

ALMA MATER STUDIORUM · UNIVERSITÀ DI  
BOLOGNA

---

SCUOLA DI SCIENZE  
Corso di Laurea in Matematica

QUANTUM COMPUTING  
AND  
POST-QUANTUM CRYPTOGRAPHY

Tesi di Laurea in Meccanica Quantistica e Crittografia

Relatore:  
Chiar.mo Prof.  
Marco Lenci

Presentata da:  
Giuseppe Murolo

Correlatrice:  
Chiar.ma Dott.ssa  
Simona Chiarelli

Sessione unica  
a. a. 2020-2021



## **Abstract**

One of the main practical implications of quantum mechanical theory is quantum computing, and therefore the quantum computer. Quantum computing (for example, with Shor's algorithm) challenges the computational hardness assumptions, e.g., of the factoring problem and the discrete logarithm problem, that anchor the safety of cryptosystems. So the scientific community is studying how to defend cryptography. There are two defense strategies: the quantum cryptography (which involves the use of quantum cryptographic algorithms on quantum computers) and the post-quantum cryptography (based on classical cryptographic algorithms, but resistant to quantum computers). For example, the National Institute of Standards and Technology (NIST) is collecting and standardizing post-quantum ciphers, as it established DES and AES as symmetric cipher standards in the past.

In this thesis we start by giving an introduction on quantum mechanics, in order to introduce quantum computing and analyze Shor's algorithm. The differences between quantum and post-quantum cryptography are then analyzed. Subsequently the focus was given to certain mathematical problems assumed to be resistant to quantum computers. To conclude, we study and compare post-quantum digital signature cryptographic algorithms selected by NIST.



# Contents

<b>Introduzione</b>	<b>7</b>
<b>Introduction</b>	<b>11</b>
<b>1 Quantum Computing</b>	<b>15</b>
1.1 Quantum bit . . . . .	18
1.2 Measurement problem . . . . .	19
1.3 Physical interpretation of the qubit . . . . .	20
1.4 Quantum register . . . . .	21
1.5 Entangled states . . . . .	23
1.6 Quantum logic gate . . . . .	24
1.7 Quantum circuits . . . . .	29
1.8 Quantum Parallelism . . . . .	32
1.9 Mathematical structure of quantum mechanics . . . . .	35
<b>2 Shor's algorithm and its Impact on cryptography</b>	<b>39</b>
2.1 RSA public-key cryptosystem . . . . .	40
2.2 Shor's algorithm . . . . .	46
2.2.1 Reduction of factorization to order calculation . . . . .	46
2.2.2 The quantum algorithm to calculate the order: a particular case . . . . .	48
2.2.3 The quantum discrete Fourier transform . . . . .	52
2.2.4 The most general case . . . . .	57
2.3 Impact on cryptography . . . . .	62

<b>3</b>	<b>Quantum cryptography</b>	<b>67</b>
3.1	BB84: quantum key exchange protocol . . . . .	68
<b>4</b>	<b>Post-Quantum Cryptography</b>	<b>75</b>
4.1	Standardization: the NIST challenge . . . . .	80
<b>5</b>	<b>Post-Quantum Cryptography hard problems</b>	<b>83</b>
5.1	Hard Problems in Lattices for cryptography . . . . .	85
5.2	Hash function for cryptography . . . . .	90
5.3	Code-based cryptography . . . . .	94
5.4	Multivariate-equation-based cryptography . . . . .	99
5.5	Supersingular isogeny-based cryptography . . . . .	103
5.5.1	Elliptic curves . . . . .	105
5.5.2	Supersingular isogeny Diffie-Hellman key exchange: SIDH111	
5.5.3	Post-Quantum hard problems on elliptic curves . . . . .	114
<b>6</b>	<b>Post-quantum Cryptography in embedded world</b>	<b>117</b>
6.1	Winternitz signature scheme . . . . .	118
6.1.1	A possible attack protected by a checksum . . . . .	124
6.1.2	Key re-using: a problem for WOTS . . . . .	125
6.2	XMSS and LMS . . . . .	128
6.2.1	Long-term public key economize: Merkle tree . . . . .	128
6.2.2	XMSS: The eXtended Merkle Signature Scheme . . . . .	130
6.3	Comparison of digital signature algorithms for practical use . . . . .	136
6.3.1	A comparison between XMSS and LMS . . . . .	136
6.3.2	A comparison of NIST's finalist digital signature schemes	142
	<b>Conclusion</b>	<b>155</b>
	<b>Appendices</b>	<b>157</b>
<b>A</b>	<b>Continued Fractions</b>	<b>159</b>

---

<b>B</b>	<b>The group structure of an elliptic curve</b>	<b>163</b>
B.1	Geometric interpretation of the sum between two points . . .	163
B.2	Coordinates of the sum between two points . . . . .	167
B.3	Elliptic Curve Discrete Logarithm Problem . . . . .	169
<b>C</b>	<b>Algebraic varieties</b>	<b>171</b>
	<b>Bibliography</b>	<b>177</b>
	<b>List of figures</b>	<b>185</b>





# Introduzione

Il tema centrale sviluppato in questa tesi è l'evoluzione della crittografia conseguente allo sviluppo del computer quantistico.

È infatti diventato di ampio interesse ricercare valide alternative agli attuali algoritmi crittografici (usati da ognuno di noi nella vita di tutti i giorni, per esempio nei cellulari, nei computer, etc.) che sono minacciati da un potenziale sviluppo in larga scala del computer quantistico.

Un articolo datato Aprile 2016, pubblicato dal NIST (*National Institute of Standards and Technology*), asserisce che il progressivo sviluppo delle tecnologie quantistiche renderà insicuro il comunissimo algoritmo RSA entro il 2030. Di conseguenza è nata la necessità di standardizzare primitive crittografiche — che sono algoritmi crittografici di basso livello, consolidati, usati per costruire protocolli crittografici — resistenti anche ai computer quantistici. Poiché si è visto che la maggior parte dei sistemi a chiave privata sono facilmente modificabili in modo da ottenere algoritmi quanto-resistenti, gli sforzi della comunità scientifica (e di questa tesi) si sono concentrati sulla crittografia a chiave pubblica.

Nel 2016 il NIST ha annunciato la Post-Quantum Cryptography Standardization (dove con "post-quantum" si intendono algoritmi che girano su computer classici, ma che sono di resistenza quantistica). È una competizione promossa dal NIST, che ha il fine di standardizzare algoritmi post-quantistici.

Nello sviluppo di questa tesi, la mia attenzione si è in particolare rivolta agli algoritmi post-quantistici di firma digitale, essendo questi ultimi negli interessi della *Marelli*, l'azienda che ha supportato questo studio. In partico-

lare, la Marelli sta cercando un algoritmo post-quantistico di firma digitale da implementare sulle ECU (le unità di controllo elettronico delle automobili, conosciute più comunemente come "centraline elettroniche") di loro produzione. Questo algoritmo andrà a sostituire i precedenti, i quali andranno in disuso.

Una buona parte di questa tesi, specialmente nei Capitoli 4,5,6, è stata ricavata da documenti del NIST, o documenti prodotti per la competizione promossa dal NIST.

Questa tesi si struttura in 6 capitoli.

Il primo presenta le basi della meccanica quantistica necessarie alla computazione quantistica. L'algoritmo di Shor è quindi studiato nel secondo capitolo: il suo avvento ha profondamente rivoluzionato il mondo della crittografia. Infatti, tanti dei problemi la cui intrattabilità è alla base dei molti algoritmi classici, verrebbero risolti in tempo polinomiale.

È interessante osservare che, per trattare la crittografia post-quantistica, non è necessario studiare preliminarmente la meccanica quantistica e quindi la computazione quantistica. Tanti libri classici sulla crittografia post-quantistica, infatti, adottano questo approccio.

I Capitoli 3 e 4 introducono i concetti, rispettivamente, di crittografia quantistica (ovverosia, un tipo di crittografia che utilizza, dalla sua, la computazione quantistica) e crittografia post-quantistica. In particolare, nel Capitolo 3, è studiato il protocollo BB84 di scambio di chiavi, come esempio di crittografia quantistica.

Il Capitolo 5 analizza molti dei problemi — creduti essere quanto-resistenti — alla base degli attuali sistemi di crittografia post-quantistica.

Nel Capitolo 6 gli algoritmi di firma digitale post-quantistica sono studiati dal punto di vista di una potenziale applicazione. Dopo aver introdotto gli schemi XMSS e LMS, si sviluppa infatti un confronto circa le performance, le dimensioni dei parametri e la sicurezza, degli algoritmi di firma selezionati dal NIST (che sono: Rainbow, Falcon, Crystals-Dilithium).

Questa tesi si conclude con 3 Appendici contenenti particolari temi in qualche modo correlati agli argomenti trattati nella tesi, ma la cui spiegazione avrebbe distolto il lettore dal flusso del discorso generale.

Questa tesi offre, oltre alle basi della computazione quantistica, una ampia visione sul mondo della crittografia post-quantistica, nella sua più recente evoluzione.



# Introduction

The central theme of this thesis is the analysis of the evolution of cryptography, consequent to the development of the quantum computer.

In fact it is of wide interest to find alternatives to current cryptographic algorithms, because most of today's systems (which we use in everyday's life, for example in mobile phones, in computers, etc.) are jeopardized from a potential development and use of the quantum computer on a large scale.

A NIST (*National Institute of Standards and Technology*) report, published in April 2016, cites experts who state that the quantum technology could render the commonly used RSA algorithm insecure by 2030. As a result, the need to standardize quantum-secure cryptographic primitives — which are well-established, low-level cryptographic algorithms frequently used to build cryptographic protocols — was pursued. Since most symmetric primitives are relatively easy to modify in a way that makes them quantum resistant, efforts were focused on public-key cryptography.

For these reasons, the commitment of the scientific community on this issue is constantly growing. In 2016 the NIST announced the Post-Quantum Cryptography Standardization (where "post-quantum" indicates non-quantum algorithms which run on classical computers, but are supposed to be resistant to quantum computers as well). It is a program and competition to update NIST standards to include post-quantum cryptography.

In the interests of *Marelli*, the company that supported my thesis, my attention has been placed on digital signature algorithms supposed to be resistant to quantum computers. In particular, Marelli is looking for performing

post-quantum digital signature algorithms, to be implemented in the ECUs (engine control unit) of their production. These algorithms would replace the old systems that will go into disuse.

Most of this thesis, especially the topics in Chapters 4,5,6, is taken from documents by NIST, or documents produced for the NIST competition.

This thesis is structured in 6 chapters.

The first chapter presents the basics of quantum mechanics that are needed for quantum computing. Shor's algorithm is then explained in the second chapter; it is an algorithm that has revolutionized the world of cryptography. In fact, it has shown that famous problems whose hardness is at the basis of classical cryptography (such as the factoring problem and the discrete logarithm problem) are solved in a reasonable time with the use of quantum computers.

It is interesting to note that it is not necessary to know quantum computing in detail to study the functioning of post-quantum algorithms: this approach is in fact chosen by numerous post-quantum cryptography books.

In Chapters 4 and 3 the differences between, respectively, post-quantum cryptography and quantum cryptography (cryptography resistant to quantum computers which makes use of quantum computers) are highlighted. In particular, in Chapter 3, the BB84 quantum key exchange protocol is reported as an example of quantum cryptography.

Chapter 5 analyzes most of the problems (believed to be quantum-resistant) at the basis of current post-quantum cryptographic systems.

In Chapter 6, post-quantum cryptography (and in particular certain digital signature algorithms) are studied from the point of view of a potential application. After introducing and analyzing two hash-based signature algorithms (XMSS and LMS), the analysis moves to a comparison of all the performances, parameter sizes, and security of the shortlisted digital signature algorithms selected by the NIST (which are Rainbow, Falcon, Crystals-Dilithium).

The thesis is concluded by 3 Appendices, which contain topics in a way or another related to subjects covered in the thesis, the explanation of which would have diverted the reader's attention from the general discourse.

In addition to the basics of quantum computing, this thesis offers in a very broad vision on the world of post-quantum cryptography with its recent evolution.





# Chapter 1

## Quantum Computing

Everyone has heard about certain "supercomputers" called *Quantum Computers*, but what are they more precisely? What are the mysteries behind their functioning? Do they exist physically or only conceptually? What consequences would they have in our lives? Let us try to give some answers.

A computation process is essentially a physical process that is performed on a machine whose operation obeys certain physical laws. The classical theory of computation is based on an abstract model of a universal machine (the Universal Turing Machine) that works according to a set of rules and principles set out by Alan Turing in 1936 and subsequently elaborated by John von Neumann in the 1940s. These principles have remained essentially unchanged since then, despite the enormous technological advances that make it now possible to produce far more powerful devices than those that could be achieved in the first half of the twentieth century. The tacit assumption behind these principles is that a Turing machine idealizes a mechanical computing device (with potentially infinite memory) that obeys the laws of classical physics.

Quantum computing was born as an alternative paradigm based on the principles of quantum mechanics. These are the only ones able to justify the physical phenomena that occur at the microscopic level, such as inside an atom. These phenomena will be essential in the construction of electronic

computers in the near future, if Moore's law continues to apply, as expected. This law, formulated as early as the 1960s, predicted that the computational power of the best performing available computer would approximately double every two years.

In practice this law has proved its validity up to now, and currently quantum effects begin to interfere in the functioning of electronic devices as their dimensions become smaller.

The idea of realizing a computation model as an isolated quantum system began to emerge in the early 1980s, when P. Benioff, starting from considerations previously elaborated by C. Bennet, defined the reversible Turing machine: a computation can always be performed in such a way as to return to the initial state by retracing the various computation steps backwards.

Subsequently, R. Feynman argued that no classical Turing machine could simulate certain physical phenomena without incurring in an important slow-down in its performance. Conversely, a "universal quantum simulator" could perform the simulation more efficiently.

In 1985 D. Deutsch formalized these ideas in his Universal Quantum Turing Machine, which represents in quantum computational theory exactly what the Universal Turing Machine represents for classical computing and has led to the modern concept of quantum computation. Of course, the effects of the introduction of the new computational model were also felt in the field of computational complexity (as predicted by Feynman). In fact, in 1994, P. Shor demonstrated that the problem of the factorization of prime numbers (classically considered intractable) can be solved efficiently ( i.e., in polynomial time) with a quantum algorithm [54].

These considerations, combined with the technological ones mentioned previously, have led to the emergence of a research field now known as *quantum computing*.

A *quantum computer* is therefore a device that can perform quantum computations.

However, it should be pointed out that the road leading to the construction

of the quantum computer has not been immediate, and has not yet been fully run: scientists are still working on it. Let us make a small overview of the most important events on the development of the quantum computer.

The first idea of a quantum computer was exposed in two papers published by physicists Paul Benioff and Yuri Manin in 1980. As we have already mentioned, the development of the quantum computer accelerated after the publication of the Shor's algorithm in 1994 [54]. In 1996, DiVincenzo published an article on the requirements that a quantum computer must satisfy [24], but it was only five years later, in 2001, that Shor's algorithm was first used to factor the number 15 [62]. However, one had to wait until 2011 for larger integers, such as 21 or even 143, to be factored by some rudimentary implementation of a quantum computer ([40] and [66]). A year later, a research group led by Australian engineers manages to create the first working qubit, based on a silicon atom. In 2013, Google announced the birth of the Quantum Artificial Intelligence Lab, a research laboratory with the aim of studying the quantum computer in the context of machine learning. Two years later, the laboratory published a paper in which it claims to have made a quantum algorithm work on its machine. In 2016, IBM announced the first IBM cloud computing quantum cloud service. In 2017, it is again IBM that created the first 56-qubit system, which was however surpassed the following year by Google's 72-qubit processor testing. In early 2019, IBM created the first quantum computer to be used outside the laboratory, which however still seems to be far from becoming commercially available. In January 2020 IBM announced the largest Quantum Volume ever (the Quantum Volume is a metric that measures the capabilities and error rates of a quantum computer), equal to 32, on a 28 qubit quantum processor, confirming the annual doubling trend of the power of its quantum computers. In April 2020, QuTech launched Quantum Inspire, the first quantum processor based on "spin qubits" controlled by lock-In amplifier technology.

## 1.1 Quantum bit

The fundamental concept of classical computation is the *bit*. Quantum computing is based on an analogous concept, the *quantum bit*, or *qubit* in short, whose fundamental properties are described below, underlining the differences with the classical bit.

Before giving a mathematical interpretation of the qubit, it is useful to introduce the *Dirac notation*, from the name of the famous English physicist, pioneer of quantum theory, who introduced it. It represents the standard notation in quantum mechanics for representing the elements of a complex vector space.

According to this notation,  $|v\rangle$  (or *ket*) indicates a generic element of the vector space and  $|i\rangle$  indicates the  $i$ -th element of the canonical orthonormal basis.

If  $|v\rangle$  and  $|w\rangle$  are two vector of the vector space, their scalar product is indicated with:

$$\langle v|w\rangle := (v_1^*, \dots, v_n^*) \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}$$

(where the apex  $*$  indicates the complex conjugate of the complex number). In general the row vector  $(v_1^*, \dots, v_n^*)$  is denoted as  $\langle v|$ , so that  $\langle v|w\rangle$  forms a *bra-ket*.

The state of a classic bit is described by the values 0 and 1. The more direct way to represent the state of a qubit is by means of a unit vector in a complex two-dimensional vector space. The vectors  $|0\rangle$  and  $|1\rangle$  form an orthonormal basis for this vector space, known as the *standard computational basis*. Using the classical notation of linear algebra, we can represent  $|0\rangle$  with the column vector  $(1, 0)^T$  and  $|1\rangle$  with the column vector  $(0, 1)^T$ , where  $T$  indicates the transpose. The states  $|0\rangle$  and  $|1\rangle$  of a qubit can be seen as the correspondents of the states 0 and 1 of a classic bit.

The difference between bits and qubits is that a qubit can also be found in

other states than  $|0\rangle$  and  $|1\rangle$ . In fact, any linear combination

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle,$$

where  $\alpha, \beta \in \mathbb{C}$  such that  $|\alpha|^2 + |\beta|^2 = 1$ , is a possible state for a bit. In classical notation,  $|\psi\rangle$  corresponds to:

$$\begin{pmatrix} \alpha \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \beta \end{pmatrix} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

These states are often called *superpositions* of pure qubits.

We are going to use ( $\mathbb{C}$ -)vector spaces. In fact, for our purpose (quantum computing) it is sufficient, but actually, to represent quantum states, Hilbert Spaces are used rather than simple vector spaces; in effect, what is generally needed, is the presence of a scalar product. To be more precise, in general, the projective spaces of Hilbert spaces are used, where the concept of norm makes no sense: this is why, for reasons of convenience, we represent states as vectors of norm 1.

## 1.2 Measurement problem

While for a classical bit we can always determine its state and establish precisely whether it is 0 or 1, for a qubit we cannot determine its quantum state with equal precision, that is, the values of  $\alpha$  and  $\beta$ . Quantum mechanics tells us that when we measure a qubit  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ , we can only obtain the state  $|0\rangle$  with a probability equal to  $|\alpha|^2$  or the state  $|1\rangle$  with a probability equal to  $|\beta|^2$ .

For this reason the values  $\alpha$  and  $\beta$  are called *probability amplitudes* and the sum  $|\alpha|^2 + |\beta|^2$  must be 1.

Geometrically this means that the states of a qubit are normalized, i.e. unit, vectors.

We have therefore established that a qubit possesses in a number of states which is infinitely greater than that of the possible states of a classical bit.

It will be seen later that the physical realization of a qubit does not allow to observe these states directly: the "measurement" of a qubit will always give as a result the state  $|0\rangle$  or the state  $|1\rangle$ .

However, it should be noted that the result of the measurements strictly depend on the specific properties of the state (in particular, the values of  $\alpha$  and  $\beta$ ) on which the transformations have been carried out: this is where the power of quantum computing essentially resides.

### 1.3 Physical interpretation of the qubit

The abstract description of a qubit as a vector in a complex two-dimensional space has a correspondent in the real world. In particular, any physical system with at least two discrete and sufficiently separated energy levels is an appropriate candidate to represent a qubit.

To physically make a qubit the three most common approaches are those based on:

- the two different polarizations of a photon;
- the alignment of a nuclear spin in a uniform magnetic field;
- two energy levels of an electron orbiting a single atom. (In an atom the energy levels of the various electrons are discrete. Two of them can be selected to represent the logical values 0 and 1. These levels correspond to specific states excitation of electrons in the atom.)

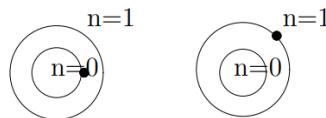


Figure 1.1: Qubit represented by an electron in a hydrogen atom

For example, using the third approach, we can consider the system constituted by the hydrogen atom H. In this system, the state  $|0\rangle$  of the qubit can be represented by the first energy level ( $n = 0$ ), corresponding to the ground state of the electron, and the state  $|1\rangle$  from the second energy level ( $n = 1$ ) corresponding to the first excited state of the electron, as in Figure 1.1.

The passage of the electron from one state to another can be achieved by subjecting the electron to a laser pulse of appropriate intensity, duration and length wave. By appropriately reducing the duration, it is possible to carry out the transition of an electron initially in the state  $|0\rangle$  to a state that is "in the middle" between  $|0\rangle$  and  $|1\rangle$ , corresponding to a superposition.

When we observe a qubit, however, the result can only be 0 or 1. As a matter of fact, the measurement we made changes the state of the qubit, causing it to collapse from its superposition to the specific state consistent with the measurement result.

These properties are explained by the principles of quantum mechanics, explained in Section 1.9.

## 1.4 Quantum register

With two classic bits we can form four possible states: 00, 01, 10, 11. In general, with  $n$  bits it is possible to construct  $2^n$  distinct states.

How many states can be obtained with  $n$  qubits? The vector space of states generated by a system of  $n$  qubits has dimension  $2^n$ : each normalized vector in this space represents a possible computational state, which we will call *n qubit quantum register*.

This exponential growth in the number of qubits in the dimensions of the state space suggests the potential ability of a quantum computer to process information at an exponentially higher speed than that of a classical computer. Note that for  $n = 300$  we obtain a number that is greater than the number of atoms in the universe.

Formally, a quantum register of  $n$  qubits is an element of a  $2^n$ -dimensional complex Hilbert space, with a computational basis formed by  $2^n$   $n$ -qubit registers:

$$|i_1\rangle \otimes |i_2\rangle \otimes \cdots \otimes |i_n\rangle \text{ with } i_j \in \{0, 1\}, \text{ for all } 1 \leq j \leq n$$

For convenience, we write this basis vector  $|i_1\rangle|i_2\rangle \dots |i_n\rangle$  or simply  $|i_1i_2 \dots i_n\rangle$ . The symbol  $\otimes$  stands for *tensor product*.

We define the tensor product between two vectors to be the following function:

$$\otimes : \mathbb{C}^k \times \mathbb{C}^l \longrightarrow \mathbb{C}^{kl}$$

where

$$v \otimes w := \begin{pmatrix} v_1w \\ v_2w \\ \vdots \\ v_kw \end{pmatrix}$$

( $v_jw$  is the multiplication of the vector  $w$  by the scalar  $v_j$ ).

*Example 1* (2-qubit case). Let consider the case of 2 qubit. The computational basis of the states space is  $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ . In algebraical notation these vectors correspond to:

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

A quantum register of 2 qubits has the form:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$$

with the condition:  $\sum_{i \in \{0,1\}^2} |\alpha_i|^2 = 1$ .

Similarily at the single qubit case, the measuring result of  $|\psi\rangle$  will be one of



the states  $i \in \{0, 1\}^2$  with probability  $|\alpha_i|^2$ .

It is interesting that in an  $n$  qubit system it is also possible to measure only a subset of the  $n$  qubits. For example, in the case of a two-qubit register, we can measure the first qubit and obtain as a result 0 with probability of  $|\alpha_{00}|^2 + |\alpha_{01}|^2$ . After this mensuration, the state will collapse to

$$\frac{|\alpha_{00}|^2|00\rangle + |\alpha_{01}|^2|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}},$$

wich is the projection of the initial state on  $\text{span}\{|00\rangle, |01\rangle\}$ , i.e., the subset of the space corresponding to the first spin equal to 0. This is a general property of quantum computers.

Note that the state is re-normalized to have length 1.

## 1.5 Entangled states

An important property of  $n$ -qubit quantum registers is that it is not always possible to factor them into the states of the component qubits. States of this type are called *entangled* and enjoy properties that cannot be found in any object of classical physics.

The spin of an entangled collection do not have their own individual status, only the entire collection corresponds to a well-defined status. Entangled spins behave as if they were closely connected to each other regardless of the distance that separates them. For example, a measurement of one of the two members (spins) of an entangled pair provides simultaneously information about the other. This property is the basis for solutions of information-processing problems that cannot be reproduced classically.

Formally: consider two arbitrary quantum systems  $A$  and  $B$ , with respective Hilbert spaces  $H_A$  and  $H_B$ . The Hilbert space of the composite system is the tensor product  $H_A \otimes H_B$ .

Fixing a basis  $\{|i\rangle_A\}_i$  for  $H_A$  and a basis  $\{|j\rangle_B\}_j$  for  $H_B$ , the most general

state in  $H_A \otimes H_B$  is of the form:

$$|\psi\rangle_{AB} = \sum_{i,j} c_{i,j} |i\rangle_A \otimes |j\rangle_B$$

Some of these states can be represented as

$$|\psi\rangle_{AB} = |\phi\rangle_A \otimes |\delta\rangle_B = \left( \sum_i a_i |i\rangle_A \right) \otimes \left( \sum_j b_j |j\rangle_B \right)$$

where  $|\phi\rangle_A \in H_A$  and  $|\delta\rangle_B \in H_B$ . States of this type are called *separable states*.

Unfortunately (or fortunately!) not all states are separable; the states that are not separable are called *entangled states*.

*Example 2* (Entanglement). It is a quick check to show that the state  $|00\rangle + |11\rangle$  is not separable in the tensor product of two independent qubit; i.e.,  $\nexists a_1, a_2, b_1, b_2 \in \mathbb{C}$  such that:

$$|00\rangle + |11\rangle = (a_1|0\rangle + b_1|1\rangle) \otimes (a_2|0\rangle + b_2|1\rangle).$$

## 1.6 Quantum logic gate

We have studied the quantum description of the states of a computation. let us now see how these states evolve giving rise to a quantum computation. Similarly to classical computers, a quantum computer is made of quantum circuits made up of elementary quantum logic gates.

In the classic case, there is a single (non-trivial) one-bit logic gate, the NOT gate, which implements the logical negation operation defined by means of a truth table in which  $1 \rightarrow 0$  and  $0 \rightarrow 1$ .

To define an analogous operation on a (single) qubit, we cannot limit ourselves to establishing its action on the basis states  $|0\rangle$  and  $|1\rangle$ , but we must also specify how a qubit that is in a superposition of the states these two. Intuitively, the NOT should swap the roles of the two ground states and

transform  $\alpha|0\rangle + \beta|1\rangle$  into  $\beta|0\rangle + \alpha|1\rangle$ . Clearly  $|0\rangle$  would turn into  $|1\rangle$  and  $|1\rangle$  into  $|0\rangle$ . The operation that implements this type of transformation is a linear operation and, as we will see, this is a general property of quantum mechanics that is experimentally justified. A convenient way to represent linear operations is by means of matrices.

The matrix corresponding to the quantum NOT is called for historical reasons  $X$ , and is defined by:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

In fact, it is easy to check that:

$$X \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix}$$

In general, an operation on a single qubit can be specified by a  $2 \times 2$  matrix. However, not all  $2 \times 2$  matrices define “legal” operations on qubits. Recall that the qubit requires normalization condition, so the same condition must also apply to the status that is obtained after carrying out the operation. In algebra, matrices which preserve the norm of the vector they multiply, are called *unitary matrices*.

**Definition 1** (Unitary matrix). A complex square matrix  $U$  is unitary if its conjugate transpose  $U^\dagger$  is also its inverse, that is, if:

$$U^\dagger U = U U^\dagger = I$$

Or equivalently:

**Definition 2** (Unitary matrix).  $U$  is unitary if and only if it preserves the scalar products:

$$\langle Uu|Uv \rangle = \langle u|v \rangle \quad \forall u, v \in \mathbb{C}^n$$

**Theorem 1.** *A linear function transforms a qubit into a qubit ( i.e., preserves normalized vectors) if and only if it is represented by a  $2 \times 2$  complex unitary matrix.*

**Definition 3** (One-Qubit quantum logic Gate). A one-qubit quantum logic gate is a transformation represented by a  $2 \times 2$  complex unitary matrix.

*Example 3* (Pauli matrices). The following matrices are known as *Pauli's matrices*, and are unitary matrices:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

In particular, the  $Z$  gate only acts on the component  $|1\rangle$  of the vector, changing its sign.

*Example 4* (Hadamard matrix). The following unitary matrix is called *Hadamard Matrix*:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

This matrix is one of the most useful operations in quantum circuits. Its effect is to transform one basis state in an another which results, after a measurement in the computational basis, to be  $|0\rangle$  or  $|1\rangle$  with equal probability.

E.g., applying  $H$  to  $|0\rangle$  we obtain:

$$H \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

Operations on quantum registers of two or more qubits are necessary to describe the transformations of compound states and in particular of the entangled states: we have seen that a register of two qubits is not always decomposed into the tensor product of the single component qubits. Consequently, in these cases we cannot simulate an operation on the two qubits by means of operations on each component qubit.

Operations on qubit registers also correspond to unitary transformation, as in the case of a single qubit, but via matrices of higher dimension.

The most important logic gates that implement classic two-bit operations are the AND, OR, XOR, NAND and NOR (summarised in table of Figure








Name	NOT	AND	NAND	OR	NOR	XOR	XNOR																																																																																																
Alg. Expr.	$\bar{A}$	$AB$	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A \oplus B$	$\overline{A \oplus B}$																																																																																																
Symbol																																																																																																							
Truth Table	<table border="1"> <thead> <tr> <th>A</th> <th>X</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	X	0	1	1	0	<table border="1"> <thead> <tr> <th>B</th> <th>A</th> <th>X</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	B	A	X	0	0	0	0	1	0	1	0	0	1	1	1	<table border="1"> <thead> <tr> <th>B</th> <th>A</th> <th>X</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	B	A	X	0	0	1	0	1	1	1	0	1	1	1	0	<table border="1"> <thead> <tr> <th>B</th> <th>A</th> <th>X</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	B	A	X	0	0	0	0	1	1	1	0	1	1	1	1	<table border="1"> <thead> <tr> <th>B</th> <th>A</th> <th>X</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	B	A	X	0	0	1	0	1	0	1	0	0	1	1	0	<table border="1"> <thead> <tr> <th>B</th> <th>A</th> <th>X</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	B	A	X	0	0	0	0	1	1	1	0	1	1	1	0	<table border="1"> <thead> <tr> <th>B</th> <th>A</th> <th>X</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	B	A	X	0	0	1	0	1	0	1	0	0	1	1	1
A	X																																																																																																						
0	1																																																																																																						
1	0																																																																																																						
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	1																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	1																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	1																																																																																																					

Figure 1.2: NOT, AND, NAND, OR, NOR, XOR, XNOR gates.

1.2) gates.

The NOT and AND gates form a *universal set*: any Boolean function can be realized through a combination of these two operations. Note that XOR alone or even together with NOT is not universal: since, for its definition, it is a *parity generator* (which means that preserves total bit parity, i.e., if the sum of the two inputs is even, it returns 0, otherwise 1), only a subset of the Boolean functions are representable by this operation. More precisely, combining parity generator we will necessarily obtain parity generator; therefore we have the impossibility of obtaining the gates that are not parity generators, such as OR (for which:  $1 \text{ OR } 1 = 1$ ). A similar argument can be made for the XOR and NOT couple.

The quantum analog of XOR is the CNOT (controlled-NOT) gate that operates on two qubits: the first is called the control qubit and the second is the target qubit. If the check is zero then the target is left unchanged; if the control is one, then the target is negated, that is:

$$|00\rangle \mapsto |00\rangle, \quad |01\rangle \mapsto |01\rangle, \quad |10\rangle \mapsto |11\rangle, \quad |11\rangle \mapsto |10\rangle.$$

Equivalently, CNOT can be seen as the transformation

$$|A, B\rangle \mapsto |A, B \oplus A\rangle,$$

where  $A$  is the control qubit,  $B$  is the target and  $\oplus$  is the sum *modulo 2*, i.e., the classical XOR operation. The CNOT gate is usually represented

graphically by the circuit in Figure 1.3.

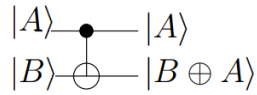


Figure 1.3: CNOT gate

Formally:

**Definition 4** (CNOT gate). The CNOT is a two-qubit quantum logic gate represented by the unitary matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

(where the matrix is written in the basis  $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ ).

It is important to note that the CNOT, like all unitary transformations, is invertible: the input can always be obtained from the output. This is not true for XOR and NAND logic gates: in general, classic operations are irreversible.

The CNOT gate and the one-qubit gates represent prototypes of all quantum logic gates. In fact, it is possible to show that, taken together, they form a universal set.

In general: the quantum version of the classic logic gate

$$f(x) : \{0, 1\} \rightarrow \{0, 1\}$$

(Boolean function) is a unitary transformation of the form:

$$U_f : |x, y\rangle \mapsto |x, y \oplus f(x)\rangle$$

where  $x, y \in \{0, 1\}$ ,  $x$  is the input and  $y$  is the register intended to contain the output. In this way, in fact, we have defined the action of  $U_f$  on the

elements of the base  $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ ; this action can therefore be extended by linearity.

To find exactly  $f(x)$  in the second register of the output, just choose  $y = 0$ . Graphically the  $U_f$  transformation is represented in Figure 1.4.

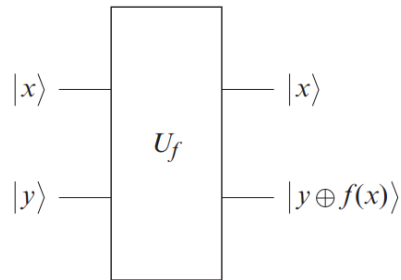


Figure 1.4:  $U_f$  representation

Note that the registers  $x$  and  $y$  can also have more than one component: in the case of  $x = x_1 \dots x_m$ ,  $y = y_1 \dots y_k$  and:

$$f(x) : \{0, 1\}^m \rightarrow \{0, 1\}^k$$

the form and the representation of  $U_f$  does not change.

The transformation  $U_f$  is considered the *reversible standard circuit* for the quantum evaluation of  $f$ .  $U_f$  is "reversible" because  $U_f^2 = I$ . In fact:

$$U_f U_f : |x, y\rangle \mapsto |x, y \oplus f(x) \oplus f(x)\rangle = |x, y \oplus 0\rangle = |x, y\rangle$$

## 1.7 Quantum circuits

A simple example of a quantum circuit is given in Figure 1.5.

The circuit, usually known as *swap*, realizes the exchange of the states of two qubits. Given in input the register of two qubits  $|a, b\rangle$ , a CNOT is performed with control qubits  $a$ . As a result  $b$  is replaced by  $a \oplus b$ . The latter is taken as a control of a second CNOT with target  $a$ . The effect is that  $a$  is replaced by  $a \oplus (a \oplus b) = b$ . A last CNOT with control  $b$  and target  $a \oplus b$  finally realizes

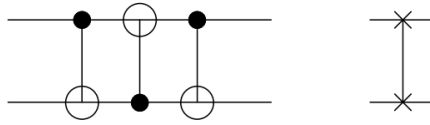


Figure 1.5: Swapping circuit realization, and schematic symbol

the exchange by replacing  $a \oplus b$  with  $a$ . Given any unitary operation  $U$  on  $n$  qubits, the *controlled- $U$*  circuit can be defined as the natural extension of the CNOT gate (cf. Figure 1.6).

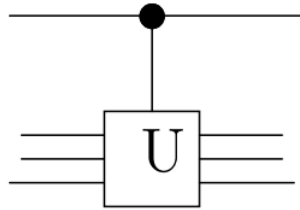


Figure 1.6: controlled- $U$  gate

The line with the black dot indicates the control qubit, while the target qubits are the  $n$  inputs of  $U$ . According to this convention, *controlled-NOT* is nothing more than a *controlled- $U$*  with  $U = X$ .

Another important operation is represented by the symbol in Figure 1.7 and consists in the measurement of a qubit  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ . As we know, the result is a classic bit  $M$  (indicated with a double line) which will be 0 or 1 with probability, respectively,  $|\alpha|^2$  and  $|\beta|^2$ .

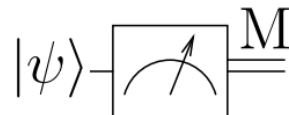


Figure 1.7: Circuit symbol for measurement.

It is interesting to know that there are quantum circuits capable of trans-



porting quantum states from one place to another (*Quantum teleportation*). If instead we ask ourselves if it is possible to build a circuit that makes the copy of a qubit, unfortunately the answer is no, as we are going to explain. One could think of using a CNOT with control qubit containing the qubit  $|x\rangle$  to be copied and the target register initially placed at  $|0\rangle$  (see Figure 1.8). The result would be copying  $x$  to the target. Actually this is true for classical

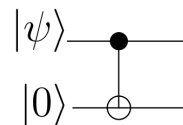


Figure 1.8: CNOT: a wrong circuit to "copy" a qubit.

bits (or for the states of the computational basis) but not for a generic qubit  $|\psi\rangle = a|0\rangle + b|1\rangle$ . In fact, consider the circuit consisting of a CNOT which has as input the qubits  $|\psi\rangle$  (as control) and  $|0\rangle$  (as target), i.e., the register  $|\psi\rangle|0\rangle$ . Our goal is to obtain  $|\psi\rangle|\psi\rangle$  as result.

We observe that

$$|\psi\rangle|\psi\rangle = a^2|00\rangle + ab|01\rangle + ab|10\rangle + b^2|11\rangle.$$

This state, unless  $ab = 0$ , is in general different from the result of our circuit:

$$CNOT(|\psi\rangle|0\rangle) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} a \\ 0 \\ b \\ 0 \end{pmatrix} = \begin{pmatrix} a \\ 0 \\ 0 \\ b \end{pmatrix} = a|00\rangle + b|11\rangle$$

So this circuit does not copy a qubit.

This quantum property is called *no-cloning*. Let us present it in the form of a theorem.

**Theorem 2** (No-cloning). *There is no unitary transformation  $M$  such that,  $M(|\psi\rangle|0\rangle) = |\psi\rangle|\psi\rangle$ , for each state  $|\psi\rangle$ .*

*Proof.* Let us suppose by absurdity that  $M$  as in the statement exists. So we can choose two states  $|\psi\rangle$  and  $|\phi\rangle$  such that  $0 < \langle\psi|\phi\rangle < 1$ . They exist: for example  $|0\rangle$  and  $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ .

