

UNIVERSITY OF BERGEN  
DEPARTMENT OF INFORMATICS

---

Quantum Computing, how it is  
jeopardizing RSA, and Post-Quantum  
Cryptography

---

*Author:* Anna Fossen-Helle

*Supervisor:* Matthew Parker

*Co-supervisor:* Sondre Rønjom



UNIVERSITETET I BERGEN  
*Det matematisk-naturvitenskapelige fakultet*

June, 2020

## **Abstract**

Quantum computers are a fact and with the quantum computers follows quantum algorithms. How will quantum computing affect how we look at public-key cryptography? And more specifically: how will it affect the most widely used public-key algorithm RSA? The impact of quantum computing is unimaginable and it will affect a massive amount of applications like e-commerce, social networks, mobile phones, generally our day to day life.

A solution has been presented: Post-Quantum Cryptography. Even though Post-Quantum primitives have been suggested, there is not yet any algorithms that has been chosen to replace our current public-key standards. A standardizing process was started in 2016 by NIST and is still ongoing.

## **Acknowledgements**

I would like to thank my supervisors, Matthew G. Parker and Sondre Rønjom at the Selmer Center, for guidance, support and patience through the process of writing my thesis. Also a special thanks to Martha Norberg Hovd, PhD candidate at Simula@UiB, for being of assistance whenever I needed it.

Also I would like to thank my fellow students and the Department of Informatics for providing me with a great study environment. It has been a joy to be able attend all the student functions and informative Department seminars.

Last but not least, a big thanks to my family and friends, both at home and here in Bergen, for all the support and love throughout my studies. I am forever grateful.

**Anna Fossen-Helle**

16 June, 2020

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Cryptography . . . . .	1
1.1.1	Basics of cryptography . . . . .	2
1.1.2	The purpose of cryptography . . . . .	2
1.2	Complexity . . . . .	3
1.3	Outline . . . . .	4
<b>2</b>	<b>Public-key cryptography</b>	<b>6</b>
2.1	Symmetric vs. asymmetric cryptography . . . . .	6
2.2	Important public-key algorithms . . . . .	10
2.3	Number theory for public-key algorithms . . . . .	11
2.3.1	Euclidean algorithm . . . . .	11
2.3.2	Extended Euclidean algorithm . . . . .	13
2.3.3	Euler's Phi function . . . . .	14
2.3.4	Fermat's Little Theorem and Euler's Theorem . . . . .	15
<b>3</b>	<b>The RSA cryptosystem</b>	<b>17</b>
3.1	Encryption and decryption . . . . .	17
3.2	Key generation and proof of correctness . . . . .	19
3.3	Finding large primes . . . . .	22
3.4	Attacks on RSA . . . . .	22
3.4.1	Protocol attacks . . . . .	23
3.4.2	Mathematical attacks . . . . .	23
3.4.3	Side-channel attacks . . . . .	24
<b>4</b>	<b>Quantum mechanics</b>	<b>25</b>
4.1	The postulates of quantum mechanics . . . . .	25

4.2	Linear algebra . . . . .	26
4.2.1	Bases and linear independence . . . . .	28
4.2.2	Linear operators and matrices . . . . .	28
4.2.3	The Pauli matrices . . . . .	29
4.2.4	Inner products . . . . .	30
4.2.5	Eigenvectors and eigenvalues . . . . .	31
4.2.6	Tensor products . . . . .	31
4.3	Entanglement . . . . .	32
4.4	Conditions for quantum computation . . . . .	35
4.4.1	Representation of quantum information . . . . .	35
4.4.2	Performance of unitary transformations . . . . .	35
4.4.3	Preparations of fiducial initial states . . . . .	36
4.4.4	Measurement of output result . . . . .	37
<b>5</b>	<b>Quantum computation</b>	<b>39</b>
5.1	Quantum bits . . . . .	39
5.1.1	Single qubits . . . . .	39
5.1.2	Multiple qubits . . . . .	40
5.2	Quantum gates and circuits . . . . .	41
5.3	The quantum Fourier transform . . . . .	43
5.4	Phase estimation . . . . .	47
5.5	Order-finding and factoring . . . . .	50
5.5.1	Order-finding . . . . .	50
5.5.2	Factoring . . . . .	53
5.6	How does it apply to RSA? . . . . .	54
<b>6</b>	<b>Post-quantum cryptography</b>	<b>56</b>
6.1	Post-quantum primitives . . . . .	59
6.2	Post-quantum candidates . . . . .	61
<b>7</b>	<b>Conclusion</b>	<b>63</b>
7.1	Path forward . . . . .	63
	<b>Bibliography</b>	<b>65</b>

# List of Figures

1.1	Overview of the research field of cryptology [31, p.3]. . . . .	2
2.1	Basics of symmetric-key encryption [31, p.150] . . . . .	7
2.2	Analogy for symmetric cryptography [31, p.151]. . . . .	7
2.3	Analogy for asymmetric encryption [31, p.151]. . . . .	8
2.4	Basic protocol for public-key encryption [31, p.152]. . . . .	9
2.5	Basic key transport protocol with AES as an example of a symmetric cipher [31, p.153]. . . . .	9
3.1	Principal approach to generating primes for RSA [31, p.187] . . . . .	22
5.1	Illustration of unitary transformation [32, p.6]. . . . .	42
5.2	Illustration of the <b>CNOT</b> transformation [32, p.7]. . . . .	42
5.3	Efficient circuit for the quantum Fourier transform. This circuit is easily derived from the product representation (5.14) for the quantum Fourier transform. Not shown are the swap gates at the end of the circuit which reverse the order of the qubits, or normalization factors of $1/\sqrt{2}$ in the output [29, p.219]. . . . .	45
5.4	The first stage of the phase estimation procedure. Normalization factors of $1/\sqrt{2}$ have been omitted, on the right [29, p.222]. . . . .	48
5.5	Schematic of the overall phase estimation procedure. The top $t$ qubits (the '\ ' denotes a bundle of wires, as usual) are the first register, and the bottom qubits are the second register, numbering as many as required to perform $U$ . $ u\rangle$ is an eigenstate of $U$ with eigenvalue $e^{2\pi i\varphi}$ . The output of the measurement is an approximation to $\varphi$ accurate to $t - \lceil \log(2 + \frac{1}{2\epsilon}) \rceil$ bits, with probability of success at least $1 - \epsilon$ [29, p.223]. . . . .	49
5.6	Quantum circuit for the order-finding algorithm. This circuit can also be used for factoring, using the reduction given in the next section [29, p.229]. . . . .	51

# List of Tables

4.1	Summary of some standard quantum mechanical notation for notions from linear algebra [29, p.62]. . . . .	27
5.1	Additional notation for quantum circuits [29]. . . . .	43
6.1	Impact of quantum computing on common cryptographic algorithms [16, p.2]	58
6.2	Continued: Impact of quantum computing on common cryptographic algorithms [1, p.24] . . . . .	58
6.3	Public-key encryption, key-establishment and digital signature algorithm candidates remaining in round 2 of the NIST competition [30]. . . . .	62

# Chapter 1

## Introduction

We are entering a new era of computing and quantum computers are becoming more realized. Quantum computers have existed for quite a few years now but they are extremely expensive and exist at near zero Kelvin - so they won't be on your desk any time soon. There is research into topological quantum computers, which would exist at room temperature, but they are still just theoretical - although there has been progress recently in this area. For my thesis I want to look at one of the most widely used public-key algorithms, namely RSA, and how the existence of quantum computing is jeopardizing the security of the algorithm, based on the factoring problem. The security of public-key cryptosystems are based on the difficulty of the number theoretic problems that are used in the cryptosystems, so further on I will introduce post-quantum cryptographic algorithms that may extinguish the threat from quantum algorithms that have been created to solve factoring.

