abelian group. Now it is explained how one can define the operations.

The addition of the point at infinity, which herein is denoted by $\mathcal{O}$, plays the role of the identity of the addition defined over the elliptic curve, i.e., for all $P \in E$

$$P + \mathcal{O} = \mathcal{O} + P = P$$

Now lets explain how the operation is determined.

Let $P = (x_P, y_P)$ be a point in the elliptic curve E and define $-P$ as the point with coordinates $(x_P, -y_P)$ if E is defined over $\mathbb{F}_p$ and define $-P$ as the point with coordinates $(x_P, x_P + y_P)$ if E is defined over $\mathbb{F}_{2^m}$. Clearly, one has $-P \in E$ for all $P \in E$.

Notice that if P and Q are two points on the curve E, then they uniquely determine a third point, that will be defined as their sum, which is the intersection of the curve with the line through P and Q. If that line is tangent to the curve at a point, then that point is counted twice and if the line is parallel to the y-axis, the third point is defined as the point "at infinity". Exactly one of these conditions then holds for any pair of points on an elliptic curve.

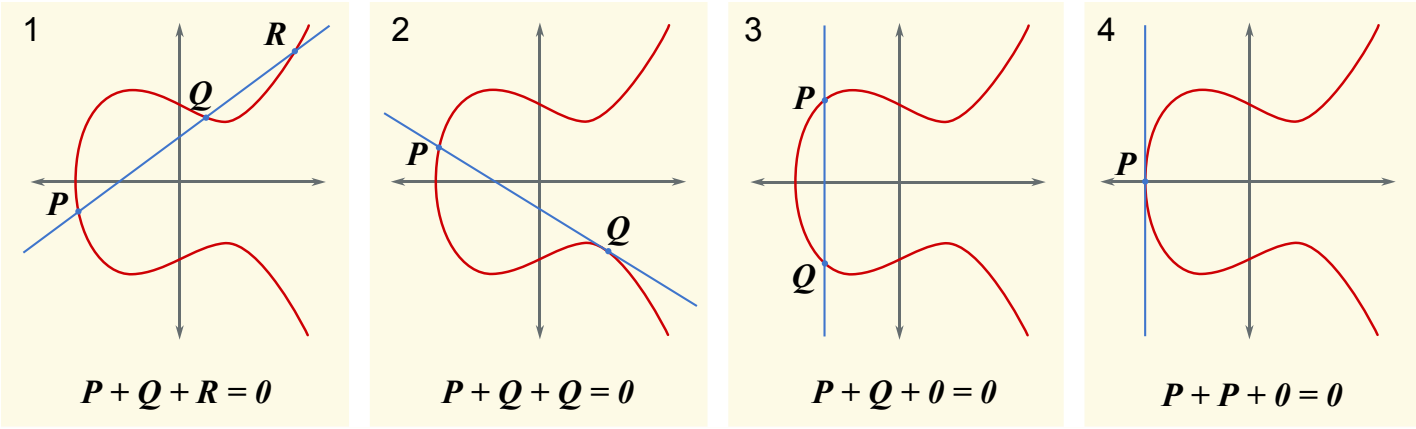


Figure 10.2: Some examples of how to define $p + q$ in an elliptic curve

Thus the operation $+$ over E can be defined more concretely as follows:

- if $Q = \mathcal{O}$ then $P + Q = P$;
- if $Q = -P$ then $P + Q = \mathcal{O}$;
- if $Q \neq P$ then define $P + Q$ to be the point $R = (x_R, y_R)$ where in the case E is defined using $\mathbb{F}_p$, $x_R = \lambda^2 - x_P - x_Q$, $y_R = \lambda(x_P - x_R) - y_P$, and $\lambda = \dfrac{y_Q - y_P}{x_Q - x_P}$. On the other hand, if E is defined using $\mathbb{F}_{2^m}$, $x_R = \lambda^2 + \lambda + x_P + x_Q + a$, $y_R = \lambda(x_P + x_R) + x_R + y_P$, and $\lambda = \dfrac{y_P + y_Q}{x_P + x_Q}$.[1]
- if $Q = P$ then define $P + Q$ to be the point $R = (x_R, y_R)$ where in the case E is defined using $\mathbb{F}_p$, $x_R = \lambda^2 - 2x_P$, $y_R = \lambda(x_P - x_R) - y_P$, and $\lambda = \dfrac{3x_P^2 + a}{2y_P}$. On the other hand, if E is defined using $\mathbb{F}_{2^m}$, $x_R = \lambda^2 + \lambda + a$, $y_R = x_P^2 + (\lambda + 1)x_R$, and $\lambda = x_P + \dfrac{y_P}{x_P}$.[2]

It is straightforward to prove E together with this binary operation is a group. Moreover, one can use Hasse's theorem on elliptic curves to estimate the number point on the elliptic curve.

[1] Geometrically, $P + Q$ is the inverse over the axis x of the third point of intersection of the cubic with the line through P and Q.

[2] Geometrically, $2P$ is the inverse of the third point of intersection of the cubic with its tangent line at P.

The theorem states this result showing that that number is close to the size of the underlying field. More precisely:

Theorem 10.4.2 (*Hasse theorem*)

Let E be an elliptic curve over a field of the form $\mathbb{F}_q$ where $q > 3$ is a prime number or $q = 2^m$ for some m. Then

$$(\sqrt{q} - 1)^2 \leq \#E \leq (\sqrt{q} + 1)^2$$

The exact computation of the cardinality of E can be done in polynomial time, due to Schoof, which runs in is $O((\log p)^8)$. Thus, if one can use further properties of this group, namely a cyclic subgroup in which the Discrete Logarithm problem is intractable then one can implement a generalized ElGamal cryptographic system in the elliptic curves. The following theorem gives the idea one can do that.

Theorem 10.4.3

Let E be an elliptic curve defined over $\mathbb{F}_q$ where $q > 3$ is a prime number. Then there exist positive integers n_1 and n_2 such that $(E, +)$ is isomorphic to $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$. Furthermore, $n_2 | n_1$ and $n_2 | (p - 1)$.

Clearly, one can have $n_2 = 1$, which happens if and only if E is a cyclic group. Also, if $\#E$ is a prime, or product of distinct primes, then E must be a cyclic group. In all the other cases one knows by the previous theorem if the integers n_1 and n_2 are computed, then one can compute the cyclic subgroup that can potentially be used to define an ElGamal generalized cryptographic system.

Notice that one can apply generic algorithms to the elliptic curve Discrete Logarithm problem. However, one can explore an explicit isomorphism between elliptic curves and finite fields that leads to efficient algorithms for certain classes of elliptic curves. This technique, due to Menezes, Okamoto and Vanstone, can be applied to some particular examples within a special class of elliptic curves called super singular curves that were suggested for use in cryptographic systems.

On the other hand one can define another kind of cryptographic system based on elliptic curves other than the generalized ElGamal cryptographic system. That scheme is presented now:

Cryptographic system 10.4.1 (*Menezes - Vanstone Cryptographic system*)

Consider p a prime number, E as the elliptic curve underlined by $\mathbb{Z}_p^*$ and H a cyclic subgroup of E. The Menezes - Vanstone Cryptographic system is the tuple $\mathcal{EV} = (P, C, K, E, D, R, r)$ where:

1. $P = \mathbb{Z}_p^* \times \mathbb{Z}_p^*$;
2. $C = E \times \mathbb{Z}_p^* \times \mathbb{Z}_p^*$;
3. *K consists of tuples $k = (\alpha, a, \beta)$ such that $\beta = a\alpha$ in E and α is a generator of H;*
4. $R = E^2$;
5. *Given a tuple (α, a, β), $r(\alpha, a, \beta) = (\alpha, \beta)$;*
6. *For each $k = (\alpha, a, \beta) \in K$ and all $x = (x_1, x_2) \in P$, $e_{r(k)}(x) = (y_1, y_2, y_3)$ where $y_1 = c\alpha$, where $c \in \mathbb{Z}$ is chosen at random and $y_2 = c_1 x_1 \pmod p$ and $y_3 = c_2 x_2 \pmod p$ where $(c_1, c_2) = c\beta$.*
7. *$d_{(\alpha,a,\beta)}(y_1, y_2, y_3) = (x_1, x_2)$ where $x_1 = y_2 c_1^{-1} \pmod p$ and $x_2 = y_2 c_2^{-1} \pmod p$, where $(c_1, c_2) = a y_1$.*

CHAPTER 11

Another approach to factorization - Shor's quantum algorithms

In last couple of chapters, it was discussed cryptography based of presumably hard problems such as factoring composite numbers and the discrete logarithm problem and all the security that every cryptography relies on is the fact that no one have ever came with an efficient algorithm to solve these problems. But since the advent of quantum computing this apparent idea of security it is dangerous as it is similar to be walking on thin ice. In fact, the apparent idea of ??security is so misleading that since the advent of quantum computation it is known that cryptography as described before, i.e., based on the hardness of factoring composite numbers and the discrete logarithm is no longer possible as it had been proved that with a mechanism that is capable of dealing with quantum information at hand, one can factor number and determine the discrete logarithm among other apparently hard problems for classical computers very efficiently.

Hence, in this chapter it is discussed a very different approach to cryptology, i.e., the science of code breaking. It is based on the quantum computation, a conceptually different paradigm of computation that is governed by the rules of the quantum mechanics.

There are several reasons to justify the study of quantum computation. In particular, several experiments had showed that one can, in fact, implement a quantum mechanics models to

capture accurately real physical systems and the interactions between them. Furthermore these experiments show that this model is more efficient than classical model of Turing machines in the sense that are problems provably exponentially more efficiently solvable in a quantum computing model than in a classical one; It is also robust, since one can perform error correction, to make implementations robust to noise. This is an essential feature for implementability, which several analog computing models cannot achieve.

Bits, or "binary digits" lie at the heart of all modern digital equipment and nowadays computers use voltage levels to encode bits. All digital devices deal with them and one can use their properties without even noticing them. One can read a bit to learn its value, one can copy, erase or negate it; and one can read some of the bits inside a digital device without changing the other bits that one did not read. In fact such properties seem so obvious that no one even bothers to question these assumptions. However, Richard Feynman anticipated that very small physical devices would be governed by quantum mechanics rather than classical mechanics and, accordingly, would not necessarily behave similarly as their larger counterparts. At the subatomic level a bit can be 0 or 1 at the same time and if one wishes to know, for sure its value, one has to perform an observation. This interaction with the system makes a permanent record of the result and hence its behavior becomes definite. That sounds crazy, but that is what quantum mechanics tells that actually happen.

As devices become smaller and smaller the physical systems used to encode those bits will become smaller and their behavior will need to be described using quantum mechanics.

At this level the common sense assumptions about how bits ought to behave (read, copy, erase, negate them without causing them to change in any way) might not be true anymore. In fact, at the quantum level one cannot necessarily read a bit without changing its value; one cannot copy, or negate it without perturbing it; one may be unable to erase it; and sometimes when one reads the smallest piece of information, these actions can change the state to another with which the reader had never interacted with. Thus, bits encoded in quantum-scale objects cease to behave like normal bits ought.

Quantum computation and quantum information are built upon an analogous concept to bit, which are known as quantum bit or shortly by qubit. The idea is that quantum computation systems is one can encode information in a physical states of the system. For example, when

the spin of a electron is measured it can be found in only two possible values, either "spin up" ($|\uparrow\rangle$) parallel to the axis or "spin down" ($|\downarrow\rangle$) anti-parallel to the axis along which the measurement was taken. This intrinsic discreteness, a manifestation of quantization, allows the spin of an electron to be considered as a natural binary digit or "bit". The resulting system is the quantum bit.

.....

SECTION 11.1

Notation

Now it is presented the classical notation used in the quantum computational field. The information in quantum is quite different from the classical information. Rather than having just two possible states for each unit of information, one can have a quantum bit into an infinite range. In fact, the quantum bit exists in a superposition of two states. A quantum bit is usually represented in Dirac notation as a "ket" of the form $|\phi\rangle$ and in general one considers two directions, more accurately spins, $|0\rangle$ or $|1\rangle$. Thus $|\psi\rangle$ is a linear combination of the form $\alpha|0\rangle + \beta|1\rangle$, where α and β are complex numbers.

The possible infinite amount of information in a quantum bit has a drawback. One can not observe them, i.e, the observation of a quantum bit is doable up to the penalty to collapse of the superposition into one of the directions considered. Either $|0\rangle$ or $|1\rangle$.

This presents the first restriction that one must consider for each quantum bit. In fact, a quantum bit is of the form $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ where α and β must satisfy $|\alpha|^2 + |\beta|^2 = 1$.

Similarly to what happens in the classical information, the power of the quantum information is given possibility of combining two or more basic information units. In the quantum information theory, to combine several quantum bits, instead of simply concatenate, similar to classical, we need to adopt the tensor product, represented by $\otimes$.