Then we get that:

$$M(|\psi\rangle|0\rangle) = |\psi\rangle|\psi\rangle \text{ and } M(|\phi\rangle|0\rangle) = |\phi\rangle|\phi\rangle.$$

Multiplying the two vectors, we obtain:

$$\langle M(|\psi\rangle|0\rangle) | M(|\phi\rangle|0\rangle) \rangle = \langle |\psi\rangle|\psi\rangle | |\phi\rangle|\phi\rangle \rangle$$

Using now that  $M$  is unitary for the first member, and the distributive property of the tensor product with respect to the scalar product (it is in fact easy to verify that:  $\langle v \otimes w | v' \otimes w' \rangle = \langle v | v' \rangle \langle w | w' \rangle$ ) for the first and the second member, we find:

$$\langle\psi|\phi\rangle\langle 0|0\rangle = \langle\psi|\phi\rangle\langle\psi|\phi\rangle.$$

But now, using  $\langle 0|0\rangle = 1$ , we have a contradiction, because  $0 < \langle\psi|\phi\rangle < 1$  by assumption. □

## 1.8 Quantum Parallelism

On a quantum computer one can evaluate a function  $f(x)$  on different values of  $x$  at the same time, with "just one application". This is a fundamental feature of quantum circuits.

Let us consider a Boolean function of the form:

$$f(x) : \{0, 1\} \rightarrow \{0, 1\}$$

To calculate  $f(x)$  by means of a quantum computation, the transformation  $f(x)$  must be defined as a unitary transformation  $U_f$ . As seen previously, this can be done by applying on the input state  $|x\rangle \otimes |y\rangle = |x, y\rangle$  an appropriate sequence of quantum logic gates (which we will indicate with a "black box" called  $U_f$ ) that transforms  $|x, y\rangle$  into the state  $|x, y \oplus f(x)\rangle$ , called the target

register. If  $y = 0$  then the final state of the second qubit will contain exactly the value of  $f(x)$ .

In the circuit in Figure 1.9, the input is

$$|x\rangle|y\rangle = |x\rangle \otimes |y\rangle \text{ where } x := \frac{|0\rangle + |1\rangle}{\sqrt{2}}, y := |0\rangle$$

so, the value of  $x$  is a superposition of 0 and 1, and  $y = 0$ . Applying  $U_f$  to

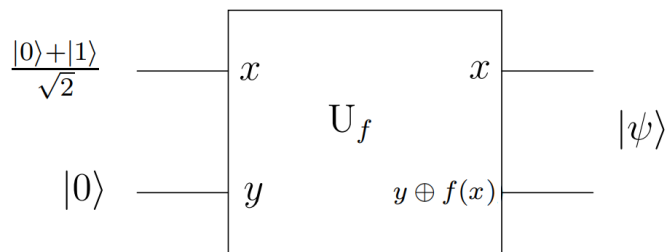


Figure 1.9: Quantum circuit to evaluate  $f(0)$  and  $f(1)$  simultaneously.

this data register we obtain

$$\frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}}$$

This state contains, implicitly, information about both the  $f(0)$  value and the  $f(1)$  value.

We then evaluated  $f$  simultaneously on  $x = 0$  and  $x = 1$ . This effect is called *quantum parallelism*. This type of parallelism is different from the classical parallelism where several circuits (each of which calculates  $f(x)$  for a single value of  $x$ ) are executed simultaneously.

We can generalize this procedure to compute functions on an arbitrary number of bits using a generalization of the Hadamard gate known as the *Walsh-Hadamard* transform. This operation consists of  $n$  Hadamard gates acting in parallel on  $n$  qubits. For example:

*Example 5* (Walsh-Hadamard for  $n=2$ ). The Walsh-Hadamard transformation for  $n = 2$ , is denoted by  $H^{\otimes 2} := H \otimes H$ , and applied to two qubits

prepared in the state  $|0\rangle$  gives as a result:

$$H^{\otimes 2}(|00\rangle) = (H|0\rangle) \otimes (H|0\rangle) = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}} = \frac{|00\rangle + |01\rangle + |10\rangle + |11\rangle}{2}$$

In general, it is easy to verify that the result of  $H^{\otimes n}$  applied to  $n$  qubits in the state  $|0\rangle$  is:

$$H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle$$

where  $x$  can be interpreted as the binary representation of a number from 0 to  $2^n - 1$ . Hence, the Walsh-Hadamard transform, produced an equiprobable superposition of all the states of the computational basis of  $n$  qubits. We observe that to obtain a superposition of  $2^n$  states, only  $n$  logic gates  $H$  are needed.

Any transformation of the form  $U_f = |x, y\rangle \mapsto |x, y \oplus f(x)\rangle$  for a classic Boolean function  $f$  is linear and therefore acts on a superposition  $\sum_x a_x |x\rangle$  of input values as follows:

$$U_f : \sum_x a_x |x, 0\rangle \mapsto \sum_x a_x |x, f(x)\rangle$$

Consider the effect of applying  $U_f$  to the superposition of values from 0 to  $2^n - 1$  obtained from the Walsh-Hadamard transformation:

$$U_f : (H^{\otimes n}|0\rangle)|0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle|0\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle|f(x)\rangle$$

After only one application of  $U_f$ , the superposition now contains all of the  $2^n$  function values  $f(x)$  entangled with their corresponding input value  $x$ . This effect is the general version of the quantum parallelism.

Since  $n$  qubits enable us to work simultaneously with  $2^n$  values, quantum parallelism in some sense circumvents the time/space trade-off of classical parallelism through its ability to hold exponentially many computed values in a linear amount of physical space. However, this effect is less powerful than it may initially appear.

In fact it is possible to gain only limited information from this superposition:

these  $2^n$  values of  $f$  are not independently accessible. We can gain information only from measuring the states, but measuring in the standard basis will project the final state onto a single input/output pair  $|x, f(x)\rangle$ , and a random one at that.

## 1.9 Mathematical structure of quantum mechanics

So far we have talked about quantum systems, quantum states, evolution and measurement of quantum states, etc., but we have not yet defined these terms in a formal way. The mathematical structure at the base of quantum computing is the *quantum theory*. Let us introduce it.

A physical system is generally described by three basic ingredients: its *states*, its *observables* (i.e., all quantities which can be measured about the current state of physical system) and its *dynamics* (the law where by the state of a sistem changes in time).

Quantum mechanics provides the most accurate and complete description of the laws that govern the physical world. The mathematical formalism on which it is based and the physical reality it describes are related by means of some fundamental postulates.

### Postulate 1: States

Each isolated physical system has an associated space (called the state space of the system), that is the projective of a complex Hilbert space. The system is completely described by its state vector, which is a unit vector in the state space.

The state space of a composite physical system is the tensor product space tensor of spaces of the single components. If the system is composed of  $n$  subsystems and the  $i$ -th component is in the state  $|\psi_i\rangle$ , then the state of the

total system is  $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle$ . This describes how to construct the state space of a quantum system composed of two or more distinct physical systems starting from the state spaces of the component systems. In particular, in addition to each separable state (obtained with a vector product of elementary states), there are the superposition of these (the so-called entangled states), fundamental in the field of quantum computation.

The simplest isolated physical system is the qubit. The associated Hilbert space is  $\mathbb{C}^2$ . The computational basis formed by  $|0\rangle$  and  $|1\rangle$  is an orthonormal basis and the condition that every vector  $|\psi\rangle = a|0\rangle + b|1\rangle$  (with  $a, b \in \mathbb{C}$ ) is a unit vector is equivalently expressed by  $|a|^2 + |b|^2 = 1$ , or by means of the inner product from  $\langle\psi|\psi\rangle = 1$  (where  $a$  and  $b$  are called the amplitudes of the vector  $|0\rangle$  and of the vector  $|1\rangle$ , respectively). We also postulate that a state of a quantum system is actually an equivalence class of vectors which differ by multiplication of a complex unitary scalar; the unit vector is the representative of this class. Hence the state  $e^{i\theta}|\psi\rangle$ , where  $|\psi\rangle$  is a state and  $\theta$  is a real number, is equivalent to  $|\psi\rangle$ . The factor  $e^{i\theta}$  is called the *global phase factor*. This identification of states with different global phases is justified by the fact that the measurement statistics (cf. Postulate 3) of these states are the same. For example, we identify  $a|\psi\rangle + b|\psi\rangle$  with  $e^{i\theta}(a|\psi\rangle + b|\psi\rangle)$  but not with  $a|\psi\rangle + be^{i\theta}|\psi\rangle$ .

## Postulate 2: Observables

In quantum mechanics, observables are identified by self-adjoint operators  $A$  on  $H$ . This requirement is due to the fact that these operators are "diagonalizable", with a positive spectrum (spectral theorem). For further information: [65].

More precisely, assuming for simplicity that  $A$  has a discrete spectrum, we know that the eigenvalue equation

$$A|\psi_m\rangle = m|\psi_m\rangle$$

admits only real eigenvalues  $m \in \mathbb{R}$ , and that the corresponding eigenvectors form an orthonormal basis for  $H$ .

We can rephrase these properties in terms of projection operators: if we define  $P_i$  as the projection on the eigenspace relative to the eigenvalue  $m$ , then we find

$$A = \sum_m m P_m.$$

Furthermore, the  $P_m$  operators satisfy the *completeness equation* 1.1:

$$\sum_m P_m = I, \quad (1.1)$$

where  $I$  is the identity operator on  $H$ .

### Postulate 3: (Discrete) Time Evolution

We provide a more simplified version of the most refined version of the third postulate: we are going to describe a closed quantum system, in a *discrete* time evolution (it is sufficient for us, because in quantum computing we study systems only before and after certain computational operations; so for the purposes of our study, the dynamics will be discrete).

The discrete evolution of a closed quantum system is described by a unitary transformation: the state  $|\psi\rangle$  of the system at time  $t_1$  is in relation to the state  $|\psi'\rangle$  of the system at time  $t_2$  by means of a unitary operator  $U$  which depends only on  $t_1$  and  $t_2$ ,

$$|\psi'\rangle = U|\psi\rangle.$$

The version for a closed quantum system in continuous time is slightly more complex: the evolution of a quantum state is described by the famous Schrödinger equation (for more details, see [46, 21]).

In particular, if  $H$  does not depend on time  $t$ , it is possible to reformulate the Schrödinger equation, obtaining that there must exist an evolution operator  $U_t$  (depending on  $t \in \mathbb{R}$ ) so that:

$$\psi(t) = U_t \psi_0,$$

where  $\psi_0$  is the initial state.

**Postulate 4: Measure**

The only possible results of a measurement of the observable  $M$  correspond to the eigenvalues  $m$  of  $M$ .

After a measurement of  $M$  in the state  $|\psi\rangle$ , the probability of obtaining the result  $m$  is:

$$p(m) = \langle\psi|P_m|\psi\rangle = \|P_m|\psi\rangle\|^2.$$

After this measurement, with its results  $m$ , the state collapse immediately in the state:

$$\frac{P_m|\psi\rangle}{\sqrt{p(m)}}.$$

The completeness equation (1.1) expresses the fact that the sum of the probabilities of the results of a measurement must be 1. In fact:

$$1 = \langle\psi|\psi\rangle = \langle\psi|(\sum_m P_m)|\psi\rangle = \sum_m \langle\psi|P_m|\psi\rangle = \sum_m p(m)$$

This type of measurement is often called *complete projective measurement* (or *Von Neumann measurement*) because the observable  $M$  is determined by any set of orthogonal projection operators  $P_m$  which satisfy the completeness relation and such that  $P_m P_{m'} = \delta_{mm'} P_m$ .

Measuring in the orthonormal basis  $|m\rangle$ , means performing a projective measurement with projection operators  $= |m\rangle\langle m|$  (which a convenient notation for  $P_m$ . Notice in fact that  $P_m|\psi\rangle = |m\rangle\langle m|\psi\rangle = \langle m|\psi\rangle|m\rangle$ ).



## Chapter 2

# Shor's algorithm and its Impact on cryptography

In this Chapter we are interested in studying and analyzing a fundamental result of computational quantum mechanics: the Shor's algorithm, conceived by the scientist in 1994. As already mentioned, this algorithm has provided a boost in the interest of quantum computers and, in general, quantum computing. This is because, through the use of this algorithm (on a quantum computer), most of the cryptographic systems used on a large scale today (such as the famous public-key cryptosystem RSA) are "broken", in the sense that they are solved in acceptable times. In fact, the Shor's algorithm allows to solve some classical computational problems, considered difficult to be solved having only classical computers available, such as, for example, the "*factoring problem*" (see Definition 5: the security of RSA relies on this problem) in polynomial time. The impact of Shor's algorithm on cryptography, and of quantum-computing in general, is deepened in Section 2.3.

Before starting to study Shor's algorithm, let us start by describing, in the next Section, the famous RSA algorithm, that Shor would break. This is useful for getting an idea of how public key cryptography works and how a cryptographic algorithm exploits "hard" math problems.

## 2.1 RSA public-key cryptosystem

Suppose there are two individuals, Alice and BoB, who want to exchange a secret message, but also want to exchange it encrypted, because it is potentially exposed on an insecure transmission channel. This encrypted message must be untranslatable by a potential attacker, but must be decryptable for the two individuals. One way to do this is to use a system based on the public-key cryptography.

Public-key cryptography, or asymmetric cryptography, is a cryptographic system that uses pairs of keys. Each pair consists of a public key (which may be known to others) and a private key (which may not be known by anyone except the owner). The generation of such key pairs depends on cryptographic algorithms which are based on mathematical problems termed one-way functions. Effective security requires keeping the private key private; the public key can be openly distributed without compromising security.

In such a system, any person can encrypt a message using the intended receiver's public key, but that encrypted message can only be decrypted with the receiver's private key.

This scheme has the advantage of not having to manually pre-share symmetric keys (a fundamentally difficult problem) while gaining the higher data throughput advantage of symmetric-key cryptography.

Public key algorithms are fundamental security primitives in modern cryptosystems, including applications and protocols which offer assurance of the confidentiality, authenticity and non-repudiability of electronic communications and data storage.

RSA is an important public-key cryptosystem that is widely used for secure data transmission. It is also one of the oldest. The word "RSA" is an acronym, and it comes from the surnames of Ron Rivest, Adi Shamir and Leonard Adleman, who publicly described the algorithm in 1977.

An RSA user creates and publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers are kept secret. Messages can be encrypted by anyone, via the public key, but can only be

decoded by someone who knows the prime numbers.

The security of RSA relies on the practical difficulty of factoring the product of two large prime numbers, the "factoring problem":

**Definition 5** (Factoring problem). Given  $n \in \mathbb{N}$ , such that there exist two large prime numbers  $p, q \in \mathbb{N}$ , for which  $n = p \cdot q$ , the problem is to find these two primes, given only  $n$ .

Breaking RSA encryption is known as the RSA problem. There are no published methods to defeat the system if a large enough key is used.

The RSA algorithm involves four steps: *key generation*, *key distribution*, *encryption*, and *decryption*.

A basic principle behind RSA is the observation that it is practical to find three very large positive integers  $e$ ,  $d$ , and  $n$ , such that with modular exponentiation for all integers  $m$  (with  $0 \leq m < n$ ):

$$(m^e)^d = m \pmod{n}$$

and that knowing  $e$  and  $n$ , or even  $m$ , it can be extremely difficult to find  $d$ . RSA involves a public key and a private key. The public key can be known by everyone and is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time by using the private key. The public key is represented by the integers  $n$  and  $e$ , and the private key by the integer  $d$  (although  $n$  is also used during the decryption process, so it might be considered to be a part of the private key too).  $m$  represents the message (previously prepared with a certain technique explained below).

## KEY GENERATION

The keys for the RSA algorithm are generated in the following way:

1. Choose two distinct (large) prime numbers  $p$  and  $q$ .

- For security purposes, the integers  $p$  and  $q$  should be chosen at random and should be similar in magnitude but differ in length by a few digits to make factoring harder. Prime integers can be efficiently found using a primality test
  - $p$  and  $q$  are kept secret.
2. Compute  $n = pq$ .
    - $n$  is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
    - $n$  is released as part of the public key.
  3. Compute  $\lambda(n)$ , where  $\lambda$  is Carmichael's totient function<sup>1</sup>. Since  $n = pq$ ,  $\lambda(n) = \text{lcm}(\lambda(p), \lambda(q))$  (where "lcm" is the least common multiple), and since  $p$  and  $q$  are prime,  $\lambda(p) = \phi(p) = p - 1$  (where  $\phi$  is Euler's totient function<sup>2</sup>), and likewise  $\lambda(q) = q - 1$ . Hence  $\lambda(n) = \text{lcm}(p - 1, q - 1)$ .
    - $\lambda(n)$  is kept secret.
    - The lcm may be calculated through the Euclidean algorithm, since  $\text{lcm}(a, b) = |ab|/\text{GCD}(a, b)$ , where GCD stands for the greatest common divisor.
  4. Choose an integer  $e$  such that  $1 < e < \lambda(n)$  and  $\text{GCD}(e, \lambda(n)) = 1$ ; that is,  $e$  and  $\lambda(n)$  are coprime.
    - $e$  having a short bit-length and small Hamming weight<sup>3</sup> results in more efficient encryption - the most commonly chosen value for

---

<sup>1</sup>In number theory, a branch of mathematics, the Carmichael function  $\lambda(n)$  of a positive integer  $n$  is the smallest positive integer  $m$  such that  $a^m = 1 \pmod{n}$

<sup>2</sup>In number theory, Euler's totient function counts the positive integers up to a given integer  $n$  that are relatively prime to  $n$ . It is also called Euler's phi function. In other words, it is the number of integers  $k$  in the range  $1 \leq k \leq n$  for which the greatest common divisor  $\text{GCD}(n, k)$  is equal to 1.

<sup>3</sup>The Hamming weight of a string is the number of symbols that are different from the zero-symbol of the alphabet used. It is thus equivalent to the Hamming distance from the all-zero string of the same length. For the most typical case, a string of bits, this is

$e$  is  $216 + 1 = 65537$ . The smallest (and fastest) possible value for  $e$  is 3, but such a small value for  $e$  has been shown to be less secure in some settings.

- $e$  is released as part of the public key.
5. Determine  $d$  as  $d = e^{-1} \pmod{\lambda(n)}$ ; that is,  $d$  is the modular multiplicative inverse of  $e$  modulo  $\lambda(n)$ .

- This means: solve for  $d$  the equation  $d \cdot e = 1 \pmod{\lambda(n)}$ ;  $d$  can be computed efficiently by using the extended Euclidean algorithm<sup>4</sup>, since, thanks to  $e$  and  $\lambda(n)$  being coprime, said equation is a form of Bézout's identity, where  $d$  is one of the coefficients.
- $d$  is kept secret as the private key exponent.

Finally, the public key consists of the modulus  $n$  and the public (or encryption) exponent  $e$ .

The private key consists of the private (or decryption) exponent  $d$ , which must be kept secret.  $p$ ,  $q$ , and  $\lambda(n)$  must also be kept secret because they can be used to calculate  $d$ . In fact, they can all be discarded after  $d$  has been computed.

## KEY DISTRIBUTION

Suppose that Bob wants to send information to Alice. If they decide to use RSA, Bob must know Alice's public key to encrypt the message, and Alice must use her private key to decrypt the message.

---

the number of 1's in the string, or the digit sum of the binary representation of a given number and the  $l_1$  norm of a bit vector. In this binary case, it is also called the population count, popcount, sideways sum, or bit summation.

<sup>4</sup>In arithmetic and computer programming, the extended Euclidean algorithm is an extension to the Euclidean algorithm, and computes, in addition to the greatest common divisor of integers  $a$  and  $b$ , also the coefficients of Bézout's identity, which are integers  $x$  and  $y$  such that:  $ax + by = \text{GCD}(a, b)$ .

To enable Bob to send his encrypted messages, Alice transmits her public key  $(n, e)$  to Bob via a reliable, but not necessarily secret, route. Alice's private key  $(d)$  is never distributed.

### ENCRYPTION

After Bob obtains Alice's public key, he can send a message  $M$  to Alice.

To do it, he first turns  $M$  (strictly speaking, the un-padded plaintext) into an integer  $m$  (strictly speaking, the padded plaintext), such that  $0 \leq m < n$  by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext  $c$ , using Alice's public key  $e$ , corresponding to

$$c = m^e \pmod{n}$$

This can be done reasonably quickly, even for very large numbers, using modular exponentiation. Bob then transmits  $c$  to Alice.

### DECRYPTION

Alice can recover  $m$  from  $c$  by using her private key exponent  $d$  by computing

$$c^d = (m^e)^d = m \pmod{n}.$$

Given  $m$ , she can recover the original message  $M$  by reversing the padding scheme.

### Using RSA: an example

Here is an example of RSA encryption and decryption. The parameters used here are artificially small.

1. Choose two distinct prime numbers:  $p = 61, q = 53$ .
2. Compute  $n = pq = 3233$ .
3. Compute the Carmichael's totient function of the product as  $\lambda(n) = \text{lcm}(p - 1, q - 1)$  giving  $\lambda(3233) = 780$ .

4. Choose any number  $1 < e < 780$  that is coprime to 780. Choosing a prime number for  $e$  leaves us only to check that  $e$  is not a divisor of 780. Let  $e = 17$ .
5. Compute  $d$ , the modular multiplicative inverse of  $e \pmod{\lambda(n)}$ , yielding  $d = 413$ , as  $1 = (17 \times 413) \pmod{780}$ .

The public key is  $(n = 3233, e = 17)$ . For a padded plaintext message  $m$ , the encryption function is:

$$c(m) = m^e \pmod{n} = m^{17} \pmod{3233}.$$

The private key is  $(n = 3233, d = 413)$ . For an encrypted ciphertext  $c$ , the decryption function is:

$$m(c) = c^d \pmod{n} = c^{413} \pmod{3233}.$$

For instance, in order to encrypt  $m = 65$ , we calculate

$$c = 65^{17} \pmod{3233} = 2790.$$

To decrypt  $c = 2790$ , we calculate

$$m = 2790^{413} \pmod{3233} = 65.$$

Both of these calculations can be computed efficiently using the square-and-multiply algorithm for modular exponentiation. In real-life situations the primes selected would be much larger; in our example it would be trivial to factor  $n = 3233$  (obtained from the freely available public key) back to the primes  $p$  and  $q$ .  $e$ , also from the public key, is then inverted to get  $d$ , thus acquiring the private key.

Practical implementations use the Chinese remainder theorem to speed up the calculation using modulus of factors (for example, see [48]).

## 2.2 Shor's algorithm

Let us describe the famous Shor's quantum algorithm for finding the prime factors of a composite number  $N$ , in polynomial time, undermining the security of much of today's classic cryptography.

Think of a large number, such as one with 300 digits in decimal notation (since such numbers are used in cryptography). Despite  $N$  is large, the number of qubits necessary to store it is relatively small. Considering that in general  $\log_2 N$  is not an integer, let us define the number of qubits necessary to store  $N$ , as:

$$n = \lceil \log_2 N \rceil$$

A quantum computer with  $n$  qubits can store  $N$  or any other smaller positive integer. Obviously, the number of prime factors of  $N$  is at most  $n$  ( $N = 2^n$ ). In the next paragraph we see a problem whose resolution implies the resolution of the factoring problem. Shor's algorithm solves factoring problem using this observation.

### 2.2.1 Reduction of factorization to order calculation

In this Subsection, we see that, to solve the problem of factoring  $N$  (we can suppose that  $N$  is odd: if  $N$  is even, we can keep dividing by 2 until the result turns out to be odd), we can reduce it to finding the order of a random integer  $x$  less than  $N$ . The order of  $x$  modulo  $N$  is defined as the least positive integer  $r$  such that:

$$x^r \equiv 1 \pmod{N}.$$

Taking  $x < N$ , obviously if  $x$  and  $N$  have common factors, then  $\text{GCD}(x, N)$  (where "GCD" stands for the greatest common divisor) gives a factor of  $N$ . In summary, if we pick up at random a positive integer  $x$  less than  $N$  and calculate  $\text{GCD}(x, N)$ , either we have a factor of  $N$  or we learn that  $x$  is coprime to  $N$ .



In particular, we are going to show that, in the latter case, if  $x$  satisfies the conditions:

- (i) its order  $r$  is even
- (ii)  $x^{r/2} - 1 \not\equiv 0 \pmod{N}$ ,  $x^{r/2} + 1 \not\equiv 0 \pmod{N}$

then  $\text{GCD}(x^{r/2} - 1, N)$  and  $\text{GCD}(x^{r/2} + 1, N)$  yield factors of  $N$  we were looking for.

If one of the conditions is not true, we start over until finding a proper candidate  $x$ .

The method would not be useful if these two assumptions were too restrictive, but fortunately that is not the case.

So let us take a random  $x < N$  coprime with  $N$ , and let us suppose (i) and (ii); since  $r$  is even, we can define  $y$  by

$$x^{r/2} \equiv y \pmod{N}.$$

Note that  $y$  satisfies  $y^2 \equiv 1 \pmod{N}$ , or equivalently  $(y-1)(y+1) \equiv 0 \pmod{N}$ , which means that  $N$  divides  $(y-1)(y+1)$ .

Since (ii) holds, the factors  $y-1$  and  $y+1$  satisfy

$$0 < y-1 < y+1 < N.$$

So  $N$  cannot divide  $y-1$  nor  $y+1$  "separately", but both  $y-1$  and  $y+1$  have factors of  $N$ . In other words, we found that in this case  $\text{GCD}(y-1, N)$  and  $\text{GCD}(y+1, N)$  yield non trivial factors of  $N$ .

If  $N$  has other factors besides  $\text{GCD}(y-1, N)$  and  $\text{GCD}(y+1, N)$ , they can be calculated applying the algorithm recursively.

*Example 6* ( $N=21$ ). Consider  $N = 21$  as an example. The next sequence of modulo  $N$  equivalences show that the order of  $x = 2$  modulo 21 is  $r = 6$ :

$$2^4 \equiv 16 \pmod{21}$$

$$2^5 \equiv 11 \pmod{21}$$

$$2^6 = 11 \times 2 \equiv 1 \pmod{21}$$

Therefore,  $y \equiv 2^3 \equiv 8$  modulo 21.  $\text{GCD}(y - 1, N)$  yields the factor 7 and  $\text{GCD}(y + 1, N)$  yields the factor 3 of 21.

Actually, this method systematically fails if  $N$  is a power of an odd prime, but an alternative efficient classical algorithm for this particular case is known.

It remains to apply the method for odd composite integers that are not a power of some prime number.

It is possible to prove that the probability of finding  $x$  coprime to  $N$  satisfying the conditions (i) and (ii) is high; in fact the following Proposition holds:

**Proposition 3** (High probability of picking  $x$  with good proprieties). *If  $N$  is odd composite integer that is not a power of a prime number, the probability of finding  $x < N$  such that satisfies the conditions (i) and (ii) is:*

$$1 - \frac{1}{2^{k-1}}$$

where  $k$  is the number of prime factors of  $N$ .

In the worst case ( $N$  has 2 factors), the probability is greater than or equal to  $1/2$ .

The prof of the Proposition is cumbersome. The more curious readers can read it in Appendix B of [26].

At first sight, it seems that we have just described an efficient classical algorithm to find a factor of  $N$ .

That is not true, since an efficient classical algorithm to calculate the order of an integer  $x$  modulo  $N$  is not known. On the other hand, there is (after Shor's work) an efficient *quantum* algorithm. Let us describe it.

### 2.2.2 The quantum algorithm to calculate the order: a particular case

Consider the circuit of Fig. 2.1. It calculates the order  $r$  of the positive

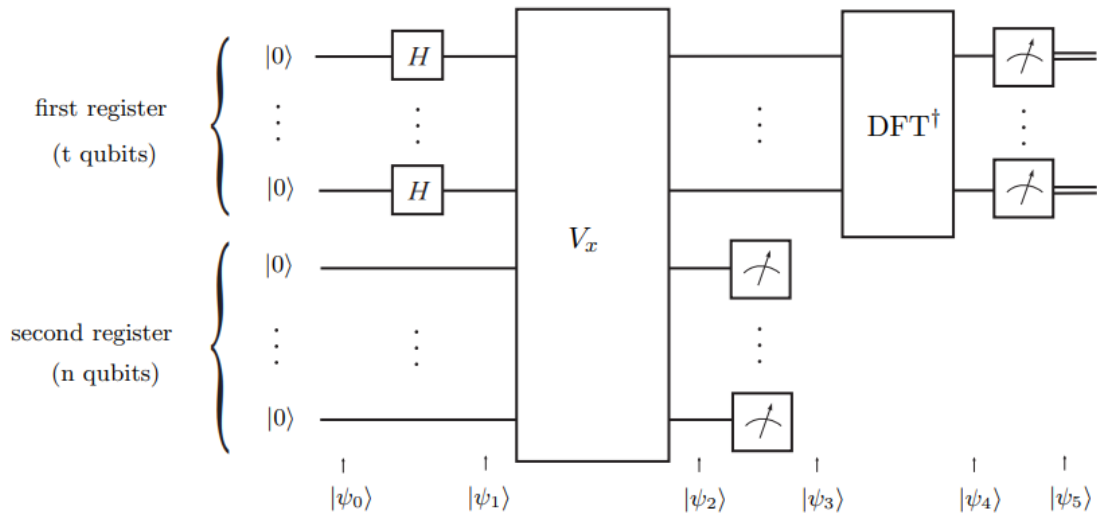


Figure 2.1: Quantum circuit for finding the order of the positive integer  $x$  modulo  $N$ .

integer  $x$  less than  $N$ , coprime to  $N$ .  $V_x$  is the unitary linear operator

$$V_x(|j\rangle|k\rangle) = |j\rangle|k + x^j\rangle$$

where  $|j\rangle$  and  $|k\rangle$  are the states of the first and second registers, respectively. The arithmetical operations are performed modulo  $N$ , so  $0 \leq k + x^j < N$ .  $DFT$  is the Discrete Fourier Transform operator which will be described in Subsection 2.2.3.

The first register has  $t$  qubits, where  $t$  is generally chosen such that

$$N^2 \leq 2^t < 2N^2.$$

As an exception, if the order  $r$  is a power of 2, then it is enough to take  $t = n$ . In this subsection we consider this very special case and leave the general case for next Subsection 2.2.4. We will keep the variable  $t$  in order to generalize the discussion later on.

The states of the quantum computer are indicated by  $|\psi_0\rangle, \dots, |\psi_5\rangle$  in Fig.

2.1. The initial state is:

$$|\psi_0\rangle = \underbrace{|0\dots 0\rangle}_t \underbrace{|0\dots 0\rangle}_n.$$

The application of the Hadamard operator  $H$  on each qubit of the first register yields

$$|\psi_1\rangle = \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|0\rangle. \quad (2.1)$$

The first register is then in a superposition of all states of the computational basis with equal amplitudes given by  $\frac{1}{\sqrt{2^t}}$ . When we apply  $V_x$  to  $|\psi_1\rangle$ :

$$\begin{aligned} |\psi_2\rangle &= V_x |\psi_1\rangle \\ &= \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} V_x(|j\rangle|0\rangle) \\ &= \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|x^j\rangle. \end{aligned}$$

The state  $|\psi_2\rangle$  is a remarkable one. Because  $V_x$  is linear, it acts on all for  $2^t$  values of  $j$ , so this generates all powers of  $x$  simultaneously (quantum parallelism). Some of these powers are 1, which correspond to the states

$$|0\rangle|1\rangle, |r\rangle|1\rangle, |2r\rangle|1\rangle, \dots, \left|\left(\frac{2^t}{r} - 2\right)r\right\rangle|1\rangle, \left|\left(\frac{2^t}{r} - 1\right)r\right\rangle|1\rangle. \quad (2.2)$$

This explains the choice for  $V_x$ .

At the quantum level, the values of  $j$  that yield  $x^j \equiv 1$  modulo  $N$  are “known”. But this quantum information is not fully available at the classical level: a classical information of a quantum state is obtained by practical measurements and, at this point, it does not help if we measure the first register, since all states in the  $|\psi_2\rangle$  superposition have equal amplitudes.

The first part of the strategy to find  $r$  is to observe that the first register of the states 2.2 is periodic. So the information we want is a period.

In order to simplify the calculation, let us measure the second register. Before doing this, we will rewrite  $|\psi_2\rangle$  collecting equal terms in the second register. Since  $x^j$  is a periodic function with period  $r$ , substitute  $ar + b$  for  $j$  in  $|\psi_2\rangle$  superposition, where  $0 \leq a \leq (2^t/r) - 1$  and  $0 \leq b \leq r - 1$ . Recall that we are supposing that  $t = n$  and  $r$  is a power of 2, therefore  $r$  divides  $2^t$ . So  $|\psi_2\rangle$  is rewritten as:

$$|\psi_2\rangle = \frac{1}{\sqrt{2^t}} \sum_{b=0}^{r-1} \left( \sum_{a=0}^{\frac{2^t}{r}-1} |ar + b\rangle \right) |x^b\rangle. \quad (2.3)$$

In the second register, we have substituted  $x^b$  for  $x^{ar+b}$ , since  $x^r \equiv 1$  modulo  $N$ . Now the second register is measured. Any output  $x^0, x^1, \dots, x^{r-1}$  can be obtained with equal probability. Suppose that the result is  $x^{b_0}$ .

The state of the quantum computer is now collapsed in:

$$|\psi_3\rangle = \sqrt{\frac{r}{2^t}} \left( \sum_{a=0}^{\frac{2^t}{r}-1} |ar + b_0\rangle \right) |x^{b_0}\rangle. \quad (2.4)$$

Note that after the measurement, the constant before the sum is renormalized, since there are  $2^t/r$  terms in the sum.

Fig. 2.2 shows the probability of obtaining the states of the computational basis upon measuring the first register.

The probabilities form a periodic function with period  $r$ . Their values are zero except for the states

$$|b_0\rangle, |r + b_0\rangle, |2r + b_0\rangle, \dots, |2^t - r + b_0\rangle.$$

How can one find out the period of a function efficiently? The answer is in the Fourier transform.

The Fourier transform of a periodic function with period  $r$  is a new periodic function with period proportional to  $1/r$ . This makes a difference for finding  $r$ .

The Fourier transform is the second part of the strategy. The whole method

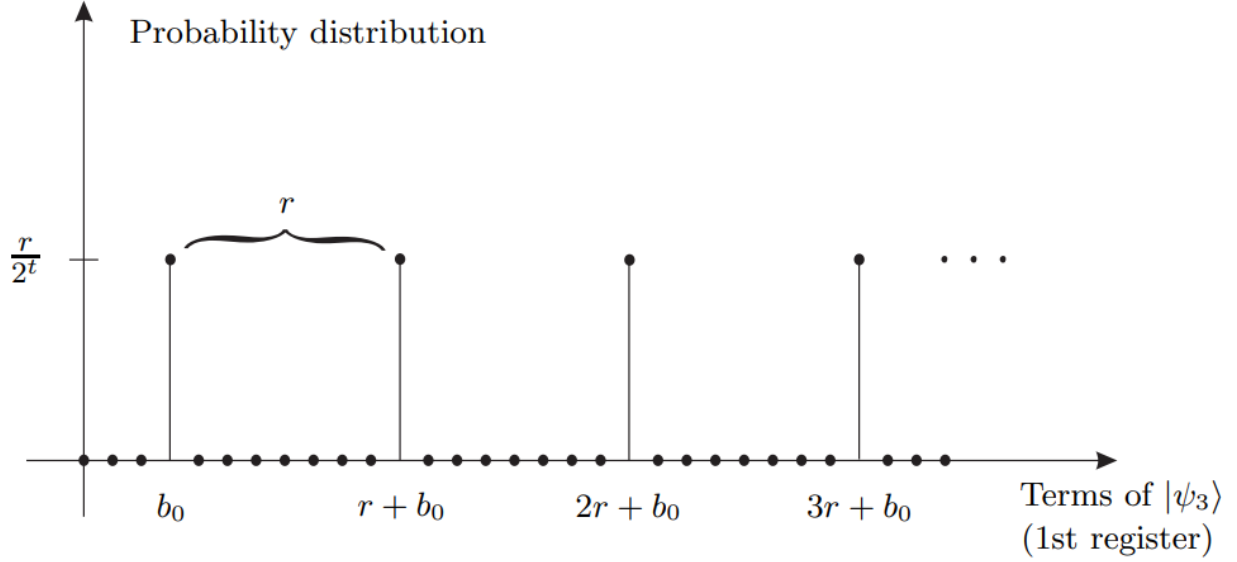


Figure 2.2: Probability distribution of  $|\psi_3\rangle$  measured in the computational basis (for the case  $b_0 = 3$  and  $r = 8$ ). The horizontal axis has  $2^t$  points. The number of peaks is  $2^t/r$  and the period is  $r$ .

relies on an efficient quantum algorithm for calculating the Fourier transform, which is not available classically. We will not analyze it in this work in order not to be too dispersive in the explanation of Shor's algorithm, but it is possible to read it for example in Chapter 8 of [35].

### 2.2.3 The quantum discrete Fourier transform

**Definition 6 (DFT).** The Discrete Fourier transform (*DFT*) of the function  $F : \{0, \dots, N - 1\} \rightarrow \mathbb{C}$  is a new function  $\tilde{F} : \{0, \dots, N - 1\}$  defined as:

$$\tilde{F}(k) = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i j k / N} F(j). \quad (2.5)$$

We can apply the Fourier transform either on a function or on the states of the computational basis. The Fourier transform applied to the state  $|k\rangle$  of

the computational basis  $|0\rangle, \dots, |N-1\rangle$  is:

$$DFT(|k\rangle) = |\psi_k\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i j k / N} F|j\rangle. \quad (2.6)$$

The Fourier transform is a unitary linear operator. So, if we know how it acts on the states of the computational basis, we also know how it acts on a generic state:

$$|\psi\rangle = \sum_{a=0}^{N-1} F(a)|a\rangle.$$

**Proposition 4.** *The set  $\{DFT(|0\rangle), \dots, DFT(|N-1\rangle)\} = \{|\psi_0\rangle, \dots, |\psi_{N-1}\rangle\}$  from 2.6, forms a new orthonormal basis. In other words, the Fourier transform is a unitary operator.*

The Proof of orthonormality in Proposition 4 is easy, let us see it. Our goal is to show that:

$$\langle \psi_{k'} | \psi_k \rangle$$

where  $k, k' \in \{0, \dots, N-1\}$ ; we can use the identity:

$$\frac{1}{N} \sum_{j=0}^{N-1} e^{2\pi i j (K)/N} = \begin{cases} 1 & \text{if } K \text{ is a multiple of } N \\ 0 & \text{otherwise} \end{cases} \quad (2.7)$$

which is useful in the context of Fourier transforms. So:

$$\langle \psi_{k'} | \psi_k \rangle = \frac{1}{N} \sum_{j=0}^{N-1} e^{2\pi i j (k-k')/N} = \begin{cases} 1 & \text{if } k - k' \text{ is a multiple of } N \\ 0 & \text{otherwise} \end{cases} \quad (2.8)$$

Note that the only the only way for  $k' - k$  to be a multiple of  $N$  is for  $k = k'$ . It is easy to verify that the identity (2.7) is true. If  $K$  is a multiple of  $N$ , then each addendum  $e^{2\pi i j K / N} = 1$  and the first case of the identity follows. If  $K$  is not a multiple of  $N$ , (2.7) is true even if  $N$  is not a power of 2; to better understand why, let us look at the Fig. 2.3. This figure shows each term  $e^{2\pi i j K / N}$  ( $j = 0, \dots, 6$ ) for the case  $K = 1$  and  $N = 7$ , as vectors in the complex plane. Note that the sum of vectors must be zero by a symmetry

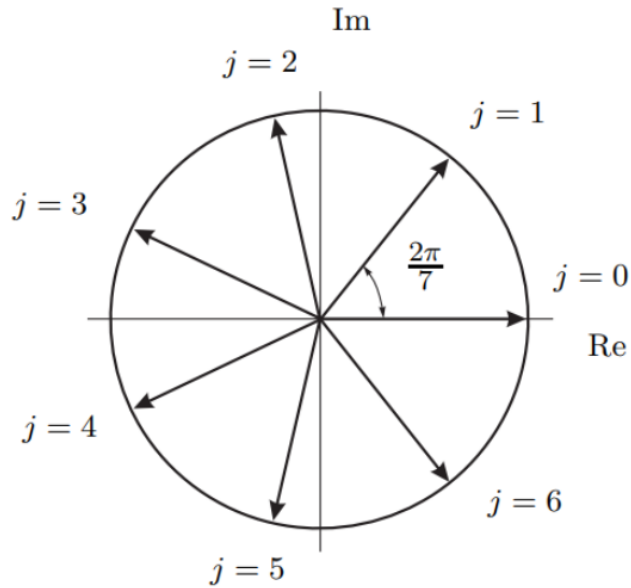


Figure 2.3: Vectors  $e^{2\pi ij/7}$ , ( $j = 0, \dots, 6$ ) in the complex plane. Their sum is zero by symmetry arguments. This is an example of Eq. (2.7) for  $N = 7$ ,  $k = 1$ .

argument: the distribution of vectors is isotropic.