### 1.1 Cryptography

Cryptography is one of the two components covered by the research field of cryptology. Cryptology is the word used for the study of secrecy. It is divided into two main components, cryptography, as mentioned, and cryptanalysis. Cryptography is the component for the study of building algorithms for encryption and decryption. This component is also divided in to three parts, which is symmetric cryptography, asymmetric cryptography and protocols. Cryptanalysis is the other component of cryptology which is the study of ciphertext analysis to recover hidden information [21, p.322]. For my thesis I will mainly focus on cryptography and more specified: asymmetric(public-key) cryptography.



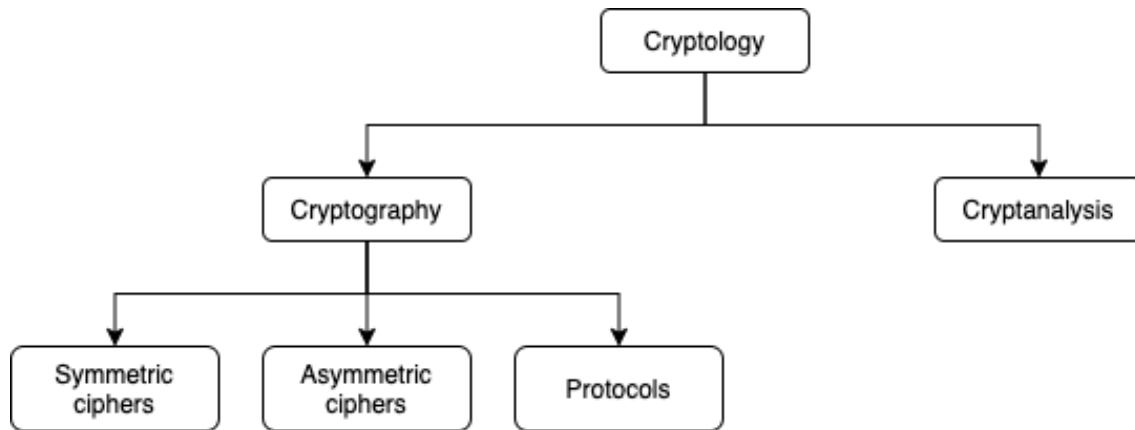


Figure 1.1: Overview of the research field of cryptology [31, p.3].

### 1.1.1 Basics of cryptography

In the cryptography universe, there are three well known people. Namely Alice, Bob and Eve. Alice and Bob acts as the parts of sender and receiver of messages and Eve acts as eavesdropper in the conversation. Even though Alice and Bob would like to think that the conversation is exclusively just between them, this is not strictly a fact. A malicious user, like Eve, can eavesdrop into the conversation or even pose as one of the users if proper security measures are not taken.

### 1.1.2 The purpose of cryptography

Cryptography is used to preserve some functionalities or services to overcome potential threats. These functionalities or services can be represented by the acronym CAIN[21, p.324]:

- Confidentiality
- Authentication
- Integrity
- Nonrepudiation

1. *Confidentiality* is ensured by using encryption algorithms on the information stored or manipulated. It consists of *preventing unauthorized access* to information for user that are not supposed to access it. These users are able to access the ciphertext after the encryption of the information, but they are not able to decrypt the ciphertext in order to get their hands on the information [21, p.324–325].

2. *Authentication* of the participants of message or information exchange. In order to preserve this functionality one must be able to recognize identity theft. Alice and Bob may authenticate themselves to each other by proving that they know a secret  $S$  and that they are the only ones that knows it [21, p.325].
3. *Integrity* of the messages sent or information stored. This is about checking that nothing has been falsified during transmission. We want the information to be unchanged while transmitted from Alice to Bob. A malicious person, Eve, may change the information in the message during transmission. Integrity checks are in place to notify Alice and Bob when the integrity of the transmission has been compromised [21, p.325].
4. *Nonrepudiation* of information is not a protection against a third party, but rather a protection for protagonists against each other. If Alice sends a message  $M$ , she must not be able to afterward pretend to Bob that she did not send it, or that she has sent a message  $M'$  and that it was actually misunderstood. For this, one associates signature-based algorithms [21, p.325–326].

## 1.2 Complexity

Going forward, it is important to state that all problems belong to a complexity class. *Computational complexity theory* is the subject of classifying the difficulty of various computational problems, both classical and quantum. The most basic idea is that of a *complexity class*. A complexity class can be thought of as a collection computational problems, all of which share some common feature with respect to the computational resources needed to solve these problems.

Two of the most important complexity classes are **P** (Polynomial) and **NP** (Non-deterministic). Roughly speaking, **P** is the class of computational problems that can be solved quickly on a classical computer. **NP** is the class of problems which have solutions which can be quickly checked on a classical computer. To understand the distinction between **P** and **NP**, consider the problem of finding the prime factors of an integer,  $n$ . So far as is known, there is no fast way of solving this problem on a classical computer, which suggests that the problem is not in **P**. On the other hand, if somebody tells you that some number  $p$  is a factor of  $n$ , then we can quickly check that this is correct by dividing  $p$  into  $n$ . Therefore, factoring is a problem in **NP**.

It is clear that **P** is a subset of **NP**, since if one is able to solve a problem it also implies

that one can check the solution. What is not so clear is whether or not there are problems in **NP** that are not in **P**. Perhaps the most important unsolved problem in theoretical computer science is to determine whether these two classes are different.

Most researchers believe that **NP** contains problems that are not in **P**. In particular, there is an important subclass of the **NP**-problems, the **NP**-complete problems, that are of especial importance for two reasons. First, there are thousands of problems that are known to be **NP**-complete. Second, any given **NP**-complete problem is in some sense 'at least as hard' as all other problems in **NP**. More precisely, an algorithm to solve a specific **NP**-complete problem can be fitted to solve any other problem in **NP**, with a small overhead. In particular, if  $\mathbf{P} \neq \mathbf{NP}$ , then it will follow that no **NP**-complete problem may be efficiently solved on a classical computer.

It is not known whether quantum computers can be used to quickly solve all the problems in **NP**, despite the fact that they can be used to solve some problems, like factoring, which is believed to be in **NP** but not in **P**. Note that factoring is not known to be an **NP**-complete problem, otherwise we would already know how to efficiently solve all problems in **NP** using quantum computers. It would certainly be very exciting if it were possible to solve all the problems in **NP** efficiently on a quantum computer[29, p.40–42].

### 1.3 Outline

- **Chapter 2** Public-key cryptography: in this chapter I will introduce public-key cryptography and some number theoretic problems that are widely used in public-key schemes. For this chapter and the next I will be using my old textbook from an introductory course in cryptography that I attended at the University of Bergen, named "Understanding Cryptography" [31] from 2010.
- **Chapter 3** The RSA cryptosystem: in this chapter I will go into depth in the RSA cryptosystem. As mentioned earlier, this will be the public-key cryptosystem that my main focus will be on.
- **Chapter 4** Quantum mechanics: this chapter will introduce quantum mechanics which is the base for quantum computing. This chapter and the next will be heavily based on the book "Quantum Computation and Quantum Information: *10th anniversary edition*" [29] and lecture notes [32] from John Preskill, Professor of Theoretical Physics, also based on the book.
- **Chapter 5** Quantum Computation: here I will introduce quantum computing and

algorithms used to break public-key algorithms, mainly RSA.

- **Chapter 6** Post-quantum cryptography: in this chapter I will introduce post-quantum cryptography, and the post-quantum algorithm competition initiated by NIST (National Institute of Standards and Technology). I will also give a brief overview of post-quantum primitives that may extinguish the threats from quantum algorithms in the previous chapter on quantum computing. This chapter is mainly based on the "Report on Post-Quantum Cryptography" [16] released by NIST in 2016.
- **Chapter 7** Conclusion.