SECTION 11.2

The notion of a qubit and its representation

As mentioned before, qubit is the counterpart in quantum computation to bits in the classical computational model. Although a qubit is a realization of a physical state one can describe them as mathematical objects with certain specific properties. Here, as in common literature, it is adopted the notation invented by Paul Dirac, which has been called since then known as "Dirac-notation". The great advantage of dealing with qubits as abstract entities is to make it possible to construct a general theory of quantum computation and quantum information which does not depend upon a specific system for its realization.

In Dirac notation, and for any realization of the quantum system, a qubit in a physical state that represents the bit value 0, one will write the qubit state using an angular looking bracket, $|0\rangle$, which is called a "ket" vector. Likewise, a qubit in a physical state representing the bit value 1 will be written $|1\rangle$. Mathematically, kets are a shorthand notation for column vectors, with $|0\rangle$ and $|1\rangle$ corresponding to:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Two symbols are not enough to represent completely the quantum system. In fact, a state, whenever it is not being observed, it may be in a combination of those two observable states (called in a superposition) but when one observes it, it collapses in the classical bits $|0\rangle$ or $|1\rangle$. Intuitively, a system in quantum state is in all classical states at the same time! Thus, a very significant difference between bits and qubits is that a qubit can be in a state other than $|0\rangle$ or $|1\rangle$.

It is also possible to form linear combinations of the form $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$. The values α and β are complex numbers and called the amplitudes of $|i\rangle$ in $|\phi\rangle$.

The laws of quantum mechanics determine that one cannot examine a qubit to determine the

values of the amplitudes. Recall that once a measure is done, i.e., an observation is performed, the quantum bit collapses in one of the classical states. Hence one is only able to acquire much more restricted information about the quantum state using a proper normalization that guarantees that these classical states are reached, 0 with probability $|\alpha|^2$, and 1 with probability $|\beta|^2$, and thus, the amplitudes satisfy $|\alpha|^2 + |\beta|^2 = 1$.

Thus, a single qubit is an element of the vector space $\mathbb{C}^2$. In fact, geometrically, one can see a qubit as an element of a sphere to radius 1 (known as the Bloch sphere). In general a qubit's state is a unit vector in a two-dimensional complex vector space.

One can use a similar idea to work with more than one qubit. They are formally represented by a tensor product space of several qubit systems. For example, a 2-qubit system has 4 basis states: $|0\rangle \otimes |0\rangle$, $|0\rangle \otimes |1\rangle$, $|1\rangle \otimes |0\rangle$ and $|1\rangle \otimes |1\rangle$ where the first element represents the sate of the first qubit and the second element represents the state of the second qubit. For simplicity, it is common to represent $|0\rangle \otimes |0\rangle$ by the following short notation $|00\rangle$.

More generally, a register of n qubits has 2^n basis states, which is of the form $|b_1\rangle \otimes \cdots \otimes |b_{2^n}\rangle = |b_1 \cdots b_{2^n}\rangle$. Since any bit string of length n can be viewed as numbers between 0 and $2^n - 1$, one can also write the basis states as numbers $|0\rangle$, ..., $|2^{n-1}\rangle$. A quantum register of n qubits can be in any superposition

$$\alpha_0|0\rangle + \cdots + \alpha_{2^n-1}|2^n - 1\rangle$$

or

$$\alpha_0|0\cdots 0\rangle + \alpha_1|0\cdots 01\rangle + \cdots + \alpha_{2^n-1}|1\cdots 1\rangle$$

and the amplitudes α_i satisfy $\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$. If one performs a measure in the standard basis, we obtain a n-bit state, and in particular one gets the state $|j\rangle$ with probability $|\alpha_j|^2$.

Here we analyze how quantum computation is performed and the kind of operations we can apply in quantum information.

It is known and it will be discussed later in this chapter, that quantum computation must obey some rules in order to be consistent. Notice that, for example, a quantum qubit $\alpha|0\rangle + \beta|1\rangle$ must satisfy that $|\alpha|^2 + |\beta|^2 = 1$. It can be seen in several ways, but remember, for example the Bloch sphere, where all vectors are expected to have norm 1. Thus, when one applies

a quantum operation to a qubit, at the end of the evolution, and in fact in between, one expects that the resulting quantum bit satisfies the same rule. This give the main criteria to the kind of operations one must deal with in quantum computation. Indeed, any quantum transformation must be unitary. Consequently all quantum transformations preserve inner products and hence also norms.

It is now surveyed the most used quantum operations by presenting several examples of quantum circuits and explain their effect on quantum states.

Now it is paid attention to the description of how one can use quantum model of representing information to do computations, i.e., changes to the quantum states, by describing how qubits can evolve within a qauntum system. There are several approaches to describe the evolution of the system that all equivalent.

As discussed in the introduction of this chapter all transformation that one can apply to a quantum register must be unitary. The formal definition of this notion is as follows:

Definition 11.2.1

Suppose A is any linear operator on a Hilbert space, V.
Then, there exists unique linear operator $A^\dagger$ on V such that for all vectors $v, w \in V$,

$$\langle v|Aw\rangle = \left\langle A^\dagger v|w\right\rangle$$

This linear operator is known as the adjoint or Hermitian conjugate of the operator A. From the definition it is easy to see that $(AB)^\dagger = B^\dagger A^\dagger$.
If the operator A is represented by a matrix, the action of the Hermitian conjugation operation corresponds to take the matrix of A to the conjugate-transpose matrix, $A^\dagger = (A^*)^T$, where the * indicates complex conjugation, and T indicates the transpose operation.
A transformation A such that $A(A)^\dagger = I$ is called unitary.

Example: Let A be the operator described by the matriz

$$A = \begin{bmatrix} 1+i & 2i \\ 1 & 4-i \end{bmatrix}$$

Then its adjoint is:

$$A^\dagger = \left(\begin{bmatrix} 1+i & 2i \\ 1 & 4-i \end{bmatrix}^* \right)^T = \left(\begin{bmatrix} 1-i & -2i \\ 1 & 4+i \end{bmatrix} \right)^T = \begin{bmatrix} 1-i & 1 \\ -2i & 4+i \end{bmatrix}$$

An operator A whose adjoint is itself, i.e., $A^\dagger = A$ is called a Hermitian or self-adjoint operator. An important example of Hermitian operators is the projectors. If W is a subspace of dimension k of V (of dimension $d > k$) then the projection defined of V on W is defined by

$$P = \sum_{i=1} k|i\rangle\langle i|$$

where we assume that the base of V is $|1\rangle, \cdots |d\rangle$.

An operator A is normal if $AA^\dagger = A^\dagger A$. Clearly, an operator which is Hermitian is also normal.

Theorem 11.2.1

A normal matrix is Hermitian if and only if it has real eigenvalues.

A matrix U is unitary if $U^\dagger U = UU^\dagger = I$, i.e., $U^{-1} = U^\dagger$. It is easily checked that an operator is unitary if and only if each of its matrix representations is unitary. A unitary operator is normal and has a spectral decomposition. Geometrically, unitary operators are important because they preserve inner products between vectors.

The reader showed check these equivalences to start to familiarize with these concepts.

• • •

11.2.1 Scoot Aaronson explains the Shor's algorithm

Before presenting the algorithm, in this subsection, it is very interesting to present the discussion about Shor's algorithm in Scott Aaronson's blog [**?**]. The description that he pulled out is very simple and clear about this quantum algorithm and what it is all about. He is a very smart guy and is specially gifted to put complex things into a very understandable way.

After this discussion, it presented the ground concepts need to understand the Shor's algorithm. Namely, it will discuss the quantum Fourier transformation, the quantum algorithm of period finding (the kernel of speed up when compared with the classical counterpart) and then the algorithm itself. An example that illustrates the algorithm is also given.

The importance of this algorithm is mainly due to the security code used in almost every place we look at, the RSA code. Breaking RSA is just a matter of finding the prime factors of a large integer N. If we try to do it classically it requires exponential resources. The idea is to use the quantum phenomena to design a fast quantum factoring algorithm and it is necessary to exploit some structure in the factoring problem.

The properties that are important to explore in the factoring problem is the uniqueness on the factorization itself and the reducibility of factoring to another problem, called period-finding. Consider the sequence of powers of two:

$$2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, \cdots$$

and consider them " mod 15": 2, 4, 8, 1, 2, 4, 8, 1, 2, 4, $\cdots$.

Taking the powers of 2 mod 15 gives us a periodic sequence, whose period is 4. This a general property and is a beautiful pattern discovered by Euler in the 1760's: Let N be a product of two prime numbers, p and q, and consider the sequence x mod N, x^2 mod N, x^3 mod N, x^4 mod N, $\cdots$. If x is not divisible by p or q, the sequence will repeat with some period that evenly divides $(p-1)(q-1)$.

So for example, if $N = 15$, then the prime factors of N are $p = 3$ and $q = 5$, so $(p-1)(q-1) = 8$. And indeed, the period of the sequence was 4, which divides 8.

So, this means that if we can find the period of the sequence x mod N, x^2 mod N, x^3 mod N, x^4 mod N... then we can learn something about the prime factors of N! Indeed we can get a divisor of $(p-1)(q-1)$. Indeed, it's more than something: it turns out that if we could learn several random divisors of $(p-1)(q-1)$ (for example, by trying different random values of x), then with high probability we could put those divisors together to learn $(p-1)(q-1)$ itself. And once we knew $(p-1)(q-1)$, we could then use some more little tricks to recover p and q, the prime factors we wanted. In fact if r is the period can be shown that with probability $\geq 1/4$, r is even and $x^{r/2}+1$ and $x^{r/2}-1$ are not multiples of N and then $x^r - 1 \equiv x^{r/2} - 1 \equiv (x^{r/2}+1)(x^{r/2}-1) \equiv 0$, and thus $x^{r/2}+1$ or $x^{r/2}-1$ will share a proper factor with N. (So we get it using the Euclides' algorithm for the great common

divisor). This is why finding the period does not seem to lead to a fast classical factoring algorithm.

But we have a quantum computer! The key point is that we are no longer trying to find a needle in an exponentially-large haystack, something we know is hard even for a quantum computer. Instead, we are now trying to find the period of a sequence, which is a global property of all the numbers in the sequence taken together. And that makes a big difference.

If we want to get this period-finding idea to work, we'll have to answer two questions:

1. Using a quantum computer, can we quickly create a superposition over $x \mod N$, $x^2 \mod N$, $x^3 \mod N$, and so on?
2. Supposing we did create such a superposition, how would we figure out the period?

It is easy to create a superposition over all integers r, from 1 up to N or so. To produce $x^r \mod N$ we can do repeated squaring. If $N = 17$, $x = 3$ and $r = 14$. Then the first step is to represent r as a sum of powers of 2: $r = 2^3 + 2^2 + 2^1$. Then

$$x^r = 3^{14} = 3^{2^3+2^2+2^1} = 3^{2^3} + 3^{2^2}3^{2^1} = ((3^2)^2)^2(3^2)^2 3^2.$$

Another advantage is to do all the multiplications $\mod N$, thereby preventing the numbers from growing out of hand at intermediate steps.

So given a superposition over all the elements of a periodic sequence, how do we extract the period of the sequence?

Finally we come to the heart of the matter - the part of the algorithm that actually depends on quantum mechanics. To get the period, Shor used the quantum Fourier transform, or QFT.

To explain it consider the famous experiment where they stick people for weeks in a sealed room without clocks or sunlight, and the people gradually shift from a 24-hour day to a 25- or 26- or 28-hour day? One day, you wake up at 9am, the next day

at 11am, the day after that at 1pm, etc. Lets say I tell you that I woke up at 5pm this afternoon. From that fact alone, what can you conclude about how long my "day" is?

The answer, of course, is not much! Now, though, I want you to imagine that the bedroom wall is covered with strange clocks: one of them makes a full revolution every 17 hours, one of them every 26 hours, one of them every 24.7 hours, and so on for just about every number of hours you can imagine and beneath each clock is a posterboard with a thumbtack in it. When you first moved into the bedroom, each thumbtack was in the middle of its respective board. But now, whenever you wake up in the "morning" the first thing you do is to go around the room, and move each thumbtack exactly one inch in the direction that the clock hand above it is pointing. Now, the new question: by examining the thumbtacks in the room, is it possible to figure out what sort of schedule you're keeping?

The answer is Yes. As an example, suppose you keep a 26-hour day. Then what would happen to the thumbtack below the 24-hour clock? It is not hard to see that it would undergo periodic motion: sure, it would drift around a bit, but after every 12 days it would return to the middle of the board where it had started. One morning you had move the thumbtack an inch in this direction, another morning an inch in that, but eventually all these movements in different directions would cancel each other out.

On the other hand and "again supposing you were keeping a 26-hour day" what would happen to the thumback below the 26-hour clock? Here the answer is different. For as far as the 26-hour clock is concerned, you have been waking up at exactly the same time each "morning" Every time you wake up, the 26-hour clock is pointing the same direction as it was the last time you woke up. So I'll keep moving the thumbtack one more inch in the same direction, until it's not even on the posterboard at all!

It follows, then, that just by seeing which thumbtack traveled the farthest from its starting point, you could figure out what sort of schedule you were on. And that, basically, is the quantum Fourier transform.