Usually it is said that the interference is destructive in this case.

Using the identity (2.7), we can define the inverse Fourier transform, which is similar to that in Definition (6), just with a minus sign on the exponent.

Note that  $DFT^{-1} = DFT^\dagger$ , since  $DFT$  is a unitary operator.

For the details of a quantum circuit to perform efficiently the Fourier transform, as mentioned above, see Chapter 8 of [35].

Now we will continue the calculation process of the circuit of Fig. 2.1, left at  $|\psi_3\rangle$ : we are ready to find out the next state of the quantum computer  $|\psi_4\rangle$ .

Applying the inverse Fourier transform only on the first register, using the



definition of  $DFT$  and the linearity of  $DFT^\dagger$ , we obtain:

$$\begin{aligned} |\psi_4\rangle &= DFT^\dagger(|\psi_3\rangle) = DFT^{-1}(|\psi_3\rangle) = \\ &= \sqrt{\frac{r}{2^t}} \sum_{a=0}^{\frac{2^t}{r}-1} \left( \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} e^{-2\pi i j(ar+b_0)/2^t} |j\rangle \right) |x^{b_0}\rangle. \end{aligned}$$

Inverting the summation order, we have

$$|\psi_4\rangle = \frac{1}{\sqrt{r}} \left( \sum_{j=0}^{2^t-1} \left[ \frac{1}{2^t/r} \sum_{a=0}^{\frac{2^t}{r}-1} e^{\frac{-2\pi i j a}{2^t/r}} \right] e^{-2\pi i j b_0/2^t} |j\rangle \right) |x^{b_0}\rangle. \quad (2.9)$$

Using identity (2.7), we see that the expression in square brackets is not zero if and only if  $j$  is a multiple of  $2^t/r$ , and so if and only if  $j = k2^t/r$  with  $k = 0, \dots, r-1$ .

When  $j$  takes such values, the expression in the square brackets is equal to 1. So, removing the  $j$  rewritten then in terms of  $k$ , and re-indexing the first sum in  $k$ , we have:

$$|\psi_4\rangle = \frac{1}{\sqrt{r}} \left( \sum_{k=0}^{r-1} e^{-2\pi i \frac{k}{r} b_0} \left| \frac{k2^t}{r} \right\rangle \right) |x^{b_0}\rangle \quad (2.10)$$

In order to find  $r$ , the expression for  $|\psi_4\rangle$  has two advantages over the expression for  $|\psi_3\rangle$  (Eq. (2.4)):  $r$  is in the denominator of the ket label and the random parameter  $b_0$  moved from the ket label to the exponent occupying now a harmless place.

Fig. 2.4 shows the probability distribution of  $|\psi_4\rangle$  measured in the computational basis.

Measuring the first register, we get the value  $k_0 2^t/r$ , where  $k_0$  can be any number between 0 and  $r-1$  with equal probability (the peaks in Fig. 2.4). If we obtain  $k_0 = 0$ , we have no clue at all about  $r$ , and the algorithm must be run again.

If  $k_0 \neq 0$ , we divide  $k_0 2^t/r$  by  $2^t$ , obtaining  $k_0/r$ . Neither  $k_0$  nor  $r$  are known. If  $k_0$  is coprime to  $r$ , we simply select the denominator.

If  $k_0$  and  $r$  have a common factor, the denominator of the reduced fraction

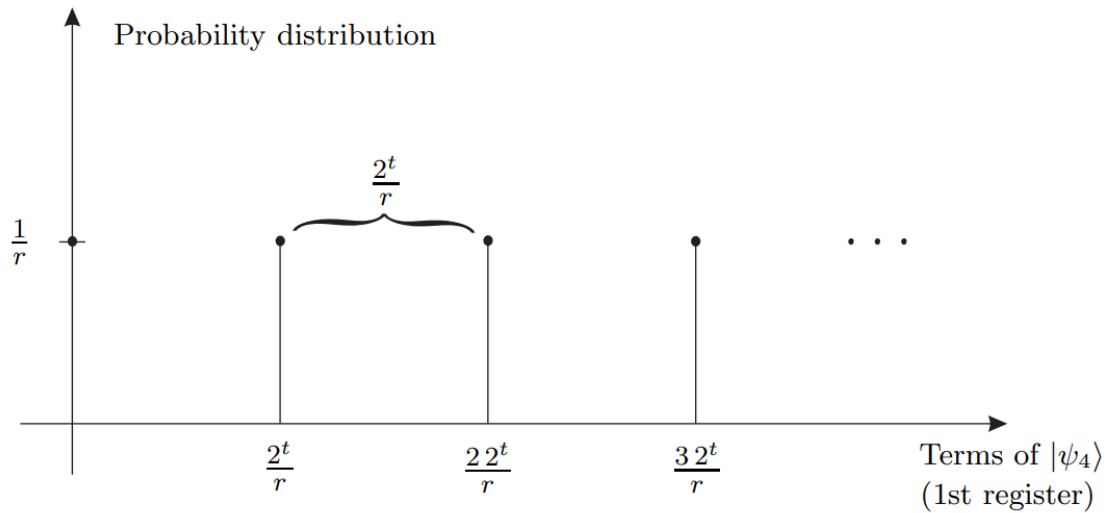


Figure 2.4: Probability distribution of  $|\psi_4\rangle$  measured in the computational basis. The horizontal axis has  $2^t$  points, only the non-null terms are shown. The number of peaks is  $r$  and the period is  $2^t/r$ .

$k_0/r$  is a factor of  $r$  but not  $r$  itself.

Suppose that the denominator is  $r_1$ . Let  $r = r_1 r_2$ . Now the goal is to find  $r_2$ , which is the order of  $x^{r_1}$ .

We run again the quantum part of the algorithm to find the order of  $x^{r_1}$ . If we find  $r_2$  in the first round, the algorithm halts, otherwise we apply it recursively. The recursive process does not last, because the number of iterations is less than or equal to  $\log_2 r$ .

*Example 7.* Take  $N = 15$  as an example, which is the least nontrivial composite number.

The set of numbers less than 15, coprime to 15 is  $\{1, 2, 4, 7, 8, 11, 13, 14\}$ . The numbers in the set  $\{4, 11, 14\}$  have order 2 and the numbers in the set  $\{2, 7, 8, 13\}$  have order 4. Therefore, in any case  $r$  is a power of 2 and the factors of  $N = 15$  can be found in a 8-bit quantum computer (because  $t + n = 2\lceil \log_2 15 \rceil = 8$ ).

### 2.2.4 The most general case

In the previous subsections, we have considered a special case when the order  $r$  is a power of 2 and  $t = n$  ( $t$  is the number of qubits in the first register—see Fig. 2.1—and  $n = \lceil \log_2 N \rceil$ ).

We present the most general case by analyzing it in an example, in order to better follow the discussion. So in this section, we consider the factorization of  $N = 21$ , that is the next nontrivial composite number.

We must choose  $t$  such that  $2^t$  is between  $N^2$  and  $2N^2$ , which is always possible [54]. For  $N = 21$ , the smallest value of  $t$  is 9. This is the simplest example allowed by the constraints, but enough to display all properties of Shor's algorithm. The first step is to pick up  $x$  at random such that  $1 < x < N$ , and to test whether  $x$  is coprime to  $N$ . If not, we easily find a factor of  $N$  by calculating  $\text{GCD}(x, N)$ . If yes, the quantum part of the algorithm starts. Suppose that  $x = 2$  has been chosen. The goal is to find out that the order of  $x$  is  $r = 6$ . The quantum computer is initialized in the state

$$|\psi_0\rangle = |0\rangle|0\rangle,$$

where the first register has  $t = 9$  qubits and the second has  $n = 5$  qubits. Next step is the application of  $H^{\otimes 9}$  on the first register yielding (see Eq. (2.1))

$$|\psi_1\rangle = \frac{1}{\sqrt{512}} \sum_{j=0}^{511} |j\rangle|0\rangle.$$

The next step is the application of  $V_x$  (defined in the previous Subsection), which yields:

$$\begin{aligned} |\psi_2\rangle &= \frac{1}{\sqrt{512}} \sum_{j=0}^{511} |j\rangle|2^j \bmod N\rangle \\ &= \frac{1}{\sqrt{512}} \left( |0\rangle|1\rangle + |1\rangle|2\rangle + |2\rangle|4\rangle + |3\rangle|8\rangle + |4\rangle|16\rangle + |5\rangle|11\rangle + \right. \\ &\quad |6\rangle|1\rangle + |7\rangle|2\rangle + |8\rangle|4\rangle + |9\rangle|8\rangle + |10\rangle|16\rangle + |11\rangle|11\rangle + \\ &\quad \left. |12\rangle|1\rangle + \dots \right) \end{aligned}$$

Notice that the above expression has the following pattern: the states of the second register of each “column” are the same. Therefore we can rearrange the terms in order to collect the second register:

$$\begin{aligned}
 |\psi_2\rangle = \frac{1}{\sqrt{512}} & \left[ (|0\rangle + |6\rangle + |12\rangle + \cdots + |504\rangle + |510\rangle)|1\rangle + \right. \\
 & (|1\rangle + |7\rangle + |13\rangle + \cdots + |505\rangle + |511\rangle)|2\rangle + \\
 & (|2\rangle + |8\rangle + |14\rangle + \cdots + |506\rangle)|4\rangle + \\
 & (|3\rangle + |9\rangle + |15\rangle + \cdots + |507\rangle)|8\rangle + \\
 & (|4\rangle + |10\rangle + |16\rangle + \cdots + |508\rangle)|16\rangle + \\
 & \left. (|5\rangle + |11\rangle + |17\rangle + \cdots + |509\rangle)|11\rangle \right]. \tag{2.11}
 \end{aligned}$$

This feature was made explicit in Eq. (2.3). Because the order is not a power of 2, here there is a small difference: the first two lines of Eq. (2.11) have 86 terms, while the remaining ones have only 85.

Now one measures the second register, yielding one of the following numbers equiprobably:  $\{1, 2, 4, 8, 16, 11\}$ . Suppose that the result of the measurement is 2, then

$$|\psi_3\rangle = \frac{1}{\sqrt{86}} (|1\rangle + |7\rangle + |13\rangle + \cdots + |505\rangle + |511\rangle)|2\rangle \tag{2.12}$$

Notice that the state  $|\psi_3\rangle$  was renormalized in order to have unit norm. It does not matter what is the result of the measurement; what matters is the periodic pattern of (2.12) .

The period of the states of the first register is the solution to the problem and the Fourier transform can reveal the value of the period. So, the next step is the application of the inverse Fourier transform on the first register

of  $|\psi_3\rangle$ :

$$\begin{aligned}
|\psi_3\rangle &= DFT^\dagger(|\psi_3\rangle) \\
&= DFT^\dagger\left(\frac{1}{\sqrt{86}}\sum_{a=0}^{85}|6a+1\rangle\right)|2\rangle \\
&= \frac{1}{\sqrt{512}}\sum_{j=0}^{511}\left(\left[\frac{1}{\sqrt{86}}\sum_{a=0}^{85}e^{-2\pi i\frac{6ja}{512}}\right]e^{-2\pi i\frac{j}{512}}|j\rangle\right)|2\rangle, \quad (2.13)
\end{aligned}$$

where we have used Eq.(2.6) and have rearranged the sums. The last equation is similar to Eq. (2.9), but with an important difference. In previous Subsection, we were assuming that  $r$  divides  $2^t$ . This is not true in the present example (6 does not divide 512), therefore we cannot use the identity (2.7) to simplify the term in brackets in Eq. (2.13). This term never vanishes, but its main contribution is still around  $j = 0, 85, 171, 256, 341, 427$ , which are obtained rounding  $\frac{512k_0}{6}$  for  $k_0$  from 0 to 5; compare to the discussion that follows Eq. (2.10).

To see this, let us plot the probability of getting the result  $j$  (in the interval 0 to 511) by measuring the first register of the state  $|\psi_4\rangle$ . From (2.13), we have that the probability is:

$$Prob(j) = \frac{1}{\sqrt{512 \times 86}} \left| \sum_{a=0}^{85} e^{-2\pi i\frac{6ja}{512}} \right|^2. \quad (2.14)$$

The plot of  $Prob(j)$  is shown in Fig. 2.5. We see the peaks around  $j = 0, 85, 171, 256, 341, 427$ , indicating a high probability of getting one of these values, or some value very close to them. In between, the probability is almost zero. The sharpness of the peaks depends on  $t$  (number of qubits in the first register). The lower limit  $2t \geq N^2$  ensures a high probability in measuring a value of  $j$  carrying the desired information. A careful analysis of the expression (2.14) is performed in [38] and a meticulous study of the peak form is performed in [25].

Let us analyze the possible measurement results.

If we get  $j = 0$  (first peak), the algorithm has failed in this round. It must

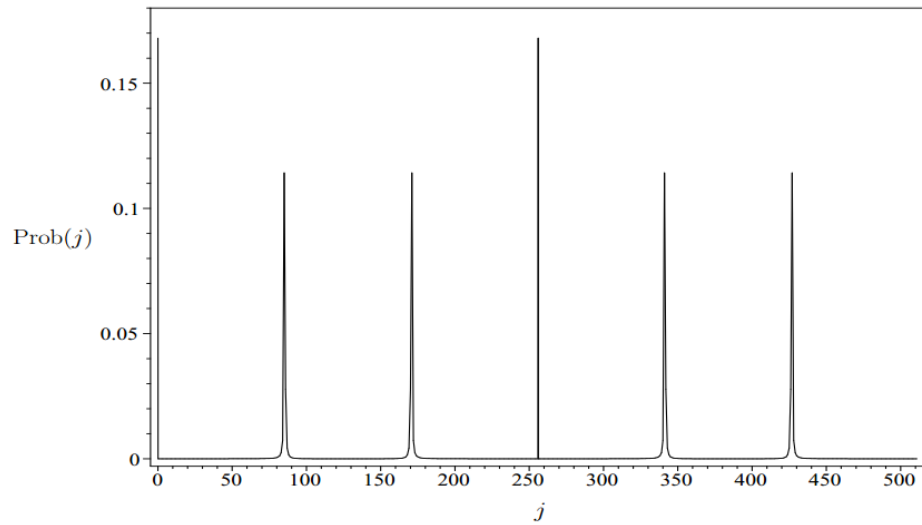


Figure 2.5: Plot of  $Prob(j)$  against  $j$ . Compare to the plot of Fig. 2.4, where peaks are not spread and have the same height.

be run again.

We keep  $x = 2$  and rerun the quantum part of the algorithm.

The probability of getting  $j = 0$  is low: from Eq. (2.14) we have that  $Prob(0) = 86/512 \approx 0.167$ . Now suppose we get  $j = 85$  (or any value in the second peak). We divide by 512 yielding  $85/512$ , which is a rational approximation of  $k_0/6$ , for  $k_0 = 1$ . How can we obtain  $r$  from  $85/512$ ?

The method of continued fraction approximation allows one to extract the desired information ( see Appendix A ). A general continued fraction expansion of a rational number  $j_1/j_2$  has the form

$$\frac{j_1}{j_2} = a_0 + \frac{1}{a_1 + \frac{1}{\dots + \frac{1}{a_p}}} \quad (2.15)$$

usually represented as  $[a_0, a_1, \dots, a_p]$ , where  $a_0$  is a non-negative integer and  $a_1, \dots, a_p$  are positive integers. The  $q$ -th convergent ( $0 \leq q \leq p$ ) is defined as the rational number  $[a_0, a_1, \dots, a_q]$ . It is an approximation to  $j_1/j_2$  and has a denominator smaller than  $j_2$ . This method is easily applied by inversion of the fraction followed by integer division with rational remainder. Inverting  $85/512$  yields  $512/85$ , which is equal to  $6 + 2/85$ .

We repeat the process with  $2/85$  until we get numerator 1. The result is:

$$\frac{85}{512} = \frac{1}{6 + \frac{1}{42 + \frac{1}{2}}}. \quad (2.16)$$

So, the convergents of  $85/512$  are  $1/6$ ,  $42/253$ , and  $85/512$ . We must select the convergents that have a denominator smaller than  $N = 21$  (since  $r < N$ )<sup>5</sup>. This method yields  $1/6$ , and then  $r = 6$ . We check that  $26 \equiv 1$  modulo 21, and the quantum part of the algorithm ends with the correct answer. The order  $r = 6$  is an even number, therefore  $\text{GCD}(2^{(6/2)} \pm 1, 21)$  gives two non trivial factors of 21.

A straightforward calculation shows that any measured result in the second peak (say  $81 \leq j \leq 89$ ) yields the convergent  $1/6$ .

Consider now the third peak, which corresponds to  $k_0/6$ ,  $k_0 = 2$ . We apply again the method of continued fraction approximation, which yields  $1/3$ , for any  $j$  in the third peak (say  $167 \leq j \leq 175$ ). In this case, we have obtained a factor of  $r$  ( $r_1 = 3$ ), since  $2^3 \equiv 8 \not\equiv 1$  modulo 21. We run the quantum part of the algorithm again to find the order of 8. We eventually obtain  $r_2 = 2$ , which yields  $r = r_1 r_2 = 3 \times 2 = 6$ .

The fourth and fifth peaks yield also factors of  $r$ . The last peak is similar to the second, yielding  $r$  directly.

The general account of the succeeding probability is as follows. The area under all peaks is approximately the same:  $\approx 0.167$ . The first and fourth peaks have a nature different from the others—they are not spread.

To calculate their contribution to the total probability, we take the basis equal to 1. The area under the second, third, fifth, and last peaks are calculated by adding up  $Prob(j)$ , for  $j$  running around the center of each peak.

So, in approximately 17% cases, the algorithm fails (1st peak). In approx-

---

<sup>5</sup>The inequality  $r < N$  comes from the two inequalities  $r \leq \phi(N) \leq N$ , where  $\phi(\cdot)$  is the Euler function ( $\phi(N)$  gives the number of positive integers less than  $N$ , coprime to  $N$ ). In particular, the first inequality follows from the Euler's theorem:  $x^{\phi(N)} \equiv 1 \pmod{N}$ , where  $x$  is a positive integer coprime to  $N$ . The second inequality follows easily from the definition of  $\phi$ .

imately 33% cases, the algorithm returns  $r$  in the first round (2nd and 6th peaks). In approximately 50% cases, the algorithm returns  $r$  in the second round or more (3rd, 4th, and 5th peaks).

Now we calculate the probability of finding  $r$  in the second round. For the 3rd and 5th peaks, the remaining factor is  $r_2 = 2$ . The graph equivalent to Fig. 2.5 in this case has 2 peaks, then the algorithm returns  $r_2$  in 50% cases. For the 4th peak, the remaining factor is  $r = 3$  and the algorithm returns  $r_2$  in 66.6% cases. This amounts to  $\frac{2 \times 50\% + 66.6\%}{30}$  of 50%, which is equal to around 22%.

In summary, the success probability for  $x = 2$  is around 55%.

## 2.3 Impact on cryptography

With interest growing in developing universal quantum computers, analyze the security of a cryptographic algorithm from the point of view of a quantum computer is of vital importance to the security of our data.

Nowadays, we are aware that some cryptosystems, believed to be safe against classical computers, are vulnerable to quantum computers: there are in fact quantum algorithms that are able to break current cryptographic systems, efficiently (in polynomial time, in the number of digits of the integer).

The main example of a quantum algorithm killer of many classical cryptosystems is the Shor's algorithm: the quantum-computer discrete-logarithm algorithm that breaks for example RSA, Diffie-Hellman, elliptic curve Diffie-Hellman (ECDH), DSA and ECDSA. In fact, since the birth of this algorithm, the integer factorization problem, the discrete logarithm problem or the elliptic-curve discrete logarithm problem can be solved in a polynomial time on a sufficiently powerful quantum computer running Shor's algorithm. Another quantum algorithm, Grover's algorithm ([29, 10]), does have some applications to these systems; but Grover's algorithm is not as shockingly fast as Shor's algorithm. Grover's algorithm is able to boost the search speed in an unsorted database. Grover's algorithm is an algorithm that can



potentially affect all cryptosystems, for instance it can boost a brute force attack, i.e., an attack that looks for a key, recursively trying all the possible combinations until the right one is found.

This particular quantum attack, however, can be avoided simply by increasing the size of the keys in all cryptographic systems: this is why Groove's algorithm had less impact on today's cryptography, than Shor's one.

So, the question is: is cryptography dead?

A closer look reveals, fortunately, that there is no justification for the leap from "quantum computers destroy RSA and some other cryptosystem" to "quantum computers destroy cryptography". There are many important classes of classic cryptographic systems based on different mathematical phenomena. All of these systems are believed to resist classical computers and quantum computers. Nobody has figured out a way to apply Shor's or Grover's algorithm to solve this problems.

Can any other quantum attack be successfully be carried out on these systems? Perhaps. This is a familiar risk in cryptography, which is why the community invests huge amounts of time and energy in cryptanalysis. Sometimes cryptanalysts find a devastating attack, demonstrating that a system is useless for cryptography; for example, every usable choice of parameters for the Merkle-Hellman knapsack public-key encryption system [34] is easily breakable. Sometimes cryptanalysts find attacks that are not so devastating but that force larger key sizes. Sometimes cryptanalysts study systems for years without finding any efficient attacks, and the cryptographic community begins to build confidence that the best possible attack has been found—or at least that real—world attackers will not be able to come up with anything better.

Consider, for example, the following three factorization attacks against RSA [11]:

- 1978: The original paper by Rivest, Shamir, and Adleman mentioned a new algorithm, Schroeppe's "linear sieve," that factors any RSA mod-

ulus  $n$  - and thus breaks RSA - using  $2^{(1+o(1))(\log(n))^{1/2}(\log(\log(n)))^{1/2}}$  simple operations. Here  $\log = \log_2$ . Forcing the linear sieve to use at least  $2^b$  operations means choosing  $n$  to have at least  $(0.5 + o(1))b^2 / \log(b)$  bits. Warning:  $0.5 + o(1)$  means something that converges to 0.5 as  $b \rightarrow \infty$ . It does not say anything about, e.g.,  $b = 128$ . Figuring out the proper size of  $n$  for  $b = 128$  requires looking more closely at the speed of the linear sieve.

- 1988: Pollard introduced a new factorization algorithm, the "number-field sieve." This algorithm, as subsequently generalized by Buhler, Lenstra, and Pomerance, factors any RSA modulus  $n$  using  $2^{(1.9\dots+o(1))(\log(n))^{1/3}(\log(\log(n)))^{2/3}}$  simple operations. Forcing the number-field sieve to use at least  $2^b$  operations means choosing  $n$  to have at least  $(0.016\dots + o(1))b^3 / (\log(b))^2$  bits. Today, twenty years later, the fastest known factorization algorithms for classical computers still use  $2^{(constant+o(1))(\log(n))^{1/3}(\log(\log(n)))^{2/3}}$  operations. There have been some improvements in the constant and in the details of the  $o(1)$ , but one might guess that  $1/3$  is optimal, and that choosing  $n$  to have roughly  $b^3$  bits resists all possible attacks by classical computers.
- 1994: Shor introduced an algorithm [54] that factors any RSA modulus  $n$  using  $(\log(n))^{2+o(1)}$  simple operations on a quantum computer of size  $\log(n)^{1+o(1)}$ . Forcing this algorithm to use at least  $2^b$  operations means choosing  $n$  to have at least  $2^{(0.5+o(1))b}$  bits an intolerable cost for any interesting value of  $b$ .

Consider, for comparison, attacks on another thirty-year-old public-key cryptosystem, namely McEliece's hidden-Goppa-code encryption system [42]. The original McEliece paper presented an attack that breaks codes of "length  $n$ " and "dimension  $n/2$ " using  $2^{(0.5+o(1))n/\log(n)}$  operations. Forcing this attack to use  $2^b$  operations means choosing  $n$  at least  $b(2 + o(1))\log(b)$ . Several subsequent papers have reduced the number of attack operations by an impressively large factor, roughly  $n \log(n) = 2^{(\log(n))^2}$ , but  $(\log(n))^2$  is much smaller

than  $0.5n/\log(n)$  if  $n$  is large; the improved attacks still use  $2^{(0.5+o(1))n/\log(n)}$  operations. One can reasonably guess that  $2^{(0.5+o(1))n/\log(n)}$  is best possible. Quantum computers do not seem to make much difference, except for reducing the constant 0.5. If McEliece's cryptosystem is holding up so well against attacks, why are we not already using it instead of RSA? The answer, in a nutshell, is efficiency, specifically key size. McEliece's public key uses roughly  $n^2/4 \approx b^2(\log(b))^2$  bits, whereas an RSA public key—assuming the number-field sieve is optimal and ignoring the threat of quantum computers—uses roughly  $(0.016\dots)b^3/(\log(b))^2$  bits. If  $b$  were extremely large then the  $b^{2+o(1)}$  bits for McEliece would be smaller than the  $b^{3+o(1)}$  bits for RSA; but real-world security levels such as  $b = 128$  allow RSA key sizes of a few thousand bits, while McEliece key sizes are closer to a million bits.

In addition to looking for classical algorithms resistant to quantum computers we have a second lifeline: in a world where the opponent has a quantum computer at their disposal, we too, potentially, can have one, and it would therefore be possible to implement quantum cryptographic algorithms (on quantum computer) resistant to quantum computing.

Let us briefly summarize these two strategies, known as Post-Quantum Cryptography and Quantum respectively:

- *Post-Quantum Cryptography*: it is based on the use of classical cryptographic algorithms, to be implemented, therefore, on classical computers, (possibly strengthened by choices of more robust parameters) which are thought to be resistant also to a quantum computer.
- *Quantum Cryptography*: the idea is to use (in the scenario in which they are available to us) quantum computers (and other quantum devices) to create quantum algorithms resistant to quantum (and classical) attacks;

Post-quantum cryptography and quantum cryptography are fundamentally different, but both have their place in strengthening security in a potential future where we have large quantum computers.

These two strategies, however, will be explored in the next chapters of this

thesis.

# Chapter 3

## Quantum cryptography

Quantum cryptography actually uses quantum mechanical principles as the basis of the security.

For example, quantum key distribution uses these quantum mechanical properties to create a shared key and distribute it, while being certain that a third party has not eavesdropped. For quantum states, we have several properties of nature that gives quantum information an extra level of security, first of all: quantum states collapse when they are measured. If an attacker tried to read out information in an entanglement based protocol, the quantum states would no longer be in a superposition. Additionally, the no-cloning theorem for quantum mechanics states that it's impossible to copy a quantum state. So, an attacker could not copy the quantum information being transmitted and do operations on their copy. If someone tries to eavesdrop, Alice and Bob will know.

Quantum cryptography is based on physical laws similar to those listed above, and not our knowledge and understanding of mathematics and hard problems. This means that it will remain secure no matter how much (quantum or classical) computing power grows over the years, and it does not matter how much mathematics develops finer resolution methodologies for mathematics problems (like factorization).

So why don't we just use quantum cryptography if it's so secure?

Quantum cryptography requires specialized equipment. For example, you need photon detectors, beamsplitters, and other equipment to make it work. We cannot put all of this into a small device like your phone.

Also, just because the encryption itself is fundamentally secure by the laws of physics, that doesn't mean no attacks can ever occur.

Even without quantum computers, intrusion does occur even on currently secure classical cryptographic algorithms. It's not because some computer can break the encryption. Side channel attacks can occur. These happen because of weakness in the implementation of the cryptosystem instead of a weakness in the algorithm itself. Even though you can be sure that no one has directly intercepted the photons in the process of creating the key in the quantum key distribution protocols, side channel attacks do exist in quantum cryptography as well.

The hardware requirements make it very difficult to use quantum cryptography everywhere. So, we will still need a post-quantum cryptographic protocol to secure the majority of devices.

Long distance quantum communication utilizes quantum properties. *BB84* and *E91* (entanglement-based) are the most famous communication protocols for quantum key exchange. These protocols generate a shared secure key.

To better understand what we are talking about, let us analyze how BB84 works.

### 3.1 BB84: quantum key exchange protocol

BB84 is a quantum key distribution scheme developed by Charles Bennett and Gilles Brassard in 1984. It is the first quantum cryptography protocol invented. The protocol is provably secure [55], relying on two conditions:

- i) the quantum property that information eavesdropping is only possible at the expense of disturbing the signal, if the two states one is trying

to distinguish are not orthogonal (see no-cloning Theorem and Measurement problem in Chapter 1);

- ii) the existence of an authenticated public classical channel in which to communicate.

In the BB84 scheme, Alice wishes to send a private key to Bob, let us see how she does it.

1. Alice starts by choosing two strings of bits,  $a$  and  $b$ , each  $n$  bits long. The first string will be the message to send to Bob, while the second, chosen randomly, represents the basis in which to encode each bit of the first string.
2. She then encodes these two strings of bits as a tensor product of  $n$  qubits:

$$|\psi\rangle = \bigotimes_{i=1}^n |\psi_{a_i b_i}\rangle = |\psi_{a_1 b_1}, \dots, \psi_{a_n b_n}\rangle$$

where  $a_i$  and  $b_i$  are the  $i$ -th bits of  $a$  and  $b$  respectively, and  $|\psi_{a_i b_i}\rangle$  represents the state obtained by encoding the bit  $a_i$  in the basis given by the bit  $b_i$  in the following way: if  $b_i = 0$  Alice will use the standard computational basis (  $\{|0\rangle, |1\rangle\}$  ), while if  $b_i = 1$  she will use the Hadamard basis (  $\{|+\rangle, |-\rangle\} := \{H(|0\rangle), H(|1\rangle)\} = \{\frac{|0\rangle+|1\rangle}{\sqrt{2}}, \frac{|0\rangle-|1\rangle}{\sqrt{2}}\}$  ). More specifically, all possible cases are the following:

$$\begin{aligned} |\psi_{00}\rangle &= |0\rangle, \\ |\psi_{10}\rangle &= |1\rangle, \\ |\psi_{01}\rangle &= \frac{|0\rangle + |1\rangle}{\sqrt{2}} = |+\rangle, \\ |\psi_{11}\rangle &= \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |-\rangle. \end{aligned}$$

The qubits are now in states that are not mutually orthogonal, and thus it is impossible to distinguish all of them with certainty without knowing  $b$ .

The process just described allows Alice to encode a string of bits  $a$  into a string of qubits  $|\psi\rangle$ , using the information of the bits string  $b$ .

3. Alice sends  $|\psi\rangle$  over a public and authenticated quantum channel to Bob.
4. Bob receives a state  $\epsilon(|\psi\rangle)$ , where  $\epsilon$  represents both the effects of noise in the channel and eavesdropping by a third party we will call Eve. After Bob receives the string of qubits, both Bob and Eve have their own states. However, since only Alice knows  $b$ , it makes it virtually impossible for either Bob or Eve to distinguish the states of the qubits. Also, after Bob has received the qubits, we know that Eve cannot be in possession of a copy of the qubits sent to Bob, by the no-cloning Theorem, unless she has made measurements. Her measurements, however, risk disturbing a particular qubit with probability of 0.5 if she guesses the wrong basis.
5. Bob proceeds to generate a string of random bits  $b'$  of the same length as  $b$ . Then, for each  $i$ , he measures the  $i$ -th qubit he has received from Alice  $\epsilon(|\psi_{a_i, b_i}\rangle)$ , using the standard computational basis if  $b'_i = 0$ , using the Hadamard basis otherwise. So he obtain a bits string  $a'$ .

The idea is to think of this Bob's measurement of  $\epsilon(|\psi\rangle)$  as a process of decoding  $|\psi\rangle$  into  $a'$  using the information contained in  $b'$ . When Bob's choice of measurement base coincides with that of Alice's coding base (for example, when they both use the Hadamard one), the result of the measurement must be the same, whereas where they use different bases, there is no correlation between its result and Alice's original choice.

We observe in particular that when (with probability of 1/2) Bob chooses the same measurement base as Alice, then, assuming that there are no interceptors or disturbances, they share the same bit. When instead (with probability 1/2) Bob chooses a measurement base other than Alice, the resulting bit agrees with the bit sent by Alice only half



of the time.

At this point, Bob announces publicly that he has received Alice's transmission (from now on Alice and Bob exchange only classic information through a public communication channel).

6. Alice, as a measurement of  $|\psi\rangle$  was made, knows she can now safely announce  $b$ , i.e., the bases in which the qubits were coded.

Bob communicates over a public channel with Alice to determine which  $b_i$  and  $b'_i$  are not equal.

Both Alice and Bob now discard the bits in  $a$  and  $a'$  where  $b$  and  $b'$  do not match. Now they know that, except for the effects of  $\epsilon$ , the remaining bits are the same: so they are sharing a percentage of bits of  $a$ .

7. From this remaining  $k$  bits where both Alice and Bob measured in the same basis, Alice randomly chooses a part of the key, say  $k/2$  bits, and discloses her choices over the public channel. Both Alice and Bob announce these bits publicly and run a check to see whether more than a certain number of them agree, calculating the percentage error  $R$  caused by  $\epsilon$ .
8. If the percentage  $R$  is too high, Alice and Bob restart the protocol from the beginning; otherwise they take the remaining bits, which are a raw key (" $RK$ "), and which will form the future secret key. Then they proceed with refining the two  $RK$  in order to correct any errors between them, and in order to make the key as secure as possible. For this purpose they use *information reconciliation* and *privacy amplification* techniques on the remaining bits of their  $RK$ . Let us describe these two techniques.
9. The reconciliation of information is a classic error correction code: the aim is to identify and correct the unequal bits between the Alice's and Bob's  $RK$ , obviously without giving useful information to Eve. Alice

and Bob divide the remaining bits of the  $RK$  into subsets of length  $l$ , chosen in such a way that there is a very low probability that there is more than 1 error per subset. This choice will be an important assumption for the continuation of the scheme.

For each subset, Alice and Bob do the parity check (the parity  $P$  of a binary string  $b_1 \dots b_l$  is defined as  $P = b_1 \oplus \dots \oplus b_l$ ). Since Alice and Bob will exchange these parities, in order not to give Eve information about the bits present in the substrings of length  $l$  of the  $RK$ , both eventually eliminate each time the last bit of the substring from the  $RK$ .

If for a substring the parities match (thanks to the assumption of the improbability of having more than 1 error per string) Alice and Bob assume that the  $l$  bits are equal; then the first  $l - 1$  bits of the substring are kept in the  $RK$ .

If for a substring the parity test does not give a positive result, then, through a binary search for parity, the different bit is identified and eliminated. Also in this phase it is important that, every time the parity is counted and shared with the other, the last bit of the tested string is eliminated from the  $RK$  by Alice and Bob, in order not to give information to Eve from their checks of parity.

At the end of this phase, with very high probability, Alice and Bob have the same bit in their  $RK$  string.

10. Privacy amplification is a method for reducing (and effectively eliminating) Eve's partial information about Alice and Bob's key. This partial information could have been gained both by eavesdropping on the quantum channel during key transmission (thus introducing detectable errors), and on the public channel during information reconciliation (where it is assumed Eve gains all possible parity information). Privacy amplification uses Alice and Bob's  $RK$  to produce a new shorter key, in such a way that Eve has only negligible information about the

new key. This can be done using a hash function, chosen at random from a publicly known set of such functions, which takes as its input a binary string of length equal to the key and outputs a binary string of a chosen shorter length. The amount by which this new key is shortened is calculated, based on how much information Eve could have gained (the percentage  $R$ ) about the old key, in order to reduce the probability of Eve having any knowledge of the new key to a very low value.

This new key obtained is the secret key sought, now shared between Alice and Bob.

The Figure 3.1 shows an example of how BB84 works (before the use of information reconciliation and privacy amplification techniques).

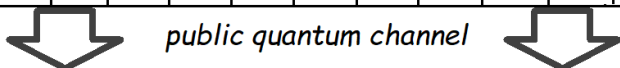
		i:	1	2	3	4	5	6	7	8	9	10	11
Alice:	$a=(a_i)$	0	1	0	0	1	1	1	0	1	0	1	
	$b=(b_i)$	st	st	H	st	H	H	st	H	st	H	H	
	$ \psi_i\rangle$	$ 0\rangle$	$ 1\rangle$	$ -\rangle$	$ 0\rangle$	$ +\rangle$	$ +\rangle$	$ 1\rangle$	$ -\rangle$	$ 1\rangle$	$ -\rangle$	$ +\rangle$	
		 <i>public quantum channel</i>											
Bob:	$b'=(b'_i)$	H	st	st	st	st	H	st	H	H	st	H	
	results	?	1	?	0	?	1	1	0	?	?	1	

Figure 3.1: BB84 quantum key exchange protocol.

The channel represented in the figure, for simplicity, is considered to be noise-free, and also the absence of Eve is assumed (in other words  $\epsilon$  is the identity function). To facilitate the reading of the scheme, the value 0 of the bit  $b_i$  (or  $b'_i$ ) has been indicated with "st" (for Standard Computational Basis), while the value 1 with "H" (Hadamard Base). When the  $i$ -th bit of  $a$ ,  $a_i$ , was found to be shared with probability 1, ( i.e., when  $b_i = b'_i$ ), its value is reported in the last line. Otherwise a "?" is returned, which means "unacceptable bit".