# Chapter 2

## Public-key cryptography

Now that we know the basics of which services cryptography provides, we can move on to public-key cryptography. Public-key cryptography, also known as asymmetric cryptography, was publicly introduced by Whitfield Diffie, Martin Hellman and Ralph Merkle in 1976.

In this section I will show that asymmetric cryptography is very different from symmetric cryptography and algorithms like AES or DES. Most public-key algorithms are based on number-theoretic functions. This is quite different from symmetric ciphers, where the goal is usually not to have a compact mathematical description between input and output. Even though mathematical structures are often used for small blocks within symmetric ciphers, for instance, in the AES S-box, this does not mean that the entire cipher forms a compact mathematical description [31, p.149–150].

### 2.1 Symmetric vs. asymmetric cryptography

#### Symmetric cryptography

In order to understand the basics of asymmetric cryptography, we first need to understand the basics of symmetric cryptography. Here is a scheme to illustrate symmetric-key encryption [31, p.150]:

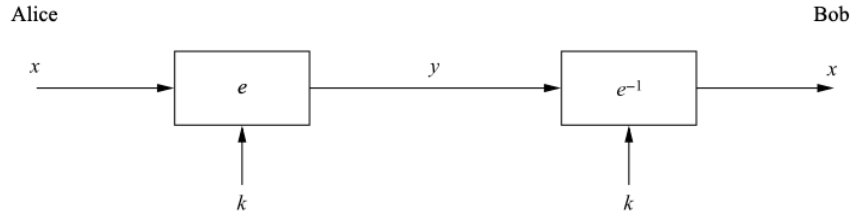


Figure 2.1: Basics of symmetric-key encryption [31, p.150]

A system is symmetric when it satisfies these two criteria:

1. The *same secret key* is used for encryption and decryption.
2. The encryption and decryption *functions* are very similar.



Figure 2.2: Analogy for symmetric cryptography [31, p.151].

There is a simple analogy for symmetric cryptography, as shown in Fig. 2.2. Assume there is a safe with a strong lock. Only Alice and Bob are able to open it with their copy of the key to the lock. The action of encrypting of a message can be seen as putting the message in the safe and locking it. In order to read it, i.e., decrypt the message, Bob uses his key and opens the safe.

Modern symmetric algorithms such as AES or 3DES are very secure, fast and are frequently used. However, there are several shortcomings associated with symmetric-key schemes, which is further discussed below [31, p.150].

**Key distribution problem** The key must be established between Alice and Bob using a secure channel. Remember that the communication link for the message is not secure, so sending the key over the channel directly - which would be the most convenient way of transporting it - can't be done [31, p.150].

**Number of keys** Even if the key distribution problem is solved, we must potentially deal with a very large number of keys. If each pair of users needs a separate pair of keys in a network with  $n$  users, there are:

$$\frac{n \cdot (n - 1)}{2}$$

key pairs, and every user has to store  $n - 1$  keys securely. Even for mid-size networks, say, a corporation with 2000 people, this requires more than 4 million key pairs must be generated and transported via secure channels [31, p.150–151].

**No protection against cheating by Alice or Bob** Alice and Bob have the same capabilities, since they possess the same key. As a consequence, symmetric cryptography cannot be used for applications where we would like to prevent cheating by either Alice or Bob as opposed to cheating by an outsider like Eve. For instance, in e-commerce applications it is often important to prove that Alice actually sent a certain message, say, an online order for a pair of shoes. If only symmetric cryptography is used and Alice changes her mind later, she can always claim that Bob, the vendor, has falsely generated the electronic purchase order. Preventing this is called *nonrepudiation* as you may recall from earlier in section 1.1.2 and can be achieved using asymmetric cryptography [31, p.151].

### Asymmetric cryptography

In order to overcome these drawbacks in symmetric cryptography, Diffie, Hellmann and Merkle had a revolutionary proposal based on the idea that it is not necessary that the key possessed by the person who *encrypts* the message (here: Alice) is secret. The crucial part is that Bob, the receiver, can only *decrypt* the message using a secret key. In order to realize a system like this, Bob publishes a public encryption key which is known to everyone. Bob also has a matching secret key, which is used for decryption. Thus, Bob's key  $k$  consists of two parts, a public part,  $k_{pub}$  and a private part,  $k_{pr}$  [31, p.152].



Figure 2.3: Analogy for asymmetric encryption [31, p.151].

A simple analogy of such a system is shown in Fig. 2.3. This system works quite similarly to the old mailbox on the corner of a street: Everyone can put a letter in the box, i.e., encrypt, but only a person with a private (secret) key can retrieve letters, i.e., decrypt. If we assume we have cryptosystems with such a functionality, a basic protocol for public-key encryption looks as illustrated in Fig. 2.4 [31, p.152].

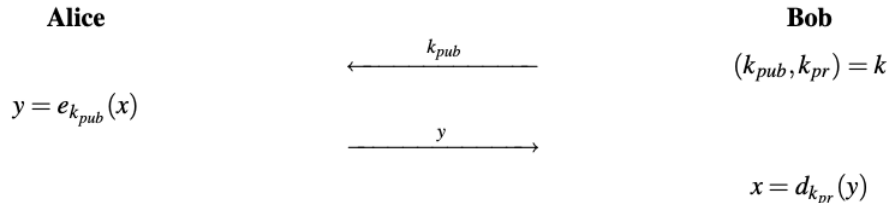


Figure 2.4: Basic protocol for public-key encryption [31, p.152].

By looking at that protocol one might argue that even though we can encrypt a message without a secret channel for key establishment, we still cannot exchange a key if we want to encrypt with, say, AES. However, the protocol can be easily modified for this use. What we have to do is to *encrypt a symmetric key*, e.g., an AES key, using the public-key algorithm. Once the symmetric key has been decrypted by Bob, both parties can use it to encrypt and decrypt messages using symmetric ciphers. Fig. 2.5 shows a basic key transport protocol where AES is used as the symmetric cipher for illustration purposes. The main advantage of the protocol in Fig. 2.5 over the protocol in Fig. 2.4 is that the payload is encrypted with a symmetric cipher, which tends to be much faster than an asymmetric algorithm[31, p.152].

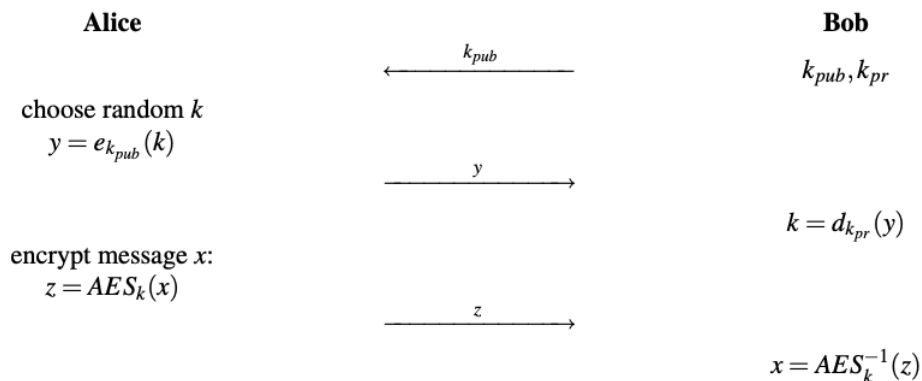


Figure 2.5: Basic key transport protocol with AES as an example of a symmetric cipher [31, p.153].



From the discussion so far, it looks as though asymmetric cryptography is a desirable tool for security applications. The question remains how one can build public-key algorithms. In the next chapter I will introduce one of the most used schemes, RSA. I would also like to mention that there are widely used schemes based on the Discrete Log Problem and Elliptic Curves even though I will not go into detail on such schemes. What they all have in common is that they are built from a common principle, the one-way function. The informal definition is as follows [31, p.152–153]:

**Definition 2.1.1.** One-way function

*A function  $f()$  is a one-way function if:*

1.  $y = f(x)$  is computationally easy, and
2.  $x = f^{-1}(y)$  is computationally infeasible.