A little more precisely, the *QFT* is a unitary transformation that maps one vector of complex numbers to another vector of complex numbers. The input vector has a nonzero entry corresponding to every time when you wake up, and zero entries everywhere else. The output vector records the positions of the thumbtacks on the posterboards (which one can think of as points on the complex plane). So what we get, in the end, is a linear transformation that maps a quantum state encoding a periodic sequence, to a quantum state encoding the period of that sequence.

Another way to think is in terms of interference. The key point about quantum mechanics - the thing that makes it different from classical probability theory - is that, whereas probabilities are always nonnegative, amplitudes in quantum mechanics can be positive, negative, or even complex. And because of this, the amplitudes corresponding to different ways of getting a particular answer can "interfere destructively" and cancel each other out.

And that is exactly what is going on in Shor's algorithm. Every "parallel universe" corresponding to an element of the sequence contributes some amplitude to every "parallel universe" corresponding to a possible period of the sequence. The catch is that, for all periods other than the "true" one, these contributions point in different directions and therefore cancel each other out. Only for the "true" period do the contributions from different universes all point in the same direction. And that is why, when we measure at the end, we will find the true period with high probability.

Scoot Aaronson

• • •

11.2.2 The Shor's algorithm for factoring

Before presenting the algorithm it is needed some background on quantum Fourier transformations as these elements are crucial for the understanding of the result.

• • •

11.2.2.1 Quantum Fourier transformation

If computers that you build are quantum, then spies everywhere will all want 'em.
Our codes will all fail, and they'll read our email. Till we get crypto that's quantum, and daunt 'em.

Jennifer and Peter Shor

The quantum Fourier transform is a linear transformation on quantum bits, and is the quantum counterpart of the discrete Fourier transform and can be efficiently implemented using $O(n^2)$ quantum gates (Hadamard and controlled phase shifts). The classical discrete Fourier transform requires $O(n2^n)$ gates to be implemented. However, the quantum Fourier transform acts on a quantum state, whereas the classical Fourier transform acts on a vector and it does not speed up the classical task of computing Fourier transforms of classical data.

Classically the Fourier transform takes a vector of complex numbers, $x_0, \cdots x_{N-1}$ where the length N of the vector is a fixed parameter and outputs the transformed data, a vector of complex numbers $y_1, \cdots, y_{N-1}$ defined by

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{\frac{2 \cdot \pi \cdot i \cdot j \cdot k}{N}}$$

The quantum Fourier transform is the quantum counterpart of this transformation. It is a linear operator that on the basis states of an orthonormal basis $|0\rangle \cdots |N-1\rangle$ operates by:

$$|k\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{\frac{2\cdot\pi\cdot i\cdot j\cdot k}{N}} |j\rangle$$

Thus, on a generic quantum state $\sum_{j=0}^{N-1} x_j|j\rangle$ the quantum Fourier transformation acts transforming it into $\sum_{j=0}^{N-1} y_j|j\rangle$ where $y_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} x_k e^{\frac{2\cdot\pi\cdot i\cdot j\cdot k}{N}}$ is the classical Fourier transformation.

Equivalently, the quantum Fourier transform can be viewed as the following unitary matrix where $\omega = e^{\frac{2\pi i}{N}}$, the N^{th} square root of 1:

$$QFT = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{pmatrix}$$

Theorem 11.2.2

The quantum Fourier transformation is unitary.

Proof: To prove this theorem it is enough to check if any two columns in the transformation are orthogonal. Notice that a transformation is unitary if and only if it preserves the inner product if and only if transforms orthogonal basis into orthogonal basis.

Thus all it is needed to check if the inner product of any two columns is equal to the Kronecker symbol, i.e.:

$$\langle i|QFT^* QFT|j\rangle = \delta_{i,j} = \begin{cases} 1 & \text{if } i = j \\ \\ 0 & \text{if } i \neq j \end{cases}$$

Computing the inner product one has

$$\langle i|QFT^* QFT|j\rangle = \frac{1}{N}\sum_{k=0}^{N-1} \overline{\omega^{i\cdot k}}\omega^{j\cdot k} = \frac{1}{N}\sum_{k=0}^{N-1} \omega^{k(j-i)}$$

which is a geometric series with ratio ω^{j-i}. Thus if $i = j \mod N$, then each term is equal to $\omega^0 = 1$ and the inner product is then equal to $N/N = 1$.

If $i \neq j$, then the sum is

$$1 + \omega^{j-i} + \omega^{2(j-i)} + \cdots + \omega^{(N-2)(j-i)} + \omega^{(N-1)(j-i)}.$$

Notice that if we multiply the sum by ω^{j-i} one gets,

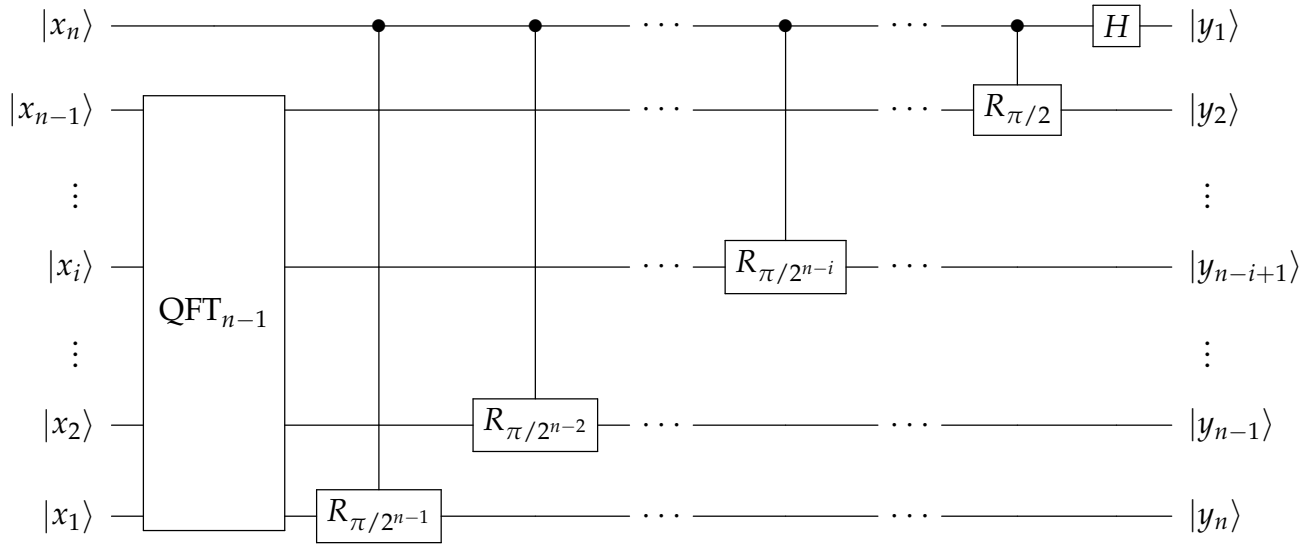
$$\omega^{j-i} + \omega^{2(j-i)} + \omega^{3(j-i)} + \cdots + \omega^{(N-1)(j-i)} + \omega^{N(j-i)}.$$

But, since ω is a n^{th} root of 1, $\omega^{N(j-i)} = 1$. Thus the multiplication it is just an rearrangement of the sum and thus, since $\omega^{j-i} \neq 0$ one concludes that

$$1 + \omega^{j-i} + \omega^{2(j-i)} + \cdots + \omega^{(N-2)(j-i)} + \omega^{(N-1)(j-i)} = 0.$$

■

The following circuit implements the quantum Fourier transformation:



The quantum Fourier transform can be approximately implemented for any N and in the case where $N = 2^n$ one has a special representation of the quantum Fourier transformation with respect to an orthonormal basis consisting of the vectors $|0\rangle, \cdots |2^n - 1\rangle$. Notice that $x = x_1 2^{n-1} + x_2 2^{n-2} + \cdots x_n 2^0$ can be represented by $|x\rangle = |x_1 \cdots x_n\rangle = |x_1\rangle \cdots |x_n\rangle$. Also, if x is a decimal number between 0 and 1 one can represent it using the decimal notation as

$$[0.x_1 \cdots x_m] = \sum_{k=0} x_k 2^{-k}.$$

Thus the quantum Fourier transform can be given by the following product representation:

$$|x_1, x_2, \cdots, x_n\rangle \rightarrow \begin{array}{l} \frac{1}{\sqrt{N}}(|0\rangle + e^{2\pi i[0.x_n]}|1\rangle) \otimes (|0\rangle + e^{2\pi i[0.x_{n-1}x_n]}|1\rangle) \otimes \\ \otimes \cdots \otimes (|0\rangle + e^{2\pi i[0.x_1 \cdots x_{n-1}x_n]}|1\rangle) \end{array}$$

This representation give the direction to construct an efficient quantum circuit computing the Fourier transform, a proof that the quantum Fourier transform is unitary, and provides insight into algorithms based upon the quantum Fourier transform.

• • •

11.2.2.2 The period finding

The problem of factoring numbers on a classical computer turns out to be equivalent to another problem, the order-finding problem. This equivalence is important as it turns out that quantum computers are able to quickly solve the order-finding problem, and thus can factor quicker than one would do in a classical computer.

The order-finding problem is believed to be a hard problem classically, in the sense that, given N an integer, no algorithm is known to solve it within polynomial resources, i.e., in $O(\log N)$ steps.

Here it is described the procedure to find the period of a function. One is given a function f and an integer N with the promise that there is a period, call it $a \leq N$, such that for all x, y one has $f(x) = f(y)$ if and only if $y \in \{x, x^a, x^{2a}, \cdots\}$. The period finding problem is just to determine a.

Problem 11.2.1 (*period finding*)

Instance: *A function f;*

Promise: *f is periodic;*

Question: *Find the period of the function.*

Algorithm 11.2.2 (*Period finding*)

Input: *Let L be equal to $\log N$. The inputs to the algorithm are:*

1. *A black box implementing the unitary transformation*

$$U_{x,N}(|j\rangle|k\rangle) = |j\rangle\left|x^j k \mod N\right\rangle$$

   where x is co-prime with N;
2. $t = 2L + 1 + \left\lceil \log\left(2 + \frac{1}{2e}\right)\right\rceil$ *qubits initialized to* $|0\rangle$.
3. *L qubits on the state* $|1\rangle$.

Output: *The least integer $r > 0$ such that $x^r \equiv 1(\mod N)$.*

Runtime: $0(L^3)$ *operations.*

Success probability: $0(1)$.

Step 1 *Apply the Quantum Fourier transformation to the first register of $|0\cdots 0\rangle|1\cdots 1\rangle$ to get:*

$$\frac{1}{\sqrt{2^t}}\sum_{j=0}^{2^t-1}|j\rangle|1\rangle$$

Step 2 *Apply $U_{x,N}$, and the result is:*

$$\frac{1}{\sqrt{2^t}}\sum_{j=0}^{2^t-1}|j\rangle\left|x^j \mod N\right\rangle$$

Step 3 *Measuring the second register we get a random $|x^s \mod N\rangle$ with $s \leq r$ and the first register collapses to a superposition of $|s\rangle$, $|s+r\rangle$, $|s+2r\rangle$, $\cdots$, $|q-r+s\rangle$, where $q = \left\lfloor \frac{N}{r} \right\rfloor (-1)$, (the -1 term is used in the case $r|q$).*

Step 4 **If** $r|q$ **then** *the last state collapses into:*

$$\frac{1}{\sqrt{q}}\sum_{j=0}^{q/r-1}|jr+s\rangle$$

and one can apply a quantum Fourier transformation to obtain:

$$\sum_{j=0}^{q/r-1}\sum_{b=0}^{q-1}e^{2\pi i\frac{(jr+s)b}{q}}|b\rangle = \sum_{b=0}^{q-1}e^{2\pi i\frac{sb}{q}}\left(\sum_{j=0}^{q/r-1}\left(e^{2\pi i\frac{sb}{q}}\right)^j\right)|b\rangle$$

Notice that $\sum_{j=0}^{q/r-1}\left(e^{2\pi i\frac{sb}{q}}\right)^j$ is a geometric sum. Notice that if a is the reason of the geometric sum, then $a^0+\cdots+a^n = \frac{1-a^{n+1}}{1-a}$ which is equal to q if sb/r is an integer and 0 otherwise. Thus, in the first register, one only observes a state of the form $b = c\frac{q}{r}$ with $c < r$ and random.

$$b = c\frac{q}{r} \Leftrightarrow \frac{b}{q} = \frac{c}{r}$$

If c is co-prime with r (which happens with probability $\Omega(1/\log\log r)$) the algorithm is done, since one knows b and q. In the case it is not known if c is co-primality, running $O(\log\log N)$ times this procedure one gets a factor of r and using the least common multiplier one can recover r.

else *i.e. $r \not| q$, then one can not any longer write $\frac{b}{q} = \frac{c}{r}$ but with very high probability one observes b such that*

$$\left|\frac{b}{q} - \frac{c}{r}\right| \leq \frac{1}{2q}.$$

Since there is only one fraction with denominator $< N$ in an interval of length $\frac{1}{q} \leq \frac{1}{N^2}$ (see Theorem 11.2.3 below) one can compute $\frac{c}{r}$ using the continued fraction algorithm (see Theorem 11.2.4).