# Chapter 4

## Post-Quantum Cryptography

From a technological point of view there is still a severe limitation coming from the extreme difficulty of producing single photons, and thus quantum bits. So far they are produced with highly attenuated lasers but that in turn can also produce couples of correlated photons.

They are easily exploited to mount an attack, since a measurement on one photon of the couple doesn't reveal an alteration when the other photon is measured, and recover the secret key entirely or partially.

It is exactly for this reason that quantum cryptography "fails" in the embedded world, and post-quantum cryptography comes into play. We cannot afford to underestimate the progresses made in the realization of quantum computer. We need an alternative in case the quantum computer will become reality in the next decade or so, or before quantum cryptography will be reliable.

Among the numerous classes of classical cryptographic algorithms, there are many that are believed to be resistant to quantum attacks. The strategy of using these classical quantum computer-resistant algorithms is called post-quantum cryptography, and the algorithms are termed post-quantum secure. Let us give an overview of the different approaches that are used in post-quantum cryptography (before going into more detail in Chapter 5):

- **Lattice-based cryptography.** This type of cryptography builds its

own security on the difficulty of solving certain problems in multidimensional lattices, or on the difficulty of recognizing perturbed equations from unperturbed ones, that is in the same spirit of the code-based algorithm but in a lattice framework.

The most important lattice-based computational problem is the Shortest Vector Problem (SVP), which asks us to approximate the minimal Euclidean length of a non-zero lattice vector. See Section 5.1.

The example of this algorithm class that has perhaps attracted the most interest, not the first example historically, is the Hoffstein-Pipher-Silverman “NTRU”[30] public-key-encryption system (1998).

- **Hash-based cryptography.** Is the generic term for cryptographic algorithm based on the security of hash functions (non-invertible functions, such that the search for a pre-image is computationally intractable). See Section 5.2.

The classic example is Merkle’s hash-tree public-key signature system (1979) (topic that has been deepened in Chapter 4), building upon a one-message-signature idea of Lamport and Diffie.

- **Code-based cryptography.** Is the area of research that focuses on the study of cryptosystems based on error-correcting codes, following the seminal work of McEliece’s, hidden Goppa-code public-key encryption system (1978) (see Section 5.3).

These algorithms are inspired to the telecommunication-engineering problem of recovering the correct signal transmitted over a noisy channel. The signal is encoded with a particular “code”, prior to the sending. This allows to correct up to a certain number of errors. Encoding a message is simply adding a certain number of parity/check bits to the message. A parity bit could be, for example, the sum modulo 2 of a couple of bits, therefore it helps in retrieving the original values of the bits if one error occurred in the communication.

- **Multivariate-equations cryptography.** Is a kind of *asymmetric*<sup>1</sup> cryptographic based on multivariate (usually quadratic) polynomials over a finite field. This class of algorithm builds its own security on solving simultaneously a random set of quadratic equations in more than one indeterminate. The encryption/decryption is mainly based on the evaluation of such equations at particular points.

See Section 5.4.

- **Supersingular isogeny-based cryptography** In this case, the security is based on computing isogenies between supersingular elliptic curves. To better understand what we are talking about, see Section 5.5 in which the theme is widely discussed.

The proposal of isogeny-based cryptography as post-quantum cryptography, although promising, is however relatively recent, and therefore still little studied and far from being standardized. This is generally a cons, because cryptographic schemes before being accepted as resistant (especially quantum resistant) must survive years of study and attempted forcing by the scientific community.

The most famous post-quantum cryptosystem based on supersingular isogenies is the supersingular isogeny Diffie-Hellman (SIDH) key exchange by Jao and De Feo [31].

---

<sup>1</sup>*Asymmetric* cryptography is another name for indicate "Public-key cryptography"; an asymmetric-key cryptographic system uses pairs of keys. Each pair consists of a public key (which may be known to others) and a private key (which may not be known by anyone except the owner). The generation of such key pairs depends on cryptographic algorithms which are based on mathematical problems termed one-way functions. Effective security requires keeping the private key private; the public key can be openly distributed without compromising security. On the other hand, a *Symmetric*-key algorithms are algorithms for cryptography that use the same cryptographic keys for both the encryption of plaintext and the decryption of ciphertext. The keys, in practice, represent a shared secret between two or more parties that can be used to maintain a private information link. The requirement that both parties have access to the secret key is one of the main drawbacks of symmetric-key encryption.

- **Zero-knowledge based cryptographic schemes** use the idea of *zero-knowledge proofs* where one party (the prover) can prove to another party (the verifier) that a given statement is true while the prover avoids conveying any additional information apart from the fact that the statement is indeed true. The essence of zero-knowledge proofs is that it is trivial to prove that one possesses knowledge of certain information by simply revealing it; the challenge is to prove such possession without revealing the information itself or any additional information. A zero-knowledge proof of some statement must satisfy three properties:
  - Completeness: if the statement is true, the honest verifier (that is, one following the protocol properly) will be convinced of this fact by an honest prover.
  - Soundness: if the statement is false, no cheating prover can convince the honest verifier that it is true, except with some small probability.
  - Zero-knowledge: if the statement is true, no verifier learns anything other than the fact that the statement is true. In other words, just knowing the statement (not the secret) is sufficient to imagine a scenario showing that the prover knows the secret. This is formalized by showing that every verifier has some simulator that, given only the statement to be proved (and no access to the prover), can produce a transcript that "looks like" an interaction between the honest prover and the verifier in question.

The first two of these are properties of more general interactive proof systems. The third is what makes the proof zero-knowledge.

Zero-knowledge proofs are not proofs in the mathematical sense of the term because there is some small probability, the soundness error, that a cheating prover will be able to convince the verifier of a false statement. In other words, zero-knowledge proofs are probabilistic "proofs" rather



than deterministic proofs. However, there are techniques to decrease the soundness error to negligibly small values.

*Example 8* (A zero-knowledge proof). Imagine your friend is red-green colour-blind (while you are not) and you have two balls: one red and one green, but otherwise identical. To your friend they seem completely identical and he is skeptical that they are actually distinguishable. You want to prove to him they are in fact differently-coloured, but nothing else; in particular, you do not want to reveal which one is the red and which is the green ball.

Here is the proof system. You give the two balls to your friend and he puts them behind his back. Next, he takes one of the balls and brings it out from behind his back and displays it. He then places it behind his back again and then chooses to reveal just one of the two balls, picking one of the two at random with equal probability. He will ask you, "Did I switch the ball?" This whole procedure is then repeated as often as necessary.

By looking at their colours, you can, of course, say with certainty whether or not he switched them. On the other hand, if they were the same colour and hence indistinguishable, there is no way you could guess correctly with probability higher than 50%.

Since the probability that you would have randomly succeeded at identifying each switch/non-switch is 50%, the probability of having randomly succeeded at all switch/non-switches approaches zero ("soundness"). If you and your friend repeat this "proof" multiple times (e.g. 20 times), your friend should become convinced ("completeness") that the balls are indeed differently coloured.

The above proof is zero-knowledge because your friend never learns which ball is green and which is red; indeed, he gains no knowledge about how to distinguish the balls.

## 4.1 Standardization: the NIST challenge

Several standardization bodies have recognized the urgency of switching to cryptosystems that remain secure against attacks by quantum computers. This is an important development because many applications of cryptography require all parties to use the same cryptographic system: standardization is thus a prerequisite for widespread deployment. Sometimes de-facto standards are set without standardization bodies, but formal standardization processes are widely viewed as reducing cryptographic risks.

The National Institute of Standards Technology (NIST)<sup>2</sup> has launched, in 2016, a post-quantum algorithms standardization program and competition [6], called *NIST Post-Quantum Cryptography Standardization*.

23 signature schemes and 59 encryption/KEM (Key encapsulation mechanisms) schemes were submitted by the initial submission deadline at the end of 2017, of which 69 total were deemed complete and proper and participated in the first round. The main characteristics on which the algorithms are evaluated are: security, efficiency and parameters sizes.

The NIST aims to complete this analysis by early 2022 and confirm the candidates for standardization.

Nowadays, there have been 3 rounds of selections. A total of 7 algorithms, of which 3 are signature schemes, have advanced to the third round. In addition to the 7 finalist algorithms, the NIST has included 8 other different algorithms among the "alternative candidates": they are algorithms that have not been completely discarded and are still under study, to which, however, the finalists are preferred today.

---

<sup>2</sup>The National Institute of Standards and Technology (NIST) is a physical sciences laboratory and non-regulatory agency of the United States Department of Commerce. Its mission is to promote American innovation and industrial competitiveness. NIST's activities are organized into laboratory programs that include nanoscale science and technology, engineering, information technology, neutron research, material measurement, and physical measurement. From 1901 to 1988, the agency was named the National Bureau of Standards.

**Finalists**

Type	PKE/KEM	Signature
Lattice	<ul style="list-style-type: none"> <li>• CRYSTALS-KYBER</li> <li>• NTRU</li> <li>• SABER</li> </ul>	<ul style="list-style-type: none"> <li>• CRYSTALS-DILITHIUM</li> <li>• FALCON</li> </ul>
Code-based	<ul style="list-style-type: none"> <li>• Classic McEliece</li> </ul>	
Multivariate		<ul style="list-style-type: none"> <li>• Rainbow</li> </ul>

**Alternate candidates**

Type	PKE/KEM	Signature
Lattice	<ul style="list-style-type: none"> <li>• FrodoKEM</li> <li>• NTRU Prime</li> </ul>	
Code-based	<ul style="list-style-type: none"> <li>• BIKE</li> <li>• HQC</li> </ul>	
Hash-based		<ul style="list-style-type: none"> <li>• SPHINCS+</li> </ul>
Multivariate		<ul style="list-style-type: none"> <li>• GeMSS</li> </ul>
Supersingular elliptic curve isogeny	<ul style="list-style-type: none"> <li>• SIKE</li> </ul>	
Zero-knowledge proofs		<ul style="list-style-type: none"> <li>• Picnic</li> </ul>

Figure 4.1: Finalists and Alternative Candidates of NIST Post-Quantum Cryptography Standardization Program Selection, round three.

In Figure 4.1 there are two tables containing the names of the finalists and the alternative candidates at the third round of NIST competition, together with the type of cryptography on which the algorithms are based, and the use of each (PKE stands for Public-Key cryptography Encryption).

It is interesting to note that among the finalist algorithms there are no Hash-based algorithms; this is because NIST has carried out a parallel standardization process for hash-based digital signature (HBS) quantum resistant algorithms [7].

In detail, the HBS algorithms of interest in this thesis are XMSS and LMS

[19], discussed later in Section 6.2.

Significant interest was manifested in the standardization of such schemes at that time, because the underlying technology was well understood. In particular, the security of an HBS scheme, when implemented properly, relies only on the preimage resistance of its component cryptographic hash function. This property is already the basis for the security of many NIST-approved cryptographic algorithms and protocols, and no quantum computing algorithms are known that would pose a practical threat in the foreseeable future.

# Chapter 5

## Post-Quantum Cryptography hard problems

The post-quantum schemes are, by definition, *post-quantum secure*. In this Chapter we start by discussing what is meant by this.

To define security of a (post-quantum) cryptographic scheme the following is required. Firstly, there must be a protection goal that the cryptographic scheme is supposed to achieve. For example, encryption schemes protect *confidentiality* and digital signature schemes provide *integrity*, *authenticity*, and *non-repudiation* (for details see [43]). Secondly, there must be an adversary model that describes the goals of a potential adversary and the capabilities and resources that the adversary can use. For example, in the *ciphertext-only security model* for encryption schemes, the adversary searches for plaintexts that correspond to given ciphertexts and can only see ciphertexts; in the *chosen ciphertext model*, the adversary can encrypt plaintexts of her choice (for more information about this topic, see the Subsection 6.1.2). Thirdly, the time period for which a cryptographic scheme is supposed to achieve its security goals must be known. For example, one-time passwords only need to be kept confidential until they have been used while conventional passwords must be protected until they expire.

We now describe how the security of a cryptographic scheme  $S$  is established. An algorithmic problem  $P$  is selected whose hardness guarantees the security of  $S$ . No polynomial time algorithm for solving  $P$  must be known as in this case  $P$  would be considered easy to solve. In all classical cryptography, and in particular in the post-quantum one, the security of the schemes is based on "hard" algorithmic problems (which we will see in this chapter). For example, in the case  $S$  is RSA algorithm,  $P$  is the integer factoring problem.

Once quantum computers reach maturity, integer factoring, along with some other problems, can no longer be used as the security basis of cryptographic schemes since polynomial-time algorithms for their resolution will be available.

If  $P$  cannot be solved in polynomial time, an instance<sup>1</sup> of  $S$  is selected that achieves the desired security level. Such an instance is determined by choosing the necessary parameters and keys. For example, the RSA encryption scheme is instantiated (i.e., an instance is created) by choosing two parameters: the *RSA-modulus* and the *RSA-encryption exponent*. Likewise, the underlying algorithmic problem can be instantiated. In the case of the integer factorization problem, an instance is determined by the number to be factored.

Each instance  $I_S$  of  $S$  is associated with an instance  $I_P$  of  $P$  whose intractability guarantees the security of  $I_S$  in the chosen security model. In order for  $I_S$  to be secure for a sufficiently long time period, the instance  $I_P$  must remain intractable during this time period. So there are two tasks in this context. First, connecting  $I_S$  to some  $I_P$  and second, determining the hardness of the instances of  $P$ . The first task is either addressed using a mathematical reduction proof or, if this is not possible, by applying heuristical arguments.

---

<sup>1</sup>An instance of an algorithm based (resp. of a computational problem underlying the algorithm) is formed by the input parameters of the algorithm (resp. of the problem). The same algorithm (resp. the computational problem), changing the choice of its initial parameters, can present a more or less complex resolution: for example RSA (resp. the factoring problem) with parameters of few digits is simple to carry out.

In the case of RSA and the relevant security models, no reduction proof is known. The second task is to analyze the hardness of  $P$ . Such an analysis provides a lower bound for the computational resources required to solve a given instance of  $P$ . There are different models for measuring the resources. As the necessary technical details about quantum computers are still unknown, such a more detailed analysis of post-quantum security is not yet possible. This is why post-quantum security currently refers to a cryptographic scheme being associated in the above sense to a computational problem that is not solvable in polynomial time on a quantum computer. This includes the impossibility of solving this problem on a conventional computer. Throughout this chapter, we will expose several computational problems believed to be intractable (not solvable in polynomial times) even by quantum computers. There will be computational problems from different fields of mathematics.

For each of the problems exposed, there exist many post-quantum cryptographic algorithms that base their security on the intractability of the related problem.

## 5.1 Hard Problems in Lattices for cryptography

The discussion that follows leads us to the following conjecture:

*Conjecture:* There is no polynomial time quantum algorithm that approximates lattice problems (that will be exposed) to within polynomial factors.

Latex-based post-quantum algorithms are numerous, and in general they appear to be very promising. For example, among the 15 finalists/alternative candidates of the Nist competition, 7 are latex-based (see Figure 4.1).

Among these, those of digital signature (such as Crystals-Dilithium and Falcon) will be better analyzed in Chapter 6.

Let us now define in detail the main latex-based computational problems of quantum strength.

**Definition 7** (Lattices). Let  $b_1, \dots, b_n$  be  $n$  linearly independent vectors in  $\mathbb{R}^n$  (the *basis* for the lattice). A *lattice* of *rank*  $n$  is defined as the set of all integer combinations:

$$\mathcal{L}(b_1, \dots, b_n) = \left\{ \sum_{i=1}^n x_i b_i \mid x_i \in \mathbb{Z} \text{ for } 1 \leq i \leq n \right\} \quad (5.1)$$

A basis can be represented by the matrix  $B = [b_1, \dots, b_n] \in \mathbb{R}^{n \times n}$  having the basis vectors as columns. Using matrix notation, the lattice generated by a matrix  $B \in \mathbb{R}^{n \times n}$  can be defined as  $\mathcal{L}(B) = \{Bx \mid x \in \mathbb{Z}^n\}$ .

It is not difficult to see that if  $U$  is a *unimodular matrix* ( i.e., an integer square matrix with determinant  $\pm 1$ ), the bases  $B$  and  $BU$  generate the same lattice. More precisely, the following Proposition holds:

**Proposition 5.** *Let  $B$  and  $C$  two bases. Then  $\mathcal{L}(B) = \mathcal{L}(C)$  if and only if there exists a unimodular matrix  $U$  such that  $B = CU$ .*

*Proof.* First assume  $B = CU$  for some unimodular matrix  $U$ . Notice that if  $U$  is unimodular, then  $U^{-1}$  is also unimodular. In particular, both  $U$  and  $U^{-1}$  are integer matrices, and  $B = CU$  and  $C = BU^{-1}$ . Since  $U$  and its inverse are two bijections of  $\mathbb{Z}^n$ , it follows that  $\mathcal{L}(B) \subseteq \mathcal{L}(C)$  and  $\mathcal{L}(C) \subseteq \mathcal{L}(B)$ , i.e., the two matrices  $B$  and  $C$  generate the same lattice.

Now assume  $B$  and  $C$  are two bases for the same lattice  $\mathcal{L}(B) = \mathcal{L}(C)$ . Then, by definition of lattice, using the two inclusions that form the equality between the two lattices, there exist integer square matrices  $V$  and  $W$  such that  $B = CW$  and  $C = BV$ . We have to show that they have a unitary determinant.

Combining these two equations we get  $B = BVW$ , or equivalently,  $B(I - VW) = 0$ . Since the columns of  $B$  are linearly independent,  $\ker(B) = \{0\}$ , so it must be  $I - VW = 0$ , i.e.,  $VW = I$ . In particular,  $\det(V)\det(W) = \det(VW) = \det(I) = 1$ . Since matrices  $V$  and  $W$  have integer entries,  $\det(V), \det(W) \in \mathbb{Z}$ , and it must be  $\det(V) = \det(W) \in \{\pm 1\}$ .  $\square$



In particular, any lattice admits several bases, and this fact is at the core of many cryptographic applications.

**Definition 8** (Minimum length). Let  $\mathcal{L}$  a lattice. For a fixed norm, we can define the minimum length:

$$\lambda_1 = \min_{0 \neq x \in \mathcal{L}} \|x\| = \min_{x, y \in \mathcal{L}, y \neq x} \|x - y\|$$

Let us now describe the most well-known computational problems on lattices, supposed to be post-quantum hard problem.

**Definition 9** (Shortest Vector Problem, SVP). Given a lattice  $\mathcal{L}(B)$ , find a (nonzero) lattice vector  $Bx$  (with  $x \in \mathbb{Z}^n$ ) of length  $\|Bx\| = \lambda_1$ .

SVP is one of the best known and most used lattice problem, and many cryptosystems, such as NTRU [30], are secure under the assumption that SVP is hard.

In fact, one typically considers the approximation variant of SVP where the goal is to output a lattice vector whose length is at most some approximation factor  $\gamma(n)$  times the length of the shortest nonzero vector, where  $n$  is the dimension of the lattice:

**Definition 10** (Approximated Shortest Vector Problem,  $\text{SVP}_\gamma$ ). Given a lattice  $\mathcal{L}(B) \subseteq \mathbb{R}^n$  and fixed a factor  $\gamma = \gamma(n) \geq 1$ , find a (nonzero) lattice vector  $Bx$  (with  $x \in \mathbb{Z}^n$ ) of length at most  $\|Bx\| \leq \gamma\lambda_1$ .

It is clear that the closer the  $\gamma$  is to 1, the closer the problem comes to being the exact version of SVP.

This type of problem weakening can also be done for the other problems presented in this section. The modus operandi is completely analogous, so we will avoid describing these generalizations.

**Definition 11** (Closest Vector Problem, CVP). Given a lattice  $\mathcal{L}(B)$  and a target point  $t \in \mathbb{R}^n$  (not necessarily in the lattice), find a lattice vector  $Bx$  within distance  $\|Bx - t\| = \min_{Bx' \in \mathcal{L}} \|Bx' - t\| =: \mu$  from the target.

**Definition 12** (Shortest Independent Vectors Problem, SIVP). Given a lattice  $\mathcal{L}(B)$ , find  $n$  linearly independent lattice vectors  $S = [s_1, \dots, s_n] = [Bx_1, \dots, Bx_n]$  minimizing the quantity:  $\max_i \|s_i\|$ .

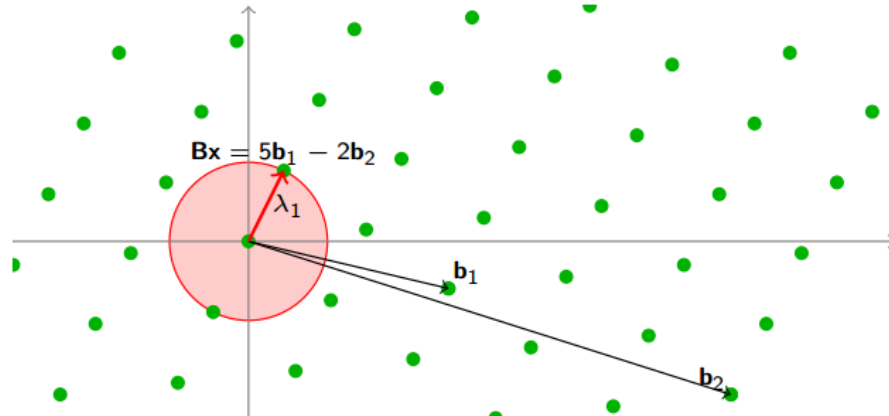


Figure 5.1: 2-dimensional SVP example

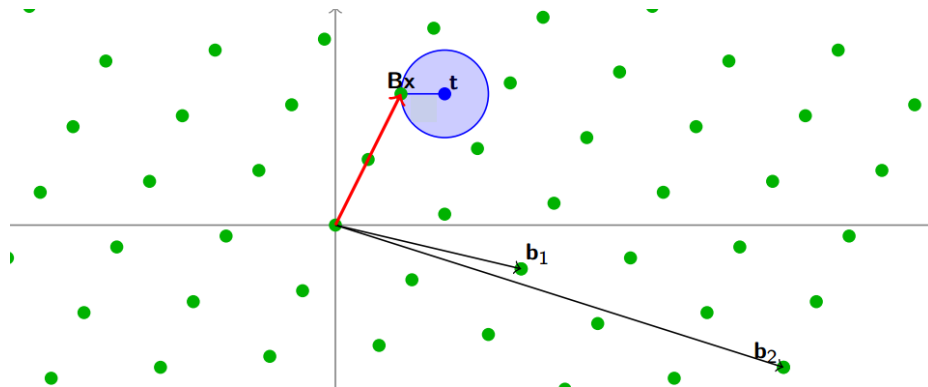


Figure 5.2: 2-dimensional CVP example

The problems defined above are not unrelated. For example, the closest vector problem is a generalization of the shortest vector problem (we will write  $SVP \leq CVP$ ): this means that solving the CVP implies being able to solve the SVP.

The naive method to find the shortest vector by calling the CVP to find the

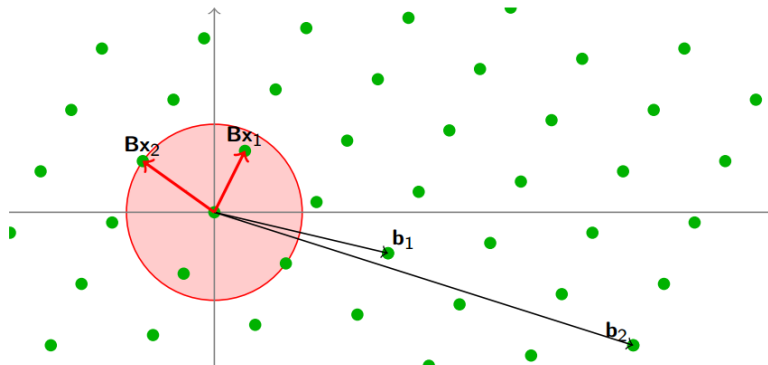


Figure 5.3: 2-dimensional SIVP example

closest vector to 0 does not work because 0 is itself a lattice vector and the algorithm could output 0.

An idea of the reduction from SVP to CVP follows. Let us start by saying that we are going to reduce the shortest vector problem to the solution of  $n$  instances of the closest vector problem.

Suppose that the input to the SVP is the basis for lattice generated by  $B = [b_1, b_2, \dots, b_n]$ . Consider the basis  $B^i = [b_1, \dots, 2b_i, \dots, b_n]$  and let  $x_i$  be the vector returned by  $\text{CVP}(B^i, b_i)$ . The claim is that the shortest vector in the given lattice  $\mathcal{L}(B)$  is the shortest vector in the set  $\{x_i - b_i \mid i = 1, \dots, n\}$  (which can be easily found with a simple check between  $n$  vectors). In conclusion, we therefore affirm that this reduction has a polynomial cost in  $n$ .

To learn more about these topics, the reader is referred to the book [44], which, for example, provides a detailed explanation of the reduction mentioned above.

The most well-known and widely studied algorithm for lattice problems is the LLL algorithm, developed in 1982 by Lenstra, Lenstra, and Lovász [37]. This is a polynomial time algorithm for SVP (and for most other basic lattice problems) that achieves an approximation factor of  $\gamma(n) = 2^{O(n)}$ . As bad as this might seem, the LLL algorithm is surprisingly useful, with applications ranging from factoring polynomials over the rational numbers, to integer programming, as well as many applications in cryptanalysis (such as attacks on

knapsack-based cryptosystems and special cases of RSA).

In 1987, Schnorr presented an extension of the LLL algorithm [52] leading to somewhat better approximation factors at the price of an increased running time.

Several variants of Schnorr's algorithm exist. Unfortunately, all these variants achieve more or less the same exponential approximation guarantee.

If one insists on an *exact* solution to SVP, or even just an approximation to within  $\text{poly}(n)$  factors, the best known algorithm has a running time of  $2^{O(n)}$  [9]. The space requirement of this algorithm is unfortunately also exponential which makes it essentially impractical. Other algorithms require only polynomial space, but run in exponential time.

## 5.2 Hash function for cryptography

Post-quantum algorithms based on hash functions appear to be very promising, so much so that NIST has decided to start an early and alternative selection, in which the XMSS and LMS algorithms have been standardized; to deepen these algorithm, refer to the Chapter 6.

Their functioning is relatively simple, as are the assumptions about their safety. In a sense that we will now go into, the thing that is required is that hash functions used are "difficult to reverse".

A hash is a particular function which produces a sequence of bits, called a *digest*, (or a *string*) that is related to the incoming data.

**Definition 13.** (Hash function) Let  $\Sigma$  be an alphabet,  $n \in \mathbb{N}$  fixed,  $h$  is called a hash function (of length  $n$ ) if:

$$\begin{aligned} h : \Sigma^* &\rightarrow \Sigma^n \\ x &\mapsto h(x) \end{aligned}$$

Where  $\Sigma^*$  indicate the set of strings composed of symbols of  $\Sigma$  of any length (including the null one), while with  $\Sigma^n$  the set of strings of length  $n$ .

Hash is a non-invertible function (for cardinality reasons is not 1-1) that maps a string of arbitrary length to a string of predefined length. There are several algorithms that implement hash functions with particular properties that depend on the application. For example, it is known how to build hash functions starting from lattice problems (see [51]).

There is no 1-1 correspondence between the hash result and the initial input, since the possible input are more than the possible hashed digest, and so, according to the pigeonhole principle, at least one hashed digest will correspond to more texts. When two texts produce the same hash, it is called a *collision*:

**Definition 14** (Collision). Let  $h$  an hash function. The couple  $(x, x') \in \Sigma^* \times \Sigma^*$  is a collision if  $x \neq x'$  and  $h(x) = h(x')$ .

In the cryptographic field, the quality of a hash function is measured directly by the difficulty in identifying two texts that generate a collision. To discourage the use of hashing algorithms previously considered safe, it was in fact sufficient for a single group of researchers to generate a collision. This is what happened for example for the SNEFRU, MD2, MD4, MD5 and SHA-1 algorithms.

In particular in cryptographic applications the hash function, in addition to asking for that for all input message  $x$  in  $\Sigma^*$  is easy (has polynomial complexity) to calculate its image  $h(x)$ , is required to have the following properties:

- *resistance to preimage* or one-way: the search for an input string that gives a hashed string equal to a given hashed string is computationally intractable. Formally:  $h$  is one-way if  $\forall y \in \Sigma^n$  one can efficiently find  $x$  such that  $h(x) = y$  with negligible probability.
- *resistance to the second preimage*: it is computationally intractable to search for an input string that gives a hashed string equal to that of a

given string. Formally:  $h$  is second preimage resistant if  $\forall x \in \Sigma^*$  one can efficiently find  $x'$  such that  $h(x) = h(x')$  with negligible probability.

- *collision resistance*: it is computationally intractable to search for a collision. Formally: one can efficiently find a collision  $(x, x')$  with negligible probability.

So, a cryptographically secure hash function should not allow to go back, in a time comparable with the use of the hash itself, to a text that can generate it.

Let us note, with the next proposition, that being a one-way function does not imply being a collision resistant function.

**Proposition 6.** *There exist one-way functions that are not collision resistant.*

*Proof.* An example of a non-collision resistant one-way function is the following:

$$h : \Sigma^* \rightarrow \Sigma^m$$

$$x \mapsto h(x) := x^2 \bmod n$$

where  $n = pq$ ,  $p$  and  $q$  are prime numbers,  $\Sigma = \{0, 1\}$  (and so one can think at  $\Sigma^*$  as the binary version of integer numbers, with a bit that gives us the information about the sign).

This  $h$  is one-way function because making the preimage of it is equivalent to solving the problem of extracting square roots of  $x$  modulo  $n = pq$ , which is a well-known algebra problem equivalent to the factorization of  $n = pq$ , to solve which they have not been found (not quantum) polynomial algorithms. Obviously  $h$  is not resistant to collisions because it is easy to find a collision  $\forall x \in \Sigma^*$ . Just take the decimal equivalent  $x_0 \in \mathbb{Z}$  of the binary  $x \in \Sigma^*$ , set  $y_0 := -x_0$  and take  $y \in \Sigma^*$ , the binary version of  $y_0$ . It turns out that  $(x, y)$  is a collision for  $h$ .  $\square$

There is an obvious reduction from Collision to Second preimage. Suppose that "FindSecondPreimage" is any algorithm for solving efficiently the search of Second preimage. If we choose  $x \in X$  at random and run  $\text{FindSecondPreimage}(x)$ , then we obtain a collision with non negligible probability. Thus we obtain the following well-known and easy result.

**Theorem 7.** *Collision resistance  $\geq$  Resistance to the second preimage.*

In [58] it is shown that it is possible to obtain a "good" reduction also from Collision to Preimage for a given hash function  $h : X \rightarrow Y$ , provided that either of the following two assumptions are satisfied:

- the hash function  $h$  is "close to uniform"<sup>2</sup>, or
- there is an oracle<sup>3</sup> that solves the Preimage problem with a "good"

---

<sup>2</sup>A hash function  $h$  is an *uniform hash function* if an adversary cannot distinguish the output from  $f$  with a uniform distribution over its range. If the domain of  $h$  is of finite cardinality, for example  $h : \{0, 1\}^k \rightarrow \{0, 1\}^n$ , asking the uniformity is reduced to asking that  $|h^{-1}(y)| = k/n$ .

<sup>3</sup>The *oracle*, in this context, is an abstract entity capable of solving some problem, which for example may be a decision problem or a function problem. The problem does not have to be computable; the oracle is not assumed to be a Turing machine or computer program. The oracle is simply a "black box" that is able to produce a solution for any instance of a given computational problem:

- A decision problem is represented as a set  $A$  of natural numbers (or strings). An instance of the problem is an arbitrary natural number (or string). The solution to the instance is "YES" if the number (string) is in the set, and "NO" otherwise.
- A function problem is represented by a function  $f$  from natural numbers (or strings) to natural numbers (or strings). An instance of the problem is an input  $x$  for  $f$ . The solution is the value  $f(x)$ .

In other words, an oracle is something which can immediately give you the answer to some problem, usually an infeasible or impossible problem. For example, a "Halting-problem Oracle" could tell you immediately whether a certain program on a certain input halts or not, even though the halting problem is uncomputable to us mere mortals. However, sometimes it is possible to prove some useful properties by pretending certain oracles exist. In cryptography, for instance, oracles are most often used to show that, even if our attackers had access to some seemingly-impossible oracle, they still wouldn't have any (significant) advantage to breaking the scheme security.

success probability for every possible input  $y \in Y$  (in other words, it is assumed the possibility of solving the Preimage problem).

Neither of these assumptions are entirely satisfactory. The first assumption is likely to be true with high probability for random hash functions; however, it seems to be impossible to verify for hash functions used in practice. The second assumption ignores the possibility that there could exist practical preimage-finding algorithms that are successful on some (but not all) inputs. In light of these results, it is advisable that it is best to require both collision resistance and preimage resistance as necessary properties for a hash function to be considered secure, due to the lack of a completely satisfactory reduction from the problem of finding collisions to the problem of finding preimages. Hash functions are the basis of various cryptographic algorithms of post-quantum interest. To learn more, see Chapter 6.

### 5.3 Code-based cryptography

Linear Codes are originally used for Digital Communication and based on Coding Theory. Coding theory is an important study which attempts to minimize data loss due to errors introduced in transmission from noise, interference or other forces. Data to be transmitted is encoded by the sender as linear codes which is decoded by the receiver. Data encoding is accomplished by adding additional information to each transmitted message to enable the message to be decoded even if errors occur.

Different codes are being studied to provide solutions for various problems occurring in applications. The most prominent type of error-correcting codes are called linear codes. The linear codes can be represented by matrices. It is computationally difficult to decode messages without knowing the underlying linear code. This hardness underpins the security of the code-based (post-quantum) cryptosystem which includes all cryptosystems, symmetric or asymmetric, whose security relies, partially or totally, on the hardness of decoding in a linear error correcting code, possibly chosen with some partic-



ular structure or in a specific family of linear codes.

Before talking more specifically about these cryptosystems, let us start recalling some basic definitions, and then listing some difficult problem cases in code theory for (post-quantum) cryptography.

**Definition 15** (Linear Code). Let  $\mathbb{F}_q$  be a finite field of order  $q$ . An  $(n, k)$ -code over  $\mathbb{F}_q^n$  is a  $k$ -dimensional linear subspace  $C$  of the linear space  $\mathbb{F}_q^n$ . Elements of  $\mathbb{F}_q^n$  are called *words*, and elements of  $C$  are *codewords*. We call  $n$  the length, and  $k$  the dimension of  $C$ .

**Definition 16** (Hamming distance, weight). The *Hamming distance*  $d(x, y)$  between two words  $x, y$  is the number of coordinates in which  $x$  and  $y$  differ. That is,  $d(x, y) = |\{i : x_i \neq y_i\}|$ , where  $x = (x_1, \dots, x_n)$  and  $y = (y_1, \dots, y_n)$ . In particular,  $w(x) := d(x, 0)$  is called the *Hamming weight* of  $x$ , where  $0$  is the vector containing  $n$  zeros. The *minimum distance of a linear code*  $C$  is the minimum Hamming distance between any two distinct codewords in  $C$ .

**Definition 17** (Generator matrix). A *generator matrix* of an  $(n, k)$ -linear code  $C$  is a  $k \times n$  matrix  $G$  whose rows form a basis for the vector subspace  $C$ . We call a code *systematic* if it can be characterized by a generator matrix  $G$  of the form  $G = (I_{k \times k} \| A_{k \times (n-k)})$ , (concatenation of two matrices) where  $I_{k \times k}$  is the  $k \times k$  identity matrix and  $A$ , an  $k \times (n - k)$  matrix.

**Definition 18.** (Parity-check matrix) A *parity-check matrix* of an  $(n, k)$ -linear code  $C$  is an  $(n - k) \times n$  matrix  $H$  whose rows form a basis of the orthogonal complement of the vector subspace  $C$ , i.e., it holds that,

$$C = \{c \in \mathbb{F}_q^n : Hc^T = 0\}$$

**Definition 19** (Syndrome of a vector). Let  $C$  be an  $(n, k)$ -linear code over  $\mathbb{F}_q$  and let  $H$  be a parity-check matrix for  $C$ . For any  $w \in \mathbb{F}_q^n$ , the *syndrome* of  $w$  is the word  $S(w) = wH^T \in \mathbb{F}_q^{n-k}$ . (Strictly speaking, as the syndrome depends on the choice of the parity-check matrix  $H$ , it is more appropriate to denote the syndrome of  $w$  by  $S_H(w)$  to emphasize this dependence. However, for simplicity of notation, the subscript  $H$  is dropped whenever there is no risk of ambiguity.)

**Definition 20** (Goppa code). Let  $g(z) = \sum_{i=0}^t g_i z^i \in \mathbb{F}_{q^m}[z]$ , and let  $L = \{a_1, \dots, a_n\} \subseteq \mathbb{F}_{q^m}$  such that, for all  $a_i \in L$ ,  $g(a_i) \neq 0$ . Then the code defined by

$$\left\{ c = (c_1, \dots, c_n) \in \mathbb{F}_{q^m} \left| \sum_{i=1}^n \frac{c_i}{z - a_i} \equiv 0 \pmod{g(z)} \right. \right\}$$

is called *Goppa code* with parameters  $g(z)$  and  $L$ .

In what follows, we recall 4 hard problems in coding theory which are supposed to have the property that they cannot be solved in polynomial time in the worse case. In other words, this property ensures the existence of some hard instances, not the hardness of every instance. These problems provide the foundation for code-based cryptography.

**Definition 21** (General decoding problem). Given an  $(n, k)$  code  $C$  over  $\mathbb{F}_q$ , an integer  $t_0$  and a word  $x$ , find a codeword  $c \in C$  with  $d(x, c) \leq t_0$ .

**Definition 22** (Syndrome Decoding (SD) Problem). Given a matrix  $H$  and a vector  $s$  (a word), both over  $\mathbb{F}_q$ , and a nonnegative integer  $t_0$ , find a vector  $x \in \mathbb{F}_q^n$  with Hamming weight  $w(x) = t_0$  such that  $Hx^T = s^T$ .

**Definition 23** (Goppa Parameterized Syndrome Decoding (GPSD) problem). Given a binary matrix  $H$  of size  $2^m \times r$  and a syndrome  $s$ , decide whether there exists a codeword  $c$  of weight  $r/m$  such that  $Hc^T = s^T$ .

**Definition 24** (Goppa Code Distinguishing (GD) problem). Given an  $r \times n$  matrix  $H$ , decide whether  $H$  is the parity check matrix of a Goppa code.

To learn more about what you have just read, see [56].

Code-based cryptography is one of the main post-quantum techniques available. Robert McEliece proposed the first code-based cryptosystem in 1978 [42]. It belongs to a very narrow class of public-key primitives<sup>4</sup> that have resisted all cryptanalytic attempts up to now, despite its large key size.