Obviously, the adjectives "easy" and "infeasible" are not particularly exact. In mathematical terms, a function is easy to compute if it can be evaluated in polynomial time, i.e., its running time is a polynomial expression. In order to be useful in practical crypto schemes, the computation  $y = f(x)$  should be sufficiently fast that it does not lead to unacceptably slow execution times in an application. The inverse computation  $x = f^{-1}(y)$  should be so computationally intensive that it is not feasible to evaluate it in any reasonable time period, say, 10,000 years, when using the best known algorithm [31, p.153].

There are two popular one-way functions which are used in practical public-key schemes. The first is the integer factorization problem, on which RSA is based. Given two large primes, it is easy to compute the product. However, it is very difficult to factor the resulting product. In fact, if each of the primes has 150 or more decimal digits, the resulting product cannot yet be factored, even with thousands of PCs running for many years. The other one-way function that is used widely is the discrete logarithm problem [31, p.153].

## 2.2 Important public-key algorithms

In public-key cryptography, there are only three major families which are of practical relevance. They can be classified by their underlying computational problem:

- **Integer-Factorization schemes** Several public-key schemes are based on the fact that it is difficult to factor large integers. The most prominent representative of this algorithm family is RSA [31, p.155].

- **Discrete Logarithm schemes** There are several algorithms which are based on what is known as the discrete logarithm problem in finite fields. The most prominent examples include the Diffie-Hellmann key exchange, Elgamal encryption or the Digital Signature Algorithm(DSA) [31, p.155].
- **Elliptic Curve (EC) schemes** A generalization of the discrete logarithm algorithm are elliptic curve public-key schemes. The most popular examples include Elliptic Curve Diffie-Hellmann key exchange (ECDH) and the Elliptic Curve Digital Signature Algorithm (ECDSA) [31, p.155].

The first two families were proposed in the mid-1970s, and elliptic curves were proposed in the mid-1980s. There are no known attacks against any of these schemes if the parameters, especially the operand and key lengths, are chosen carefully. The RSA scheme based on integer factorization will be explained in more detail in the next chapter, but not schemes from the other families. This is because the consequences of solving the problem of integer factorization has the highest impact out of the three families. It is also important to note that each of these three families can be used to provide the main public-key mechanisms of key establishment, nonrepudiation through digital signatures and encryption of data [31, p.155].

## 2.3 Number theory for public-key algorithms

I will now show a few techniques from number theory which are essential for public-key cryptography. This will be the Euclidean algorithm, Euler's phi function, Fermat's Little Theorem and Euler's theorem. All are important for asymmetric algorithms, especially for understanding the RSA crypto scheme.

### 2.3.1 Euclidean algorithm

I start with the problem of computing the *greatest common divisor (gcd)*. The gcd of two positive integers is denoted by

$$\text{gcd}(r_0, r_1)$$

and is the largest positive number that divides both  $r_0$  and  $r_1$ . For instance  $\text{gcd}(21, 9) = 3$ . For small numbers, the gcd is easy to calculate by factoring both numbers and finding the largest common factor. The gcd is the product of all common prime factors:

$$2 \cdot 3 = 6 = \gcd(30, 84).$$

For the large numbers appearing in public-key schemes, factoring is often not possible, and a more efficient algorithm is used for gcd computations, the Euclidean algorithm. The algorithm, often referred to as Euclid's algorithm, is based on the simple principle that

$$\gcd(r_0, r_1) = \gcd(r_0 - r_1, r_1),$$

where we assume that  $r_0 > r_1$ , and that both numbers are positive integers. The process can also be applied iteratively:

$$\gcd(r_0, r_1) = \gcd(r_0 - r_1, r_1) = \gcd(r_0 - 2r_1, r_1) = \cdots = \gcd(r_0 - mr_1, r_1)$$

as long as  $(r_0 - mr_1) > 0$ . The algorithm uses the fewest number of steps if the maximum value of  $m$  is chosen. This is the case if we compute:

$$\gcd(r_0, r_1) = \gcd(r_0 \bmod r_1, r_1).$$

Since the first term  $(r_0 \bmod r_1)$  is smaller than the term  $r_1$ , they are usually swapped:

$$\gcd(r_0, r_1) = \gcd(r_1, r_0 \bmod r_1).$$

The core observation from this process is that we can reduce the problem of finding the gcd of two given numbers to that of the gcd of two smaller numbers. This process can be applied recursively until  $\gcd(r_l, 0) = r_l$  is finally obtained. Since each iteration preserves the gcd of the previous iteration step, it turns out that this final gcd of the original problem, i.e.,

$$\gcd(r_0, r_1) = \cdots \circ \gcd(r_l, 0) = r_l.$$

The Euclidean algorithm is very efficient, even with the very long numbers typically used in public-key cryptography. The number of iterations is close to the of digits of the input operands. That means, for instance, that the number of iterations of a gcd involving 1024-bit numbers is 1024 times a constant. Of course, algorithms with a few thousand iterations can easily be executed on today's PCs, making the algorithms very efficient in practice [31,

p.157–160].

### 2.3.2 Extended Euclidean algorithm

Above it was shown that finding the gcd of two integers  $r_0$  and  $r_1$  can be done by recursively reducing the operands. However, it turns out that finding the gcd is not the main application of the Euclidean algorithm. An extension of the algorithm allows us to compute modular inverses, which is of major importance in public-key cryptography. In addition to computing the gcd, the *extended Euclidean algorithm* (EEA) computes a linear combination of the form:

$$\gcd(r_0, r_1) = s \cdot r_0 + t \cdot r_1$$

where  $s$  and  $t$  are integer coefficients. This equation is often referred to as *Diophantine equation* [31, p.160].

The question now is: how are the coefficients  $s$  and  $t$  computed? The idea behind the algorithm is that the standard Euclidean algorithm is executed, but the current remainder  $r_i$  is expressed in every iteration as a linear combination of the form:

$$r_i = s_i r_0 + t_i r_1. \tag{2.1}$$

If we succeed with this, we end up in the last iteration with the equation:

$$r_l = \gcd(r_0, r_1) = s_l r_0 + t_l r_1 = s r_0 + t r_1.$$

This means that the last coefficient  $s_l$  is the coefficient  $s$  in Eq. 2.1 we are looking for, and also  $t_l = t$  [31, p.160].

Now we want to establish a recursive formulae for computing  $s_i$  and  $r_i$  in every iteration. Assume we are in iteration with the index  $i$ . In the two previous iterations we computed the values

$$r_{i-2} = [s_{i-2}]r_0 + [t_{i-2}]r_1 \tag{2.2}$$

$$r_{i-1} = [s_{i-1}]r_0 + [t_{i-1}]r_1 \tag{2.3}$$

In the current iteration  $i$  we first compute the quotient  $q_{i-1}$  and the new remainder  $r_i$  from  $r_{i-1}$  and  $r_{i-2}$  [31, p.161]:

$$r_{i-2} = q_{i-1} \cdot r_{i-1} + r_i.$$

This equation can be written as:

$$r_i = r_{i-2} - q_{i-1} \cdot r_{i-1} \tag{2.4}$$

Recall that the goal is to represent the new remainder  $r_i$  as a linear combination of  $r_0$  and  $r_1$  as shown in Eq. 2.1. The core step for achieving this happens now: in Eq. 2.4 we simply substitute  $r_{i-2}$  by Eq. 2.2 and  $r_{i-1}$  by Eq. 2.3 [31, p.161]:

$$r_i = (s_{i-2}r_0 + t_{i-2}r_1) - q_{i-1}(s_{i-1}r_0 + t_{i-1}r_1)$$

If we arrange the terms we obtain the desired result:

$$\begin{aligned} r_i &= [s_{i-2} - q_{i-1}s_{i-1}]r_0 + [t_{i-2} - q_{i-1}t_{i-1}]r_1 \\ r_i &= [s_i]r_0 + [t_i]r_1 \end{aligned} \tag{2.5}$$

Eq. 2.5 also gives us immediately the recursive formulae for computing  $s_i$  and  $t_i$ , namely  $s_i = s_{i-2} - q_{i-1}s_{i-1}$  and  $t_i = t_{i-2} - q_{i-1}t_{i-1}$ . These recursions are valid for index values  $i \geq 2$ . Like any recursion, we need starting values for  $s_0, s_1, t_0$  and  $t_1$ . These initial values is set to be  $s_0 = 1, s_1 = 0, t_0 = 0$  and  $t_1 = 1$  [31, p.161–162].