Theorem 11.2.3

Let ϕ be a number between 0 and 1. If α/β is a rational number such that

$$\left|\frac{\alpha}{\beta} - \phi\right| \leq \frac{1}{2\beta^2}$$

then $\frac{\alpha}{\beta}$ is convergent of the continued fraction for ϕ and can be computed fast by th Euclid's algorithm.

Proof: TBW... ■

In the case considered, this is sufficient since it is known that ϕ up to a precision of $2L+1$ and then it follows that

$$\left|\frac{b}{q} - \frac{c}{r}\right| \leq 2^{-2L-1} \leq \frac{1}{2r^2}$$

as $r \leq N \leq 2^L$.

Theorem 11.2.4

Let x be any rational number. One can write x as

$$[a_0, \cdots, a_n] = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{1... + \cfrac{1}{M}}}}.$$

Proof: TBW ■

Example 11.2.3.

1. *The continued fraction of* $\frac{100}{13}$ *is* $[7,1,2,4]$

$$\frac{100}{13} = 7 + \frac{9}{13} = 7 + \frac{1}{\frac{13}{9}} = 7 + \frac{1}{1 + \frac{4}{9}} = 7 + \frac{1}{1 + \frac{1}{\frac{9}{4}}}$$

$$= 7 + \frac{1}{1 + \frac{1}{2 + \frac{1}{4}}} = [7,1,2,4]$$

2. *The continued fraction of* $\frac{31}{13}$ *is* $[2,2,1,1,2]$

$$\frac{31}{13} = 2 + \frac{5}{13} = 2 + \frac{1}{\frac{13}{5}} = 2 + \frac{1}{2 + \frac{3}{5}} = 2 + \frac{1}{2 + \frac{1}{\frac{5}{3}}} = 2 + \frac{1}{2 + \frac{1}{1 + \frac{2}{3}}}$$

$$= 2 + \frac{1}{2 + \frac{1}{1 + \frac{1}{\frac{3}{2}}}} = 2 + \frac{1}{2 + \frac{1}{1 + \frac{1}{1 + \frac{1}{2}}}}$$

$$= [2,2,1,1,2]$$

...

11.2.2.3 Putting together all the ingredients

It is now time to put all the pieces together and present the quantum algorithm for factoring. This algorithm has a major breakthrough in the end of last decade of the last century. It was presented by Peter Shor.

Algorithm 11.2.4 (*Shor*)

Input: *A number N that is assumed to be composed.*

Output: *A non trivial factor of N.*

Runtime: *Expected $O((\log N)^3)$.*

Step 1: *Choose randomly $r \in \{2, \cdots, N-1\}$.*

Step 2: *Compute $gcd(x, N)$.*

Step 3: **If** *$GCD(x, N) > 1$ then apply the Euclid's algorithm to get a non trivial facto of N,* **else** *continue to the next step.*

Step 4: *Use the order-finding subroutine to find the order r of x modulo N.*

(From a pigeon and hole principle it is know that in $x^1, \cdots, x^{N+1}$ there are at least two numbers that are equal.

Assume that $b > a$ and $x^a \equiv x^b \mod N$. Since all $x^1, \cdots, x^{N+1}$ are co prime with N then $x^{a-b} \equiv 0 \mod N$).

Step 5: *With probability at least $1/4$, r is even, and $x^{r/2} + 1$ is not a multiple of N, i.e., $x^{r/2} + 1 \not\equiv 0 \mod N$.*

Step 6: *Test, using the Euclid's algorithm if $gcd(x^{r/2} - 1, N) > 1$ or $gcd(x^{r/2} + 1, N) > 1$. Otherwise return to the first step and restart.*

Notice that if one is fortuned to get randomly a factor of N then there is no need to run the rest of the algorithm. Step 3 calls the order-finding subroutine, to compute the order r of x modulo N and is indeed the only part of the algorithm that requires the use quantum computers to be sufficiently fast, namely to run in polynomial time. The last step completes the algorithm, since by the next theorem, with probability at least one-half r will be even and $x^{r/2}+1 \not\equiv 0 \mod N$. Notice that, $x^r \equiv 1 \mod N \Leftrightarrow (x^{r/2})^2 \equiv (x^{r/2}-1)(x^{r/2}+1) \equiv 0 \mod N$. So applying the Euclid's algorithm one can get a non trivial factor of N.

Theorem 11.2.5

Consider a positive integer N and assume that $N = p_1^{\alpha_1} \cdot \cdots \cdot p_m^{\alpha_m}$ is composition in primes of N. Let x be an integer chosen uniformly at random in $\{1, \cdots, N-1\}$ that is co-prime to N. If r in the order of x modulo N, then

$$\Pr(r \text{ is even and } x^{r/2} \equiv -l \mod N) \geq 1 - \frac{1}{2^m}.$$

• • •

11.2.2.4 An example of the Shor's algorithm for n=15

Algorithm 11.2.5

Input: $N = 15$.

Output: *A factor of* 15*, namely 3 or 5.*

Step 1 *Choose $n = 4$ so that $15 < S = 2^4 = 16$.*

Step 2 *Choose y so that $gcd(y, 15) = 1$. For example, if $y = 13$.*

Step 3 *Initialize two qubit registers to state 0, i.e., $|\psi_0\rangle = |0\rangle|0\rangle$.*

Step 4 *Apply Hadamard to the first qubit to get:*

$$|\psi_1\rangle = \frac{1}{\sqrt{16}} \sum_{k=0}^{15} |k\rangle|0\rangle = \sum_{k=0}^{15} \frac{1}{4}|k\rangle|0\rangle$$

Step 5 *Apply unitarily the function $f(k) = 13^k \mod 15$ an store t in the second register:*

$$|\psi_2\rangle = \frac{1}{4}\sum_{k=0}^{15}|k\rangle|f(k)\rangle$$

Step 6 *Apply the quantum Fourier transformation on the first qubit to get:*

$$|\psi_3\rangle = \frac{1}{16}\sum_{u=0}^{15}|u\rangle\sum_{k=0}^{15}e^{\frac{2\pi iuk}{16}}|f(k)\rangle$$

Step 7 *Since 13 has period r (unknown in general, in this case 4), the function f is periodic.*

In this case the period happens to divide 16, so that one can write $k = m + jr$, where $0 < m < r$ and $0 < j < 16/r$, and $f(k)$ can be written as $f(m)$.

Then the new state can be expressed as

$$\begin{aligned}|\psi_3\rangle &= \frac{1}{16}\sum_{u=0}^{15}|u\rangle\sum_{m=0}^{r-1}|f(k)\rangle e^{\frac{2\pi ium}{16}}\sum_{j=0}^{16/r-1}e^{\frac{2\pi iurj}{16}}\\ &= \sum_{u=0}^{15}|u\rangle b_u\sum_{m=0}^{r-1}|f(k)\rangle e^{\frac{2\pi ium}{16}}\end{aligned}$$

where $b_u = 1/r$ if $16|ur$ and 0 otherwise.

Step 8 *Measure the state of the first register. Let P_u denote the projection onto the state u, then the probability of observing u is*

$$\langle\psi_3|P_u|\psi_3\rangle = r|b_u|^2 = \begin{cases}\frac{1}{r} & \text{if } 16|ur\\ 0 & \text{otherwise}\end{cases}$$

Step 9 *Use u to deduce the value r:*

1. *If no inference can be made or if r is odd, return to Step 2 and repeat.*
2. *If $r = 2s$ is even, and $y^s = -1 \mod (15)$, return to Step 2 and repeat.*
3. *If $gcd(15, j^s \pm 1) > 1$, quit;*
4. *Otherwise, return to Step 2 and repeat.*

In this case $r = 4$ *and* $4|16$ *the only* u*'s with a positive probability of being observed are multiples of* 4 *(* $u = 0, 4, 8, 12$*), each one occurring at Step 1 with probability* 0.25*. Since* $ur = 16k$*, for some integer* k*, one can infer nothing about* r *if* $u = 0$*. The remaining cases give equations for* r *of the form* $r = 4k$*,* $r = 2k$ *and* $3r = 4k$*. In the cases that* r *and* k *are relatively prime, the algorithm gives the value of* $r = 4$ *for the period. One knows that value to be correct and that 15 has to divide the product of* $13^2 - 1$ *and* $13^2 + 1$*, enabling us to find both of its factors.*

If one obtains $r = 2$*, giving values of 12 and 14 for* $y^2 \pm l$*. In this case as the numbers are so small, one get a factor of* 15 *in this case, since one of three consecutive numbers is divisible by 3. However,* $r = 2$ *is not the period, and in general we are not guaranteed to obtain a factor of* N *if one gets the wrong value of* r*. Hence, one can say that this will be successful in factoring* $N = 15$ *at the end of a single run of the procedure with probability of* $1/2$*.*

Part IV

Applications of Cryptography

CHAPTER 11

In this part of this project it is discussed several application of cryptography. Informally speaking, in this part it is analyzed how one can use the concepts explained in the previous part for example to sign a document electronically, how one can convince another of the possession of a secret without revealing the secret, identification processes among others.

CHAPTER 12

Digital Signtures

The chapter name says it all. Digital signatures are computer/digital based counterpart of legal handwritten signature in a piece of paper of authentication of electronic records. The purpose of this chapter is to explain how a digital signature can be created and how it achieves the same functionality as that of a handwritten signature. The idea is to use asymmetric cryptography explained in Part III.

The idea of a digital signature is due to Diffie and Hellman that in 1976 described the basic properties that such scheme must have and conjectured that such schemes could be designed. Only when Rivest, Shamir, and Adleman proposed RSA as public key cryptographic system, was possible to design a protocol of digital signature, proving that the concept is sound although it is not secure. In 1988, Goldwasser, Micali, and Rivest rigorously define the security requirements of digital signature schemes by describing a hierarchy of attack models for signature schemes, and present the now so called GMR signature scheme, a digital signature scheme that is proved to be secure even when forged chosen message attack are considered. Nowadays there are widely marketed software package to offer digital signature such as Lotus Notes (the first to appear), released in 1989, which used the RSA algorithm.

The reader may wonder why digital signatures are important and what kind of problems (legal/practical) that one might find when trying to implement such systems. In this chapter it is not discussed the legal issues that one might find and point out briefly the importance of of

them. To illustrate the thematic, first it is presented an example for motivating the discussion of digital signatures.

Example 12.0.6. *Consider the following scenario: a company, an employee (E), the boss (B) and an email from B informing E of an raise in his salary in the following terms:*

Email 12.0.7 (Decision of raising your salary request)

> **From:** *boss@company.org*
>
> **To:** *employee@company.org*
>
> *Dear Collaborator E:*
>
> *I am delight to inform you that your salary raise has been approved.*
>
> *Best regards,*
>
> *Boss B*

How can E be sure that this email is for real and how can he convince the human resources department of that raise? The reader may say "Well there is Boss B name at the end of the message" or "also it came from Boss B company email", so this email is for real. But what is the validity of this message? How does the reader know that this message was not created by someone else in B's place that had simply had access to the email sender server?[1]

Even in the case there was a digital copy of B's signature what was the validity of the message?

This kind of issues are the base for the study of digital signatures, i.e., process of signing digital documents without ink based signatures that have "legal" value or more simply compromised the signer with the document.

Now it is clearer the importance of digital signatures. In the real world, more than ever, the computer is a tool present in daily life and it is become more and more common one have

[1] Notice that a similar situation with post mail may happen. The reader can send to a physical letter to his/her father in the name of the Republic President without its consent saying that him will the awarded a medal. But in this case, without President's ink signature, the validity of this message is null. Nowadays the most common email servers do not allow the sent of an email without being an authorized user.

the need to ensure himself about the origin of a message and also when their use is needed for similar reasons that would use a signature to sign a paper document. The use of ink signatures or authenticity stamps, digital signatures are no longer a practical to use traditional signature. Thus, since it is become prominent the digital documents they are the answer to provide evidences of provenance, identity, and status of an digital document. They also provide insurance that the content of the message is consented and approved by the one how signs it. So they become useful tools in many applications such as electronic commerce and emails for proving the authenticity of messages.

In order to have a digital signature with similar properties to the real signature one must have the following properties similarly to what one would expect form a handwritten signature:

Authenticity which means that the digital signature would help to assure that the signer is who he or she claims to be and that can be checked. This is particular important for example in a financial context. If in a case someone goes into a bank with instructions to pay a bill and the bank pays it even when the bank collaborator is not convinced of the provenance of the instruction can be a serious question and a severe mistake.

Integrity which implies that the content has not been changed or tampered since it was digitally signed, preventing the possibility of a changed after the signing was made without turning the document into an unsigned one. In particular no one can reuse the signature to signed new documents without the consent of the sender, which will have to use its ”pen” to sign the new document.