The McEliece public-key encryption scheme was proposed almost 40 years

---

<sup>4</sup>*Cryptographic primitives* are well-established, low-level cryptographic algorithms that are frequently used to build cryptographic protocols for computer security systems.

ago and hasn't been threatened essentially since then. McEliece's original idea was to use as ciphertext a word of a carefully chosen linear error-correcting code (a binary Goppa code, in this case) to which random errors were added. An arbitrary basis of the code — a generator matrix — is the public key, allowing anyone to encrypt (see Figure 5.4).

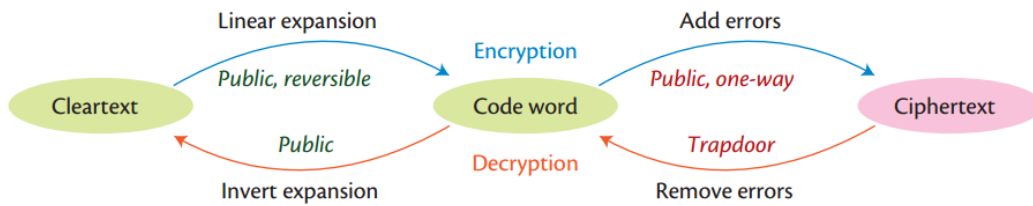


Figure 5.4: Code-based public-key encryption. The ciphertext is a noisy code word that only the legitimate user can correct to recover the cleartext.

Legitimate users who know a secret trapdoor — a fast (that is, polynomial time) decoding algorithm for the code — can remove the errors and recover the cleartext.

Adversaries are reduced to a generic decoding problem, which is believed to be hard on average, including against quantum adversaries.

The McEliece's scheme security relies on two computational assumptions (previously defined):

- hardness of decoding in a random linear code (General decoding problem);
- the public key (a generator matrix) is hard to distinguish from a random matrix (Goppa Code Distinguishing problem).

The first problem, the hardness of generic decoding, is an old problem of coding theory for which only exponential time solutions are known; is also believed to be hard on average. An improvement is possible and would then require an increase in system parameters, but a significant breakthrough is unlikely. Much like factoring and discrete logarithms for number theory-based

cryptosystems, research on this topic must be maintained at the highest level to ensure enough confidence in the system and adjust its parameters when needed.

The second problem, public-key indistinguishability, is much more open. To state it properly, the system must be instantiated. For instance, McEliece proposed using the family of binary Goppa codes, for which the indistinguishability assumption holds so far. For some other families, Reed-Solomon codes, concatenated codes, low-density parity check codes, and so on, the assumption doesn't hold and the corresponding instances of McEliece are unsafe. Providing families of codes for which the indistinguishability assumption holds is a key issue in code-based cryptography.

A quantum resistance of the McEliece scheme is also assumed at the moment. There is a quantum attack strategy against the McEliece public key cryptosystem (McEliece PKC for short), which uses, in the "research step", the Grover's quantum algorithm; however this attack would either require an iterative application of the algorithm (which is not possible) or a memory of size of the whole search space. Thus there is no possibility to significantly speed-up the search step by Grover's algorithm.

Table in Figure 5.5 gives an overview for the advantage of quantum computers over classical computers in attacking the McEliece PKC. One can see, that the expected advantage does not lead to significantly different security estimations for the McEliece PKC.

McEliece parameters $m, t$	Workload Cryptanalysis (in binary operations)		Minimal number of Qubits	Quantum-computer bit security
	classic computer	quantum		
11, 32	$2^{91}$	$2^{86}$	25	80
11, 40	$2^{98}$	$2^{94}$	50	88
12, 22	$2^{93}$	$2^{87}$	29	80
12, 45	$2^{140}$	$2^{133}$	28	128

Figure 5.5: Attacking the McEliece PKC. Table from [47].

Public-key encryption can also be achieved with the Niederreiter scheme [45], which is equivalent to the McEliece scheme in terms of security.

In addition, two other important functionalities can be achieved from codes: *zero-knowledge authentication* and digital signatures.

The first Zero-knowledge authentication protocol was proposed by Jacques Stern in 1993 [57]. Some variants have followed, and all amount to the same idea: one party picks a code word  $x$ , keeps it secret, and publishes a noisy version of it, say  $y = x + e$ , with  $e$  of small weight. Then, this party can prove interactively to another party that it knows a code word close to the public word  $y$  without ever revealing any information about  $x$ .

There's a generic way to produce digital-signature schemes from zero-knowledge protocols using the Fiat-Shamir paradigm [27]. This can be achieved using the Stern protocol. Note that, against quantum adversaries, the construction requires some modifications [61]. The resulting digital signature scheme is easy to implement and enjoys relatively small key sizes (a few hundred bytes) but produces rather large signatures (one or a few hundred kilobytes).

Another method to build digital signatures is the "hash and sign" paradigm in which users consider the digest of the message to be signed as a ciphertext and produce the corresponding cleartext as the signature.

In this scenario, the public key can be used to check the signature's validity. Unfortunately, the McEliece encryption primitive is not invertible (isn't surjective) and therefore it cannot be used for authentication or for signature schemes. Moreover, the scheme uses large public keys, has significant signing complexity, and doesn't scale very well.

## 5.4 Multivariate-equation-based cryptography

A multivariate public key cryptosystem (MPKC for short) has a set of nonlinear (usually quadratic) polynomials over a finite field as its public map. Its main security assumption is backed by the hardness of the problem to solve nonlinear equations over a finite field. The corresponding mathe-

matical structure to a system of polynomial equations, is the ideal generated by those polynomials, so, philosophically speaking, multivariate cryptography relate to mathematics that handles polynomial ideals, namely algebraic geometry.

This family is considered as one of the major families of PKCs that could resist potentially even the powerful quantum computers of the future. There has been fast and intensive development in Multivariate Public Key Cryptography in the last two decades. The original idea of MPKC was presented by Matsumoto and Imai [41], and their scheme is commonly referred to as the MI scheme. After MI scheme was proposed, several encryption systems were proposed. Unfortunately, most of them, including MI, were broken after several security analyses, but, on the other hand, other constructions are still viable. For example, the *Rainbow* (multivariate-quadratic-equation based) scheme for digital signatures [22] is one of the finalists in the NIST post-quantum cryptosystems competition.

Let us analyze the question more formally.

As envisioned by Diffie and Hellman, a public key cryptosystem depends on the existence of class of “*trapdoor one-way functions*”. A trapdoor function is a function that is easy to compute in one direction, yet difficult to compute in the opposite direction (finding its inverse) without a special information, called the *trapdoor*.

In the Multivariate (Public-Key) Cryptography the trapdoor one-way function takes the form of a multivariate quadratic polynomial map over a finite field. Namely the public key is in general given by a tuple  $\mathcal{P}$  of  $m$  nonlinear (usually quadratic) multivariate (in  $w = (w_1, \dots, w_n)$ ) polynomials:

$$\mathcal{P} = (p_1(w), \dots, p_m(w))$$

where

$$z_k = p_k(w) := \sum_i P_{ik} w_i + \sum_i Q_{ik} w_i^2 + \sum_{i>j} R_{ijk} w_i w_j$$

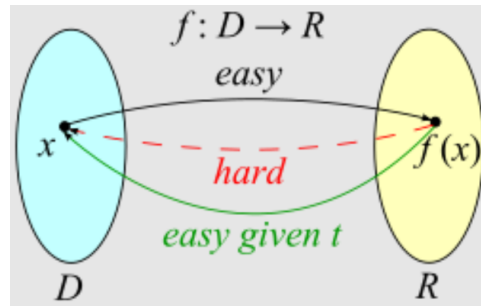


Figure 5.6: The idea of trapdoor function. A trapdoor function  $f$  with its trapdoor  $t$  can be generated by an algorithm.  $f$  can be efficiently computed, i.e., in probabilistic polynomial time. However, the computation of the inverse of  $f$  is generally hard, unless the trapdoor  $t$  is given.

with all coefficients and variables in  $\mathbb{K} = \mathbb{F}_q$ , the field with  $q$  elements. The evaluation of these polynomials at any given value corresponds to either the encryption procedure or the verification procedure (in the case of digital signatures scheme).

Inverting a multivariate quadratic map is equivalent to solving a set of quadratic equations over a finite field, or the following problem:

**Definition 25** (Multivariate quadratic problem (MQ)). Solve the system  $p_1(w) = p_2(w) = \dots = p_m(w) = 0$ , where each  $p_i$  is a quadratic in  $w = (w_1, \dots, w_n)$ . All coefficients and variables are in  $\mathbb{K} = \mathbb{F}_q$ .

There is the analogous version of the problem in which the nonlinear polynomials  $p_i$  are not only quadratic, but of any other degree, and this is called the MP problem (multivariate-polynomials problem).

MQ and MP are in general hard problems (not solvable in polynomial time). The one who is communicating using an MPKC will have to know a particular trapdoor that allows him to invert easily the polynomials in  $\mathcal{P}$ . Of course, a random set of quadratic equations would not have a trapdoor and hence not be usable in an MPKC, so one has to "choose" which maps to use, in order to know the trapdoor.

Since we are no longer dealing with "random" or "generic" systems, but sys-

tems where specific trapdoors exist, the security of MPKCs is then not totally guaranteed by the hardness of MQ, and effective attacks may exist for any chosen trapdoor. The history of MPKCs therefore evolves as we understand more and more about how to design secure, to "hide", multivariate trapdoors.

Even if we restrict ourselves to cryptosystems for which the public key is a  $m$ -tuple of polynomials  $\mathcal{P} = (p_1, \dots, p_m)$  in variables  $w = (w_1, \dots, w_n)$  where all variables and coefficients are in  $\mathbb{F}_q$ , the way to hide the trapdoor is not unique.

We are now going to describe a standard construction of an MPKC scheme. The way used in this construction to "hide" the trapdoor, is a classical one; it is based on the following problem, supposed hard to be solved:

**Definition 26** (Extended Isomorphism of Polynomials problem (EIP)). Given a class  $\mathcal{C}$  of multivariate nonlinear maps (named *central maps*) from  $\mathbb{K}^n$  to  $\mathbb{K}^m$ , and a map  $\mathcal{P}$  expressible as  $\mathcal{P} = T \circ Q \circ S$ , where  $T$  and  $S$  are invertible affine maps (resp. over  $\mathbb{K}^m$  and over  $\mathbb{K}^n$ ) and  $Q \in \mathcal{C}$ , find a decomposition of  $\mathcal{P}$  of the form  $\mathcal{P} = T_0 \circ Q_0 \circ S_0$ , with affine invertible maps  $S_0$  and  $T_0$  and  $Q_0 \in \mathcal{C}$ .

So let us take as public key the quadratic multivariate map  $\mathcal{P} = T \circ Q \circ S : \mathbb{K}^n \rightarrow \mathbb{K}^m$ , where  $Q$  is a central map belonging to a certain class of quadratic maps whose inverse can be computed relatively easily, hidden by the composition with  $S$  and  $T$ :

$$\begin{aligned} \mathcal{P}: \mathbb{K}^n &\xrightarrow{S} \mathbb{K}^n \xrightarrow{Q} \mathbb{K}^m \xrightarrow{T} \mathbb{K}^m \\ w &\mapsto x \quad \mapsto y \quad \mapsto z \end{aligned}$$

where:

$$Q(x_1, \dots, x_n) = Q(x) = (q_1(x), \dots, q_m(x)) = y = (y_1, \dots, y_m)$$



and in particular, because of affinity of  $S$  and  $T$  maps:

$$\begin{aligned}x &= (x_1, \dots, x_n) = S(w) = M_s w + c_s \\z &= (z_1, \dots, z_m) = T(y) = M_t w + c_t\end{aligned}$$

where  $M_s$  and  $M_t$  are full-rank square matrix, and  $c_s, c_t$  are the translation vectors.

The  $x_j$  are called the central variables. The polynomials giving  $y_i$  in  $x$  are called the central polynomials. The key of a MPKC is the design of the central map  $Q$ .

The public key consists of the polynomials in  $\mathcal{P}$ . In practice, this is always the collection of the coefficients of the  $p_i$ 's, compiled in some order conducive to easy computation.

The secret key consists of the informations in  $S$ ,  $T$ , and  $Q$ . That is, we collect  $(M_s^{-1}, c_s)$ ,  $(M_t^{-1}, c_t)$  and whatever parameters there exist in  $Q$ .

To verify a signature or to encrypt a block, one simply computes  $z = P(w)$ . To sign or to decrypt a block, one computes  $y = T^{-1}(z)$ ,  $x = Q^{-1}(y)$  and  $w = S^{-1}(x)$  in turn (in other words, he calculates  $\mathcal{P}^{-1}(z)$ ). Here, calculate  $Q^{-1}(y)$  means finding one (of the possibly many, because in general  $Q$  is not injective) pre-images of  $x$  under the central map  $Q$ .

## 5.5 Supersingular isogeny-based cryptography

In this Section, we focus on the supersingular elliptic curve isogeny-based cryptography as a post-quantum candidate. We will first give a brief general overview of the isogeny-based cryptography; then we will present the question more formally, providing the mathematical prerequisites useful to deal with the subject (in the Subsection 5.5.1), exposing the problems based on elliptic curves considered "difficult" to be solved even by quantum computers (in the Subsection 5.5.3), describing SIDH, the most famous cryptographic

scheme based on elliptic curves, supposed to be quantum-resistant (in the Subsection 5.5.2).

Among the earliest Elliptic Cryptography (ECC) systems, there are those based on the Elliptic Curve Discrete Logarithm Problem (ECDLP<sup>5</sup>; for more details about this Problem, see the Appendix B, Section B.3).

ECDLP is much more difficult than the problem of factorising prime numbers, with the same field size, and therefore with the same security, this cryptography requires smaller public keys, and therefore more easily usable than those used by the RSA method. Despite this, however, ECDLP did not resist Shor's quantum algorithm.

Couveignes [20] and Stolbunov [59] independently discovered a cryptosystem that relies on the computational difficulty of finding *isogenies* between *ordinary*<sup>6</sup> elliptic curves. Soon after Childs et al. [18] found a quantum subexponential attack on this cryptosystem, due to the commutative structure of the endomorphism ring in the ordinary setting. In order to circumvent this attack, De Feo, Jao and Plût [31] proposed to use isogenies between *supersingular* elliptic curves, because the supersingular endomorphism rings are not commutative: thus the SIDH (Supersingular isogeny Diffie–Hellman) key exchange scheme was born. In addition, in supersingular elliptic curves there is the useful property that its ring of endomorphisms on the algebraic closure of a field is particularly large.

De Feo, Jao and Plût proposed also to transmit auxiliary information, that is conjectured not to compromise the security of the scheme. In fact, this cryptosystem remains secure to this day, with the best known attack requiring exponential time with both classical and quantum computers.

Furthermore, it should be noted that the performance of the supersingular

---

<sup>5</sup>ECDLP consists in, fixed an elliptic curve on a finite field and two points  $P, Q$  on it, to find an integer  $k$  such that  $Q = k \cdot P$ . In particular, the meaning of the multiplication  $k \cdot P$  will become clear by reading this section.

<sup>6</sup>Elliptic curve can be of two types: ordinary or supersingular. These two opposing attributes, together with other definition (like "isogeny"), will be specified in the course of this Section.

protocol outperforms the ordinary protocol.

### 5.5.1 Elliptic curves

In this section we assume that the reader is familiar with the basics of abstract algebra and geometry (especially projective geometry).

Throughout this section we denote by  $K$  a field, and by  $\bar{K}$  its algebraic closure. We also let  $\text{char}(K)$  denote the characteristic of  $K$ .

What is written in this section, from now on, is mainly taken from [60], where there are the sources to deepen each topic and the proof of the theorems (which have been omitted here).

**Definition 27** (Elliptic curve). An elliptic curve  $E$  over a field  $K$ , denoted by  $E/K$ , is given by the non-singular (*non-singularity* means that an elliptic curve does not intersect itself) projective curve of the form:

$$E : Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3,$$

for  $a_1, a_2, a_3, a_4, a_6 \in K$ , along with a base point  $\mathcal{O}_E$  of (projective) coordinates  $\mathcal{O}_E = [X, Y, Z] = [0, 1, 0]$ , which is referred to as the *point at infinity*.

So elliptic curves are particular projective curves with a special base point. Throughout this text we will be using the compact short Weierstrass equation form for elliptic curves over  $K$ ; in fact, if we assume that the characteristic of  $K$  is different than 2 and 3 (which is always the case for isogeny-based cryptography), it can be shown that one can transform the elliptic curve equation into:

**Definition 28** (Weierstrass Equation). An elliptic curve defined over  $K$  is the locus in  $\mathbb{P}_2(\bar{K})$  of an equation of the form:

$$E : Y^2Z = X^3 + aXZ^2 + bZ^3$$

with  $a, b \in K$  and  $4a^3 + 27b^2 \neq 0$  (to ensure non-singularity), where  $\mathcal{O}_E = [0, 1, 0]$ .

The point at infinity is the only point on the line  $Z = 0$ , all other points have coordinates  $[x, y, 1]$ . The pairs  $(x, y)$  are the solutions of an affine equation, and are defined as  $x = X/Z$  and  $y = Y/Z$ . Hence, we can rewrite the previous equation in its affine form:

$$E : y^2 = x^3 + ax + b.$$

An important invariant of an elliptic curve is its *discriminant*  $\Delta$ , which is a quantity associated with the cubic equation describing the elliptic curve. As far as our study is concerned, we can reduce ourselves to considering only the short Weierstrass form equations. In this case the discriminant can be defined as:

$$\Delta = -16(4a^3 + 27b^2).$$

One can check the non-singularity condition by computing the discriminant of the equation of the curve, since a cubic polynomial has only simple roots over  $\bar{K}$  if and only if the discriminant is non-zero.

This also explains why the condition  $4a^3 + 27b^2 \neq 0$  in the previous definition is necessary for an elliptic curve.

**Theorem 8** (Bézout's theorem). *Suppose that  $V$  and  $W$  are two plane projective curves defined over a field  $K$  that do not have a common component (this condition means that  $W$  and  $V$  are defined by polynomials, which are not multiples of a common non constant polynomial; in particular, this holds for a pair of "generic" curves). Then the total number of intersection points of  $W$  and  $V$  with coordinates in an algebraically closed field which contains  $K$ , counted with their multiplicities, is equal to the product of the degrees of  $W$  and  $V$ .*

Since an elliptic curve is non-singular and it is defined by a cubic equation, Bézout's theorem tells us that any straight line in  $\mathbb{P}_2(\bar{K})$  intersects the curve in exactly three points.

By requiring that three co-linear points sum to zero, we can define a *group law* that provides elliptic curves with a natural group structure. The group

law can be expressed by rational polynomials with coefficients in  $K$ , and allows us to "add" and "subtract" points, and also to multiply a point by an integer.

**Definition 29** (Group law "+"). Let  $E : y^2 = x^3 + ax + b$  be an elliptic curve. Let  $P_1 = (x_1, y_1)$  and  $P_2 = (x_2, y_2)$  be two points on  $E$  different from the point at infinity; then we can define a group law "+" on  $E$  as follows:

- $P + \mathcal{O}_E = \mathcal{O}_E + P = P$  for any point  $P \in E$ ;
- If  $x_1 = x_2$  and  $y_1 = -y_2$ , then  $P_1 + P_2 = \mathcal{O}_E$  (this allows us to define the *inverse* of a point  $P$ :  $-P = -(x, y) := (x, -y)$ );
- Otherwise, set

$$\lambda := \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{if } P_1 \neq P_2 \\ \frac{2x_1^2 + a}{2y_1} & \text{if } P_1 = P_2 \end{cases}$$

then the point  $P_1 + P_2 = (x_3, y_3)$  is defined by

$$x_3 := \lambda^2 - x_1 - x_2, \quad y_3 := -\lambda x_3 - y_1 + \lambda x_1.$$

One can show that the above law produces an Abelian group, which we denote by  $E(K)$ , with  $\mathcal{O}_E$  acting as the identity element. The  $k$ -th scalar multiple of a point  $P$ ,  $k \cdot P = P + \dots + P$ , will be denoted by  $[k]P$ .

Although we have avoided showing where the group law comes from, it exploits the Bézout's theorem, and it has important geometric bases, that can help make the law more "visual" and intuitive; the more interested reader can deepen these arguments in Appendix B.

**Definition 30** (Multiplication map). Let  $E$  be an elliptic curve over a field  $K$ , and let  $P \in E(\bar{K})$ . For any  $m \in \mathbb{Z}$ , define the map  $[m] : E(\bar{K}) \rightarrow E(\bar{K})$  to be the multiplication-by- $m$  map, where  $[m]P := P + \dots + P$ .

The multiplication map can be defined in a similar way for negative integers:  $[-m]P := [m](-P)$ , where  $-P$  is the inverse of  $P$ .

**Definition 31** (The  $m$ -torsion subgroup). For any  $m \in \mathbb{Z}$ , and an elliptic curve  $E(K)$ , the set

$$E[m] := \{P \in E(\bar{K}) \mid [m]P = \mathcal{O}_E\}$$

is a subgroup of  $E(K)$ , called the  $m$ -torsion subgroup of  $E(K)$ , and an element  $P \in E[m]$  is called an  $m$ -torsion point.

The  $m$ -torsion subgroup can be viewed as the kernel of the multiplication-by- $m$  map, while all the points in  $E[m]$  have order dividing  $m$ .

**Theorem 9.** *Let  $E$  be an elliptic curve over a field  $K$ , and let  $m$  be a positive integer. If  $\text{char}(K) = 0$  or  $\text{char}(K)$  does not divide  $m$ , then*

$$E[m] \cong \mathbb{Z}/m\mathbb{Z} \oplus \mathbb{Z}/m\mathbb{Z}.$$

It follows from this Theorem that there are points  $P, Q \in E[m]$  such that linear combinations of  $P$  and  $Q$  generate all of the group  $E[m]$  as

$$\{aP + bQ \mid a, b \in \mathbb{Z}/m\mathbb{Z}\} = E[m].$$

This property will be useful to define the SIDH keys exchange scheme.

For  $P, Q \in E$ , we denote the set of all linear combinations of  $P$  and  $Q$  by  $\langle P, Q \rangle$ .

Now we are going to study the algebraic relationships, and different kinds of maps, between elliptic curves.

**Definition 32** (Morphism of elliptic curve). Let  $E$  and  $E'$  be two elliptic curves over  $K$ . A *morphism*  $\phi : E \rightarrow E'$  over  $K$  is a polynomial mapping with coefficients from  $K$ . If the curves are in projective coordinates we can write it as:

$$\phi([X, Y, Z]) = [\phi_0[X, Y, Z], \phi_1[X, Y, Z], \phi_2[X, Y, Z]]$$

where  $\phi_0, \phi_1, \phi_2$  are homogeneous polynomials of equal degree satisfying the defining equation of  $E'$ .

If we are using Weierstrass coordinates, a morphism  $\phi$  becomes a rational map:

$$\phi(x, y) = \left( \frac{\phi_0[x, y, 1]}{\phi_2[x, y, 1]}, \frac{\phi_1[x, y, 1]}{\phi_2[x, y, 1]} \right).$$

Each morphism has an integer *degree*, such that a degree  $m$  morphism implies that the kernel of the morphism has cardinality  $m$ . That is, the morphism is  $m$ -to-1 from  $E$  to  $E'$ .

**Definition 33** (Homomorphism of elliptic curve). Let  $E$  and  $E'$  be two elliptic curves over  $K$ . A *homomorphism*  $\phi : E \rightarrow E'$  over  $K$  is a morphism of elliptic curves such that:

$$\phi(P + Q) = \phi(P) + \phi(Q),$$

for all  $P, Q \in E(K)$ . That is,  $\phi$  respects the group law and structure of the curve.

**Theorem 10.** Let  $E$  and  $E'$  be two elliptic curves, and  $\phi : E \rightarrow E'$  a morphism. Then,  $\phi$  is a homomorphism  $\Leftrightarrow \phi(\mathcal{O}_E) = \mathcal{O}_{E'}$ .

**Proposition 11.** The map  $[m]$ ,  $\forall m \in \mathbb{Z}$ , defines an homomorphism from  $E$  to itself (we say that  $[m]$  is an endomorphism). The degree of  $[m]$  is  $m^2$ .

Another important type of morphism is isomorphism, which we can think of as an invertible homomorphism, and can define as follows:

**Definition 34** (Isomorphism of elliptic curves). Let  $E$  and  $E'$  be two elliptic curves. An *isomorphism of elliptic curves*  $\phi : E \rightarrow E'$  is an isomorphism over  $\bar{K}$  of algebraic varieties (see Appendix C)<sup>7</sup> such that  $\phi(\mathcal{O}_E) = \mathcal{O}_{E'}$ . If there is an isomorphism from  $E$  to  $E'$ , then we write  $E \cong E'$  and we say that  $E$  and  $E'$  are isomorphic elliptic curves.

Isomorphism classes of elliptic curves, are labeled by an invariant, called the  $j$ -invariant, whose origins can be traced back to complex analysis.

---

<sup>7</sup>In order to speak about isomorphisms between algebraic varieties (affine or projective) it is necessary to give a certain amount of background. We do not do it here, lest it divert attention from the study of elliptic curves. We include in the Appendix C.

**Definition 35** (*j*-invariant:  $j(E)$ ). Let  $E/K : y^2 = x^3 + ax + b$  be an elliptic curve defined over a field  $K$ , define

$$j(E) = 1728 \frac{4a^3}{4a^3 + 27b^2} \in K$$

to be the *j*-invariant of  $E$ .

**Theorem 12.** *The quantity  $j(E)$  is invariant for a  $K$ -isomorphism class of elliptic curves. The converse is also true for an algebraically closed field. So, two curves are isomorphic over  $\bar{K}$  if and only if they have the same *j*-invariant.*

**Definition 36** (Isogeny). Let  $\phi : E \rightarrow E'$  be a map between elliptic curves. These conditions are equivalent:

- $\phi$  is a surjective group morphism,
- $\phi$  is a group morphism with finite kernel,
- $\phi$  is a non-constant algebraic map of projective varieties such that  $\phi(\mathcal{O}_E) = \mathcal{O}_{E'}$ .

If they hold,  $\phi$  is called an *isogeny*. Two curves are called *isogenous* if there exists an isogeny between them.

Using Theorem 10, we can see that the isogenies are homomorphisms. Additionally, by definition of endomorphism, we see that an endomorphism is an isogeny from a curve  $E$  to itself. Perhaps, the simplest examples of isogenies are the so-called *pointed isomorphisms*, which are basically isomorphisms that preserve the point at infinity of the two curves.

*Example 9.* On any elliptic curve, an isogeny from  $E$  to itself is the map  $[m]$ , for  $m \neq 0$ . So being isogenous is, for example, a reflexive relation.

It may be interesting for the reader to know that being isogenous is actually an *equivalence relation*.

For elliptic curves over finite fields, Tate showed that being isogenous is equivalent to having the same cardinality.



**Theorem 13** (J. Tate). *Let  $n \in \mathbb{N}$ ,  $q = p^n$ , where  $p$  is a prime number; also let  $E/\mathbb{F}_q$  and  $E'/\mathbb{F}_q$  be elliptic curves. Then,  $E$  and  $E'$  are isogenous over  $\mathbb{F}_q$  if and only if*

$$\#E(\mathbb{F}_q) = \#E'(\mathbb{F}_q)$$

**Definition 37** (Supersingular and Ordinary elliptic curve). Let  $E/K$  be an elliptic curve, where  $K = \mathbb{F}_q$ ,  $q = p^r$ ,  $p$  prime number,  $1 \leq r \in \mathbb{N}$ . Then one can show that either:

$$E[p^r] \cong \{0\} \text{ or } E[p^r] \cong \mathbb{Z}/p^r\mathbb{Z}.$$

In the first case,  $E$  is called *supersingular*. Otherwise it is called *ordinary*. In other words, an elliptic curve is supersingular if and only if the group of geometric points of order  $p$  is trivial.

**Theorem 14.** *Let  $\phi : E \rightarrow E'$  be an isogeny.  $E$  is supersingular if and only if  $E'$  is supersingular.  $E$  is ordinary if and only if  $E'$  is ordinary.*

## 5.5.2 Supersingular isogeny Diffie-Hellman key exchange: SIDH

### SIDH PARAMETERS:

Before explaining the SIDH protocol, we need to establish the common public parameters used in these protocols as part of the setup procedure.

- Choose a prime  $p$  of the form  $l_A^{e_A} l_B^{e_B} f \pm 1$ , where  $f$  is a small cofactor. Primes of this form are known to be dense, therefore, for any choice of natural numbers  $l_A, l_B, e_A, e_B$ , the prime number theorem in arithmetic progression guarantees that only  $O(\log(p))$  trials are needed in expectation before a suitable prime is found. It should be noted that generally we take  $l_A = 2$  and  $l_B = 3$ . Currently, the most commonly used prime is  $p = 2^{372} 3^{239} - 1$ .

- Choose a supersingular curve  $E$  over the field  $\mathbb{F}_{p^2}$ , which has cardinality  $(p \mp 1)^2 = (l_A^{e_A} l_B^{e_B} f)^2$ . Such a curve can be computed efficiently using the Bröker's algorithm (see [12]).
- Choose torsion basis  $\{P_A, Q_A\}$  of  $E[l_A^{e_A}]$  and  $\{P_B, Q_B\}$  of  $E[l_B^{e_B}]$  (we are using Theorem 9). This can be computed via a simple randomized algorithm that scales random points of  $E$ , and tests linear independence using an argument called *Weil pairing* (see [60]). This approach succeeds with a very high probability. It should be noted that the choice of basis has no effect on the security.

The parameters  $p, E, l_A, e_A, l_B, e_B, P_A, Q_A, P_B$  and  $Q_B$ , compose the public parameters of the system.

### SIDH PROTOCOL:

Alice and Bob want to securely exchange a secret key  $s_k$ .

1. Alice randomly chooses two elements  $m_A, n_A \in \mathbb{Z}/l_A^{e_A}\mathbb{Z}$  not both congruent to 0, and computes a secret isogeny  $\phi_A : E \rightarrow E_A$  with  $\ker(\phi_A) = \langle R_A \rangle = \langle [m_A]P_A + [n_A]Q_A \rangle$ . To do this, Alice uses formulas due to Vélu which allow to build an isogeny from its ker (see [63]). She transmits to Bob  $E_A$  along with the auxiliary input  $\phi_A(P_B), \phi_A(Q_B)$ , the image of the other torsion base under the secret isogeny.
2. Bob randomly chooses two elements  $m_B, n_B \in \mathbb{Z}/l_B^{e_B}\mathbb{Z}$  not both congruent to 0, and computes, using Vélu's formulas, a secret isogeny  $\phi_B : E \rightarrow E_B$  with  $\ker(\phi_B) = \langle R_B \rangle = \langle [m_B]P_B + [n_B]Q_B \rangle$ . He transmits to Alice,  $E_B$  along with the auxiliary input  $\phi_B(P_A), \phi_B(Q_A)$ .
3. Using the auxiliary input, Alice computes (using Vélu) an isogeny  $\phi'_A : E_B \rightarrow E_{AB}$  with  $\ker(\phi'_A) = \langle S_A \rangle = \langle [m_A]\phi_B(P_A) + [n_A]\phi_B(Q_A) \rangle$ . Bob proceeds analogously to compute an isogeny  $\phi'_B : E_A \rightarrow E_{BA}$  with  $\ker(\phi'_B) = \langle S_B \rangle = \langle [m_B]\phi_A(P_B) + [n_B]\phi_A(Q_B) \rangle$ .

4. It can be seen that the two curves  $E_{AB}$  and  $E_{BA}$  are isomorphic ( $\phi'_A \circ \phi_B \cong \phi'_B \circ \phi_A$ ), and in particular have the same  $j$ -invariant. Hence, the shared key is  $s_k = j(E_{AB}) = j(E_{BA})$ .

Alice	Bob
$m_A, n_A \leftarrow_s \mathbb{Z}/\ell_A^{e_A} \mathbb{Z}$	$m_B, n_B \leftarrow_s \mathbb{Z}/\ell_B^{e_B} \mathbb{Z}$
$R_A \leftarrow [m_A]P_A + [n_A]Q_A$	$R_B \leftarrow [m_B]P_B + [n_B]Q_B$
$\phi_A : E \rightarrow E/\langle R_A \rangle$	$\phi_B : E \rightarrow E/\langle R_B \rangle$
$\phi_A(P_B), \phi_A(Q_B)$	$\phi_B(P_A), \phi_B(Q_A)$
$\xrightarrow{E_A, \phi_A(P_B), \phi_A(Q_B)}$ $\xleftarrow{E_B, \phi_B(P_A), \phi_B(Q_A)}$	
$S_A \leftarrow [m_A]\phi_B(P_A) + [n_A]\phi_B(Q_A)$	$S_B \leftarrow [m_B]\phi_A(P_B) + [n_B]\phi_A(Q_B)$
$\phi'_A : E/\langle R_B \rangle \rightarrow (E/\langle R_B \rangle)/\langle S_A \rangle$	$\phi'_B : E/\langle R_A \rangle \rightarrow (E/\langle R_A \rangle)/\langle S_B \rangle$
$j_A \leftarrow j((E/\langle R_B \rangle)/\langle S_A \rangle)$	$j_B \leftarrow j((E/\langle R_A \rangle)/\langle S_B \rangle)$

Figure 5.7: Scheme of the SIDH protocol

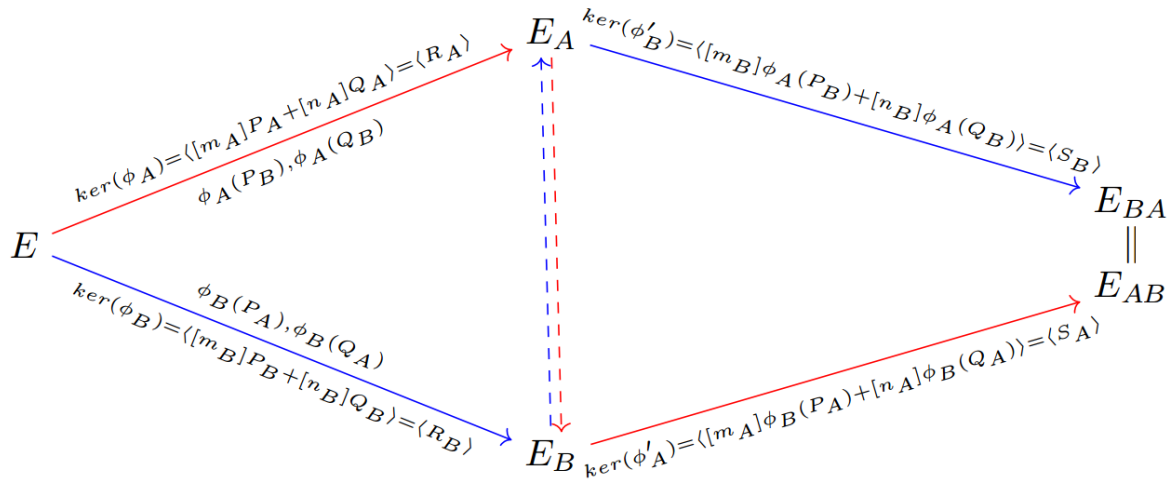


Figure 5.8: Isogeny-Diagram at the base of the SIDH protocol

Note that all the elliptic curves involved are supersingular (using Theorem 14). It should be noted that the auxiliary inputs provide an eavesdropper with the ability to evaluate  $\phi_A$  on all of  $E[\ell_B^{e_B}]$  (respectively,  $\phi_B$  on all of  $E[\ell_A^{e_A}]$ ). Though, it is conjectured that this leakage does not affect the security of the protocols in any critical way, and essentially no information about

$\phi_A$  or  $\phi_B$  is revealed. The key exchange is schematized in Figure 5.7 and depicted in the diagram of the Figure 5.8.

### 5.5.3 Post-Quantum hard problems on elliptic curves

In this subsection we study computational problems that supersingular isogeny-based post-quantum cryptography is based on, which are the problems related to the SIDH key exchange.

We are using the same notation of the SIDH protocol.

**Definition 38** (Supersingular Computational DiffiHellman (SSCDH) Problem). Given the curves  $E_A, E_B$  and the points  $\phi_A(P_B), \phi_A(Q_B), \phi_B(P_A), \phi_B(Q_B)$ , find the  $j$ -invariant of  $E/\langle [m_A]P_A + [n_A]Q_A, [m_B]P_B + [n_B]Q_B \rangle$ .

**Definition 39** (Supersingular Decisional DiffiHellman (SSDDH) Problem). Given a tuple sampled with probability  $1/2$  from one of the following two:

- $(E_A, E_B, \phi_A(P_B), \phi_A(Q_B), \phi_B(P_A), \phi_B(Q_A), E_{AB})$ , where

$$E_{AB} \cong E/\langle [m_A]P_A + [n_A]Q_A, [m_B]P_B + [n_B]Q_B \rangle$$

- $(E_A, E_B, \phi_A(P_B), \phi_A(Q_B), \phi_B(P_A), \phi_B(Q_A), E_C)$ , where

$$E_C \cong E/\langle [m'_A]P_A + [n'_A]Q_A, [m'_B]P_B + [n'_B]Q_B \rangle$$

where  $m'_A, n'_A$  are chosen at random from  $\mathbb{Z}/l_A^{e_A}\mathbb{Z}$  and not both divisible by  $l_A$ , and similarly for  $m'_B, n'_B$ ,

determine from which distribution the tuple is sampled.

These problems are conjectured to be computationally infeasible (in polynomial time) also by quantum computers.

The SIDH key exchange protocol provides some auxiliary information, therefore, these problems are not the most natural isogeny problems. Hence, it is important to contrast them with the more general isogeny computation problem, which does not give any auxiliary information.

**Definition 40** (General Isogeny Computation Problem). Given  $j, j' \in \mathbb{F}_{p^2}$ , find an isogeny  $\phi : E \rightarrow E'$ , if it exists, where  $j(E) = j$  and  $j(E') = j'$ .

Comparing the SIDH problems with the above more general isogeny problem, we notice that the SIDH problems contain much auxiliary information.



## Chapter 6

# Post-quantum Cryptography in embedded world

This chapter studies, from several points of view, post-quantum digital signature schemes selected among the algorithms chosen by NIST for the final phase of its post-quantum competition, and among algorithms selected in another parallel selection made by NIST for hash-based algorithms [19]. The reason for this parallel standardization is that NIST specifically stated that *stateful*<sup>1</sup> schemes were not allowed in the NIST post-quantum competition, because they could not be implemented using the API (Application Program Interfaces) that NIST has defined (which does not allow any state). That would appear to be reasonable, as stateful hash based signature methods do need extra care to implement safely.

The focus on digital signature algorithms is due to the fact that it was in the interests of *Marelli*, the company that supported this thesis, to have a broad overview of these, with the aim of being able to choose one of these to implement on their engine control unit (ECU).

A digital signature is a mathematical scheme for verifying the authenticity of digital messages or documents. Digital signatures employ asymmetric

---

<sup>1</sup>The meaning of the words "stateful" and "stateless" will be clarified in the Subsection 6.1.2.

cryptography. A valid digital signature, where the prerequisites are satisfied, ensures that:

- the message was created by a known sender (*authenticity* property);
- the message was not altered in transit (*integrity* property);
- the signer cannot successfully claim they did not sign a message (*non-repudiation* property).