### 2.3.3 Euler’s Phi function

This is a number theoretic tool that is useful for public-key cryptography, and especially for RSA. We consider the ring  $\mathbb{Z}_m$ , i.e., the set of integers  $0, 1, \dots, m - 1$ . We are interested in the problem of knowing *how many* numbers in this set that are relatively prime to  $m$ . This quantity is given by *Euler’s phi function*, which is defined as follows [31, p.164–165]:  
Definition[chapter]

**Definition 2.3.1.** Euler’s phi function

*The number of integers in  $\mathbb{Z}_m$  relatively prime to  $m$  is denoted by  $\Phi(m)$ .*

From examples with small numbers, one can count all the integers in  $\mathbb{Z}_m$  which are relatively prime. This is a relatively naive way to compute Euler's phi function and is completely out of reach for the large numbers used in public-key cryptography. Fortunately, there exists a relation to calculate it much easier if know the factorization of  $m$ :

**Theorem 2.3.1.** *Let  $m$  have the following canonical factorization*

$$m = p_1^{e_1} \cdot p_2^{e_2} \cdot \dots \cdot p_n^{e_n}$$

*where the  $p_i$  are distinct prime numbers and  $e_i$  are positive integers, then*

$$\Phi(m) = \prod_{i=1}^n (p_i^{e_i} - p_i^{e_i-1}).$$

Since the value of  $n$ , i.e., the number of distinct prime factors, is always quite small even for large numbers  $m$ , evaluating the product symbol  $\prod$  is computationally easy [31, p.165–166].

It is important to stress that the factorization of  $m$  needs to be known in order to calculate Euler's phi function quickly this way. As you'll see in the next chapter, this property is at the heart of the RSA public-key scheme: Conversely, if the factorization of a certain number is known, Euler's phi function can be computed and the ciphertext can be decrypted. If the factorization is not known, then the phi function can not be computed and, hence, the ciphertext can not be decrypted [31, 166].

### 2.3.4 Fermat's Little Theorem and Euler's Theorem

The next to theorems I will describe is quite useful in public-key cryptography. I will start with *Fermat's Little Theorem*. This theorem is quite useful for primality testing and in many other aspects of public-key cryptography. For the RSA crypto scheme this theorem is useful for finding large primes. The theorem gives a seemingly surprising result if we do exponentiations modulo an integer.

**Theorem 2.3.2.** *Let  $a$  be an integer and  $p$  be a prime, then:*

$$a^p = a \pmod{p}$$

Note that arithmetic in finite fields  $GF(p)$  is done modulo  $p$ , and hence, the theorem holds for all integers  $a$  which are elements of a finite field  $GF(p)$ . The theorem can be stated in the form:

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

which is often useful in cryptography. One application is the computation of the inverse in a finite field. The equation can be rewritten as  $a \cdot a^{p-2} \equiv 1 \pmod{p}$ . This is exactly the definition of the multiplicative inverse. Thus, it immediately follows that we have a way of inverting an integer  $a$  modulo a prime:

$$a^{-1} \equiv a^{p-2} \pmod{p} \tag{2.6}$$

Note that this inversion method only holds if  $p$  is a prime.

Performing the exponentiation in Eq. (3.1) is usually slower than using the Extended Euclidean algorithm. However, there are situations where it is advantageous to use Fermat's Little Theorem, e.g., on smart cards or other devices which have a hardware accelerator for fast exponentiation anyway. This is not uncommon because many public-key algorithms require exponentiation[31, p.166–167].

A generalization of Fermat's Little Theorem to any integer moduli, i.e., moduli that are not necessarily primes, is *Euler's theorem*.

**Theorem 2.3.3.** *Let  $a$  and  $m$  be integers with  $\gcd(a, m) = 1$ , then:*

$$a^{\Phi(m)} \equiv 1 \pmod{m}.$$

Since it works in modulo  $m$ , it is applicable to integer rings  $\mathbb{Z}_m$  [31, p.167–168].

This concludes my section on public-key cryptography. I will now move on to looking at the RSA cryptosystem.

# Chapter 3

## The RSA cryptosystem

The RSA crypto scheme, also referred to as the Rivest-Shamir-Adleman algorithm, is one of the most widely used asymmetric cryptographic schemes. RSA was patented in the USA (but not in the rest of the world) until 2000.

There are many applications for RSA, but in practice it is most often used for:

- encryption of small pieces of data, especially for key transport.
- digital signatures.

However, it should be noted that RSA encryption is not meant to replace symmetric ciphers because it is several times slower than ciphers such as AES. This is because of the many computations involved in performing RSA. Thus, the main use of the encryption feature is to securely exchange a key for a symmetric cipher (key transport). In practice, RSA is often used together with a symmetric cipher such as AES, where the symmetric cipher does the actual bulk data encryption [31, p.174].

The underlying one-way function of RSA is the integer factorization problem: Multiplying two large primes is computationally easy, while factoring the resulting product is very hard. Euler's theorem and Euler's phi function play important roles in RSA [31, p.174].

### 3.1 Encryption and decryption

RSA encryption and decryption is done in the integer ring  $\mathbb{Z}_n$  and modular computations play a central role. RSA encrypts plaintext  $x$ , where we consider the bit string representing  $x$  to be an element in  $\mathbb{Z}_n = \{0, 1, \dots, n - 1\}$ . As a consequence the binary value of the plaintext  $x$  must be less than  $n$ . The same holds for the ciphertext. Encryption with the



public key and decryption with the private key are as shown below:

**RSA Encryption:** Given the public key  $(n, e) = k_{pub}$  and the plaintext  $x$ , the encryption function is:

$$y = e_{k_{pub}}(x) \equiv x^e \pmod{n} \quad (3.1)$$

where  $x, y \in \mathbb{Z}_n$ .

**RSA Decryption:** Given the private key  $d = k_{pr}$  and the ciphertext  $y$ , the decryption function is:

$$x = d_{k_{pr}}(y) \equiv y^d \pmod{n} \quad (3.2)$$

where  $x, y \in \mathbb{Z}_n$ .

In practice,  $x$ ,  $y$ ,  $n$  and  $d$  are very long numbers, usually 1024 bit long or more. The value  $e$  is sometimes referred to as *encryption exponent* or *public exponent*, and the private key  $d$  is sometimes called *decryption exponent* or *private exponent*. If Alice wants to send an encrypted message to Bob, Alice needs to have his public key  $(n, e)$ , and Bob decrypts with his private key  $d$  [31, p.175].

Even without knowing more details, we can already state a few requirements for the RSA cryptosystem [31, p.175]:

1. Since an attacker has access to the public key, it must be computationally infeasible to determine the private-key  $d$  given the public-key values  $e$  and  $n$ .
2. Since  $x$  is only unique up to the size of the modulus  $n$ , we cannot encrypt more than  $l$  bits with one RSA encryption, where  $l$  is the bit length of  $n$ .
3. It should be relatively easy to calculate  $x^e \pmod{n}$ , i.e., to encrypt, and  $y^d \pmod{n}$ , i.e., to decrypt. This means we need a method for fast exponentiation with very long numbers.
4. For a given  $n$ , there should be many private -key/public-key pairs, otherwise an attacker might be able to perform a brute-force attack.