Non-repudiation in the sense that it proves without error that the signer cannot deny that he had signed a valid document. This property of digital signatures is crucial for situations where are conflict between the two parties. For example, one part signs a contract and the recipient needs to prove to a judge that in fact the first one did indeed “certify” and signed that contract.

A very important difference between handwrite signature and digital signature is the fact that in the former case, the signature itself is part of the physical document while in the latter is not attached physically to the message. So, for that reason the digital signature process has to “bind” the signature to the message. Another important difference concerns the verification

process. A conventional signature can be verified by comparison to an other authentic signatures and thus it is not very secure method as it is relatively easy to forge someone else's signature. On the other hand digital signatures can be verified by means of publicly known verification algorithm which everyone can verify making the forgery process much harder. Public verifiability implies that signatures are transferable, i.e. a signature can be shown to a third party that also can then verify itself that it has a legitimate signature. These properties, transferability and public verifiability are essential for the application of digital signatures to certificates and public-key infrastructures. A third major difference between the two signature processes mentioned above is the possibility of copy. In the traditional signatures all of them are distinguishable for each other while in the digital signature, the signatures are all identical and thus one must be careful in order to prevent a signed digital message from being reused. For sake of understanding, consider the example where Alice signs a check to pay Bob's 100 dollars. Of course, Alice wants Bob to be able to do so but it must be the case Bob cannot use the same check or signature more than once. So the message itself should contain information, such as a date, which prevents the reusage.

• •

SECTION 12.1

Definition and examples

In this section it is presented the definition of digital signature scheme and presented some examples of such schemes. From the discussion above, one can realize that these type of schemes must have two basic parts:

Signing process usually an algorithm depending on a private key that binds the signature to a message that is performed by the person who wants to sign a digital document;

Verification process usually an algorithm depending on a public key that allows anyone to verify the consistence of the signature.

A possible definition of digital signature is presented next.

Definition 12.1.1 (*Digital signature scheme*)

A digital signature scheme is a tuple $\mathcal{DS} = (P, A, K, R, r, S, V)$ *such that the following conditions are satisfied:*

- P *is a finite set of possible messages;*
- A *is a finite set of possible signatures;*
- K*, the keyspace, which is a finite set of possible keys;*
- R*, the public part of the key, which is a finite set;*
- $r : K \to R$*, is the revealing function, that gives the public part of the key. It is required that this function run in polynomial time;*
- *For each* $k \in K$*, there is a signing algorithm* $sig_k \in S$ *(running in polynomial time) and a corresponding verification algorithm* $ver_k \in V$ *(also running in polynomial time).*

  Each $sig_k : P \to A$ *and* $ver_k : P \times A \to \{true, false\}$ *are functions such that for every message* $x \in P$ *and for every signature* $y \in A$

$$ver_k(x, y) = \begin{cases} true & \text{if } y = sig_k(x) \\ false & \text{if } y \neq sig_k(x) \end{cases}$$

A pair (x, y) *with* $x \in P$ *and* $y \in A$ *is called a signed message.*

As mentioned above the first digital signature scheme was based on the RSA public key cryptographic system. For sake of understanding it is now presented.

The idea is to use private key to sign and the public key to verify the signature.

In fact, one can sign a message x using the private key of the RSA cryptographic system and the decryption rule. Notice that only the person who knows the private key can create such signature since the decryption rule entirely depends on the private key. On the other hand, anyone can verify the signature using the public key and the encryption function of the RSA cryptographic system.

Digital Signature System 12.1.1 (*based on RSA cryptographic system*)

Consider $n = p \times q$ for primes p and q. The digital signature scheme based on RSA is the tuple $\mathcal{DS}_{\mathcal{RSA}}{}^n = (P, A, K, R, r, S, V)$ where:

1. *$P = A = \mathbb{Z}_n$;*
2. *K consists of pairs $k = (a, b) \in (\mathbb{Z}_n)^2$ such that $a \cdot b \equiv 1 \pmod{\phi(n)}$, i.e., a and b are inverse of each order modulo $\phi(n)$ where $\phi(n)$ is the n's value of Euler's function;*
3. *$R = \mathbb{Z}_n$;*
4. *Given a pair (a, b), $r(a, b) = b$;*
5. *For each $k = (a, b) \in K$ and all $x, y \in \mathbb{Z}_n$,*

$$sig_k(x) = x^a \pmod{n}$$

and

$$ver_b(x, y) = y^b \pmod{n}.$$

The motivation for the presentation of this digital signature scheme is just to work as a proof-of-concept, since, in particular, this scheme does not provide the required non-forgery property. In fact, anyone can easily forge a signature of legitimate owner by simply generating a random y and compute $x = e_k(y)$ and then compute $y = sig_k(x)$. It is left for the interested reader to check that y is a valid signature of x. Notice that one can sign a message without even knowing the private key. Moreover, the other way around possibility seems to be impossible without compromising the RSA cryptographic system security. In fact, to sign a particular message x with this system one really needs to know the private key and thus to have a general way to recover the private key from publicly available information, i.e., of breaking the RSA. This attack brings to light the need of extra requirements in order to have a "secure" digital signature scheme. For example, one can require that messages contain sufficient redundancy that a forged signature of this type does not correspond to a "meaningful" message.

It is of particular interest the possibility of combining digital signatures with public-key encryption.

Consider the situation where A wants to send x a signed messages that only B can read.

Clearly, one way to achieve this goal is to consider a combination of digital signatures and encryption scheme. First, A signs the message x, i.e., computes $y = sig_A(x)$ and then encrypt both x and y with B's public key b, to get $z = e_{b_B}(x, y)$. From the point of view of A, the process is done and then the message can be sent to B. When B receives z reverts the process, i.e, first decrypts z to get $(x, y) = d_{k_B}(z)$, where k is its private key. Then using A's public verification function B can check if $ver_A(x, y) = true$.

Another way that A could use is first encrypted the message x and then signed the result. In this case A's operation results would be $y = sig_A(z)$ where $z = e_B(x)$, and thus, A would send (z, y) to B and it would revert the process in a similar manner. But, in this case, since the signature is send openly, if and opponent O intersects the a valid pair (z, y) he could replace A's signature y by his own signature. Notice that O can sign the ciphertext $z = e_B(x)$ even without knowing the plaintext x. Then, if O replaces the original message to B by (z, y') to B, O's signature would be a valid one and B could certify that that the plaintext x wold have come from O. This justifies that one should first sign before encrypting.

SECTION 12.2

Secure digital signatures

From what was discussed in the previous section the reader may think that digital signatures can be seen as "inverses" of public-key encryption schemes, with the roles of intervener interchanged. In fact, historically signing a message has been made by decrypting it using the private key and verifying a signature by encrypting the signature using the corresponding public-key function. Also, from what it was discussed this suggestion of constructing digital signature schemes is completely useless as the result are insecure not fulfilling the requirements one would expect a digital signature to have. In their foundational paper, Goldwasser, Micali, and Rivest lay out a hierarchy of attack models against digital signatures. The most common

attacks that one consider for digital signature schemes are:

Key-only attack In this case the attacker is only given the public verification key. I tis the most general type of attack one can consider.

Message attacks In this kind of attack the enemy is able to examine some signatures corresponding to either known or chosen messages before he tries to break the scheme.

Known message attack Herein the adversary has at hand a list of previously signed messages. It is a particular case of the previous item where the adversary cannot ask for signatures of his choice.

Chosen message attack It is a sub-case of second presented item. In this case the adversary can ask for signatures of documents of his choice. These messages are chosen by the enemy and are fixed and sent to signing at once, i.e., in a nonadaptive way. Sometimes this attack is called "generic" since it does not depend on the A's public key; the same attack is used against everyone.

Directed Chosen Message Attack This attack is similar to the generic chosen-message attack, except the fact that the list of messages to be signed may be created after the opponent seeing A's public key but before any signatures are seen. This attack is called "directed" because it is made against a particular user A.

Adaptive Chosen Message Attack This kind of attack is even more general. In this case the opponent is also allowed to use A as an "oracle" not only by requesting A signatures of messages of his choice, that might depend on A's public key but also request signatures of messages which depend additionally on previously obtained signatures. It is adaptive because after each signature obtained the adversary can adaptively choose the next text to be signed and mount the attack.

In Goldwasser's, Micali's, and Rivest's own words the above list of attacks is ordered increasing by severity in the sense that with the adaptive chosen-message attack is the most general and natural attack that one can perform. Notice that the adaptive chosen-message attack is natural and is usually seen in the case of a notary public who must sign more-or-less arbitrary documents on demand. In general, the user of a signature scheme would like to feel that

he may sign arbitrary documents prepared by others without the fear of compromising his security.

Before continuing one must understand what kind of security one expects. This can be done by analysis the mean of the statement: "break the digital scheme". There are several goals that an opponent can consider as "brake" such as:

Total break Basically means that the opponent compute A's secret trap-door information, i.e, A's private key and thus can compute the function sig_A, which allows him to create valid signatures of any message.

Universal forgery In this case the opponent only wants to find an efficient signing algorithm functionally equivalent to A's signing algorithm, based on possibly different but equivalent trap-door information.

Selective forgery In this case the opponent can, with non-negligible probability, create valid signatures of messages chosen by someone else, i.e., given a message x that has not previously been signed by A he can determine with non-negligible probability y a valid signature of x.

Existential forgery The opponent is able to create a valid signature for at least one message, i.e., he can create a pair (x, y) where x is a particular message, that has not been signed yet, for which $ver_k(x, y) = true$. Notice that the opponent has no control over the message whose signature he obtains, so it may be random or nonsensical and consequently this forgery may only be a minor nuisance to A.

It is clear from the context that the term "forge a signature" means to produce a new signature. However, in a chosen-message attack it not a forged signature to get from the real signer a signature for the target message. Again, the listed kinds of "breaks" listed above are in decreasing order of severity, since the least an opponent might hope as success is to have an existential forgery. Clearly, from the design point of view, a scheme of digital signature it is desirable to design a proven secure scheme that is not even existentially forgeable.

A signature scheme cannot be unconditionally secure, i.e., it is impossible to prevent that someone can sign a message disorderly, since and opponent can always test all possible signatures $y \in A$ for a given message x, using the public algorithm ver_k, and using a brute force

attack until he finds a valid signature. Thus, as was the case with public-key cryptographic systems, the best one can expect is to find signature schemes that are at most computationally or provably secure.

• •

SECTION 12.3

The ElGamal digital signature scheme

This scheme is not really used in practice that it is the predecessor of the NSA variant called Digital Signature Algorithm which is much more widely used and is classified as a standard by the NIST. This scheme was proposed by Taher ElGamal in the middle of 80's and is based on the difficulty of discrete logarithm. The reader should not confuse the digital signature with the cryptographic system with the same name.

The ElGamal digital signature scheme is non-deterministic since there are many valid signatures for any given message, and the verification algorithm must be able to accept any of these valid signatures as authentic.

The formalization of this cryptographic system is the following:

Digital Signature System 12.3.1 (*ElGamal digital signature scheme*)

Consider p a prime such that the discrete logarithm problem is intractable in $\mathbb{Z}_p^*$ and let $\alpha \in \mathbb{Z}_p^*$ be a generator of $\mathbb{Z}_p^*$.
The ElGamal digital signature scheme is the tuple $\mathcal{DS}_{\mathcal{EG}}{}^n = (P, A, K, R, r, S, V)$ where:

1. $P = \mathbb{Z}_p^*$;
2. $A = \mathbb{Z}_p^* \times \mathbb{Z}_{p-1}$;
3. *K consists of pairs $k = (a, \beta) \in \mathbb{Z} \times \mathbb{Z}_p^*$ such that*

$$\beta \equiv \alpha^a \pmod{p};$$

4. $R = \mathbb{Z}_p^*$;
5. *Given a pair (a, β), $r(a, \beta) = \beta$;*

6. *For each* $k = (a, \beta) \in K$ *and all* $x \in \mathbb{Z}_n$,

$$sig_k(x, c) = (\gamma, \delta)$$

where $c \in \mathbb{Z}_{p-1}^*$ *is chosen uniformly at random and*

$$\gamma \equiv \alpha^c \pmod{p}$$

and

$$\delta \equiv (x - a\gamma)c^{-1} \pmod{p-1}.$$

7. *For each* $\beta, x, \gamma \in \mathbb{Z}_p^*$ *and* $\delta \in \mathbb{Z}_{p-1}$ *define*

$$ver_b(x, (\delta, \gamma)) = true \text{ if and only if } \beta^\gamma \gamma^\delta \equiv \alpha^x \pmod{p}.$$

The first thing one has to do is to see that the verification process will succeed. In fact, since $c\delta + a\gamma \equiv x \pmod{p-1}$ one has

$$\begin{aligned} \beta^\gamma \gamma^\delta &\equiv (\alpha^a)^\gamma (\alpha^c)^\delta \pmod{p} \\ &\equiv \alpha^{a\gamma} \alpha^{c\delta} \pmod{p} \\ &\equiv \alpha^{a\gamma + c\delta} \pmod{p} \\ &\equiv \alpha^x \pmod{p} \end{aligned}$$

Notice also that all the operations involved in the definition of the scheme are performed in polynomial time.