This chapter begins with the study of the Winternitz signature scheme: it is a famous hash-based digital signature scheme quantum resistant. It is also at the base of XMSS and LMS digital signature protocols: two post-quantum schemes selected by NIST for an early standardization (see [19]).

In the last section of this chapter a comparison between the post-quantum signature algorithms proposed by NIST is provided, in order to select them for a practical use.

## 6.1 Winternitz signature scheme

Winternitz One Time Signature (WOTS) is a famous quantum resistant digital signature scheme. As it is a one-time signature (OTS) scheme, the key can be used to securely sign only one message.

The study of this scheme is important for us since some digital signature algorithms make use of variants of the Winternitz signature scheme.

In the Winternitz signature scheme, the message to be signed is hashed to create a digest for making attacks significantly harder (as specified in [13]) and to have a standard length of the digest to work on. Then each digit of the digest is signed using a hash chain.

Let us describe the scheme.

Suppose Alice wants to digitally sign her message to Bob.

First of all Alice hashes the message  $m$  (using for example the hash function  $H = \text{SHA-256}$ ) which produces a (256-bit) digest:  $H(m)$ . This digest



is encoded as *base b* number (where  $b$  is a fixed integer called *the Winteritz parameter*; often the letter  $w$  is used for  $b$ ). So the digest is  $H(m) = N_0|N_1|\dots|N_k|\dots$  where every  $N_k$  is a digit with its value in  $\{0, 1, \dots, b - 1\}$ .

In the Figure 6.1 we can see how a 256-bit digest is encoded in base  $b = 16$ :

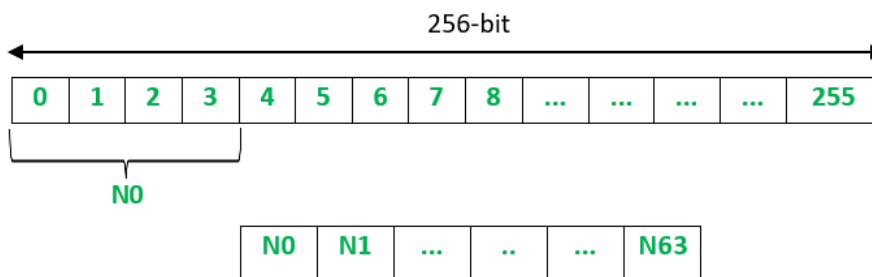


Figure 6.1: 256-bit digest encoded in base  $b = 16$

Once obtained the message digest, the signing process (for every digit of the digest) can be explained in 3 steps: Key Generation, Signature Generation and Signature Verification.

Key Generation:

1. Alice needs to create a key pair for every digit of the digest: private and public key.
  
2. To create the private key for the  $k$ -th digit of the message digest, Alice uses a random number generator to obtain a random number  $x_k$  of a fixed length (number of bits): for the parameter sets approved by the NIST recommendation in [19],  $x_k$  will be either 192 or 256 bits in length. So we assume for example 256 bits.  
This private key  $x_k$  is known only to Alice.
  
3. To create the public key, the  $k$ -th private key  $x_k$  is hashed using an hash function  $H$  (a common choice, also suggested by the NIST in [19], is to take  $H = \text{SHA-256}$ )  $b - 1$  times to obtain another number: so for every  $x_k$  we have its public key:  $H^{b-1}(x_k)$  (also of 256-bit). This total public key (the concatenation of previous)  $H^{b-1}(x_0)|H^{b-1}(x_1)|\dots H^{b-1}(x_k)|\dots$  is shared with Bob.

Unfortunately, the size of similar keys may not be comfortable enough: in fact the length of these key is 256 times the number of digits obtained.

For example, if we take  $b = 4$  or  $b = 16$  (that are common choices) we will have the total public keys of length respectively  $128 \cdot 256 = 32768$  bits (about 4 kilobyte) and  $64 \cdot 256 = 16384$  bits (about 2 kilobyte).

In the figure 6.2 we can observe how to obtain 64 public keys from 64 private keys. These keys will be used for signing a message digest of 256 bit, composed of 64 digits in *base*  $b = 16$ .

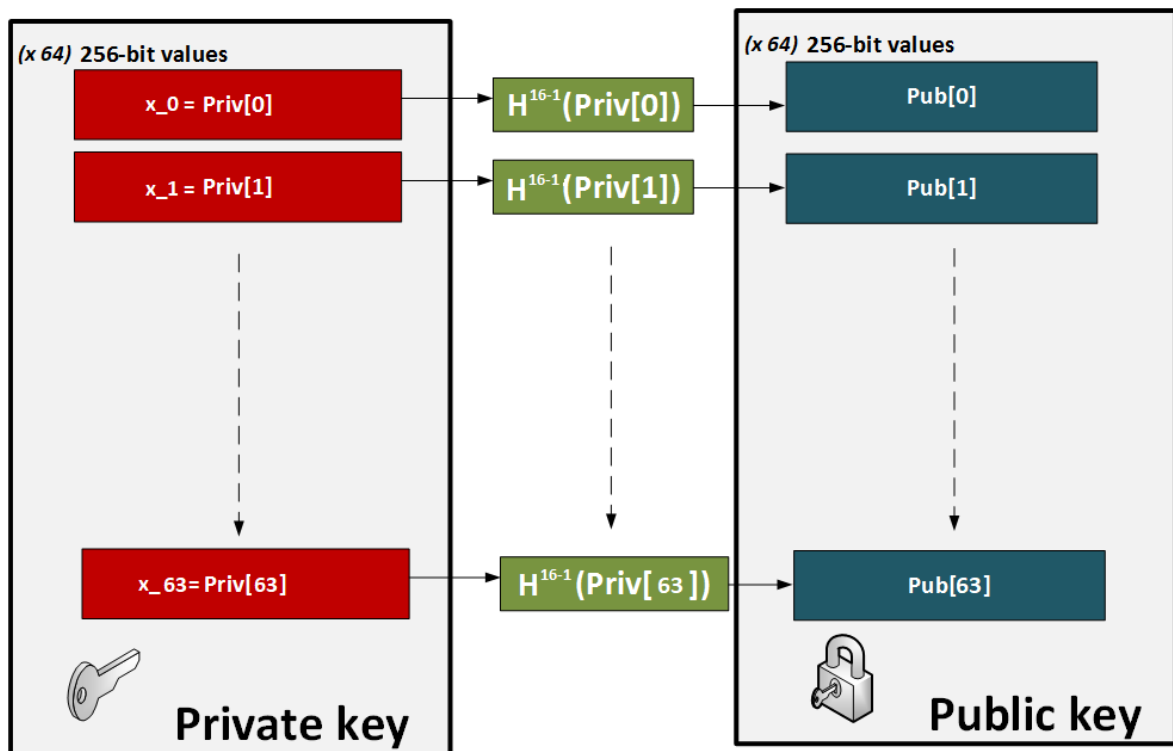


Figure 6.2: 64 keys generation in base  $b = 16$

Signature Generation:

The  $k$ -th digit of the digest,  $N_k$ , is signed by Alice by applying the hash function  $N_k$ -times to the  $k$ -th digit of the private key  $x_k$ . So the signature for the digit  $N_k$  is:  $s_k = H^{N_k}(x_k)$ . After doing this for each digit  $N_k$ , the digital signature for  $m$  is generated:

$$s_0|s_1|\dots|s_k|\dots = H^{N_0}(x_0)|H^{N_1}(x_1)|\dots|H^{N_k}(x_k)|\dots \quad (6.1)$$

Regarding the signature length, it is the same as the public key.

Figure 6.3 shows (like in the previous example, for  $b = 16$ ) how the signature,  $s_k$ , is generated.

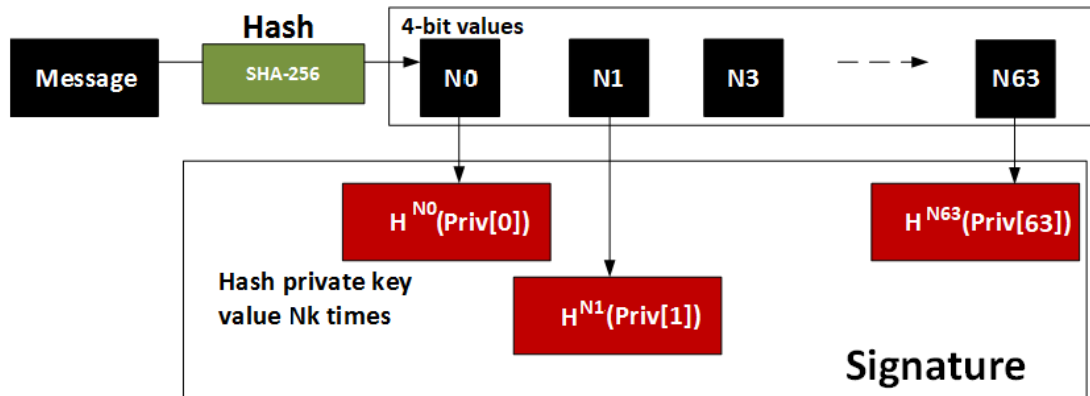


Figure 6.3: Generation of signature for 64 digits in base  $b = 16$

Signature Verification :

Bob knows Alice's public key, and we suppose he reads the digest message.

1. Bob, like Alice did, hashes the message using SHA-256 to produce the digest:  $H(m) = N_0|N_1|\dots|N_k|\dots$  (encoded in base  $b$ ).
2. Bob then hash every signature value  $s_k$ ,  $b - 1 - N_k$  times:  $H^{b-1-N_k}(s_k)$ .
3. Bob compares the result with Alice's public key, checking if  $H^{b-1-N_k}(s_k) == H^{b-1}(x_k)$  for every  $k$ . If they are a match, the signature is valid.

Figure 6.4 shows (like in the previous example, for  $b = 16$ ) how the signature,  $s_k$ , can be verified. For the first digit  $N_0$ , the hash function,  $H$ , is applied to  $s_0 = H^{N_0}(x_0)$ ,  $15 - N_0$  times, and if the resulting value is the same as the public key  $H^{15}(x_0)$ , then the signature is valid:

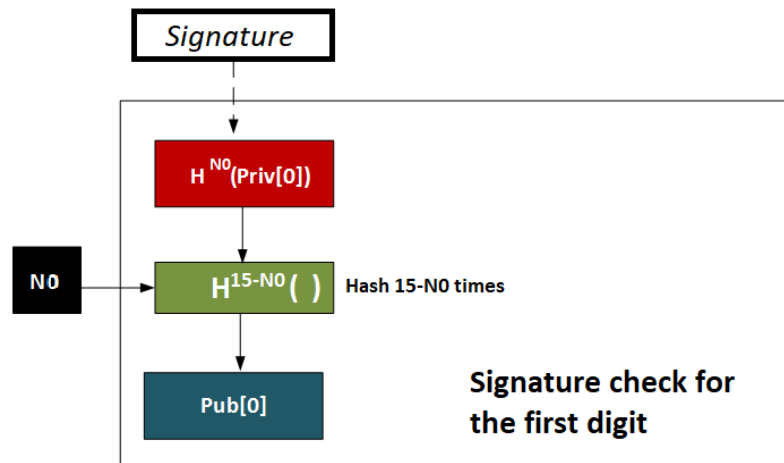


Figure 6.4: Signature check for  $N_0$  in base  $b = 16$

Next figure 6.5 shows an example of a signature for a 32-bit hashed message digest using  $b = 16$  (so the digest has 8 digits). The digest of  $H(m)$  is written as eight hexadecimal digits in the first row.

A separate hash chain is used to sign each digit with each hash chain having its own private key.

	Digest							
Digest	6	3	F	1	E	9	0	B
Private Key	$x_0$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$
Signature	$H^6(x_0)$	$H^3(x_1)$	$H^{15}(x_2)$	$H(x_3)$	$H^{14}(x_4)$	$H^9(x_5)$	$x_6$	$H^{11}(x_7)$
Public Key	$H^{15}(x_0)$	$H^{15}(x_1)$	$H^{15}(x_2)$	$H^{15}(x_3)$	$H^{15}(x_4)$	$H^{15}(x_5)$	$H^{15}(x_6)$	$H^{15}(x_7)$

Figure 6.5: A signature for a digest in base  $b = 16$  (from [19])

### 6.1.1 A possible attack protected by a checksum

Simply signing the individual digits of the digest is not sufficient because an attacker would be able to generate valid signatures for other message digests.

For example, given  $s_k = H^{N_k}(x_k)$  read by an attacker, he would be able to generate a signature for a message digest with a k-th digit of  $N_k + 1$  (instead of  $N_k$ ) by applying  $H$  to  $s_k$  once, or to a message digest with a kth digit of  $N_k + 2$  by applying  $H$  to  $s_k$  twice.

An attacker could not, however, generate a signature for a message digest with a kth digit minor than  $N_k$  as this would require finding some value  $y$  such that  $H(y) = s_k$ , which would not be feasible as long as  $H$  is preimage-resistant.

In order to prevent the above attack, the Winternitz signature scheme computes a checksum of the message digest and signs the checksum along with the digest. For an n-digit message digest, the checksum is computed as:

$$\sum_{k=0}^{n-1} (b - 1 - N_k) \tag{6.2}$$

The checksum is designed so that the value is non-negative, and any increase in a digit in the message digest will result in the decreasing of the checksum. This prevents an attacker from creating an effective forgery from a message signature since the attacker can only increase values within the message digest and cannot decrease values within the checksum.

If we compute the checksum for the example in Figure 6.5, we obtain the situation of Figure 6.6:

	Digest								Checksum	
Digest	6	3	F	1	E	9	0	B	3	D
Private Key	$x_0$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	$x_8$	$x_9$
Signature	$H^6(x_0)$	$H^3(x_1)$	$H^{15}(x_2)$	$H(x_3)$	$H^{14}(x_4)$	$H^9(x_5)$	$x_6$	$H^{11}(x_7)$	$H^3(x_8)$	$H^{13}(x_9)$
Public Key	$H^{15}(x_0)$	$H^{15}(x_1)$	$H^{15}(x_2)$	$H^{15}(x_3)$	$H^{15}(x_4)$	$H^{15}(x_5)$	$H^{15}(x_6)$	$H^{15}(x_7)$	$H^{15}(x_8)$	$H^{15}(x_9)$

Figure 6.6: Example of a signature and its checksum (from [19])

### 6.1.2 Key re-using: a problem for WOTS

As we said, the Winternitz scheme is One-Time-Signature scheme: the keys must be ‘disposable’.

This creates memory problems since one always have to create new keys. In particular, it is necessary to worry about using new keys that are always different from the previous ones, and therefore you have to worry about the previous "state"; in this sense we will say that the OTS schemes, therefore the WOTS, are *stateful*.

In the embedded world this memory problem could be uncomfortable for some users. The first question that comes to mind is: could the ‘ONE’-Time Keys become ‘FEW’-Times Keys? Can we re-use these keys more thane once without running into problems? Without making these schemes weaker and exposing us to possible attacks?

Unfortunately the answer is no: measures have to be taken that prevent OTS-key reuse in any case. This answer is given to us by [13] in which a study was carried out about this question. In particular they studied the case of just a double re-use of a OTS-key.

For WOTS with a Winternitz parameter of  $w = 16$  for a 256-bit message digest, if we use as a unit of measurement of the complexity of the attack, the computations of a hash function, there’s, for example, an attack with an attack complexity of  $2^{34}$  hash function computations. This can be done on a modern computer within few days, if not hours. For bigger values of  $w$ , the

attack complexity goes down even further.

However, as soon as we are considering attacks on quantum-computers, complexities drop at least by a square-root factor.

In [13] is analyzed the problem of how to define security for a signature scheme. Security notions are splitted into the goals an adversary (giving a name, Eva) has to achieve and the attack capabilities given to Eva.

For the goals, the relevant notions are (NB: *strong unforgeability* is omitted as it is irrelevant for this context):

- *Full break (FB)*: Eva can compute the secret key.
- *Universal forgery (UU)*: Eva can forge a signature for any given message. Eva can efficiently answer any signing query.
- *Selective forgery (SU)*: Eva can forge a signature for some message of its choice. In this case Eva commits itself to a message before the attack starts.
- *Existential forgery (EU)*: Eva can forge a signature for one arbitrary message. Eva might output a forgery for any message for which it did not learn the signature from a oracle during the attack.

On the other hand, for the attacks we got (*key-only attacks* are omitted as these allow for no signature queries at all):

- *Random message attack (RMA)*: Eva learns the public key and the signatures on a set of random messages.
- *Adaptively chosen message attack (CMA)*: Eva learns the public key and is allowed to adaptively ask for the signatures on messages of its choice.

Theoretical results of the computational complexity for two-message attacks against the Winternitz OTS are expressed in figure 6.7. The letter  $w$



is used for the *Winternitz parameter*, the letter  $m$  for the message digest length.

Security Goal	Attack Complexity	Pr[Success]
EU-CMA	$\mathcal{O}\left(\left(\frac{(w+1)(4w+1)}{6w^2}\right)^{-\frac{m+\log m}{3\log w}}\right)$	$\frac{1}{2}$
SU-CMA	$\mathcal{O}\left(\left(\frac{(w+1)(4w+1)}{6w^2}\right)^{-\frac{m+\log m}{3\log w}}\right)$	$\frac{1}{2}$
UU-CMA	$\mathcal{O}\left(\left(1 - \left(\frac{w-1}{w}\right)^2\right)^{-\frac{m+\log m}{2\log w}}\right)$	$\frac{1}{2}$
FB-CMA	$\mathcal{O}\left(\left(1 - \left(\frac{w-1}{w}\right)^2\right)^{-\frac{m+\log m}{\log w}}\right)$	$\frac{1}{2}$
EU-RMA	$\mathcal{O}\left(\left(\frac{(w+1)(4w+1)}{6w^2}\right)^{-\frac{m+\log m}{\log w}}\right)$	$\frac{1}{2}$
SU-RMA	$\mathcal{O}\left(\left(\frac{1}{w}\right)^{-\frac{m+\log m}{\log w}}\right)$	$\frac{1}{2}$
UU-RMA	-	$\left(\frac{(w+1)(4w+1)}{6w^2}\right)^{\frac{m+\log m}{\log w}}$
FB-RMA	-	$\left(1 - \left(\frac{w-1}{w}\right)^2\right)^{\frac{m+\log m}{\log w}}$

Figure 6.7: Theoretical results of the computational complexity for two-message attacks against the Winternitz OTS. If the success probability of an attack is not constant in terms of complexity, the attack complexity is given to achieve a success probability of 1/2. (Table from: [13])

For example, take the *EU-RMA* (*Existential forgery-Random message attack*) case: the adversary gets a signature of two random messages ( $M_1, M_2$ ) and has to find a third message  $M_3$  that is covered by  $M_1, M_2$ . (The difference to the CMA case is that Eva cannot optimize the choice of  $M_1, M_2$ ).

How many message the adversary has to search, to forge a third signature?

It turns out that two messages cover a third one with probability of:

$$P[\text{break}(M_1, M_2, M_3) = 1] \approx \left(\frac{(w+1)(4w-1)}{6w^2}\right)^{\frac{m+\log m}{\log w}} \quad (6.3)$$

This means that when an attacker receives two signatures of two random messages, it has to compute about  $\left(\frac{(w+1)(4w-1)}{6w^2}\right)^{\frac{m+\log m}{\log w}}$  messages to find a

covered third message.

For the common choices of  $m = 256$  and  $w = 16$ , this number is approximately  $2^{34}$ , which can be done within a few days on today's CPUs.

## 6.2 XMSS and LMS

There are two famous WOTS-based post quantum digital signature schemes selected by NIST in [19]: XMSS [15] and LMS [36].

At a high level, XMSS and LMS are very similar. They each consist of two components: a one time signature scheme (a variant of WOTS) and a method for creating a single, long-term public key from a large set of OTS public keys.

While a single, long-term public key for a OTS scheme could be created from a large set of OTS public keys by simply concatenating the keys together, the resulting public key would be unacceptably large. Let us now examine the second part of the schemes, the one relating to the creation of the long-term key.

### 6.2.1 Long-term public key economize: Merkle tree

Instead of concatenating the OTS public keys, it could be used a *Merkle hash tree* (like XMSS and LMS do), which allow for the long-term public key to be shorter in exchange for requiring an additional amount of information to be provided with each OTS key.

To create this tree, the OTS public keys are hashed once to form the leaves of the tree, and these hashes are then hashed together (concatenated in pairs) to form the next level up. Those hash values are then hashed together in pairs, the resulting hash values are hashed together, and so on until all of the public keys have been used to generate a single hash value (the root of the tree), which will be used as the long-term public key.

The figure 6.8 depicts a hash tree that contains eight OTS public keys  $k_0, k_1, \dots, k_7$ .

The eight keys are each hashed to form the leaves of the tree  $h_0, \dots, h_7$ , and the eight leaf values are hashed concatenated in pairs to create the next level up in the tree:  $h_{01}, h_{23}, h_{45}, h_{67}$ . These four hash values are again hashed in pairs to create  $h_{0-3}$  and  $h_{4-7}$ , which are hashed together to create the long-term public key:  $h_{0-7}$ . This long-term public key (the root) is the trusted point and it must be known to the verifier Bob.

The authenticity of Alice's  $i$ -th key can be tested by retracing the tree, recalculating the hashes necessary, up to the root and then verifying their equivalence with the known root.

For example if Bob received by Alice  $h_{0-7}$  as long-term public key, and he wants to verify a message signed using  $k_2$  (also communicated before by Alice with Bob), Alice would need to provide  $h_3, h_{01}$ , and  $h_{4-7}$  in addition to  $k_2$ . The verifier Bob, would compute  $h'_2 := H(k_2)$ ,  $h'_{23} := H(h'_2 \| h_3)$ ,  $h'_{0-3} := H(h_{01} \| h'_{23})$ , and  $h'_{0-7} := H(h'_{0-3} \| h_{4-7})$ .

If  $h'_{0-7}$  is the same as  $h_{0-7}$ , then  $k_2$  may be used to verify the message signature.

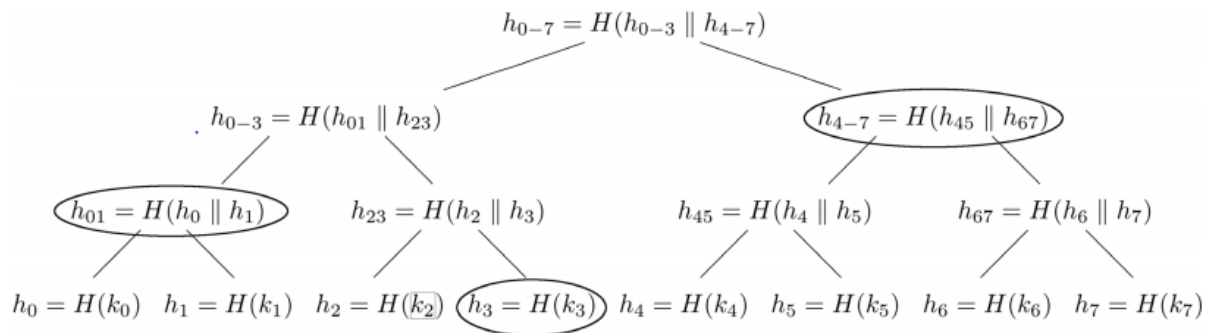


Figure 6.8: A Merkle Hash Tree

The idea of hashes tree could be generalized to form a multi-Merkle-tree: a tree of several layers of Merkle trees. Both XMSS and LMS define single tree and multi-tree variants of their signature schemes.

The trees on top and intermediate layers are used to sign the root nodes of the trees on the respective layer below. Only the bottommost tree (on the

lowest layer) is used to sign messages.

In an instance that involves two levels of trees, as shown in Figure 6.9, the OTS keys that form the leaves of the top-level tree sign the roots of the trees at the bottom level, and the OTS keys that form the leaves of the bottom-level trees are used to sign the messages.

The root of the top-level tree is the long-term public key for the signature scheme.

As described in Section 7 of [19], the use of two levels of trees can make it easier to distribute OTS keys across multiple cryptographic modules in order to protect against private key loss. A set of OTS keys can be created in one cryptographic module, and the root of the Merkle tree formed from these keys can be published as the public key for the signature scheme. OTS keys can then be created on multiple other cryptographic modules with a separate Merkle tree created for the OTS keys of each of the other cryptographic modules, and a different OTS key from the first cryptographic module can be used to sign each of the roots of the other cryptographic modules.

While there are benefits in the use of a two-level tree, it results in larger signatures and slower signature verification as each message signature will need to include two OTS signatures.

For example, if a message were signed using the OTS key  $k_{b,6}$  in Figure 6.9, the signature would need to include the signature on  $root_{b,1}$  using  $k_{a,1}$  in addition to the signature on the message using  $k_{b,6}$ .

This possible strategy to economize on the size of the keys, however, does not lighten the Winternitz scheme enough: this is why LMS and XMSS are still practically not very usable in most real situations.

### 6.2.2 XMSS: The eXtended Merkle Signature Scheme

Since XMSS and LMS are very similar signature scheme, and have similar performances at a high level, in this thesis it was chosen to deepen only one of them: XMSS. This subsection is studied from [15].

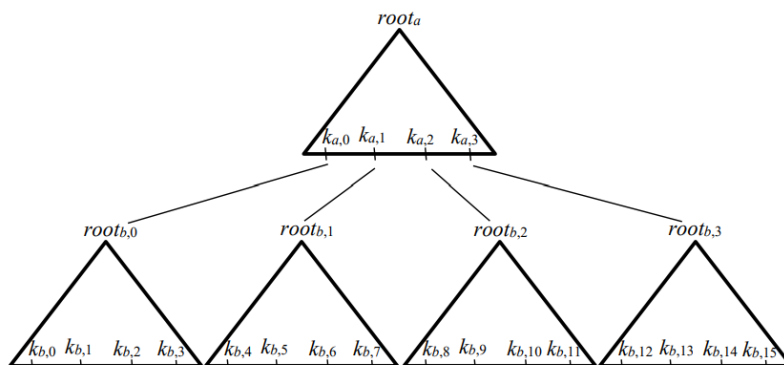


Figure 6.9: Multi level Merkle's tree

The parameters of XMSS are the following:

- $n \in \mathbb{N}$ , the security parameter,
- $\omega \in \mathbb{N}, \omega > 1$ , the Winternitz parameter,
- $m \in \mathbb{N}$ , the message length in bits,
- $F_n = \{f_K : \{0, 1\}^n \rightarrow \{0, 1\}^n | K \in \{0, 1\}^n\}$  a function family,
- $H \in \mathbb{N}$ , the tree height. XMSS allows to make  $2^H$  signatures using one keypair,
- $h_K$ , a hash function, chosen randomly with the uniform distribution from the family  $\mathcal{H}_n = \{h_K : \{0, 1\}^{2n} \rightarrow \{0, 1\}^n | K \in \{0, 1\}^n\}$ ,
- $x \in \{0, 1\}^n$ , chosen randomly with the uniform distribution. The string  $x$  is used to construct the one-time verification keys.

Those parameters are supposed publicly known.

In this thesis, is written  $\log$  for  $\log_2$ .

As mentioned above, in XMSS it is used WOTS in a slightly modified version proposed in [14], sometimes called WOTS-PRF (from Pseudo-Random Function).

For  $K, x \in \{0, 1\}^n$ ,  $e \in \mathbb{N}$  and  $f_K \in F_n$ ,  $f_K^e(x)$  is defined as follows.

We set  $f_K^0(x) := K$  and for  $e > 0$  we define  $K' := f_K^{e-1}(x)$  and  $f_K^e(x) := f_{K'}(x)$ .

In contrast to previous versions of WOTS this is a (random) walk through the function family instead of an iterated evaluation of a hash function. This modification allows to eliminate the need for a collision resistant hash function family. Also, define:

$$l_1 = \left\lceil \frac{m}{\log(\omega)} \right\rceil, l_2 = \left\lceil \frac{\log(l_1(\omega - 1))}{\log(\omega)} \right\rceil + 1, l = l_1 + l_2.$$

The secret signature key of WOTS consists of  $l$   $n$ -bit strings  $sk_i$ ,  $1 \leq i \leq l$  chosen uniformly at random. The public verification key is computed as:

$$pk = (pk_0, pk_1, \dots, pk_l) := (x, f_{sk_1}^{\omega-1}(x), \dots, f_{sk_l}^{\omega-1}(x)),$$

with  $f^{w-1}$  as defined above.

WOTS signs messages of binary length  $m$ . They are processed in base  $w$  representation. They are of the form  $M = (M_1, \dots, M_{l_1})$ ,  $M_i \in \{0, \dots, \omega-1\}$ . The checksum  $C = \sum_{i=1}^{l_1} (\omega - 1 - M_i)$  in base  $\omega$  representation is appended to  $M$ . It is of length  $l_2$ . The result is  $M||C =: (b_1, \dots, b_l)$ . The signature of  $M$  is

$$\sigma = (\sigma_1, \dots, \sigma_l) := (f_{sk_1}^{b_1}(x), \dots, f_{sk_l}^{b_l}(x)).$$

The signature is verified by constructing  $(b_1, \dots, b_l)$  and checking if:

$$(f_{\sigma_1}^{\omega-1-b_1}(pk_0), \dots, f_{\sigma_l}^{\omega-1-b_l}(pk_0)) == (pk_1, \dots, pk_l).$$

The sizes of signature, public, and secret key are  $ln$ .

For more detailed information see [14].

Let us now study the use of Merkle Hash Trees in XMSS.

It utilizes the hash function  $h_K$ . The XMSS tree is a binary tree. Denote its height by  $H$ . It has  $H + 1$  levels. The leaves are on level 0. The root is on level  $H$ . The nodes on level  $j$ , where  $0 \leq j \leq H$ , are denoted by  $Node_{i,j}$ , where  $0 \leq i \leq 2^{H-j}$ . The construction of the leaves is explained below. Level

$j$  for  $1 \leq j \leq H$  is constructed using a bitmask  $(b_{l,j} || b_{r,j}) \in \{0, 1\}^{2^n}$  chosen uniformly at random. The nodes are computed as:

$$Node_{i,j} := h_K((Node_{2i,j-1} \oplus b_{l,j}) || (Node_{2i+1,j-1} \oplus b_{r,j}))$$

for  $1 \leq j \leq H$  (the operator  $\oplus$  is used to indicate XOR bitmasking). The usage of the bitmasks is the main difference to the other Merkle tree constructions. It allows to replace the collision resistant hash function family. Figure 6.10 shows the construction of the XMSS tree.

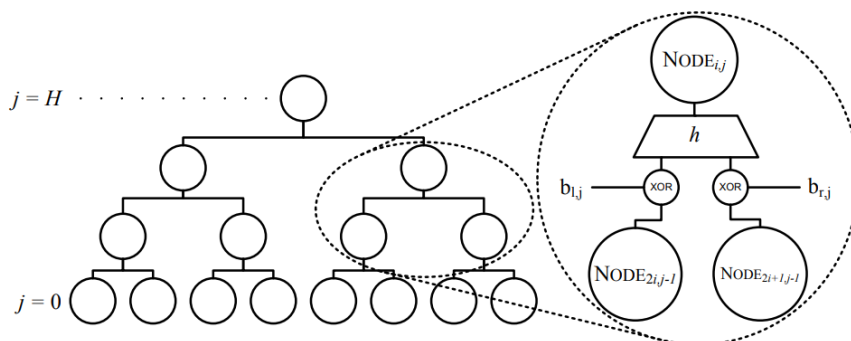


Figure 6.10: The XMSS tree construction

We explain the computation of the leaves of the XMSS tree. The XMSS tree is used to authenticate  $2^H$  WOTS verification keys, each of which is used to construct one leaf of the XMSS tree. The construction of the keys is explained at the end of this subsection. In the construction of a leaf another XMSS tree is used. It is called L-tree. The first  $l$  leaves of an L-tree are the  $l$  bit strings  $(pk_0, pk_1, \dots, pk_l)$  from the corresponding verification key. As  $l$  might not be a power of 2 there are not sufficiently many leaves. Therefore the construction is modified. A node that has no right sibling is lifted to a higher level of the L-tree until it becomes the right sibling of another node. In this construction, the same hash function as above but new bitmasks are used. The bitmasks are the same for each of those trees. As L-trees have height  $\lceil \log(l) \rceil$ , additional  $\lceil \log(l) \rceil$  bitmasks are required. The XMSS public key  $PK$  contains the bitmasks and the root of the XMSS tree.





Let us now see the Signature Key Generation.

The WOTS secret signature keys are computed using a seed  $Seed \in \{0, 1\}^n$ , the pseudorandom function family  $F_n$ , and the pseudorandom generator  $GEN$  which for  $\lambda \in \mathbb{N}, \mu \in \{0, 1\}^n$  yields

$$GEN_\lambda(\mu) = f_\mu(1) \parallel \dots \parallel f_\mu(\lambda).$$

For  $i \in \{1, \dots, 2^H\}$  the  $i$ -th WOTS signature key is

$$sk_i \leftarrow GEN_i(f_{Seed}(i)).$$

The XMSS secret key contains  $Seed$  and the index of the last signature  $i$ .

The bit length of the XMSS public key is  $(2(H + \lceil \log(l) \rceil + 1)n)$  an XMSS signature has length  $(l + H)n$  and the length of the XMSS secret signature key is  $< 2n$ .

In [15] it is also shown that XMSS is provably secure in the standard model and it is discussed the minimality of the used assumptions. In particular the following two theorems are proved: the first one enforces the EU-CMA security; the second one enforces the forward secrecy.

The forward secrecy is a feature of specific key agreement protocols that gives assurances that session keys will not be compromised (all signatures created before remain valid) even if long-term secrets used in the session key exchange are compromised. Obviously, this notion is only meaningful for key evolving signature schemes that change their secret key over time. From an attacker based point of view this translates to: if an attacker learns the actual secret key  $sk_i$ , he is still not able to forge a signature under a secret key  $sk_j$ , with  $j < i$ . This is a desirable property, especially in the context of long term secure signatures, as it allows to remove the need for timestamps and an online trusted third party.

**Theorem 15.** *If  $\mathcal{H}_n$  is a second preimage resistant hash function family (i.e., given  $y$ , for  $h \in \mathcal{H}_n$ , an adversary can find an  $x$  such that  $h(x) = y$  with a negligible probability) and  $F_n$  a pseudorandom function family, then*

*XMSS is existentially unforgeable under chosen message attacks (EU-CMA security).*

**Theorem 16.** *If  $\mathcal{H}_n$  is a second preimage resistant hash function family and  $F_n$  a pseudorandom function family, then XMSS (with the slightly modified key generation described in 4th section of [15]) is a forward secure digital signature scheme.*

## 6.3 Comparison of digital signature algorithms for practical use

### 6.3.1 A comparison between XMSS and LMS

The caveat of XMSS and LMS is that a private key can be used only once to sign a message.

Instead of concatenating the OTS public keys, it could be used a Merkle hash tree, as we saw earlier, which allow for the long-term public key to be shorter in exchange for requiring an additional amount of information to be provided with each OTS key.

The maximum number of messages an HBS (Hash Based Signatures) tree can sign is  $2^{H_{tree}}$ , where  $H_{tree}$  is the height of the tree.

For a tree of height  $H_{tree} = 40$  or more, key generation would be completely impractical, as the entire tree would need to be generated in order to form the root.

So the proposal is to concatenate, in turn, several Merkle hash trees, in order to create a Multi level Merkle's tree.

The maximum number of signed messages of the multi-level tree is  $2^{(H_{tree_1} + H_{tree_2} + \dots + H_{tree_Z})}$ , where  $H_{tree_1}, H_{tree_2}, \dots, H_{tree_Z}$  are the heights of the subtrees at level 1, 2, ..., Z.

With a multi-level tree the height of each tree remains reasonable.

Key generation can be done in time  $O(2^{H_{tree_1}} + O(2^{H_{tree_2}} + \dots + O(2^{H_{tree_Z}}))$  hashes computation (the count of the number of hash compression done

in these algorithms, is a good parameter to study the performance of the schemes), which is considerably smaller than the number of signatures that can be generated. Signing and verifying can also be done with a relatively small cost increase.

On the other hand, multi-level trees do increase the length of the signature: the new signature consists of the OTS signature, and now we include an OTS signature for each tree level.

In fact one important disadvantage of post-quantum signatures is their size. In particular, stateful hash based signatures schemes (like XMSS and LMS, that are OTS) have large signatures that make them impractical in some scenarios.

LMS and XMSS use different notation to specify the same parameters.

We will use a common notation for various parameters which is summarized in Figure 6.12.

Symbols	Meaning
$w$	The Winternitz parameter, which is the base that the initial hash is interpreted. In XMSS, this is the value $w$ . In LMS, this is the value $2^w$ .
$n$	The length of the hash (in bytes). In XMSS, this is the value $n$ . In LMS, both $n$ and $m$ stand for this.
$p$	The number of Winternitz chains used in a single OTS operation. In XMSS, this is the value $len$ . In LMS, this is the value $p$ .
$h$	The height of a single Merkle tree. Both LMS and XMSS notate this as $h$ when dealing with a single level tree.
$\Sigma p$	This is the total number of Winternitz chains used in all levels of a multilevel tree. In XMSS <sup>MT</sup> , this is the value $d \cdot len$ . In HSS, this is the sum of all the $p$ values across all the levels.
$\Sigma h$	This is the total height of all the Merkle trees. In XMSS <sup>MT</sup> , this is the value $h$ . In HSS, this is the sum of all the $h$ values across all the levels.
$d$	This is the number of Merkle trees in the multi-level tree. In XMSS <sup>MT</sup> , this is the value $d$ . In HSS, this is $L$ .

Figure 6.12: Meanings for parameteres used in this Subsection.

The public key and signature sizes for LMS and XMSS and their corre-

sponding multi-level version are shown in Figure 6.13. Currently, the size of the public keys depends on only the hash function.

Even though it is not obvious at first, the signature sizes of LMS are slightly bigger than XMSS, but not significantly.

	Public Key	Signature
LMS	$24 + n$	$12 + n(p + h + 1)$
XMSS	$4 + 2n$	$4 + n(p + h + 1)$
HSS	$28 + n$	$(36d + 2nd - n - 20) + n(\Sigma p + \Sigma h)$
XMSS <sup>MT</sup>	$4 + 2n$	$\lceil \Sigma h / 8 \rceil + n(\Sigma p + \Sigma h + 1)$

Figure 6.13: Sizes (in bytes) of HBS schemes based on schemes parameters.

To quantify the public key and signature sizes differences between LMS and XMSS, we use the tables in Figure 6.14. The tables summarize sizes for the two schemes that can sign the same amount of total messages, at the same security level with the same OTS signature length.