## 3.2 Key generation and proof of correctness

A distinctive feature of all asymmetric schemes is that there is a set-up phase during which the public and private key are computed. Depending on the public-key scheme, key generation can be quite complex [31, p.175].

Here are the steps involved in computing the public and private key for an RSA cryptosystem.

### RSA Key Generation:

**Output:** public-key:  $k_{pub} = (n, e)$  and private-key:  $k_{pr} = (d)$

1. Choose two large primes  $p$  and  $q$ .
2. Compute  $n = p \cdot q$ .
3. Compute  $\Phi(n) = (p - 1)(q - 1)$ .
4. Select the public exponent  $e \in \{1, 2, \dots, \Phi(n) - 1\}$  such that

$$\gcd(e, \Phi(n)) = 1.$$

5. Compute the private key  $d$  such that

$$d \cdot e = 1 \pmod{\Phi(n)}$$

The condition that  $\gcd(e, \Phi(n)) = 1$  ensures that the inverse of  $e$  exists modulo  $\Phi(n)$ , so that there is always a private key  $d$  [31, p.175–176]. Two parts of the key generation are nontrivial: Step 1, in which the two large primes are chosen, as well as Steps 4 and 5 in which the public- and private-key are computed. The prime generation of Step 1 is quite involved and is addressed later. The computation of the keys  $d$  and  $e$  can be done at once using the extended Euclidean algorithm (EEA). In practice, one often starts by first selecting a public parameter  $e$  in the range  $0 < e < \Phi(n)$ . The value  $e$  must satisfy the condition  $\gcd(e, \Phi(n)) = 1$ . We apply the EEA with the input parameters  $n$  and  $e$  and obtain the relationship:

$$\gcd(\Phi(n), e) = s \cdot \Phi(n) + t \cdot e$$

If  $\gcd(e, \Phi(n)) = 1$ , we know that  $e$  is a valid public-key. Moreover, we also know that the parameter  $t$  computed by the extended Euclidean algorithm is the inverse of  $e$ , and thus:

$$d = t \bmod \Phi(n)$$

In case that  $e$  and  $\Phi(n)$  are not relatively prime, we simply select a new value for  $e$  and repeat the process. Note that the coefficient  $s$  of the EEA is not required for RSA and does not need to be computed [31, p.176].

What is interesting is that the message  $x$  is first raised to the  $e$ th power during encryption and the result  $y$  is raised to the  $d$ th power in the decryption, and the result of this is again equal to the message  $x$ . Expressed as an equation, this process is:

$$d_{k_{pr}}(y) \equiv d_{k_{pr}}(e_{k_{pub}}(x) \equiv (x^e)^d \equiv x^{de} \equiv x \bmod n. \quad (3.3)$$

This is the essence of RSA. I will now provide a proof why the RSA scheme works [31, p.178].

*Proof.* We need to show that the decryption is the inverse function of encryption,  $d_{k_{pr}}(e_{k_{pub}}(x)) = x$ . We start with the construction rule for the public- and private-key:  $d \cdot e \equiv 1 \bmod \Phi(n)$ . By definition of the modulo operator, this equivalent to:

$$d \cdot e = 1 + t \cdot \Phi(n),$$

where  $t$  is some integer. Inserting this expression in Eq.(3.3):

$$d_{k_{pr}}(y) \equiv x^{de} \equiv x^{1+t \cdot \Phi(n)} \equiv x^{t \cdot \Phi(n)} \cdot x^1 \equiv (x^{\Phi(n)})^t \cdot x \bmod n. \quad (3.4)$$

This means we have to prove that  $x \equiv (x^{\Phi(n)})^t \cdot x \bmod n$ . We now use Euler's Theorem, which states that if  $\gcd(x, n) = 1$  then  $1 \equiv x^{\Phi(n)} \bmod n$ . A minor generalization immediately follows:

$$1 \equiv 1^t \equiv (x^{\Phi(n)})^t \bmod n, \quad (3.5)$$

where  $t$  is any integer. For the proof, two cases are distinguished:

**First case:**  $\gcd(x, n) = 1$

Euler's Theorem holds here and we can insert Eq.(3.5) into (3.4):

$$d_{k_{pr}}(y) \equiv (x^{\Phi(n)})^t \cdot x \equiv 1 \cdot x \equiv x \pmod{n}. \quad q.e.d.$$

This part of the proof establishes that decryption is in fact the inverse function of encryption for plaintext values  $x$  which are relatively prime to the RSA modulus  $n$  [31, p.178]. I will now provide the proof for the second case:

**Second case:**  $\gcd(x, n) = \gcd(x, p \cdot q) \neq 1$

Since  $p$  and  $q$  are primes,  $x$  must have one of them as a factor:

$$x = r \cdot p \quad \text{or} \quad x = s \cdot q,$$

where  $r, s$  are integers such that  $r < q$  and  $s < p$ . Without loss of generality we assume  $x = r \cdot p$ , from which follows that  $\gcd(x, q) = 1$ . Euler's Theorem holds in the following form:

$$1 \equiv 1^t \equiv (x^{\Phi(q)})^t \pmod{q},$$

where  $t$  is any positive integer. We now look at the term  $(x^{\Phi(n)})^t$  again:

$$(x^{\Phi(n)})^t \equiv (x^{(q-1)(p-1)})^t \equiv ((x^{\Phi(q)})^t)^{p-1} \equiv 1^{(p-1)} \equiv 1 \pmod{q}.$$

Using the definition of the modulo operator, this is equivalent to:

$$(x^{\Phi(n)})^t = 1 + u \cdot q,$$

where  $u$  is some integer. We multiply this equation by  $x$ :

$$\begin{aligned} x \cdot (x^{\Phi(n)})^t &= x + x \cdot u \cdot q \\ &= x + (r \cdot p) \cdot u \cdot q \\ &= x + r \cdot u \cdot (p \cdot q) \\ &= x + r \cdot u \cdot n \\ x \cdot (x^{\Phi(n)})^t &\equiv x \pmod{n}. \end{aligned} \tag{3.6}$$

Inserting Eq.(3.6) into (3.4) yields the desired result [31, p.178–179]:

$$d_{k_{pr}} = (x^{\Phi(n)})^t \cdot x \equiv x \pmod{n}.$$

### 3.3 Finding large primes

There is one important part of the RSA crypto scheme that has not been discussed yet, and that is finding large primes  $p$  and  $q$  in Step 1 of the key generation. Since their product is the RSA modulus  $n = p \cdot q$ , the two primes should have about half the bit length of  $n$ . For instance, if we want to set up RSA with a modulus of length  $\lceil \log_2 n \rceil = 1024$ ,  $p$  and  $q$  should have a bit length of 512 bit. The general approach is to generate integers at random which are then checked for primality, as depicted in Fig.4.1, where RNG is the random number generator. The RNG should be non predictable because if an attacker can compute or guess one of the two primes, The RSA algorithm will be compromised [31, p.187].

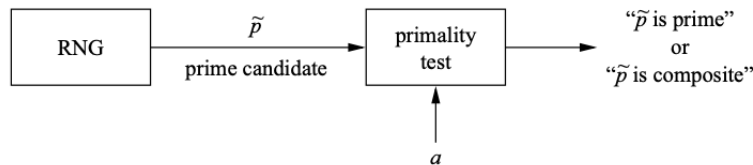


Figure 3.1: Principal approach to generating primes for RSA [31, p.187]

In order to make this approach work, we have to answer to questions [31, p.187]:

1. How many random integers do we have to test before we have a prime? (If the chance of a prime is too small, it might take too long).
2. How fast can we check whether a random number is a prime? (if the test is too slow, then the method might be impractical).