In what follows it is presented a general way of producing a digital signature scheme based on a one-way function. To sign a message of k bits, one signs each bit individually. If the z^{th} bit of the message equals $j \in \{0, 1\}$ then the i^{th} element of the signature is the value $y_{i,j}$, which is a preimage of the public key value $z_{i,j}$, while the verification procedure consists simply of checking that each element in the signature is a preimage of the public key element $z_{i,j}$.

Digital Signature System 12.3.2 (*Lamport digital signature scheme*)

Consider a positive integer k and assume that $f : Y \to Z$ is a one-way function. The Lamport digital signature scheme the tuple $\mathcal{DS} = (P, A, K, R, r, S, V)$ withe following conditions:

- $P = \{0,1\}^k$;
- $A = \{0,1\}^k$;
- K, *consists of* $2k$ *pairs* $(y_{i,j}, z_{i,j})$, *where* $i = 1, \cdots k$ *and* $j = 0,1$ *and* $y_{i,j} \in Y$ *and* $z_{i,j} = f(y_{i,j})$;
- $R = \{0,1\}^{2k}$, *the public part of the key;*
- $r(y_{i,j}, z_{i,j}) = z_{i,j}$;
- *For each* $k = \{(y_{i,j}, z_{i,j}) \in K : i = 1, \cdots k, j = 0,1\}$, *and* $x = (x_1, \cdots, x_k)$

$$sig_k(x_1, \cdots, x_k) = (y_{1,x_1}, \cdots, y_{k,x_k})$$

- *The verification function is defined as follows. Let* $(a_1, \cdots, x_k)$ *be a signed message. Then*

$$ver_k((x_1, \cdots, x_k), (a_1, \cdots, x_k)) = true \Leftrightarrow f(a_i) = z_{i,x_i}$$

Example 12.3.3. *Consider* $f(x) = 3^x \pmod{7879}$ *and* $k = 3$. *A chooses six ranodm numbers y*

i/j	0	1
1	5831	735
2	803	2467
3	4285	6449

the correspondents z's are:

i/j	0	1
1	2009	3810
2	4672	4721
3	268	5731

The sign the message $x = (1, 1, 0)$ *one gets* $(y_{1,1}, y_{2,1}, y_{3,0}) = (735, 2467, 4285)$.

To verify this signature, it suffices to compute the following:

$$3^{735} \equiv 3810 \mod 7879$$
$$3^{2467} \equiv 4721 \mod 7879$$
$$3^{4285} \equiv 268 \mod 7879$$

Notice that an opponent cannot forge a signature because he is unable to invert the one-way function f to obtain the secret y's. However, the signature scheme can be used to sign only one message securely. Given signatures on two different messages, it is an easy matter for the opponent to construct signatures for another message different from the first two (unless the first two messages differ in exactly one bit). So he can sign $2^l - 2$ new messages where $l = d(x, x')$.

• •

SECTION 12.4

The notion of hash function

It is clear from the context of signatures that for practical reasons one cannot use the entire document to sign a message as at that process would require too much information. Instead of using all the information of the message one consider a special compression of the message that works like a fingerprint of the data in such a way that, that similar to a real fingerprint, identifies uniquely identifies the message. Notice that since the idea is to compress the message it is impossible to have a uniqueness identification. So instead of uniqueness one requires to be hard to find two messages that are compressed to the same "fingerprint". The concretion of this idea is achieved by the notion of hash function.

Thus, a hash function is an algorithm mapping large data sets of variable length, called keys,

to smaller data sets of a fixed length called hash values, hash codes, hash sums, checksums or simply hashes. In particular, a cryptographic hash function can provide assurance of data integrity as it provides the data the possibility that if the original data are altered, then the "fingerprint" will no longer be valid for the original data. The fingerprint is also often referred to as a message digest and typically is a fairly short string (usually binary with 160 bits). Usually given a original message x and a changed one, for example, x' then one expects that y the digest of x is not also a digest for x'. If this is indeed the case, then the fact that x has been changed can be detected simply by computing the message digest y' for x' and verifying that $y' \neq y$.

A good cryptographic hash function is required to satisfy the following properties:

1. Easy to compute the hash value for any given message;
2. Infeasible to generate a message that has a given hash value;
3. Infeasible to modify a message without changing the hash value;
4. Infeasible to find two different messages with the same hash

With this properties one expect that a cryptographic hash function h should withstand all known types of cryptographic attacks. In particular it should be:

Preimage resistance: Related with the second item above and also with the concept of one-way function, since this property makes very unlikely that given y to find any message x such that $y = h(x)$.

Second-preimage resistance: Related with the third item above states that, given an input x it should be difficult to find a different input x' such that $h(x) = h(x')$. This property is sometimes called as weak collision resistance.

Collision resistance: Related with the second item above. It states that it is difficult to find x and x' such that $x \neq x'$ and $h(x) = h(x')$. This kind of pairs are called a hash collision. This property is also called as strong collision resistance. It imposes that the hash value has at least twice the length of a hash resistance for preimage-resistance ones since otherwise collisions may be found by a birthday attack that will be explained in due course of this section.

These properties imply that a malicious adversary cannot replace or modify the input data without changing its digest and thus, if two messages have the same digest, one can be very confident that they are identical.

Definition 12.4.1

TBW

Birthday attack...

CHAPTER 13

Zero knowledge

The concept of zero knowledge conveys the idea of two parties P and V communication to each other where one has a secret and tries to convince the other that he really knows the secret. One way that P has to prove to V that we knows the secret is to reveal the secret, but in that case the secret would no longer be a secret only known by A. So the idea is to use interaction in order of A to convince V of the "knowledge" of the secret revealing "zero" information about that secret.

In fact, the zero knowledge requirement is quite stronger than the above explanation. As it will be clear from the definition presented below, it follows that a protocol will be called "zero knowledge" if it conveys no information whatsoever about whatever rather than one bit that is related with the "yes" or "no" quality of the instance that will be explained in due course.

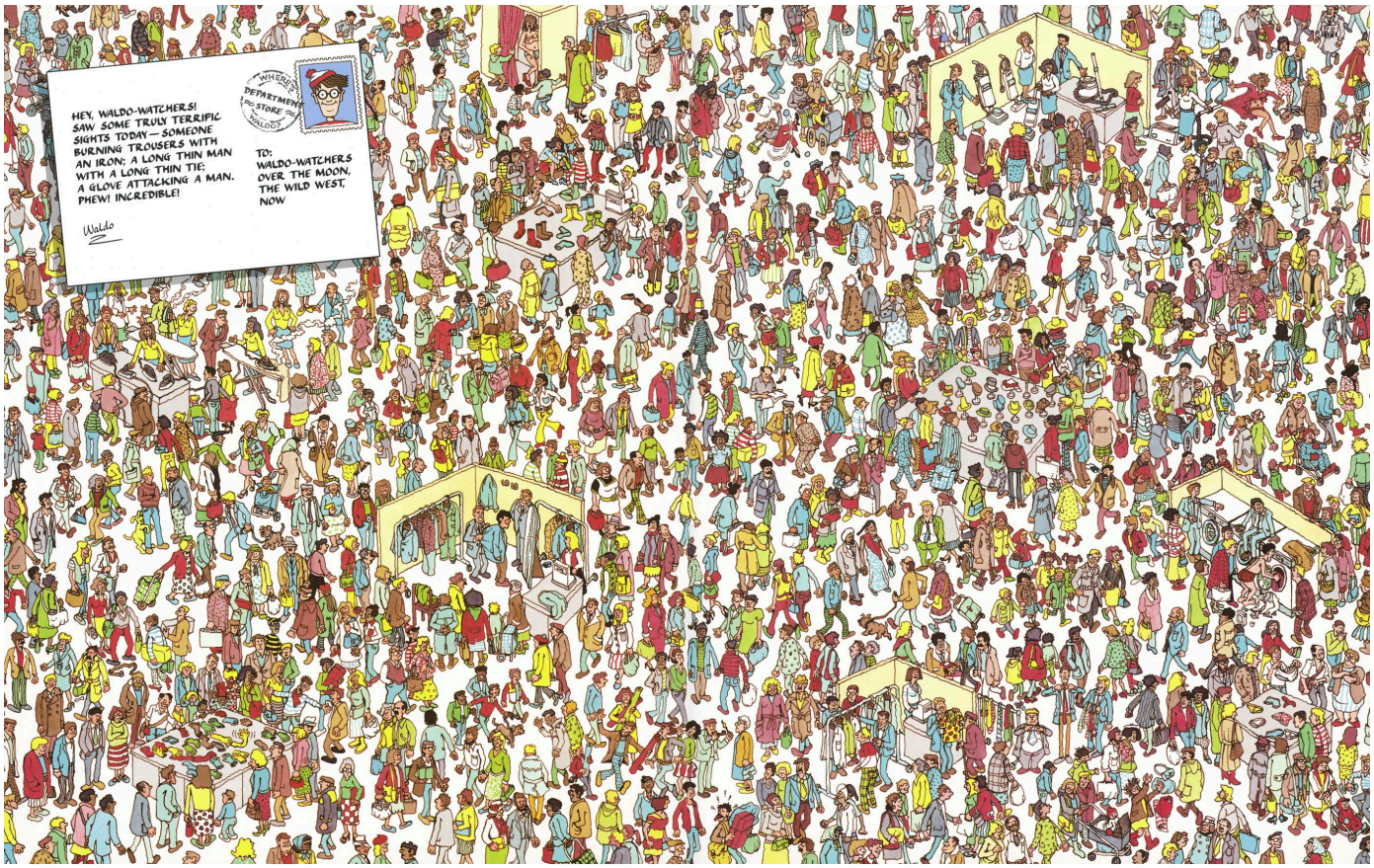
An illustrative example of a Zero Knowledge protocol can presented as follows

Example 13.0.1. *Imagine that the reader has one of "Where is Waldo?" comic books and cannot find the Waldo himself in many pages of the book. We, the writers, on the other hand are experts on this finding the Waldo in those images and the reader requires our help.*

Of course, if we point out the Waldo for the reader in each pictures of the book, then the reader would lose completely the interest of the book as he could not longer entertain himself looking for Waldo in all those pictures. Instead, and in order to keep reader's interest on the book, we could try to convince the reader that we really are experts on Where's Waldo book by interacting with the reader and the book to

show only Waldo in the picture without revealing his location. To do so we can do the following.

Place in a large table (in fact, 3 times larger than the book) a thick piece of black paper with a very tinny whole in the center where only the Waldo figure fits in. We kindly ask the reader to turn around and grab the book, find Waldo and place it in the table in such a way that the Waldo appears in the whole placed in the paper. Since the entire book in covered we can reveal the reader that in fact Waldo is in the picture and, moreover, the reader, did not get a clue where, in fact, Waldo is placed in the picture. Again, we ask the reader to turn around again and take the book under the paper and give it back to the reader so he can look for Waldo in the same picture we had been shown the Waldo.



If we repeat this procedure a large number of times, at the end, if we manage to show, in each page chosen by the reader, the Waldo, then the reader will be convinced that, in fact, we are specialists on finding Waldo without loosing the enthusiasm for search of the character. Another important aspect of this interactive way of proving is that the reader is convinced that Waldo is really in the picture.

Now it is presented the definition of zero knowledge proof system that will be adopted in these notes.

Definition 13.0.2 (*Zero knowledge proof system*)

A zero knowledge system between two parties P and V is a protocol involving the P, called the Prover, and V, called the Verifier, such that P holds a secret s and:

Soundness: *If P holds the secret s than he can convince V (up to a negligible probability) that he, in fact, holds the secret. This property establishes that in the possession of the secret there is a way of P to convince V that he knows the secret.*

Completeness: *If P does not hold the secret then he cannot convince V (with a non negligible probability) that he has the secret. This property states that if P does not know the secret no matter what he does V will reject, i.e., will not be convinced of the possession of the secret by P with significant probability.*

Zero knowledgeness: *After the end of the protocol, V gets no information whatsoever about the secret, i.e., the amount of information that V gained during the protocol is useless to determine the secret s. In particular, the verifier cannot, at the end of the protocol, convince someone that he knows the secret by trying to impersonate the prover.*

Non-transferability: *After the end of the protocol, V cannot prove to a third party that he was interacting with P. This property aims to prevent a malicious behavior of V trying to convince another party of the validity of the statement proven by P.*

In the next protocols it is presented zero knowledge proof systems for some problems that were considered in the previous chapter of these notes. In particular, it is designed a zero knowledge system for the $\mathbb{Quadratic\ Residues}$, for the $\mathbb{Isomorphic}$ graph problem, the $\mathbb{Hamiltonian}$ cycle on graphs (which is an **NP** complete problem) and for the discrete logarithm. It is also outlined a way of designing a zero knowledge proof system for any problem in **NP** based on the reduction from that language to the $\mathbb{Hamiltonian}$ cycle one.