	Public Key	Signature
LMS	56	2508
XMSS	68	2500
HSS	60	5076
XMSS <sup>MT</sup>	68	4963

(a)  $w = 16, p = 67, 2^{10}$  LMS / XMSS and  $2^{20}$  HSS / XMSS<sup>MT</sup> total messages (2 levels)

	Public Key	Signature
LMS	56	2828
XMSS	68	2820
HSS	60	5716
XMSS <sup>MT</sup>	68	5605

(b)  $w = 16, p = 67, 2^{20}$  LMS / XMSS and  $2^{40}$  HSS / XMSS<sup>MT</sup> total messages (2 levels)

	Public Key	Signature
HSS	60	8600
XMSS <sup>MT</sup>	68	8392

(c)  $w = 16, p = 67, 2^{60}$  HSS / XMSS<sup>MT</sup> total messages (3 levels)

	Public Key	Signature
HSS	60	15533
XMSS <sup>MT</sup>	68	14824

(d)  $w = 16, p = 67, 2^{60}$  HSS / XMSS<sup>MT</sup> total messages (6 levels)

Figure 6.14: Sizes (in bytes) of HBS schemes based for various parameters, and  $n = m = 32$ .

In particular: in table (a), both schemes (XMSS, LMS) can sign  $2^{10}$  mes-

sages, and their multi-level version (XMSS<sup>MT</sup>,HSS) (2 level-tree) can sign  $2^{20}$  messages; in table (b), both schemes (XMSS,LMS) can sign  $2^{20}$  messages, and their multi-level version (XMSS<sup>MT</sup>,HSS) (2 level-tree) can sign  $2^{40}$  messages; similarly, tables (c) and (d), provide the public key and signature sizes for more ( $2^{60}$ ) total multi-level signed messages and a tree height of 20 and 10 respectively (3 levels and 6 levels multi tree respectively).

In these schemes most of the time used for the algorithm is spent doing hash compressions. In [32], they made measurements of the percentage of time spent; these can be seen in Figure 6.15. This suggests that the count of the number of hash compression done in the algorithm, is a good parameter to study the performance of the schemes.

Operation	Hash Compression for OTS	Other Operations
LMS Key Gen	94.4%	5.6%
XMSS Key Gen	88.1%	11.9%
LMS Sig Gen	93.6%	6.4%
XMS Sig Gen	88.0%	12.0%
LMS Sig Ver	90.1%	9.9%
XMSS Sig Ver	85.8%	14.2%

Figure 6.15: Percentage of time spent doing hash compression operations during an OTS computation.

In particular, Figure 6.16 contains the count of the number of compression hashes performed by the algorithms, and the ratio for comparing XMSS and LMS.

To experimentally confirm their performance analysis, in [32] they measured the overall performance on the LMS and XMSS implementations (measuring the time per operation). The ratios measured are mostly in line with their analysis. The XMSS signature operations are somewhat more expensive than expected, but well within the range they would expect from implementation details.

Operation	LMS	XMSS	XMSS / LMS ratio
OTS PK Gen	603.5	2473	4.10
OTS Sign	318.25	1072	3.37
OTS Verify	285.25	1401	4.91

Figure 6.16: Number of hash compression operations expected for various OTS operations SHA-256,  $w = 16$ .

The LMS and XMSS standards are similar. They address the same issues and provide post-quantum secure digital signatures that could find different applications in practice. Thus, various protocols and implementers might find it hard to decide between the two in order to pick the more suitable scheme for their usecase.

Both XMSS and LMS make similar assumptions on their hash function; for both LMS and XMSS, a second preimage attack on the hash function would allow a single forgery, and a preimage attack that allowed the attacker to specify all but  $n$  bits would allow the attacker to generate his own Merkle tree based on a public key, allowing him to sign as many messages as he wished.

Where they differ is that XMSS strives to provide the hash function with random independent inputs for every hash evaluation; while LMS has inputs with predictable changes. This difference allows a tighter proof model for XMSS' tree hierarchy (because the attacker has to find a preimage of a hash of random inputs).

On the other hand, both systems achieve the same security level during the initial message hashing (with both LMS and XMSS providing an unpredictable prefix); as this requires a stronger assumption of the hash function (second preimage resistance), it's debatable whether XMSS' tighter proof model for the tree hierarchy is important.

Other factors to consider when making a decision between XMSS and LMS

are the signature and public key sizes, and the computation time. As seen above, only XMSS<sup>MT</sup> has slightly smaller signature sizes than HSS, while LMS performs significantly faster. In addition, while we have studied them in isolation, they need to be considered together. There are parameter sets that reduce the signature size at the cost of computation; LMS (with its cheaper computation) may make such a trade-off more acceptable, and such a reduction in signature size might more than make up the slightly larger LMS signature encoding.

To give a concrete example, in [32] made a comparison using two similar parameter sets. The results of this comparison are in Figure 6.17. This table shows that LMS (with these settings) is measured to perform moderately slower than XMSS; however the LMS signature size is almost half of the XMSS signature. One could define an equivalent XMSS parameter set with  $w = 256$ ; however that would drastically increase the amount of computation required.

Operation	HSS	XMSS <sup>MT</sup>
PK Gen	5.44 sec	3.26 sec
Sign Gen	6.49 msec	4.72 msec
Sig Ver	2.66 msec	1.76 msec
Size Public Key	60 byte	68 byte
Size Signature	2964 byte	5605 byte

Figure 6.17: Comparison of LMS ( $w = 256$ ) and XMSS ( $w = 16$ ).

In summary, XMSS with equivalent multilevel parameter sets has slightly smaller signature sizes than LMS. However, LMS performs significantly better, which allows us more options when selecting parameter sets that fit within the application constraints.

To learn more about this topic, please refer to [32].

In addition, in [17], a comparison was made between XMSS and LMS (and their variants) on the specific platform *ARM Cortex-M4*.

There is a substantial difference between these two algorithms and other finalist: XMSS and LMS are OTS. This is a big drawback for XMSS and LMS in computational terms.

Furthermore both algorithms differ considerably depending on the parameters chosen, the size of the trees used, number of multi-level trees, and the size of the hash functions used, and this makes a comparison between these two algorithms and the other finalists more complicated.

### 6.3.2 A comparison of NIST's finalist digital signature schemes

Let us start by saying that in this section we will discuss a comparison between only the digital signature algorithms; for the other algorithms one can refer for example to the recent study done in [53].

In the third round of NIST selection, the remaining algorithms for digital signature are: Falcon [5], Crystals-Dilithium [2, 3], Rainbow [23, 39, 8].

As already mentioned, Falcon and Crystals-Dilithium are lattice-based algorithms, while Rainbow is multivariate-based.

NIST has dictated 5 security levels, of increasing quality explained in Figure 6.18.

Level Security	Description
I	At least as hard to break as AES128 (exhaustive key search)
II	At least as hard to break as SHA256 (collision search)
III	At least as hard to break as AES192 (exhaustive key search)
IV	At least as hard to break as SHA384 (collision search)
V	At least as hard to break as AES256 (exhaustive key search)

Figure 6.18: 5 different security levels in NIST competition.

So each algorithm will have different specifications depending on the required security level.



## Falcon

Falcon offers the following features:

- Security: a true Gaussian sampler is used internally, which guarantees negligible leakage of information on the secret key up to a practically big number of signatures (more than 264).
- Compactness: thanks to the use of NTRU lattices, signatures are substantially shorter than in any lattice-based signature scheme with the same security guarantees, while the public keys are around the same size.
- Speed: use of fast Fourier sampling allows for very fast implementations, in the thousands of signatures per second on a common computer; verification is five to ten times faster.
- Scalability: operations have cost  $O(n \log(n))$  for degree  $n$ , allowing the use of very long-term security parameters at moderate cost.
- RAM Economy: the enhanced key generation algorithm of Falcon uses less than 30 kilobytes of RAM, a hundredfold improvement over previous designs such as NTRUSign. Falcon is compatible with small, memory-constrained embedded devices.

While resistance to quantum computers is the main drive for the design and development of Falcon, the algorithm may achieve significant adoption only if it is also reasonably efficient in our current world, where quantum computers do not really exist. Using the reference implementation on a common desktop computer (Intel® Core i5-8259U at 2.3 GHz, TurboBoost disabled), Falcon achieves the performances expressed in Figure 6.19:

Sizes (key generation RAM usage, public key size, signature size) are expressed in bytes. Key generation time is given in milliseconds. Private key

degree	keygen (ms)	keygen (RAM)	sign/s	vrfy/s	pub length	sig length
512	8.64	14336	5948.1	27933.0	897	666
1024	27.45	28672	2913.0	13650.0	1793	1 280

Figure 6.19: Falcon performance on an Intel® Core® i5-8259U CPU (“Coffee Lake” core, clocked at 2.3 GHz). Sizes of pub length and signature length in bytes.

size (not listed above) is about three times that of a signature, and it could be theoretically compressed down to a small PRNG seed (say, 32 bytes), if the signer accepts to run the key generation algorithm every time the key must be loaded.

To give a point of comparison, Falcon-512 is roughly equivalent, in classical security terms, to RSA-2048, whose signatures and public keys use 256 bytes each. On the specific system on which these measures were taken, OpenSSL’s thoroughly optimized assembly implementation achieves about 1140 signatures per second; thus, Falcon’s reference implementation, which is portable and uses no inline assembly on x86 CPUs, is already more than five times faster, and it scales better to larger sizes (for long-term security).

In conclusion, the main advantage of Falcon is its compactness. Stateless hash-based signatures often have small public keys, but large signatures. Conversely, some multivariate schemes achieve very small signatures but require large public keys. Lattice-based schemes can offer the best of both worlds, but no NIST candidate gets  $|p_k| + |signature|$  to be as small as Falcon does.

Farther, the signature generation and verification procedures are very fast. This is especially true for the verification algorithm, but even the signature algorithm can perform more than 1000 signatures per second on a moderately-powered computer.

The main drawbacks of Falcon, on the other hand, are: the key generation and signing remain complex; it has a delicate implementation: both the key

generation procedure and the fast Fourier sampling are non-trivial to understand and delicate to implement, and constitute the main shortcoming of Falcon (on the bright side, the fast Fourier sampling uses subroutines of the fast Fourier transform as well as trees, two objects most implementers are familiar with); Key generation and Signing rely on floating-point arithmetic, which, in some cases, can be problematic. Signing procedure uses floating-point arithmetic with 53 bits of precision. While this poses no problem for a software implementation, it may prove to be a major limitation when implementation on constrained devices – in particular those without a floating-point unit – will be considered.

### **Crystals-Dilithium**

In contrast to other signature proposals, Crystals-Dilithium samples from a uniform distribution avoiding the complex and inefficient sampling from a discrete Gaussian distribution.

The modular structure of Dilithium ensures that polynomial multiplication is always performed in the same ring regardless of security level, which makes it easy to switch between these levels.

Multiplication can be performed efficiently due to its NTT friendly parameters. Applying a trick to compress the public key with a factor 2, Dilithium could have the smallest public key plus signature size of lattice-based schemes that use uniform sampling.

A study on a Crystals-Dilithium implementation on Cortex-M3 and Cortex-M4 can be found in [28].

### **Rainbow**

Rainbow digital signature scheme has (relatively) small signatures (e.g. about 66 Bytes at SL I) and the signing and verification algorithms are

AVX2 + AES on Skylake  
# / sec is assuming 3GHz freq.

## CRYSTALS-Dilithium

Security Level	Public Key (Bytes)	Signature (Bytes)	pkgen	sign	verify
60	864	1196			
100	992	1843			
<b>128 (NIST II)</b>	<b>1312</b>	<b>2420</b>	<b>50K cyc 60K / sec</b>	<b>150K cyc 20K / sec</b>	<b>65K cyc 45K / sec</b>
<b>2<sup>159</sup> gates</b>					
<b>2<sup>98</sup> memory</b>					
<b>192 (NIST III)</b>	<b>1952</b>	<b>3293</b>	<b>80K cyc 35K / sec</b>	<b>200K cyc 15K / sec</b>	<b>95K cyc 30K / sec</b>
<b>2<sup>217</sup> gates</b>					
<b>2<sup>139</sup> memory</b>					
<b>256 (NIST V)</b>	<b>2592</b>	<b>4595</b>	<b>125K cyc 24K / cyc</b>	<b>230K cyc 13K / sec</b>	<b>135K cyc 22K / sec</b>
<b>2<sup>285</sup> gates</b>					
<b>2<sup>187</sup> memory</b>					
320	2912	5246			
384	3232	5892			

Figure 6.20: Crystals-Dilithium sizes, relatively to different security levels, on Skylake platform.

(relatively) fast.

Rainbow uses only linear algebra over very small finite fields, which makes it suitable for implementing the scheme on low-cost devices, without the need for a cryptographic coprocessor.

On the other hand, the public keys are rather large (e.g. 158 KB at SL I). It is possible to compress the public key size by almost a factor 3 at the expense of slower signing times.

Furthermore, the security analysis of Rainbow cannot be considered stable, because, at the moment, it has no security proof.

Rainbow offers three different versions of the same algorithm:

- Standard Rainbow;
- CZ-Rainbow;
- Compressed Rainbow.

In CZ-Rainbow the key generation process of standard Rainbow it's reversed: instead of computing the public key from the private key, it's fixed

major parts of the public key and then computed the central Rainbow map from the public key.

The CZ-Rainbow scheme is exactly based on this idea of reversing the key generation process of Rainbow.

Since a part of the public key can be randomly selected anyway, the key idea in the CZ-Rainbow scheme is to use a cryptographic PRNG to generate that part of the public key.

Using the PRNG it's possible to reduce the public key size of the Rainbow scheme by a factor of up to 70%.

However since, during the verification process, it's necessary to decompress the public key before evaluating it, the verification process of the scheme is slowed down significantly.

Besides the standard and the CZ variant, is also propose a Rainbow variant with a "compressed" key.

This compressed Rainbow variant works very similar to CZ-Rainbow, but uses additionally a seed from which two linear maps, used in the algorithm, are generated.

NIST required to evaluate the performance of the algorithms on a specific platform.

The data of this platform are given as follows:

- Processor: Intel(R) Xeon(R) CPU E3-1275 v5 @ 3.60GHz (Skylake);
- Clock Speed: 3.60GHz;
- Memory: 32GB (2x16) ECC DIMM DDR4 Synchronous 2133 MHz (0.5 ns);
- Operating System: Linux 4.8.15, GCC compiler version 9.3.

The NIST reference platform makes no use of special processor instructions. We now provide, in Figura 6.23, Figura 6.24 and Figura 6.25, the performances of the 3 different Rainbow on the platform indicated by NIST.

## Rainbow: Key and Signature Sizes

NIST security category	parameters ( $q, v_1, o_2, o_2$ )	signature size (bit)
I	(16,36,32,32)	528
III	(256,68,32,48)	1,312
V	(256,96,36,64)	1,696

NIST security category	standard Rainbow		CZ-Rainbow		compressed Rainbow	
	$ pk $	$ sk $	$ pk $	$ sk $	$ pk $	$ sk $
I	157.8	101.2	58.8	101.2	58.1	64B
III	861.4	611.3	258.4	611.3	258.4	64B
V	1,885.4	1,375.7	523.6	1,375.7	523.6	64B

Figure 6.21: In the first table, we have the parameters and the signature size in relation to the NIST security category; these parameters are equal in the 3 different version of Rainbow.

In the second table we have the sizes for Rainbow in its 3 different versions. In particular  $|pk|$  and  $|sk|$  are in expressed  $kB$ .

## Performance of Rainbow on Linux/Skylake (using AVX2 instructions)

NIST security category	standard Rainbow			CZ-Rainbow			compressed Rainbow		
	keygen	sign	verify	keygen	sign	verify	keygen	sign	verify
I	9.9M	67k	34k	10.7M	67k	3.5M	10.7M	7.0M	3.5M
III	52M	285k	132k	64M	285k	20M	64M	41M	20M
V	192M	739k	392k	235M	739k	47M	235M	118M	47M

Figure 6.22: Performance (cycles) for Rainbow in its 3 different versions (on Linux/Skylake using AVX2 instructions).

## Comparison

To obtain a comparison between Falcon and Crystals-Dilithium, we provide the following tables (from Figure 6.26 to Figure 6.27) taken from the Crystals-Dilithium presentation [4] at round 3 of NIST selection.

A recommended study on security comparisons and performance analyses of post-quantum signature algorithms was conducted in [50].

parameter set		key gen.	sign. gen.	sign. verif.
I	cycles	32 M	319 k	41 k
	time (ms)	8.98	0.09	0.01
	memory(MB)	12.94	12.79	12.94
III	cycles	197 M	1.47 M	203 k
	time (ms)	54.59	0.41	0.06
	memory(MB)	15.09	13.40	13.64
V	cycles	436 M	2.48 M	362 k
	time (ms)	121.05	0.69	0.10
	memory(MB)	17.83	14.15	14.64

Figure 6.23: Performance for standard Rainbow on the NIST platform.

parameter set		key gen.	sign. gen.	sign. verif.
I	cycles	37 M	319 k	3.5 M
	time (ms)	10.26	0.09	0.98
	memory(MB)	12.79	12.79	12.79
III	cycles	232 M	1.47 M	20.4 M
	time (ms)	10.26	0.41	5.66
	memory(MB)	14.25	13.40	13.05
V	cycles	488 M	2.48 M	46 M
	time (ms)	135.62	0.69	12.81
	memory(MB)	16.00	14.15	13.31

Figure 6.24: Performance for CZ-Rainbow on the NIST platform.

parameter set		key gen.	sign. gen.	sign. verif.
I	cycles	37 M	19 M	3.5 M
	time (ms)	10.26	5.42	0.98
	memory(MB)	12.79	12.79	12.79
III	cycles	232 M	137 M	20.4 M
	time (ms)	10.26	5.42	5.66
	memory(MB)	14.25	13.40	13.05
V	cycles	488 M	233 M	46 M
	time (ms)	135.62	64.68	12.81
	memory(MB)	16.00	14.15	13.31

Figure 6.25: Performance for Compressed Rainbow on the NIST platform.

We therefore provide, from Figure 6.28 to Figure 6.31, some Tables (taken from [1]) for a final and complete comparison between the finalist NIST post-quantum digital signature algorithms, with also the alternative candidates:

CRYSTALS-Dilithium			Falcon		
Security Level	Public Key (Bytes)	Signature (Bytes)	Security Level	Public Key (Bytes)	Signature (Bytes)
128	1312	2420	128	897	666
192	1952	3293	-	-	-
256	2592	4595	256	1793	1280

-  $\approx 2.3$  X larger (pk + sig)

+ Signing uses only uniform sampling in a (power-of-2) range. Easier to detect bugs.

+  $\approx 2.3$  X smaller (pk + sig)

- Signing uses high-precision Gaussian sampling with high-precision changing centers. Hard to detect subtle implementation mistakes which can leak the secret key

- Very difficult to mask

Signing a few messages ( $\approx 100?$ ) shouldn't leak enough even if the sampling is leaky

Figure 6.26: Comparison between Crystals-Dilithium and Falcon sizes.

Dilithium [Greconici, Kannwischer, Sprenkels 2020]			Falcon [Pornin, 2019]		
NIST Level 3	Key Gen. Speed	Key Gen. RAM	NIST Level 1	Key Gen. Speed	Key Gen. RAM
Cortex M4	6M cycles	10KB	Cortex M4	171M cycles	16KB
NIST Level 3	Sign Speed	Sign RAM	NIST Level 1	Sign Speed	Sign RAM
Cortex M4	8M cycles	70KB	Cortex M4	40M cycles	40KB
Cortex M4	26M cycles	11KB	Cortex M4	21M cycles	25KB
Cortex M4	6M cycles	21KB + 48KB Flash			+ 57KB Flash
NIST Level 3	Ver. Speed	Ver. RAM	NIST Level 1	Ver. Speed	Ver. RAM
Cortex M4	2.7M cycles	11KB	Cortex M4	0.5 M cycles	4KB

➤ 80% of Dilithium Verification Time is Keccak

Figure 6.27: Comparison between Crystals-Dilithium and Falcon performance on Cortex M3 and Cortex M4 platforms.

there were the sizes of some classical (no post-quantum) digital signature algorithms as a term of comparison, the sizes (of signature, and of public and private key) of the post-quantum algorithms (finalists and alternative candidates), the performances and the memory used.



Method	Public key size (B)	Private key size (B)	Signature size (B)	Security level
Ed25519	32	32	64	1 (128-bit) EdDSA
Ed448	57	57	112	3 (192-bit) EdDSA
ECDSA	64	32	48	1 (128-bit) ECDSA
RSA-2048	256	256	256	1 (128-bit) RSA

Figure 6.28: Sizes and security levels for ECDSA, RSA, Ed25519 and Ed448 (useful as a term of comparison with post-quantum schemes).

Method	Public key size	Private key size	Signature size	Security level
Crystals Dilithium 2 (Lattice)	1,312	2,528	2,420	1 (128-bit) Lattice
Crystals Dilithium 3	1,952	4,000	3,293	3 (192-bit) Lattice
Crystals Dilithium 5	2,592	4,864	4,595	5 (256-bit) Lattice
Falcon 512 (Lattice)	897	1,281	690	1 (128-bit) Lattice
Falcon 1024	1,793	2,305	1,330	5 (256-bit) Lattice
Rainbow Level Ia (Oil-and-Vineger) (UOV)	161,600	103,648	66	1 (128-bit) Multivariate
Rainbow Level IIIa (UOV)	861,400	611,300	164	3 (192-bit) Multivariate
Rainbow Level Vc (UOV)	1,885,400	1,375,700	204	5 (256-bit) Multivariate
Sphincs SHA256-128f Simple	32	64	17,088	1 (128-bit) Hash-based
Sphincs SHA256-192f Simple	48	96	35,664	3 (192-bit) Hash-based
Sphincs SHA256-256f Simple	64	128	49,856	5 (256-bit) Hash-based
Picnic 3 Full	49	73	71,179	3 (192-bit) Symmetric
GeMSS 128 (HFEv-)	352,188	16	33	1 (128-bit) Multivariate
GeMSS 192 (HFEv-)	1,237,964	24	53	1 (128-bit) Multivariate

Figure 6.29: Sizes (in byte) for post-quantum digital signature schemes.

In conclusion:

- Rainbow is an interesting candidate, versatile thanks to its 3 different formulations, but there are no math proof on its security, and this could prove to be a problem in the future. It also has the public key and private key sizes that are much larger than those of Crystals-Dilithium and Falcon.
- Falcon is the most compact algorithm (for sizes and performances), and that makes it an attractive algorithm for the embedded world, but Key Generation and Signing are complex and delicate.

Method	Key generation	Sign	Verify
Crystals Dilithium 2 (Lattice)	36,424	61,312	40,664
Crystals Dilithium 3	50,752	81,792	55,000
Crystals Dilithium 5	67,136	104,408	71,472
Falcon 512 (Lattice)	1,680	2,484	512
Falcon 1024	1,680	2,452	512
Rainbow Level Ia (Oil-and-Vinegar)	2,969	4,720	2,732
Rainbow Level IIIa	3,216	3,224	1,440
Rainbow Level Vc	3,736	6,896	4,928
Sphincs SHA256-128f Simple	2,192	2,248	2,544
Sphincs SHA256-192f Simple	3,512	3,640	3,872
Sphincs SHA256-256f Simple	5,600	5,560	5,184

Figure 6.30: Performance on M4 (ARM Cortex-M4 dev) measured in CPU operations per second. The data in this table were taken from [33]. Note, no Rainbow assessment has been performed in [33], so LUOV (an Oil-and-Vinegar method) has been used to give an indication of performance levels.

Method	Memory (Bytes)
Crystals Dilithium 2 (Lattice)	13,948
Crystals Dilithium 3	13,756
Crystals Dilithium 5	13,852
Falcon 512 (Lattice)	117,271
Falcon 1024	157,207
Rainbow Level Ia (Oil-and-Vinegar)	404,920
Rainbow Level IIIa	405,412
Rainbow Level Vc	405,730
Sphincs SHA256-128f Simple	4,668
Sphincs SHA256-192f Simple	4,676
Sphincs SHA256-256f Simple	5,084

Figure 6.31: Stack memory size on an ARM Cortex-M4 device (data from [33]) and measured in bytes. Note, no Rainbow assessment has been performed in [33], so LUOV (an Oil-and-Vinegar method) has been used to give an indication of performance levels.

- Chrystal-Dilithium could be a good compromise between security and compression, despite it being the one that, among the three under examination, has the worst performance.

According to the information we have today, listed above, among the three post-quantum digital signature algorithms (finalists in the NIST competition) Falcon, Crystals-Dilithium and Rainbow no one stands out who is better than

the others on all fronts. Based on the characteristics one is looking for, he can choose which of the three to use (for example if one is looking for an algorithm that has the smallest possible length of public key and private key, he can use Falcon; if one is looking for a short signature, Rainbow; if one is looking for the lowest stack memory size, Crystal-Dilithium).

We conclude by saying that it is difficult to make a comparison between these three algorithms and the previous XMSS or LMS, because the latter two are OTS and strongly depend on the numerous parameters that can be chosen (NIST has not made a big selection of these). It is no coincidence that NIST has carried out a parallel standardization for XMSS and LMS, and that in the literature it is difficult to find comparisons with the other finalists.



# Conclusion

In this thesis I analyzed quantum computing and the way in which cryptography, to defend itself, is evolving: in particular with post-quantum cryptography (which opposes quantum cryptography).

Thanks to attempts to standardize post-quantum algorithms, primarily the work of NIST, a selection phase has begun with the aim of obtaining algorithms that are as safe and usable as possible.

I then analyzed the problems which "hardness" underlies the security of post-quantum algorithms. Among the current 7 finalist algorithms, in particular, 5 are lattice-based: the problems based on lattices, to date, seem to be an excellent basis for creating algorithms, that can be implemented on classical computers, and which are also resistant to quantum computers.

In the last Chapter I analyzed the post-quantum digital signature algorithms selected by NIST (in particular the hash-based XMSS and LMS, which NIST proposed to the community in a parallel selection, since they are OTS algorithms). Furthermore, I made a comparison between them from the point of view of security, performance and parameter sizes, to have a picture of these for anyone who is evaluating which algorithm to implement. What emerged is that there is no digital signature algorithm that is better than the others on all aspects: based on the features sought, everyone can choose which algorithm to implement. For example: if one is looking for an algorithm that has the smallest possible length of public key and private key, he can use Falcon; if one is looking for a short signature, Rainbow; if one is looking for the lowest stack memory size, Crystal-Dilithium.



# Appendices





# Appendix A

## Continued Fractions

**Definition 41** (Continued Fractions). A continued fraction is a fraction of the type:

$$a_0 + \frac{1}{a_1 + \frac{1}{\dots + \frac{1}{a_p}}} \quad (\text{A.1})$$

where  $a_0 \in \mathbb{Z}$ , and the others  $a_i$  are positive integers.

We will refer to the fraction A.1 with the notation:

$$[a_0, a_1, \dots, a_p]$$

To each real number  $c$ , an expansion in continued fractions is associated in the following way. Starting from  $c$ , the sequences  $\{a_j\}_{j \geq 0}$  and  $\{r_j\}_{j \geq 0}$  are recursively constructed taking:

$$\begin{cases} a_0 = \lfloor c \rfloor \\ r_0 = c - a_0 \\ a_j = \lfloor \frac{1}{r_{j-1}} \rfloor & \forall j \geq 1 \\ r_j = \frac{1}{r_{j-1}} - \lfloor \frac{1}{r_{j-1}} \rfloor & \forall j \geq 1 \end{cases} \quad (\text{A.2})$$

obtaining:

$$a_0 + \frac{1}{a_1 + \frac{1}{\dots + \frac{1}{a_j + r_j}}} \quad (\text{A.3})$$

The rational number  $[a_0, a_1, \dots, a_j]$  is called *jth convergent* of  $c$ .

It is possible to prove that

$$[a_0, a_1, \dots, a_j] \xrightarrow{j \rightarrow \infty} c$$

and so every  $[a_0, a_1, \dots, a_j]$  is an approximation of  $c$  (the larger a term is in the continued fraction, the closer the corresponding convergent is to the irrational number being approximated; this explains the choice of the name "jth convergent of  $c$ ").

In particular: the successive approximations generated in finding the continued fraction representation of a number, that is, by truncating the continued fraction representation, are, in a certain sense, the "best possible" if one choose to define a *best rational approximation to a real number  $c$*  as a rational number  $\frac{n}{d}$ ,  $d > 0$ , that is closer to  $c$  than any other approximation with a smaller or equal positive denominator.

If  $\exists j$  s.t.  $r_j = 0$ , then the expansion ends with  $a_j$  and we obtain the equivalence:

$$c = [a_0, a_1, \dots, a_j]$$

In this case,  $c$  was in  $\mathbb{Q}$ .

The following useful results hold.

**Proposition 17.** *The expansion in continued fractions of a real number  $c$  ends if and only if  $c$  is a rational number.*

The following proposition allows us to derive the numerator and denominator of a rational number  $c$  from its expansion into continued fractions.

**Proposition 18.** *The rational number  $[a_0, a_1, \dots, a_n]$  has the form  $\frac{p_n}{q_n}$  where  $p_0 = a_0, q_0 = 1, p_1 = 1 + a_0a_1, q_1 = a_1$  and  $\forall i$  s.t.  $n \geq i \geq 2$ :*

$$p_i = a_i p_{i-1} + p_{i-2}, \quad \text{and} \quad q_i = a_i q_{i-1} + q_{i-2}.$$

*Furthermore  $GCD(p_n, q_n) = 1$  and so the fraction is always reduced.*

**Corollary 19.** *The expansion in continued fractions of a rational positive number  $\frac{p}{q}$  can be obtained in  $O(m^3)$  operations, if  $p$  and  $q$  are integers on  $m$  bits.*

**Theorem 20** (Sufficient condition to be a convergent). *Suppose that  $x$  and  $\frac{p}{q}$  are rational numbers such that  $|x - \frac{p}{q}| \leq \frac{1}{2q^2}$ . Then  $\frac{p}{q}$  is a convergent of the continued fraction for  $x$ .*



# Appendix B

## The group structure of an elliptic curve

In this appendix it is assumed that the reader has already read Subsection 5.5.1, from which the main definitions and results are inherited.

Furthermore, in the following, for illustrative purposes, we refer to the case where the field  $K$  is  $\mathbb{R}$ , but the arguments made are valid for any field.

We consider a non-singular cubic  $E$  of  $\mathbb{P}^2(K)$  (refer to Definition 27) in the case in which the characteristic of the field is different from 2 and 3.

We make this assumption only to have the possibility of representing each elliptic curve in the Weierstrass form (see Definition 28), but also in the cases of characteristic 2 or 3 the group law continues to exist.

### B.1 Geometric interpretation of the sum between two points

The sum rule "+" in the elliptic curves (see Definition 29), is also called the "tangent-chord rule" and has a very intuitive geometric explanation.

Let  $P = (x_1, y_1)$  and  $Q = (x_2, y_2)$  be two distinct points on the curve  $E$ . Then the sum  $R = P + Q$  is defined.

- Take  $r(P, Q)$ , the line that contains  $P$  and  $Q$ . This line intersects (necessarily, by — Bézout's — Theorem 8) the curve at a third point  $R$ .
- Identify the symmetrical point to  $R$  with respect to the  $x$  axis ( i.e., change the sign of the  $y$  coordinate) and thus obtain  $R' = -R := P+Q$ .

The second point of the procedure can also be stated in the following equivalent way:

- Let  $\mathcal{O}_E$  be the point at infinity of  $E$ . Then we construct the line through  $\mathcal{O}_E$  and  $R$ ,  $r(\mathcal{O}_E, R)$ . By Bézout's theorem, this will intersect  $E$  at a third point  $R'$ , which will then verify  $R' := P + Q$ .

In other words, if:

$$\begin{aligned} * : E \times E &\longrightarrow E \\ (P, Q) &\mapsto P * Q = r(P, Q) \cap E \setminus \{P, Q\} \end{aligned}$$

then:

$$\begin{aligned} + : E \times E &\longrightarrow E \\ (P, Q) &\mapsto P + Q = P * Q * \mathcal{O}_E \end{aligned}$$

The law  $*$  is clearly commutative.

For the laws  $*$  and  $+$  just defined, the closing property obviously holds; in fact if  $A, B \in E$ , also  $A + B \in E$ . Moreover, since  $*$  is commutative, also  $+$  is commutative.

The procedure for summing two points on an elliptic curve is represented in Figure B.1. In particular in case a) the two points  $P, Q$  are distinct, while in case b) a doubling is calculated, that is, we find  $P + P$ . In this case the line  $r(P, P)$  must be a line that "passes twice" in the point  $P$ ; therefore this line is the only line tangent to  $E$  at the point  $P$ .

**Theorem 21.** *With the sum law  $+$ , the set of points of the cubic  $E(K)$  with coordinates on an algebraically closed field is an abelian group whose null element is  $\mathcal{O}_E$ , and  $\forall P = (x, y) \in E$ , the inverse of  $P$  is  $-P = (x, -y)$ .*

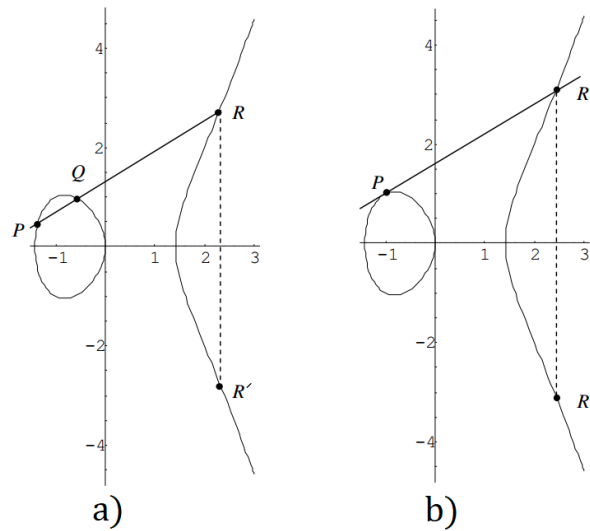


Figure B.1: Group Law: sum of two distinct points in a), doubling of point  $P$  in b).

The commutative property, the existence of the neutral element and of the inverse element are obvious. Demonstrating associativity is not easy. The reader is referred to [64]. However, we give a graphical verification of the associative property in the example in Figure B.2.

*Remark 22* (Alignment equation). Since  $P * Q = R$  has the point  $P + Q$  as its inverse, we can write  $P + Q = -R$  and therefore:

$$P + Q + R = 0$$

which is the equation of alignment of the three points  $P, Q, R$ .

Figure B.3 shows what was observed in the previous Remark. In other words:  $(P+Q)+(P*Q) = (P+Q)+[-(P+Q)] = \mathcal{O}_E$ , so  $-(P+Q) = (P*Q)$  and therefore  $P + Q$  is obtained by mirroring the point  $P * Q$  around the abscissa axis.

This implies that, for each point  $A$ , its inverse  $-A$  is obtained with the procedure:  $A = A + \mathcal{O}_E \implies -A = -(A + \mathcal{O}_E) = A * \mathcal{O}_E$  (see Figure B.4a). In the event that  $A$  and  $B$  are vertically aligned (figure B.4b ), we have that the sum  $A + B$  corresponds to  $\mathcal{O}_E$ , which is the third intersection point with

$E$  of the vertical line. Therefore  $A$  is the inverse of  $B$ .

On the other hand, Figure B.4c shows the case in which  $A$  is added to itself; since in this case the tangent in  $A$  to  $E$  (the point  $A$  has double multiplicity of intersection with the straight line) is vertical,  $A + A = \mathcal{O}_E$ , and we get  $A = -A$ ; here therefore  $A$  is the inverse of itself.

The last case, corresponding to Figure B.4d, is the one in which the point  $\mathcal{O}_E$  is added to itself, that is  $\mathcal{O}_E + \mathcal{O}_E = \mathcal{O}_E$ . Since the point  $\mathcal{O}_E$  is an inflection point, the intersection cardinality of a vertical line in it is equal to 3; therefore there are no other points on the curve touched by the line.

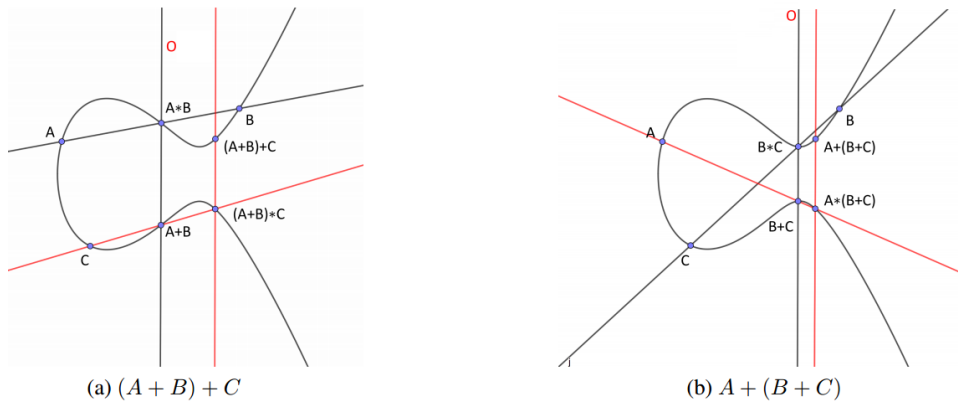


Figure B.2: Associativity for the Group Law.

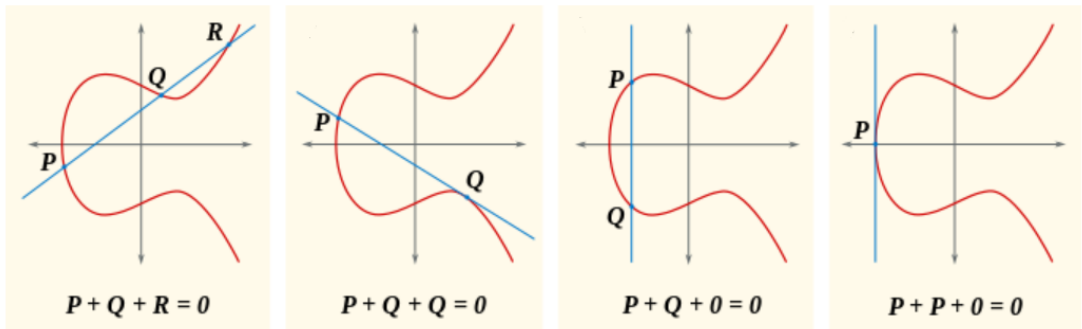


Figure B.3: Alignment equation in 4 cases.