### 3.4 Attacks on RSA

There has been many attacks proposed on RSA since it was invented in 1977. None of them were serious, and moreover, they typically exploit weaknesses in the way RSA is implemented or used rather than the RSA algorithm itself. There are three general attack categories against RSA [31, p.194].

1. Protocol attacks
2. Mathematical attacks
3. Side-channel attacks

I will now give a short explanation of the three attack categories. Forward the focus for my thesis will be on mathematical attacks and factoring of large numbers, so the explanation on mathematical attacks will be more in depth than the others.

### 3.4.1 Protocol attacks

Protocol attacks exploits the weaknesses in how RSA is being used. There has been numerous of the over the years and the more known attacks exploit the malleability of RSA. Many of them can be prevented by using padding. Modern security standards describe exactly how RSA should be used, and if one follows the guidelines, protocol attacks should not be a threat [31, p.194].

### 3.4.2 Mathematical attacks

The best mathematical cryptanalytical method known is factoring the modulus. An attacker, Eve, knows the modulus  $n$ , the public key  $e$  and the ciphertext  $y$ . Her goal is to compute the private key  $d$  which has the property the  $e \cdot d \equiv \text{mod} \Phi(n)$ . It seems that she could simply apply the extended Euclidean algorithm and compute  $d$ . However, she does not know the value of  $\Phi(n)$ . At this point factoring comes in: the best way to obtain this value is to decompose  $n$  into its primes  $p$  and  $q$ . If Eve can do this, the attack succeeds in three steps [31, p.194]:

$$\begin{aligned}\Phi(n) &= (p - 1)(q - 1) \\ d^{-1} &\equiv e \text{ mod } \Phi(n) \\ x &\equiv y^d \text{ mod } n.\end{aligned}$$

In order to prevent this attack, the modulus must be sufficiently large. This is the reason why moduli of 1024 or more bit are needed for a RSA. The proposal of the RSA scheme in 1977 sparked much interest in the old problem of integer factorization. Major progress has been done in the field of integer factorization because of the creation of RSA. These advances have been possible mainly due to improvements in factoring algorithms, an to a lesser extent due to improved computer technology. Even though factoring has become easier than the RSA designers had assumed in 1977, factoring RSA moduli beyond a certain size is still out of reach. Which exact length the RSA modulus should have is the topic of much discussion.

Many RSA applications used a bit length of 1024 bits as default [31, p.194–195].

Through the years since RSA was invented in 1977, there has been a lot of progress in factoring numbers [31, p.194–195]. As late as February 2020, the factoring of a number of the longest bit length yet was done by researchers lead by Paul Zimmermann at Inria, Nancy. The length of the number was 829 bits [35]. As you can see, 829 bits is not that much less than 1024 bits. It has been recommended to change the RSA parameters in the range of 2048-4096 bits for long-term security.

### **3.4.3 Side-channel attacks**

A third and quite different family of attacks are side-channel attacks. They exploit information about the private key which is leaked through physical channels such as the power consumption or the timing behaviour. In order to observe such channels, an attacker must typically have direct access to the RSA implementation, e.g., in a cellphone or a smart card[31, 195]. Side-channel attacks are a large and interesting field of research in modern cryptography, but I will not further explain how the attacks are performed on RSA.

# Chapter 4

## Quantum mechanics

Before we can dive in to the quantum computation in the next chapter, we need an introduction to quantum mechanics and I will now provide the basics.

### 4.1 The postulates of quantum mechanics

First, I want to start off with the postulates of quantum mechanics. The postulates are the rule set of the quantum realm and important to keep in mind. There are in total 6 postulates and they are listed below:

- **Postulate 1:** The state of a quantum mechanical system is completely specified by a function  $\Psi(r, t)$  that depends on the coordinates of the particle(s) and on time. This function, called the wave function or state function has the important property that  $\Psi^*(r, t)\Psi(r, t)d\tau$  is the probability that the particle lies in the volume element  $d\tau$  located at  $r$  at time  $t$  [33].
- **Postulate 2:** To every observable in classical mechanics there corresponds a linear, Hermitian operator in quantum mechanics [33].
- **Postulate 3:** In any measurement of the observable associated with operator  $\hat{A}$ , the only values that will ever be observed are the eigenvalues  $a$ , which satisfy the eigenvalue equation [33]

$$\hat{A}\Psi = a\Psi. \tag{4.1}$$

- **Postulate 4:** Is a system is in a state described by a normalized wave function  $\Psi$ ,



then the average value of the observable corresponding to  $\hat{A}$  is given by [33]

$$\langle A \rangle = \int_{-\infty}^{\infty} \Psi^* \hat{A} \Psi d\tau. \quad (4.2)$$

- **Postulate 5:** The wave function or state function of a system evolves in time according to the time-dependent Schrödinger equation [33]

$$\hat{H}\Psi(r, t) = i\hbar \frac{\partial \Psi}{\partial t}. \quad (4.3)$$

- **Postulate 6:** The total wave function must be antisymmetric with respect to the inheritance of all coordinates of one fermion with those of another. Electronic spin must be included in this set of coordinates [33].

## 4.2 Linear algebra

Linear algebra is the study of vector spaces and of linear operations on those vector spaces. To understand the basics of quantum mechanics, we need to have a solid grasp on linear algebra. In this section I will introduce the standard notations which are used for these concepts in the study of quantum mechanics in table [] [29, p.61].

The basic objects of linear algebra are *vector spaces*. The vector space of most interest to us is  $\mathbf{C}^n$ , the space of all  $n$ -tuples of complex numbers,  $(z_1, \dots, z_n)$ . The elements of a vector space is called *vectors*, and I will sometimes use the column matrix notation

$$\begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} \quad (4.4)$$

to indicate a vector[29, p.61]. There is an *addition* operation defined which takes pairs of vector to other vectors. In  $\mathbf{C}^n$  the addition operation for vectors is defined by

$$\begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} + \begin{bmatrix} z'_1 \\ \vdots \\ z'_n \end{bmatrix} \equiv \begin{bmatrix} z_1 + z'_1 \\ \vdots \\ z_n + z'_n \end{bmatrix}, \quad (4.5)$$

where the addition operations on the right are just ordinary additions for complex numbers[29, p.61]. Furthermore, in a vector space there is a *multiplication by a scalar* operation.

In  $\mathbf{C}^n$  this operation is defined by

$$z \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} \equiv \begin{bmatrix} zz_1 \\ \vdots \\ zz_n \end{bmatrix}, \quad (4.6)$$

where  $z$  is a *scalar*, a complex number, and the multiplications on the right are ordinary multiplication of complex numbers, sometimes referred to as *c-numbers*[29, p.61–62].

The standard quantum mechanical notation for a vector in a vector space is:

$$|\psi\rangle. \quad (4.7)$$

$\psi$  is a label for the vector, though any label is valid it is preferred to use simple labels like  $\psi$  and  $\varphi$ . The  $|\cdot\rangle$  notation is used to indicate that the object is a vector [29, p.62].

A vector space also contains the *zero-vector*, which is denoted by  $0$ . It satisfies the property that for any vector  $|v\rangle$ ,  $|v\rangle + 0 = |v\rangle$ .