In the first example it is considered the $\mathbb{Quadratic\ Residues}$ problem. For sake of completeness, and since it is used, in particular, for non prime moduli it is mentioned again its

formulation.

Problem 13.0.2 (Quadratic Residues)

Instance: *$x \in \mathbb{Z}_n^*$ where $n = p \times q$ for some unknown primes p and q;*

Question: *Is x a quadratic residue modulo n, i.e., is there a $u \in \mathbb{Z}_n^*$ such that $u^2 \equiv u \pmod{n}$?*

The zero knowledge proof system for this problem is described below. It is assumed that (P, V) interact with each other and P holds the secret u, a square root of x modulo n.

Protocol 13.0.3 (*Zero-knowledge protocol for* Quadratic Residues)

Input: *An integer n with unknown factorization $n = pq$, where p and q are prime and x a quadratic residue modulo n.*

Secret: *u a square root of x modulo n, i.e., $x = u^2 \pmod{n}$ (unknown to V).*

Number of rounds: *The following steps are repeated $m = \log n$ times.*

Step 1: *P chooses a random $v \in \mathbb{Z}_n^*$, and computes $y = v^2 \mod n$. P sends y to V.*

Step 2: *V chooses a random $i \in \{0, 1\}$ and sends it to P as a challenge;*

Step 3: *P computes $z = u^i v \mod n$;*

Step 4: *V checks if $z^2 = x^i y \mod n$;*

If all rounds are completed with success V accepts that P really knows u, the square root of x.

Usually, the **Step 1** is also known as the commitment step, the **Step 1** is also designated as the challenge step, the **Step 3** is also called as the revealing or decommitment step. The **Step 4** is called the verification process.

If x is, in fact, a quadratic residue modulo n and P knows the square root of x modulo n then he can always answer correctly to V's challenge. On the other hand if x is a non-quadratic residue

modulo n then any P' can only convince V with probability $1/2$ on each round that he knows the square root, sine the best he can do is to chose on of the two challenges to answer correctly. These two arguments show that the protocol is sound and complete.

Naively speaking, this protocol is also zero knowledge since the verifier gains no knowledge whatsoever about the secret that was used.

In fact the protocol is zero-knowledge since we can build a simulator that does the job by, at the beginning, chooses to send either v^2 or $x^{-1}v^2$. Notice that x has an inverse modulo n and that inverse can be easily found using the $\mathbb{E}\mathrm{xtented}\mathbb{E}\mathrm{uclides}$ algorithm presented in Page 180.

Theorem 13.0.1

The Protocol 13.0.3 has the zero knowledge property of Definition 13.0.2.

Proof: In order to prove the zero knowledge property, one needs to show that the verifier did not gain any knowledge about the secret. One way to do it is to show that the verifier can simulate the knowledge, in fact the entire, conversation with the prover without a single interaction with it. I.e., if one denotes by μ^j the quantity of knowledge that the verifier gained during round j of the protocol, then he could simulate the interaction in such way that $\kappa_S^j \approx \kappa_{P\leftrightarrow}^j$ which means that the two distributions, the simulated and the real one, are similar.

Notice that the verifier is given a procedure $B(\mu^j, z^{j+1})$ that is a probabilistic time algorithm that outputs the challenge of the protocol, and also another probabilistic polynomial time algorithm, called knowledge updating function, $U(\kappa^j, z^{j+1}, v) = \kappa^{j+1}$.

The simulator constructing this function is described by:

Simulator 13.0.4

Step 1 *Chose uniformly at random* $r \in \{0, 1\}$ *;*

Step 2 *If* $r = 0$ *consider* $y = v^2 \pmod n$ *and if* $r = 1$ *consider* $y = v^2x^{-1} \pmod n$*, where* v *is generated uniformly at random.*

Step 3 *If* $r \neq B(\kappa^j, z^{j+1})$*, then go back to* **Step 1***;*

Step 4 *Then* $\kappa^{j+1} = U(\kappa^j, z^{j+1}, v)$*.*

Notice that after m trials, the probability of not reaching **Step 4** is $(1/2)^m$ and hence, since B follows a Bernoulli distribution with probability $1/2$ then clearly one has $\kappa_S^j \approx \kappa_{P\leftrightarrow}^j$, as claimed. This means that if the verifier is capable of generating the proof by himself without communication with the prover than of course, all that he could gain from the interaction could also be produced by himself without the interaction. ■

Now it is discussed the impossibility of transfer the proof. It is shown that is one uses devices that have non-erasable memory then the verifier can proof to someone else that he interacted with the prover.

Theorem 13.0.2

If a device with non-erasable memory is given by a third party to V then he can prove that he had interacted with the prover.

Proof: The proof consists on design of a device D that proves the the interaction. The device is such that:

- D has a pseudorandom generator with a seed unknown to the verifier.
- D receives a value y and outputs a challenge b and records y and b on its permanent memory.
- D record the output of the challenge and the acceptance of the verifier.

Since the third party can access to the entire memory of the device then he can conclude if there was or not interaction between the parties since if the conversation were simulated there are challenges that were not properly answer and those would require the device to go back and erase its permanent memory, which is impossible. Thus, the third party got to know if there was or not communication between the prover and the verifier. ■

Another example concerning Zero Knowledge is related with $\mathbb{NP}$ complete problems. The reader may ask why this is important and if he thinks for a bit he easily realize that one zero knowledge protocol for such problem can be used to design a Zero Knowledge protocol for all $\mathbb{NP}$ language. The problem that it is consider here is the Hamiltonian cycle.

Problem 13.0.5 (Hamiltonian *graphs*)

Instance: *A graph H;*

Question: *Is there an Hamiltonian cycle on H, i.e., a sequence that passes through all the vertices once with the same start and end point?*

Before the presentation of the zero knowledge for $\mathbb{NP}$ complete problems one needs another ingredient that plays an important role in this protocol called Commitment schemes. The idea is to use them to enable a party to commit itself to a value while keeping it secret and in a later stage the commitment is "opened" and it is guaranteed that the "opening" can yield only a single value determined in the committing phase.

Heuristically, a commitment scheme is an efficient two-phase two-party protocol through which one party, called the sender, can commit itself to a value so the following two conflicting requirements are satisfied.

Secrecy: At the end of the first phase, the other party, called the receiver, does not gain any knowledge of the sender's value, even if the receiver tries to cheat.

Unambiguity: Given the transcript of the interaction in the first phase, there exists at most one value which the receiver may later accept as a legal "opening" of the commitment, even if the sender tries to cheat.

The first phase is called the commit phase and the second is called the reveal phase. So, to define a commitment scheme, in first phase it is required that a sender commit itself with a value and yield no knowledge to the receiver, and in the reveal phase the sender sends to the receiver the original value and the sequence of random coin tosses that it has used during the commit phase. The receiver will accept the value if and only if the supplied information matches its transcript of the interaction in the commit phase.

Before giving the precise definition of bit commitment scheme, we give some preliminaries definitions:

1. A receiver's view of an interaction with the sender, denoted (r, m), consists of the random coins used by the receiver r and the sequence of messages received from the sender m.

2. Let $\sigma \in \{0,1\}$. We say that a receiver's view $(r, \bar{m})$ is a possible σ-commitment if there exists a string s such that $\bar{m}$ describes the messages received by R when R uses local coins r and interacts with machine S which uses local coins s and has input $(\sigma, 1^n)$. This means,that $\bar{m} = view^{S(\sigma,1^n,s)}_{R(1^n,r)}$.

3. We say that the receiver's view $(r, \bar{m})$ is ambiguous if it is both a possible 0-commitment and a possible 1-commitment.

Definition 13.0.3

A bit commitment scheme is a pair of probabilistic polynomial-time interactive machines, denoted (S, R), the sender and receiver, such that:

- *Input: The common input is an integer n presented in unary. The private input to the sender is a bit v.*
- *Secrecy: The receiver cannot distinguish a commitment to 0 from a commitment to 1, this is, for every probabilistic polynomial-time machine R^* interacting with S, the random variables describing the output of R^* in the two cases, namely $\langle S(0), R^* \rangle (1^n)$ and $\langle S(1), R^* \rangle (1^n)$ are polynomially indistinguishable.*
- *Unambiguity: The unambiguity requirement asserts that, for all but a negligible fraction of the coin tosses of the receiver, there exists no sequence of messages which together with these coin tosses forms an ambiguous receiver view.*

This type of schemes are currently used in cryptography. They use a pairs of keys, in which one is public, and a commitment with a encrypted value.

Now it is presented a simple construction of a bit commitment scheme.

Protocol 13.0.6

Let $f : \{0,1\}^* \to \{0,1\}^*$ be a function, and $b : \{0,1\}^* \to \{0,1\}$.

commit phase: *To commit to value $v \in \{0,1\}$, using security parameter n, the sender uniformly selects $s \in \{0,1\}^n$ and sends the pair $(f(s); b(s) \oplus v)$ to the receiver.*

reveal phase: *The sender reveals the string s used in the commit phase. The receiver accepts the value v if $f(s) = a$ and $b(s) \oplus v = \sigma$ where (a, σ) is the receiver's view of the commit phase.*

Notice that the bit commitment scheme uses one-way functions. This schemes are important since they are the key in the zero knowledge of $\mathbb{NP}$ complete problems. Its usefulness is due to the fact that it allows the prover to hide private information and thus, for the non transferability property means that the verifier cannot convince anyone that the "commitment" was made by the prover as it could do the commitment himself.

Protocol 13.0.7 (*Zero-knowledge protocol for* $\mathbb{H}\mathtt{amiltonian}$)

Input: *A graph H with set of vertices $\{1, \cdots, n\}$.*

Secret: *An Hamiltonian path c in H.*

Number of rounds: *The following steps are repeated n^2 times.*

Step 1: *P chooses a random permutation π and sets $H' = \pi(H)$. For each (i, j) pair of verices, P runs the commitment phase of the bit commitment scheme to commit to "(i, j) is an edge/is not an edge in H'" and sends them to V.*

Step 2: *V chooses randomly one bit in $\{0, 1\}$ and sends it to P challenging him to show the isomorphism between the graph he got and H in the first case, and to show the Hamiltonian path in H' on the latter one.*

Step 3: *If P receives 0 he reveals the permutation π and all the keys to the commitment. If he receives 1, it reveals the keys of the commitment containing the Hamiltonian path in H'.*

Step 4: *V checks the validity of the things sent by P.*

If all rounds are completed with success V accepts that P really knows u, the square root of x.

Notice that if c' is an Hamiltonian path in H', since $c' = \pi(c)$.

The reader is now invited to describe a zero knowledge proof for any $\mathbb{NP}$ language. A hint that helps on the resolution of this task is to notice that, since $\mathtt{Hamiltonian}$ is $\mathbb{NP}$ complete then this means that there exists a polynomial time reduction from any language to $\mathtt{Hamiltonian}$, i.e., a polynomial time computable function that not only transforms instances of the given problem into instances of $\mathtt{Hamiltonian}$ but also transform non instances into non instances of $\mathtt{Hamiltonian}$. Hence, applying the reduction one is able to have a sublanguage of $\mathtt{Hamiltonian}$ to which can apply the above protocol. The details of the design and the proofs are left as exercise.

. . . . . .

SECTION 13.1

Undeniable signatures

In the previous Chapter it was discussed digital signatures schemes and it was required that the signature could be universally verified using the signer's public key. Although this property is very convenient in most applications, like e-commerce and digital contracts, one may find some disadvantages and undesirable consequences in the case of private or commercially sensitive information. In fact, specially for privacy reasons of the buyer, it is expectable that one can sign the payment authorization but anyone other than the seller and the bank should not be convinced that in fact the signer of the payment really allowed that payment. Another important example, is in the case where compromising authenticated documents are publicly released.

So, in order to prevent leaked information concerning signers privacy, David Chaum and Hans van Antwerpen in 1989 came up with the concept of a new kind of signatures, which they called undeniable signatures, that achieves the required property expressed above. In fact, in order for the recipient to convince a third party that the signer really signed the document, needs the cooperation of the signer that interactively provides information, to the third party, to verify the validity of a signature. This property enables the signer to have control on his signatures spread by verifying signatures only to concerned or authorized person and every time someone wants to verify the signatures must interest with him.

On the other hand, to prevent from a malicious signer who would claim that he did not sign a

contract, this kind of signatures provide a method that allows the signer to prove interactively that a signature is invalid. In case of dispute, legal means this disavowal procedure allows the signer to show that a valid signature is a forgery by proving interactively that the verification procedure on that message fail. On the other hand, if the message is not forged, the disavowal process would end up to verify the message and thus the the signer cannot repudiate his signatures.

Hence, undeniable signatures have the following main properties:

1. the verification process needs the signer cooperation;
2. a signer can prove that a signature was not valid;
3. a signer can not deny a valid signature;

The Chaum and van Antwerpen scheme is based on the difficulty of the discrete logarithm problem over $\mathbb{Z}_p^*$ and uses interaction also called challenge-and-response protocol as a verification process of signatures.