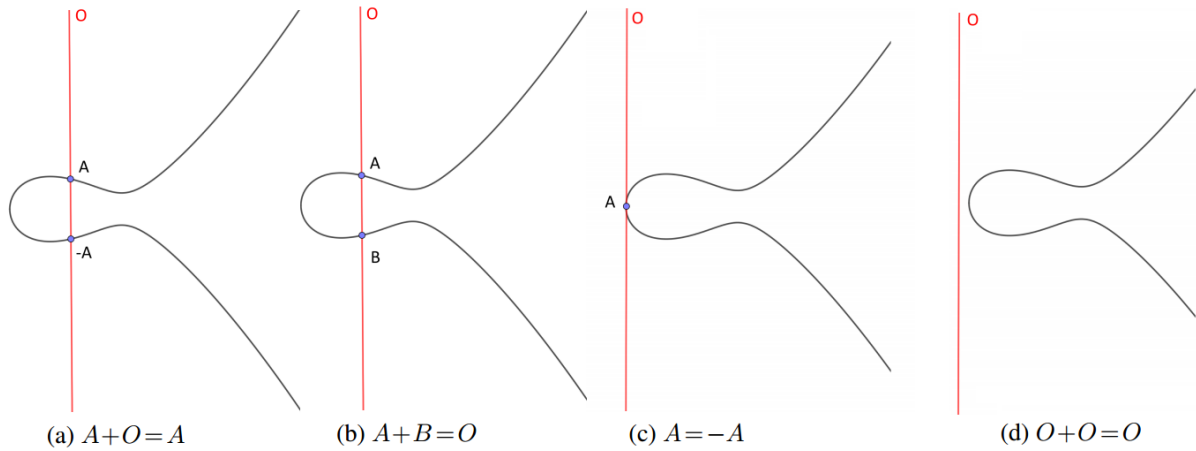


Figure B.4: Group Law: case of two points living on the same vertical line.

## B.2 Coordinates of the sum between two points

We now want to provide the equations that explicitly define the coordinates of the  $+$  law, previously introduced. In other words, let us see how to get the coordinates of  $P + Q$ , starting from the coordinates of  $P$  and  $Q$ . We will do the construction only in the cases  $\text{char}(K) \neq 2, 3$ . A similar argument can be made for characteristics 2 and 3 as well.

In particular, fixing  $\text{char}(K) \neq 3, 2$  and using the Weierstrass form  $y^2 = x^3 + Ax + B$  for the elliptic curve  $E$ , there are four different cases. Let be  $P = (x_1, y_1), Q = (x_2, y_2) \in E$ :

1.  $P \neq Q$  and  $P, Q \neq \mathcal{O}_E$ . If  $x_1 \neq x_2$  the line  $r := r(P, Q)$  has equation

$$y = m(x - x_1) + y_1 \tag{B.1}$$

where  $m = \frac{y_2 - y_1}{x_2 - x_1}$ . To find the intersection between  $r$  and  $E$ , we substitute (B.1) in the equation of  $E$ , finding

$$(m(x - x_1) + y_1)^2 = x^3 + Ax + B, \tag{B.2}$$

which can be rewritten in the form:

$$x^3 - m^2x^2 + (A + 2mx_1 - 2my_1)x + (B - m^2x_1^2 + 2my_1x_1 - y_1^2) = 0. \tag{B.3}$$

The three roots of this cubic are the abscissas of the three points of intersection of  $r$  with  $E$ .

In particular,  $x_1, x_2$  are two of the roots. Let  $x_3$  be the third. We factor the first member of the previous equation into:

$$(x-x_1)(x-x_2)(x-x_3) = x^3 - (x_1+x_2+x_3)x^2 + (x_2x_3+x_1x_3+x_1x_2)x - x_1x_2x_3.$$

Therefore, from (B.3), it follows that  $x_3 = m^2 - x_1 - x_2$  and therefore the coordinates of point  $P + Q$  are:

$$\begin{cases} x_3 &= \left(\frac{y_2-y_1}{x_2-x_1}\right)^2 - x_1 - x_2, \\ y_3 &= -y_1 + \left(\frac{y_2-y_1}{x_2-x_1}\right)(x_1 - x_3). \end{cases}$$

If instead of  $x_1 \neq x_2$ , we have  $x_1 = x_2$ , the line  $r$  intersects  $E$  in  $\mathcal{O}_E$ . Therefore, in this case, we have that  $P + Q = \mathcal{O}_E$ .

2.  $P = Q \neq \mathcal{O}_E$ . In this case, the tangent line  $t$  in  $P$  to  $E$  has equation:

$$f_x(x_1, y_1)(x - x_1) + f_y(x_1, y_1)(y - y_1) = 0,$$

where  $f_x$  and  $f_y$  are the partial derivative of the Weierstrass form  $f(x, y) := y^2 - x^3 + -Ax - B$ . And so we find:

$$(-3x_1^2 - A)(x - x_1) + 2y_1(y - y_1) = 0.$$

If  $y_1 \neq 0$ , then the equation of the tangent line becomes (B.1) with  $m = \frac{3x_1^2 + A}{2y_1}$ . Now, reasoning in a similar way to the previous case and bearing in mind that here we have  $x_1 = x_2$ , we find that the coordinates of  $P + P$  are:

$$\begin{cases} x_3 &= \left(\frac{-3x_1^2 + A}{2y_1}\right)^2 - 2x_1, \\ y_3 &= -y_1 + \left(\frac{-3x_1^2 + A}{2y_1}\right)(x_1 - x_3). \end{cases}$$

We note that, if  $y_1 = 0$ , then:

$$P + P = 2P = \mathcal{O}_E.$$

3.  $P \neq Q$  and  $P = \mathcal{O}_E$ . (respectively  $Q = \mathcal{O}_E$ ). In this case, the intersection between the straight line  $x = x_2$  (respectively  $x = x_1$ ) and the curve  $E$  contains  $P, Q$  and  $-Q = (x_2, -y_2)$  (respectively  $-P = (x_1, -y_1)$ ). Therefore we define  $P + Q = Q$  (respectively  $P + Q = P$ ).
4.  $P = Q = \mathcal{O}_E$ . In that case, by definition, we have:

$$\mathcal{O}_E + \mathcal{O}_E = \mathcal{O}_E.$$

### B.3 Elliptic Curve Discrete Logarithm Problem

Once the sum between points in an elliptic curve  $E$  has been defined, the multiplication by an integer  $k$  also makes sense:  $kP := [k]P = P + \dots + P$ . Observe that if  $k$  is very large it is not convenient to add  $P$  with itself  $k$  times; in this case, the expedient of the sum of the doubling is used, which allows to reduce the number of operations. For example, if we want to calculate  $23P$  we will proceed as follows:

$$\begin{aligned} 2P &= P + P, & 4P &= 2P + 2P, & 8P &= 4P + 4P \\ 16P &= 8P + 8P, & 23P &= 16P + 4P + 2P + P \end{aligned}$$

which requires only 7 sums. In general it is necessary to decompose  $k$  in positional notation into base 2 and verify for which exponents of 2 the coefficient is equal to 1; in the case of 23 we have:

$$23 = \sum_{i=0}^n a_i 2^i = 1 \cdot 2^4 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

This method allows to save on the number of operations, passing from an algorithm of complexity  $O(k)$  to one of  $O(\log_2 k)$ , in the case in which both the sum and the doubling are considered of complexity  $O(1)$ . The only difficulty that remains, independent of the method used to make the product, is that the coordinates of the points can grow very rapidly and unexpectedly

as  $k$  increases when working on infinite fields. However, if we use a finite field this is no longer a problem, since for any value of  $k$ , the coordinates are always confined within the elements of the field, which are finite.

So we have an efficient way to calculate the multiplication of a point of an elliptic curve by an integer.

The multiplication by an integer constitutes an important one-way function:

**Definition 42** (Direct function associated with elliptic curves). Given a point  $P \in E(K)$  and an integer  $k$ , find a point  $Q$  such that  $Q = kP$ .

In fact, while adding the points to obtain the integer product as is relatively simple, performing the inverse operation (defined below), in the case of finite field, is not at all:

**Definition 43** (Inverse function associated with elliptic curves). Given two points  $Q, P \in E(K)$ , find an integer  $k$  such that  $Q = kP$ .

This inverse problem is known in the literature as "*the problem of the discrete logarithm on elliptic curves*" (even if in reality we are not using exponentiation here, but only multiplication).

In fact, there are currently no known classical efficient algorithms to be able to reverse the operation. Hence, the inverse function associated with elliptic curves is presumably intractable when it is referred to an elliptic curve  $E(K)$  with  $K$  finite field.

This problem is therefore the basis of various cryptographic applications. The problem of the discrete logarithm used in elliptic curve cryptography is much more difficult than the problem of the factorization of prime numbers, for the same field size, and therefore for the same security this cryptography requires smaller public keys, and therefore more easily usable than those used by the RSA method.

Unfortunately, the hardness of this problem is not also of quantum nature, as it is efficiently solved by Shor's algorithm [49].

# Appendix C

## Algebraic varieties

**Definition 44** (Polynomial ring). The polynomial ring,  $K[x]$ , in  $x$  over a field (or, more generally, a commutative ring)  $K$  can be defined in several equivalent ways. One of them is to define  $K[X]$  as the set of expressions, called polynomials in  $x$ , of the form

$$p = p_0 + p_1x + p_2x^2 + \cdots + p_{m-1}x^{m-1} + p_mx^m$$

where  $p_0, p_1, \dots, p_m$ , the coefficients of  $p$ , are elements of  $K$ ,  $p_m \neq 0$  if  $m > 0$ , and  $x, x^2, \dots$ , are symbols, which are considered as "powers" of  $x$ , and follow the usual rules of exponentiation:  $x^0 = 1$ ,  $x^1 = x$ , and

$$x^k \cdot x^l = x^{k+l}$$

for any nonnegative integers  $k$  and  $l$ . The symbol  $x$  is called an *indeterminate* or *variable*.

Two polynomials are equal when the corresponding coefficients of each  $x_k$  are equal. The polynomial ring in  $X$  over  $K$  is equipped with an addition, a multiplication and a scalar multiplication.

**Definition 45** ( $V(S)$ , affine algebraic set). Let  $S \subseteq K[x]$ . Define

$$V(S) = \{P \in \bar{K}^n \mid f(P) = 0 \forall f \in S\}$$

If  $S = f_1, \dots, f_m$  then we write  $V(f_1, \dots, f_m)$  for  $V(S)$ . An *affine algebraic set* is a set  $X = V(S) \subseteq \bar{K}^n$  where  $S \subseteq K[x]$ .

Let  $K'$  be an algebraic extension of  $K$ . The  $K'$ -rational points of  $X = V(S)$  are:

$$X(K') = X \cap K'^n = \{P \in K'^n : f(P) = 0 \forall f \in S\}$$

**Definition 46** (Projective zeroes, projective  $V(S)$ , projective algebraic set). Let  $f \in K[x] = K[x_0, x_1, \dots, x_n]$  be a homogeneous polynomial. A point  $[x_0, x_1, \dots, x_n] \in \mathbb{P}^n(K)$  is a *zero* of  $f$  if  $f(x_0, \dots, x_n) = 0$  for some (hence, every) point  $(x_0, \dots, x_n)$  in the equivalence class  $[x_0, \dots, x_n]$ . We therefore write  $f(P) = 0$ . Let  $S$  be a set of polynomials and define:

$$V(S) = \{P \in \mathbb{P}^n(\bar{K}) \mid P \text{ is a zero of } f(x) \forall f(x) \in S\}.$$

A *projective algebraic set* is a set  $X = V(S) \subseteq \mathbb{P}^n(\bar{K})$  for some  $S \subseteq K[x]$ . Such a set is also called a *projective  $K$ -algebraic set*.

**Definition 47** (Zariski topology on  $X \subseteq K^n$ ). Let  $X$  be an algebraic set in  $K^n$  (respectively,  $\mathbb{P}^n[K]$ ). The *Zariski topology* is the topology on  $X$  defined as follows: the closed sets are  $X \cap Y$  for every algebraic set  $Y \subseteq K^n$  (respectively,  $Y \subseteq \mathbb{P}^n[K]$ ).

**Definition 48** (Affine variety, Projective variety). An affine  $K$ -algebraic set  $X \subseteq K^n$  is  *$K$ -reducible* if  $X = X_1 \cup X_2$  with  $X_1$  and  $X_2$  being  $K$ -algebraic sets and  $X_i \neq X$  for  $i = 1, 2$ . An affine algebraic set is  *$K$ -irreducible* if there is no such decomposition. An affine algebraic set is *geometrically irreducible* if  $X$  is  $\bar{K}$ -irreducible. An *affine variety* over  $K$  is a geometrically irreducible  $K$ -algebraic set defined over  $K$ .

A projective  $K$ -algebraic set  $X \subseteq \mathbb{P}^n[K]$  is  *$K$ -irreducible* (resp. *geometrically irreducible*) if  $X$  is not the union  $X_1 \cup X_2$  of projective  $K$ -algebraic sets  $X_1, X_2 \subseteq \mathbb{P}^n[K]$  (respectively, projective  $K$ -algebraic sets) such that  $X_i \neq X$  for  $i = 1, 2$ . A *projective variety* over  $K$  is a geometrically irreducible projective  $K$ -algebraic set defined over  $K$ .

Let  $X$  be a variety (affine or projective). A *subvariety* of  $X$  over  $K$  is a subset  $Y \subseteq X$  that is a variety (affine or projective) defined over  $K$ .

This definition matches the usual topological definition of a set being irreducible if it is not a union of proper closed subsets.

**Definition 49** (Ideal over a set). The *ideal over*  $X$  of a set  $X \in \bar{K}^n$  is:

$$I_K(X) = \{f \in K[x] : f(P) = 0 \forall P \in X(\bar{K})\}.$$

Similarly we can define for any set  $X \subseteq \mathbb{P}^n[\bar{K}]$ , the ideal over  $X$ :

$$I_K(X) = \{f \in K[x_0, \dots, x_n] : f \text{ is homogeneous and } f(P) = 0 \forall P \in X\}.$$

**Definition 50** (Affine and Homogenous coordinate ring). The *affine coordinate ring* over  $K$  of an affine algebraic set  $X \subseteq K^n$  defined over  $K$  is the quotient:

$$K[X] = K[x_1, \dots, x_n]/I_K(X).$$

If  $X$  is a projective algebraic set defined over  $K$ , then the *homogenous coordinate ring* of  $X$  over  $K$  is:

$$K[X] = K[x_0, \dots, x_n]/I_K(X).$$

Note that elements of  $K[X]$  are not necessarily homogeneous polynomials.

**Definition 51** (Function field  $K(X)$ ). Let  $X$  be an affine variety defined over  $K$ . The *function field*  $K(X)$  is the set:

$$K(X) = \{f_1/f_2 : f_1, f_2 \in K[X], f_2 \notin I_K(X)\}$$

of classes under the equivalence relation  $f_1/f_2 \equiv f_3/f_4$  if and only if  $f_1f_4 - f_2f_3 \in I_K(X)$ . In other words,  $K(X)$  is the field of fractions of the affine coordinate ring  $K[X]$  over  $K$ .

Let us now look at the projective case. Let  $X$  be a projective variety. The *function field* is:

$$K(X) = \{f_1/f_2 : f_1, f_2 \in K[X] \text{ homogeneous of the same degree, } f_2 \notin I_K(X)\}$$

with the equivalence relation  $f_1/f_2 \equiv f_3/f_4$  if and only if  $f_1f_4 - f_2f_3 \in I_K(X)$ . Elements of  $K(X)$  are called rational functions.

**Definition 52** (Function regularity). Let  $X$  be a variety and let  $f_1, f_2 \in K[X]$ . Then  $f_1/f_2$  is *defined* or *regular* at  $P$  if  $f_2(P) \neq 0$ . An equivalence class  $f \in K(X)$  is regular at  $P$  if it contains some  $f_1/f_2$  with  $f_1, f_2 \in K[X]$  (if  $X$  is projective then necessarily  $\deg(f_1) = \deg(f_2)$ ) such that  $f_1/f_2$  is regular at  $P$ .

Note that there may be many choices of representative for the equivalence class of  $f$ , and only some of them may be defined at  $P$ .

**Definition 53** (Rational map). Let  $X$  be an affine or projective variety over a field  $K$  and  $Y$  an affine variety in over  $K$ . Let  $\phi_1, \dots, \phi_n \in K(X)$ . A map  $\phi : X \rightarrow K^n$  of the form

$$\phi(P) = (\phi_1(P), \dots, \phi_n(P))$$

is *regular* at a point  $P \in X(\bar{K})$  if all  $\phi_i$ , for  $1 \leq i \leq n$ , are regular at  $P$ .

A *rational map*  $\phi : X \rightarrow Y$  defined over  $K$  is a map of the form  $\phi(P) = (\phi_1(P), \dots, \phi_n(P))$  such that, for all  $P \in X(\bar{K})$  for which  $\phi$  is regular at  $P$ , then  $\phi(P) \in Y(\bar{K})$ .

Let us now look at the projective version of the same definitions.

Let  $X$  be an affine or projective variety over a field  $K$  and  $Y$  a projective variety in  $\mathbb{P}^n[K]$  over  $K$ . Let  $\phi_1, \dots, \phi_n \in K(X)$ . A map  $\phi : X \rightarrow \mathbb{P}^n[K]$  of the form

$$\phi(P) = [\phi_0(P), \dots, \phi_n(P)]$$

is *regular* at a point  $P \in X(\bar{K})$  if there is some function  $g \in K(X)$  such that all  $g\phi_i$ , for  $0 \leq i \leq n$ , are regular at  $P$  and, for some  $0 \leq i \leq n$ , one has  $(g\phi_i)(P) \neq 0$ <sup>1</sup>.

A *rational map*  $\phi : X \rightarrow Y$  defined over  $K$  is a map of the form  $\phi(P) = [\phi_0(P), \dots, \phi_n(P)]$  such that, for all  $P \in X(\bar{K})$  for which  $\phi$  is regular at  $P$ , then  $\phi(P) \in Y(\bar{K})$ .

**Definition 54** (Morphism and isomorphism between (subsets of) varieties). Let  $X$  and  $Y$  be varieties over  $K$  and let  $U \subseteq X$  be open sets. A rational

<sup>1</sup>This last condition is to prevent  $\phi$  mapping to  $[0, \dots, 0]$ , which is not a point in  $\mathbb{P}^n[K]$ .



map  $\phi : U \rightarrow Y$  over  $K$  which is regular at every point  $P \in U(K)$  is called a *morphism* over  $K$ .

Let  $U \subseteq X$  and  $V \subseteq Y$  be open. If  $\phi : U \rightarrow Y$  is a morphism over  $K$  and  $\psi : V \rightarrow X$  is a morphism over  $K$  such that  $\phi \circ \psi$  and  $\psi \circ \phi$  are the identity on  $V$  and  $U$  respectively, then we say that  $U$  and  $V$  are *isomorphic* over  $K$ , and  $\phi, \psi$  are *isomorphisms*. If  $U$  and  $V$  are isomorphic we write  $U \cong V$ .



# Bibliography

- [1] asecuritysite.com post-quantum digital signature schemes, link : [https://asecuritysite.com/pqc/falcon01#:~:text=With%20Falcon%2D512%20\(which%20has,signature%20size%20of%201%2C280%20bytes.,](https://asecuritysite.com/pqc/falcon01#:~:text=With%20Falcon%2D512%20(which%20has,signature%20size%20of%201%2C280%20bytes.,) note : Accessed: 2022-02-16.
- [2] Crystals-dilithium: A lattice-based digital signature scheme, link : <https://eprint.iacr.org/2017/633.pdf>, note : Accessed: 2022-02-12.
- [3] Crystals-dilithium algorithm specifications and supporting documentation, link : <https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf>, note : Accessed: 2022-02-12.
- [4] Crystals-dilithium presentation, 3rd round nist, link : <https://csrc.nist.gov/CSRC/media/Presentations/crystals-dilithium-round-3-presentation/images-media/session-1-crystals-dilithium-lyubashevsky.pdf>, note : Accessed: 2022-02-12.
- [5] Falcon: Fast-fourier lattice-based compact signatures over ntru, link : <https://falcon-sign.info/falcon.pdf>, note : Accessed: 2022-02-12.
- [6] Post-quantum cryptography standardization, link : <https://csrc.nist.gov/Projects/post-quantum-cryptography/>

- post-quantum-cryptography-standardization, note : Accessed: 2021-12-15.
- [7] Post-quantum cryptography standardization, link : <https://csrc.nist.gov/projects/stateful-hash-based-signatures>, note : Accessed: 2021-12-15.
- [8] Rainbow - algorithm specification and documentation, link : <https://troll.iis.sinica.edu.tw/by-publ/recent/Rainbow3round.pdf>, note : Accessed: 2022-02-12.
- [9] Miklós Ajtai, Ravi Kumar, and Dandapani Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 601–610, 2001.
- [10] Dave Bacon. Cse 599d-quantum computing grover’s algorithm. *Department of Computer Science & Engineering. University of Washington. University of Washington, Washington*, 16, 2006.
- [11] Daniel J Bernstein. Introduction to post-quantum cryptography. In *Post-quantum cryptography*, pages 1–14. Springer, 2009.
- [12] Reinier Bröker. Constructing supersingular elliptic curves. *J. Comb. Number Theory*, 1(3):269–273, 2009.
- [13] Leon Groot Bruinderink and Andreas Hülsing. “oops, i did it again”–security of one-time signatures under two-message attacks. In *International Conference on Selected Areas in Cryptography*, pages 299–322. Springer, 2017.
- [14] Johannes Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. On the security of the winternitz one-time signature scheme. *International Journal of Applied Cryptography*, 3(1):84–96, 2013.

- 
- [15] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. Xmss-a practical forward secure signature scheme based on minimal security assumptions. In *International Workshop on Post-Quantum Cryptography*, pages 117–129. Springer, 2011.
- [16] Johannes Buchmann, Erik Dahmen, and Michael Schneider. Merkle tree traversal revisited. In *International Workshop on Post-Quantum Cryptography*, pages 63–78. Springer, 2008.
- [17] Fabio Campos, Tim Kohlstadt, Steffen Reith, and Marc Stöttinger. Lms vs xmss: Comparison of stateful hash-based signature schemes on arm cortex-m4. In *International Conference on Cryptology in Africa*, pages 258–277. Springer, 2020.
- [18] Andrew Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *Journal of Mathematical Cryptology*, 8(1):1–29, 2014.
- [19] David A Cooper, Daniel C Apon, Quynh H Dang, Michael S Davidson, Morris J Dworkin, and Carl A Miller. Recommendation for stateful hash-based signature schemes. *NIST Special Publication*, 800:208, 2020.
- [20] Jean Marc Couveignes. Hard homogeneous spaces. *IACR Cryptol. ePrint Arch.*, 2006:291, 2006.
- [21] Gian Fausto Dell’Antonio. *Aspetti matematici della meccanica quantistica*. Bibliopolis, 2011.
- [22] Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In *International conference on applied cryptography and network security*, pages 164–175. Springer, 2005.
- [23] Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. volume 3531, pages 164–175, 06 2005.

- [24] David P DiVincenzo. Topics in quantum computers. In *Mesoscopic electron transport*, pages 657–677. Springer, 1997.
- [25] Göran Einarsson. Probability analysis of a quantum computer. *arXiv preprint quant-ph/0303074*, 2003.
- [26] Artur Ekert and Richard Jozsa. Quantum computation and shor’s factoring algorithm. *Reviews of Modern Physics*, 68(3):733, 1996.
- [27] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.
- [28] Denisa OC Greconici, Matthias J Kannwischer, and Daan Sprenkels. Compact dilithium implementations on cortex-m3 and cortex-m4. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 1–24, 2021.
- [29] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.
- [30] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. Ntru: A ring-based public key cryptosystem. In *International algorithmic number theory symposium*, pages 267–288. Springer, 1998.
- [31] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *International Workshop on Post-Quantum Cryptography*, pages 19–34. Springer, 2011.
- [32] Panos Kampanakis and Scott Fluhrer. Lms vs xmss: Comparison of two hash-based signature standards. *Cryptology ePrint Archive*, 2017.

- 
- [33] Matthias J Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stofelen. pqm4: Testing and benchmarking nist pqc on arm cortex-m4. 2019.
- [34] Jeffrey C Lagarias. Knapsack public key cryptosystems and diophantine approximation. In *Advances in cryptology*, pages 3–23. Springer, 1984.
- [35] C Lavor, LRU Manssur, and R Portugal. Shor’s algorithm for factoring large integers, 2008, da. *arXiv preprint quant-ph/0303175*.
- [36] Frank T Leighton and Silvio Micali. Large provably fast and secure digital signature schemes based on secure hash functions, July 11 1995. US Patent 5,432,852.
- [37] Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261(ARTICLE):515–534, 1982.
- [38] SJ Lomonaco. Shor’s quantum factoring algorithm. In *Proceedings of Symposia in Applied Mathematics*, volume 58, pages 161–180, 2002.
- [39] Le Van Luyen et al. An improved identity-based multivariate signature scheme based on rainbow. *Cryptography*, 3(1):8, 2019.
- [40] Enrique Martin-Lopez, Anthony Laing, Thomas Lawson, Roberto Alvarez, Xiao-Qi Zhou, and Jeremy L O’Brien. Experimental realization of shor’s quantum factoring algorithm using qubit recycling. *Nature photonics*, 6(11):773–776, 2012.
- [41] Tsutomu Matsumoto and Hideki Imai. Public quadratic polynomial-tuples for efficient signature-verification and message-encryption. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 419–453. Springer, 1988.
- [42] Robert J McEliece. A public-key cryptosystem based on algebraic. *Coding Thv*, 4244:114–116, 1978.

- 
- [43] Alfred J Menezes, Scott A Vanstone, and Paul C Van Oorschot. Handbook of applied cryptography (special indian edition), 2010.
- [44] Daniele Micciancio and Shafi Goldwasser. *Complexity of lattice problems: a cryptographic perspective*, volume 671. Springer Science & Business Media, 2002.
- [45] Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Prob. Contr. Inform. Theory*, 15(2):157–166, 1986.
- [46] Michael A Nielsen and Isaac L Chuang. Quantum computation and quantum information. *Phys. Today*, 54(2):60, 2001.
- [47] Raphael Overbeck and Nicolas Sendrier. Code-based cryptography. In *Post-quantum cryptography*, pages 95–145. Springer, 2009.
- [48] Dingyi Pei, Arto Salomaa, and Cunsheng Ding. *Chinese remainder theorem: applications in computing, coding, cryptography*. World Scientific, 1996.
- [49] John Proos and Christof Zalka. Shor’s discrete logarithm quantum algorithm for elliptic curves. *arXiv preprint quant-ph/0301141*, 2003.
- [50] Manohar Raavi, Simeon Wuthier, Pranav Chandramouli, Yaroslav Balytskyi, Xiaobo Zhou, and Sang-Yoon Chang. Security comparisons and performance analyses of post-quantum signature algorithms. In *International Conference on Applied Cryptography and Network Security*, pages 424–447. Springer, 2021.
- [51] Oded Regev. Lattice-based cryptography. In *Annual International Cryptology Conference*, pages 131–141. Springer, 2006.
- [52] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical computer science*, 53(2-3):201–224, 1987.



- [53] Jose-Antonio Septien-Hernandez, Magali Arellano-Vazquez, Marco Antonio Contreras-Cruz, and Juan-Pablo Ramirez-Paredes. A comparative study of post-quantum cryptosystems for internet-of-things applications. *Sensors*, 22(2):489, 2022.
- [54] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.
- [55] Peter W Shor and John Preskill. Simple proof of security of the bb84 quantum key distribution protocol. *Physical review letters*, 85(2):441, 2000.
- [56] Harshdeep Singh. Code based cryptography: Classic mceliece. *arXiv preprint arXiv:1907.12754*, 2019.
- [57] Jacques Stern. A new identification scheme based on syndrome decoding. In *Annual International Cryptology Conference*, pages 13–21. Springer, 1993.
- [58] Douglas R Stinson. Some observations on the theory of cryptographic hash functions. *Designs, Codes and Cryptography*, 38(2):259–277, 2006.
- [59] Anton Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Advances in Mathematics of Communications*, 4(2):215, 2010.
- [60] Erkan Tairi. *Isogenies for Post-Quantum Cryptography/submitted by Erkan Tairi, BSc*. PhD thesis, Universität Linz, 2018.
- [61] Dominique Unruh. Non-interactive zero-knowledge proofs in the quantum random oracle model. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 755–784. Springer, 2015.

- 
- [62] Lieven MK Vandersypen, Matthias Steffen, Gregory Breyta, Costantino S Yannoni, Mark H Sherwood, and Isaac L Chuang. Experimental realization of shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature*, 414(6866):883–887, 2001.
- [63] Jacques Vélú. Isogénies entre courbes elliptiques. *CR Acad. Sci. Paris, Séries A*, 273:305–347, 1971.
- [64] Lawrence C Washington. *Elliptic curves: number theory and cryptography*. CRC press, 2008.
- [65] Joachim Weidmann. *Linear operators in Hilbert spaces*, volume 68. Springer Science & Business Media, 2012.
- [66] Nanyang Xu, Jing Zhu, Dawei Lu, Xianyi Zhou, Xinhua Peng, and Jiangfeng Du. Quantum factorization of 143 on a dipolar-coupling nmr system. *arXiv preprint arXiv:1111.3726*, 2011.

# List of Figures

1.1	Qubit represented by an electron in a hydrogen atom . . . . .	20
1.2	NOT, AND, NAND, OR, NOR, XOR, XNOR gates. . . . .	27
1.3	CNOT gate . . . . .	28
1.4	$U_f$ representation . . . . .	29
1.5	Swapping circuit realization, and schematic symbol . . . . .	30
1.6	controlled-U gate . . . . .	30
1.7	Circuit symbol for measurement. . . . .	30
1.8	CNOT: a wrong circuit to "copy" a qubit. . . . .	31
1.9	Quantum circuit to evaluate $f(0)$ and $f(1)$ simultaneously. . .	33
2.1	Quantum circuit for finding the order of the positive integer $x$ modulo $N$ . . . . .	49
2.2	Probability distribution of $ \psi_3\rangle$ measured in the computational basis (for the case $b_0 = 3$ and $r = 8$ ). The horizontal axis has $2^t$ points. The number of peaks is $2^t/r$ and the period is $r$ . . .	52
2.3	Vectors $e^{2\pi ij/7}$ , ( $j = 0, \dots, 6$ ) in the complex plane. Their sum is zero by symmetry arguments. This is an example of Eq. (2.7) for $N = 7$ , $k = 1$ . . . . .	54
2.4	Probability distribution of $ \psi_4\rangle$ measured in the computational basis. The horizontal axis has $2^t$ points, only the non-null terms are shown. The number of peaks is $r$ and the period is $2^t/r$ . . . . .	56

2.5	Plot of $Prob(j)$ against $j$ . Compare to the plot of Fig. 2.4, where peaks are not spread and have the same height. . . . .	60
3.1	BB84 quantum key exchange protocol. The channel represented in the figure, for simplicity, is considered to be noise-free, and also the absence of Eve is assumed (in other words $\epsilon$ is the identity function). To facilitate the reading of the scheme, the value 0 of the bit $b_i$ (or $b'_i$ ) has been indicated with "st" (for Standard Computational Basis), while the value 1 with "H" (Hadamard Base). When the $i$ -th bit of $a$ , $a_i$ , was found to be shared with probability 1, ( i.e., when $b_i = b'_i$ ), it is value is reported in the last line. Otherwise a "?" is returned, which means "unacceptable bit". . . . .	73
4.1	Finalists and Alternative Candidates of NIST Post-Quantum Cryptography Standardization Program Selection, round three.	81
5.1	2-dimensional SVP example . . . . .	88
5.2	2-dimensional CVP example . . . . .	88
5.3	2-dimensional SIVP example . . . . .	89
5.4	Code-based public-key encryption. The ciphertext is a noisy code word that only the legitimate user can correct to recover the cleartext. . . . .	97
5.5	Attacking the McEliece PKC. Table from [47]. . . . .	98
5.6	The idea of trapdoor function. A trapdoor function $f$ with its trapdoor $t$ can be generated by an algorithm. $f$ can be efficiently computed, i.e., in probabilistic polynomial time. However, the computation of the inverse of $f$ is generally hard, unless the trapdoor $t$ is given. . . . .	101
5.7	Scheme of the SIDH protocol . . . . .	113
5.8	Isogeny-Diagram at the base of the SIDH protocol . . . . .	113
6.1	256-bit digest encoded in base $b = 16$ . . . . .	119

6.2	64 keys generation in base $b = 16$ . . . . .	121
6.3	Generation of signature for 64 digits in base $b = 16$ . . . . .	122
6.4	Signature check for $N_0$ in base $b = 16$ . . . . .	123
6.5	A signature for a digest in base $b = 16$ (from [19]) . . . . .	123
6.6	Example of a signature and its checksum (from [19]) . . . . .	125
6.7	Theoretical results of the computational complexity for two- message attacks against the Winternitz OTS. If the success probability of an attack is not constant in terms of complexity, the attack complexity is given to achieve a success probability of $1/2$ . (Table from: [13]) . . . . .	127
6.8	A Merkle Hash Tree . . . . .	129
6.9	Multi level Merkle's tree . . . . .	131
6.10	The XMSS tree construction . . . . .	133
6.11	The authentication path for leaf $i$ . . . . .	134
6.12	Meanings for parameteres used in this Subsection. . . . .	137
6.13	Sizes (in bytes) of HBS schemes based on schemes parameters. . . . .	138
6.14	Sizes (in bytes) of HBS schemes based for various parameters, and $n = m = 32$ . . . . .	138
6.15	Percentage of time spent doing hash compression operations during an OTS computation. . . . .	139
6.16	Number of hash compression operations expected for various OTS operations SHA-256, $w = 16$ . . . . .	140
6.17	Comparison of LMS ( $w = 256$ ) and XMSS ( $w = 16$ ). . . . .	141
6.18	5 different security levels in NIST competition. . . . .	142
6.19	Falcon performance on an Intel® Core® i5-8259U CPU ("Coffee Lake" core, clocked at 2.3 GHz). Sizes of pub length and signature length in bytes. . . . .	144
6.20	Crystals-Dilithium sizes, relatively to different security levels, on Skylake platform. . . . .	146

6.21	In the first table, we have the parameters and the signature size in relation to the NIST security category; these parameters are equal in the 3 different version of Rainbow. In the second table we have the sizes for Rainbow in its 3 different versions. In particular $ pk $ and $ sk $ are in expressed $kB$ . . . . .	148
6.22	Performance (cycles) for Rainbow in its 3 different versions (on Linux/Skylake using AVX2 instructions). . . . .	148
6.23	Performance for standard Rainbow on the NIST platform. . .	149
6.24	Performance for CZ-Rainbow on the NIST platform. . . . .	149
6.25	Performance for Compressed Rainbow on the NIST platform. .	149
6.26	Comparison between Crystals-Dilithium and Falcon sizes. . . .	150
6.27	Comparison between Crystals-Dilithium and Falcon performance on Cortex M3 and Cortex M4 platforms. . . . .	150
6.28	Sizes and security levels for ECDSA, RSA, Ed25519 and Ed448 (useful as a term of comparison with post-quantum schemes). .	151
6.29	Sizes (in byte) for post-quantum digital signature schemes. . .	151
6.30	Performance on M4 (ARM Cortex-M4 dev) measured in CPU operations per second. The data in this table were taken from [33]. Note, no Rainbow assessment has been performed in [33], so LUOV (an Oil-and-Vinegar method) has been used to give an indication of performance levels. . . . .	152
6.31	Stack memory size on an ARM Cortex-M4 device (data from [33]) and measured in bytes. Note, no Rainbow assessment has been performed in [33], so LUOV (an Oil-and-Vinegar method) has been used to give an indication of performance levels. . . . .	152
B.1	Group Law: sum of two distinct points in a), doubling of point $P$ in b). . . . .	165
B.2	Associativity for the Group Law. . . . .	166
B.3	Alignment equation in 4 cases. . . . .	166

B.4 Group Law: case of two points living on the same vertical line. 167





# Ringraziamenti

**Disclaimer:** *questi ringraziamenti non verranno letti ad alta voce (ma nemmeno a voce bassa o sussurrati) dal sottoscritto Murolo Giuseppe, perchè sennò poi piango e non riesco a leggere.*

*Per il medesimo motivo, ogni qualvolta in cui verrà intonato il coro "...di-sco-rso, di-sco-rso!" si rimanda l'interessato a leggere questo paragrafo.*

*Continuando a leggerlo, accetterai questa clausola.*

*NB: non c'è scritto che se non continui a leggere rifiuti la clausola.*

Questo percorso per me così arduo è stato reso affrontabile grazie ad alcune persone che mi sono state vicino, nei bei momenti quanto nei brutti. Questo paragrafo è per loro. Chiedo scusa se, come è molto probabile che sia successo, ho dimenticato qualcuno. Ma se, anche per un breve momento, mi sei stato vicino o mi hai aiutato, sappi che l'ho notato e ringrazio anche te.

Voglio partire con il ringraziare i miei genitori: grazie mamma, grazie papà, il vostro amore ed il vostro imperterrito supporto è stato il mio carburante.

Ringrazio mia sorella Debora, per tutto l'aiuto diretto ed indiretto che mi ha, sempre, assicurato.

Ringrazio Marcello, praticamente un fratello per me, fin dai tempi del liceo.

Ringrazio Franca e Lucia per esserci sempre state. Voi 3 siete il mio porto sicuro a Torremaggiore.

Ringrazio quel matto di Mattarello (Mattarello è un vezzeggiativo di matto?):

un punto di riferimento essenziale qui a Bologna (prima no perchè mi chiamavi Moreno). Sei un amico raro e prezioso per me.

Grazie a Puer, amico importante, persona dal cuore d'oro.

Grazie alla Confraternita: oltre al già citato Matteo, ringrazio Ettore e Federico. Siete gli amici di sempre, so che posso contare in ogni occasione su di voi. Siete quelli con i quali ho condiviso le avventure più pazze... ad altre 1000 di queste!

Ringrazio i miei amici matematici che ho conosciuto quando mi sono ritrovato innestato a Bologna: Umberto, Trebe, Nico, Evi, Marco, Forno, Simo, Ettore, Gaia, Sacha, Francesco. Siete i coetanei che ho conosciuto dai quali ho imparato più in assoluto. E, credetemi, non mi riferisco solo alla matematica: siete un esempio per me.

Ringrazio la seconda tranche di matematici che mi ha accolto una volta approdato alla magistrale: Chiara, Lorenzo, Sofia, Titta, Anna, Madda, Mary, Federica, Marco, Davide DG, Laurina, Matteo. Siete persone fantastiche!

Tra queste ho volutamente omesso Davide: sei stato un ottimo coinquilino, ma soprattutto un ottimo amico. Ho passato bellissimi momenti con te, sei davvero un pazzo scatenato, grazie di tutto!

Dulcis in fundo: grazie a te che mi hai donato una forza immensa dalla prima sera che sono uscito con te, che sei disposta a così tanto per me, che costantemente mi ricordi cos'è l'amore, e come ci si sente ad essere davvero amati. Grazie Arianna, per tutto quello che hai fatto, che fai, per quello che rappresenti per me, per il tuo coraggio e la tua determinazione. Ti amo <3.

Gente elencata, vi voglio un sacco bene.

E ora — giocando a far finta di non vivere nel bel mezzo di una pandemia e di una guerra — beviamo un po', va', *ca c vo'*!

Giuseppe