Digital Signature System 13.1.1 (*Chaum - van Antwerpe*)

Consider $p = 2q + 1$ where p and q are primes such that the discrete logarithm problem in $\mathbb{Z}_p^*$ is intractable. Consider also $\alpha \in \mathbb{Z}_p^*$ of order q. Let $1 \leq a \leq q - 1$ and take $\beta \equiv \alpha^a \pmod{p}$.
Let also G denote the multiplicative subgroup of $\mathbb{Z}_p^*$ of order q consisting of the quadratic residue modulo p).
The Chaum - van Antwerpen signature scheme is the tuple (P, A, K, R, r, S, V) where:

1. *$P = A = G$;*
2. *The keys are picked in the set:*

$$K = \{(p, \alpha, a, \beta) : \beta \equiv \alpha^a \pmod{p}\}$$

3. *$R = \mathbb{Z}_p^3$;*
4. *Given $(p, \alpha, a, \beta) \in K$), $r(p, \alpha, a, \beta) = (p, \alpha, \beta)$;*

5. *For each $k = (p, \alpha, a, \beta) \in K$ and all $x \in G$,*

$$y = sig_k(x) = x^a \pmod{p}$$

6. *For each $b = (p, \alpha, \beta) \in R$ and all $x, y \in G$, the verification procedure is done by the execution of the following protocol:*

Protocol 13.1.2 (Verification Protocol)

Input *$x, y \in G$ and k the key;*

Output *Either 1 or 0 is y is a valid signature of x or not.*

Step 1 *The verifier chooses uniformly at random $e_1, e_2 \in \mathbb{Z}_q^*$;*

Step 2 *The verifier computes $c \equiv y^{e_1}\beta^{e_2} \pmod{p}$ and sends it to signer.*

Step 3 *The signer computes $d \equiv c^{a^{-1} \pmod{q}} \pmod{p}$ and sends it to the verifier;*

Step 4 *The verifier accepts y as a valid signature if and only if*

$$d \equiv x^{e_1}\alpha^{e_2} \pmod{p}.$$

First it is necessary to explain why the verifier accepts a valid signature.

Since $\beta \equiv \alpha^a \pmod{p}$ then $\beta^{a^{-1} \bmod q} \equiv \alpha \pmod{p}$.

A similar reason applies to $y \equiv x^a \pmod{p}$ to prove $y^{a^{-1} \pmod{q}} \equiv x \pmod{p}$.

Hence:

$$\begin{aligned} d &\equiv c^{a^{-1} \bmod q} \pmod{p} \\ &\equiv y^{e_1 a^{-1} \bmod q}\beta^{e_2 a^{-1} \bmod q} \pmod{p} \\ &\equiv (y^{a^{-1} \bmod q})^{e_1}(\beta^{a^{-1} \bmod q})^{e_1} \pmod{p} \\ &\equiv x^{e_1}\alpha^{e_1} \pmod{p} \end{aligned}$$

Now it is presented the reason why the other direction is also true by showing that if $y \not\equiv x^a \pmod{p}$ the verifier accepts y as a valid signature for x with probability at most $1/q$ which is negligible in the number of bits of x. The challenge c that the signer is given by the verifier is determined by the choice of e_1 and e_2. Both y and β are elements in G and thus, since G as order q, c is one of the q ordered pairs (e_1, e_2) that the signer cannot guess beforehand. Therefore if $y \not\equiv x^a \pmod{p}$ no matter which response d the signer sends to the verifier only one will correspond exactly the choices of the verifier is $1/q$.

Theorem 13.1.1

If $y \not\equiv x^a \pmod{p}$ then the verifier accepts y as a valid signature of x with probability at most $1/q$.

Now it is discussed the disavowal property. The signer could attempt to disavow a (valid) signature in one of the following ways:

1. refuse to participate in the verification protocol, which is considered a clearly attempt of repudiation, revealing that the signer committed with the signature;
2. perform the verification protocol incorrectly;
3. claim a signature a forgery even though the verification protocol is successful.

the last two items are more difficult to guard against but the next protocol achieves that goal. The idea of this protocol is to run the verification protocol twice and a consistency check at the end.

Protocol 13.1.3 (*Disavowal Protocol*)

Input *$x, y \in G$ and k the key, where y might be a forgery signature of x.*

Output *Either 1 or 0 is y is a valid signature of x or not.*

Step 1 *The verifier chooses uniformly at random $e_1, e_2 \in \mathbb{Z}_q^*$.*

Step 2 *The verifier computes $c \equiv y^{e_1}\beta^{e_2} \pmod{p}$ and sends it to signer.*

Step 3 *The signer computes $d \equiv c^{a^{-1} \pmod{q}} \pmod{p}$ and sends it to the verifier.*

Step 4 *The verifier checks if* $d \not\equiv x^{e_1}\alpha^{e_2} \pmod{p}$.

Step 5 *The verifier chooses uniformly at random new* $f_1, f_2 \in \mathbb{Z}_q^*$;

Step 6 *The verifier computes* $C \equiv y^{f_1}\beta^{f_2} \pmod{p}$ *and sends it to signer.*

Step 7 *The signer computes* $D \equiv C^{a^{-1} \pmod{q}} \pmod{p}$ *and sends it to the verifier;*

Step 8 *The verifier checks if* $D \not\equiv x^{f_1}\alpha^{f_2} \pmod{p}$.

Step 9 *The verifier checks if* $(d\alpha^{-e_2})^{f_1} \equiv (D\alpha^{-f_2})^{e_1} \pmod{p}$.

In order to justify the use of this protocol it must be proved that the signer can convince the verifier that an invalid signature is a forgery and, in the case that of valid signature the signer cannot convince the verifier that the signature is a forgery with significant probability. The next theorem shows that the signer can, in fact, prove that the signatures was forged.

Theorem 13.1.2

If $y \not\equiv x^a \pmod{p}$ *and both the signer and the verifier follow the above protocol, then* $(d\alpha^{-e_2})^{f_1} \equiv (d\alpha^{-f_2})^{e_1} \pmod{p}$.

Proof: Notice that:

$$\begin{aligned}
(d\alpha^{-e_2})^{f_1} &\equiv ((y^{e_1}\beta^{e_2})^{a^{-1}}\alpha^{-e_2 \bmod q})^{f_1} \pmod{p} \\
&\equiv y^{e_1 a^{-1} f_1 \bmod q}\beta^{e_2 a^{-1} f_1 \bmod q}\alpha^{-e_2 f_1 \bmod q} \pmod{p} \\
&\equiv y^{e_1 a^{-1} f_1 \bmod q}\alpha^{e_2 f_1 \bmod q}\alpha^{-e_2 f_1 \bmod q} \pmod{p} \\
&\equiv y^{e_1 a^{-1} f_1 \bmod q} \pmod{p}
\end{aligned}$$

and, with a similar reasoning:

$$(D\alpha^{-f_2})^{e_1} \equiv y^{e_1 a^{-1} f_1 \bmod q} \pmod{p}$$

and thus it verifies the consistency property checked at **Step 9** of the protocol. ■

Now it is discussed the (im)possibility of the signer repudiate a valid signature by not performing trustfully the disavowal protocol. In that case, the signer had to produce d and D such that the following congruences had to hold:

1. $y \equiv x^a \pmod{p}$;
2. $d \not\equiv x^{e_1}\alpha^{e_2} \pmod{p}$;
3. $D \not\equiv x^{f_1}\alpha^{f_2} \pmod{p}$;
4. $(d\alpha^{-e_2})^{f_1} \equiv (D\alpha^{-f_2})^{e_1} \pmod{p}$.

But then, one would have $D \equiv d_0^{f_1}\alpha^{f_2} \pmod{p}$, with $d \equiv d^{1/e_1}\alpha^{-e_2/e_1} \pmod{p}$. By Theorem 13.1.1, with probability $1 - 1/q$, y is a valid signature of d_0 and since it is assumed that y is a valid signature for x, then $x \equiv d_0 \pmod{p}$.

On the other hand, since $d \not\equiv x^{e_1}\alpha^{e_2} \pmod{p}$ i.e., $x \not\equiv d^{1/e_1}\alpha^{-e_2/e_1} \pmod{p} \equiv d_0 \pmod{p}$ which is a contradiction.

Hence one concludes that:

Theorem 13.1.3

Assume that $y \equiv x^a \pmod{p}$ and the verifier follows the disavowal protocol. If $d \not\equiv x^{e_1}\alpha^{e_2} \pmod{p}$ and $D \not\equiv x^{f_1}\alpha^{f_2} \pmod{p}$, then the probability that $(d\alpha^{-e_2})^{f_1} \equiv (D\alpha^{-f_2})^{e_1} \pmod{p}$ is at least $1 - 1/q$.

The idea behind undeniable signatures is similar to a zero knowledge proofs since, for example, it requires interaction; at the end of the verification procedure the verifier does not gained any information about the secret of the signer and also he cannot convince anyone else that he had interact with the prover.

In the sequel it is presented a variation of the undeniable signature in which the verification and the disavowal protocols are zero knowledge accordingly to Definition 13.0.2.

Consider the Chaum-van Antewerpe undeniable signature presented in Page 309 with the following protocol for verification:

Protocol 13.1.4 (*Zero knowledge verification procedure*)

Input: *A message $x \in G$ and a possible signature $y \in G$ of x.*

Output: 1 *or* 0 *depending on the validity of the signature.*

Step 1 *The verifier generates uniformly at random d, e in G.*

Step 2 *The verifier computes $x^d \alpha^e \pmod p$ and sends this value to the signer.*

Step 3 *The signer generates uniformly at random f in G.*

Step 4 *The signer computes $x^d \alpha^{e+f} \pmod p$ and $(x^d \alpha^{e+f})^a \pmod p$ and sends these values to the signer.*

Step 5 *The verifier sends back d and e to the signer.*

Step 6 *The signer returns f to the verifier.*

Step 7 *Upon receiving the f the verifier can compute $x^a \pmod p$ and check if $y \equiv x^a \pmod p$. If that is the case, then the verifier outputs 1 and otherwise outputs 0.*

Notice that if both the signer and the verifier follow the verification protocol then from $x^d \alpha^{e+f} \pmod p$, $(x^d \alpha^{e+f})^a \pmod p$ and d, e, f the verifier can reconstruct $x^a \pmod p$, the claimed signature of x. Notice that only the signer can produce $x^d \alpha^{e+f} \pmod p$ and $(x^d \alpha^{e+f})^a \pmod p$ since he is the only one that knows a.

The non transferability property follows from the fact that the verifier could, instead of performing the steps 2 and 5 could simple generate α^e and α^{xe} and claim that the protocol had succeed.

The completeness property follows form the fact that only with probability $1/p$ the signer can lead the protocol to a good end with a fake signature.

Now it is presented the disavowal protocol.

Protocol 13.1.5 (*Zero knowledge disavowal procedure*)

Input: *A message $x \in G$ and a possible signature $y \in G$ of x and k a number of rounds.*

Step 1 *The verifier generates uniformly at random d in G and $s \in \{0, \cdots, k\}$.*

Step 2 *The verifier computes $x^d \alpha^s \pmod{p}$ and $y^s \alpha^{ad} \pmod{p}$ sends this value to the signer.*

Step 3 *The signer generates uniformly at random r in G.*

Step 3 *The signer by trial and error tries to find s. If s is not found chooses s uniformly at random and using r he commits himself with the value s and send the commitment to the verifier.*

Step 5 *The verifier sends back d to the signer.*

Step 6 *The signer uses d to reconstruct the first message sent and returns r to the verifier.*

Step 7 *Upon receiving the f the verifier can compute $x^a \pmod{p}$ and check if $y \equiv x^a \pmod{p}$. If that is the case, then the verifier outputs 1 and otherwise outputs 0.*

The zero knowledge property will be seen in due course...

CHAPTER 14

Stuff

Part V

Bibliography

CHAPTER 14

Bibliography

[AB09] S. Arora and B. Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[And71] G. Andrews. *Number Theory*. Dover Publications, Inc, 1971.

[CT91] T. Cover and J. Thomas. *Elements of information theory*. Wiley-Interscience, New York, NY, USA, 1991.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[HPS08] J. Hoffstein, J. Pipher, and J. Silverman. *An Introduction to Mathematical Cryptography*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[IR90] K. Ireland and M. Rosen. *A Classical Introduction to Modern Number Theory*. Graduate Texts in Mathematics. Springer-Verlag, 1990.

[Kal02] O. Kallenberg. *Foundations of Modern Probability*. Probability and Its Applications. Springer, 2002.

[KL07] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

[Lan02] S. Lang. *Algebra*. Springer, 3rd edition, January 2002.

[MVO96] A. Menezes, S. Vanstone, and P. Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

[oST99] National Institute of Standards and Technology. *Data Encryption Standard (DES)*. FIPS - PUB 46-3, 1999.

[Pap94] C. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994.

[Sha48] C. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27, 1948.

[Sip12] M. Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, 3rd edition, 2012.

[Sti06] D Stinson. *Cryptography: Theory And Practice*. The CRC Press Series on Discrete Mathematics and its Applications. Chapman & Hall/CRC, 2006.