Notation	Description
$z^*$	Complex conjugate of the complex number $z$ . $(1 + i)^* = 1 - i$
$ \psi\rangle$	Vector. Also known as a <i>ket</i> .
$\langle\psi $	Vector dual to $ \psi\rangle$ . Also known as a <i>bra</i> .
$\langle\varphi \psi\rangle$	Inner product between vectors $ \varphi\rangle$ and $ \psi\rangle$ .
$ \varphi\rangle \otimes  \psi\rangle$	Tensor product of $ \varphi\rangle$ and $ \psi\rangle$ .
$ \varphi\rangle \psi\rangle$	Abbreviated notation for tensor product of $ \varphi\rangle$ and $ \psi\rangle$ .
$A^*$	Complex conjugate of the $A$ matrix.
$A^T$	Transpose of the $A$ matrix.
$A^\dagger$	Hermitian conjugate or adjoint of the $A$ matrix, $A^\dagger = (A^T)^*$ .
	$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^\dagger = \begin{bmatrix} a^* & c^* \\ b^* & d^* \end{bmatrix}$
$\langle\varphi A \psi\rangle$	Inner product between $ \varphi\rangle$ and $A \psi\rangle$ . Equivalently, inner product between $A^\dagger \varphi\rangle$ and $ \psi\rangle$ .

Table 4.1: Summary of some standard quantum mechanical notation for notions from linear algebra [29, p.62].

### 4.2.1 Bases and linear independence

A *spanning set* for a vector space is a set of vectors  $|v_1\rangle, \dots, |v_n\rangle$  such that any vector  $|v_n\rangle$  in the vector space can be written as a linear combination  $|v_n\rangle = \sum_i a_i |v_i\rangle$  of vectors in that set. For example, a spanning set for the vector space  $\mathbf{C}^n$  is the set

$$|v_1\rangle \equiv \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \quad |v_2\rangle \equiv \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad (4.8)$$

since any vector

$$|v\rangle = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \quad (4.9)$$

in  $\mathbf{C}^n$  can be written as a linear combination  $|v\rangle = a_1|v_1\rangle + a_2|v_2\rangle$  of the vectors  $|v_1\rangle$  and  $|v_2\rangle$ . We say that the vectors  $|v_1\rangle$  and  $|v_2\rangle$  *span* the vector space  $\mathbf{C}^n$  [29, p.62–63].

A set of non-zero vectors  $|v_1\rangle, \dots, |v_n\rangle$  are *linearly independent* if there exists a set of complex numbers  $a_1, \dots, a_n$  with  $a_i \neq 0$  for at least one value of  $i$ , such that

$$a_1|v_1\rangle + a_2|v_2\rangle + \dots + a_n|v_n\rangle = 0. \quad (4.10)$$

A set of vectors is *linearly independent* if it is not linearly dependent. It can be shown that any two sets of linearly independent vectors which span a vector space  $V$  contain the same number of elements. We call such a set *basis* for  $V$ . Furthermore, such a basis set always exists. The number of elements in the basis is defined to be the *dimension* of  $V$  [29, p.63].

### 4.2.2 Linear operators and matrices

A *linear operator* between vector spaces  $V$  and  $W$  is defined to be any function  $A : V \rightarrow W$  which is linear in its inputs,

$$A\left(\sum_i a_i |v_i\rangle\right) = \sum_i a_i A(|v_i\rangle). \quad (4.11)$$

Usually  $A|v\rangle$  is written to denote  $A(|v\rangle)$ . When a linear operator  $A$  is defined *on* a vector space, it means that  $A$  is a linear operator from  $V$  to  $V$ . An important linear operator on any vector space  $V$  is the *identity operator*,  $I_V$ , defined by the equation  $I_V|v\rangle \equiv |v\rangle$  for all vectors  $|v\rangle$ . Where no chance of confusion arises, one can drop the subscript  $V$  and just write  $I$  to denote the identity operator. Another important linear operator is the *zero operator*,

which is denoted by 0. The zero operator maps all vectors to a zero vector,  $0|v\rangle \equiv 0$ . It is clear from equation (5.8) that once the action of a linear operator  $A$  on a basis is specified, the action of  $A$  is completely determined on all inputs [29, p.63–64].

The most convenient way to understand linear operators is in terms of their *matrix representations*. In fact, the linear operator and matrix viewpoints turn out to be completely equivalent. The matrix viewpoint may be more familiar, however. To see the connection, it helps to first understand that an  $m$  by  $n$  complex matrix  $A$  with entries  $A_{ij}$  is in fact a linear operator sending vectors in the vector space  $\mathbf{C}^n$  to the vector space  $\mathbf{C}^m$ , under matrix multiplication of the matrix  $A$  by a vector in  $\mathbf{C}^n$ . More precisely, the claim that the matrix  $A$  is a linear operator just means that

$$A\left(\sum_i a_i|v_i\rangle\right) = \sum_i a_i A|v_i\rangle \quad (4.12)$$

is true as an equation where the operation is matrix multiplication of  $A$  by column vectors [29, p.64].

We've seen that matrices can be regarded as linear operators, but can linear operators be given a matrix representation? They can! This equivalence between the two viewpoints justifies our interchanging terms from matrix theory and operator theory. Suppose  $A : V \rightarrow W$  is a linear operator between vector spaces  $V$  and  $W$ . Suppose  $|v_1, \dots, |v_m\rangle$  is a basis for  $V$  and  $|w_1, \dots, |w_n\rangle$  is a basis for  $W$ . Then for each  $j$  in the range  $1, \dots, m$ , there exists complex numbers  $A_{1j}$  through  $A_{nj}$  such that

$$A|v_j\rangle = \sum_i A_{ij}|w_i\rangle. \quad (4.13)$$

The matrix whose entries are the values  $A_{ij}$  is said to form a *matrix representation* of the operator  $A$ . This matrix representation of  $A$  is completely equivalent to the operator  $A$ . Note that to make the connection between matrices and linear operators we must specify a set of input and output basis states for the input and output vector spaces of the linear operator [29, p.64].

### 4.2.3 The Pauli matrices

Four extremely useful matrices that are often used are the *Pauli matrices*. They are all 2 by 2 matrices and they go by a variety of notations. The matrices, and their variety of notations, are listed underneath [29, p.65].

$$\begin{aligned}\sigma_0 &\equiv I \equiv \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \sigma_1 &\equiv \sigma_x \equiv X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\ \sigma_2 &\equiv \sigma_y \equiv Y \equiv \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \\ \sigma_3 &\equiv \sigma_z \equiv Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\end{aligned}$$

#### 4.2.4 Inner products

An *inner product* is a function which takes as input two vectors  $|v\rangle$  and  $|w\rangle$  from a vector space and produces a complex number as output. For the time being, the notation for writing the inner product of  $|v\rangle$  and  $|w\rangle$  as  $(|v\rangle, |w\rangle)$ . The standard quantum mechanical notation for the inner product  $(|v\rangle, |w\rangle)$  is  $\langle v|w\rangle$ , where  $|v\rangle$  and  $|w\rangle$  are vectors in the inner product space, and the notation  $\langle v|$  is used for the *dual vector* to the vector  $|v\rangle$ ; the dual is a linear operator from the inner product space  $V$  to the complex numbers  $\mathbf{C}$ , defined by  $\langle v|(|w\rangle) \equiv \langle v|w\rangle \equiv (|v\rangle, |w\rangle)$  [29, p.65].

A function  $(\cdot, \cdot)$  from  $V \times V$  to  $\mathbf{C}$  is an inner product if it satisfies the requirements that:

1.  $(\cdot, \cdot)$  is linear in the second argument,

$$(|v\rangle, \sum_i \lambda_i |w_i\rangle) = \sum_i \lambda_i (|v\rangle, |w_i\rangle). \quad (4.14)$$

2.  $(|v\rangle, |w\rangle) = (|v\rangle, |w\rangle)^*$ .
3.  $(|v\rangle, |w\rangle) \geq 0$  with equality if and only if  $|v\rangle = 0$ .

We call a vector space equipped with an inner product an *inner product space*. Discussions of quantum mechanics often refers to *Hilbert space*. In the finite dimensional complex vector spaces that come up in quantum computation and quantum information, a Hilbert space is *exactly the same thing* as an inner product space. In infinite dimensions Hilbert spaces satisfy additional technical restrictions [29, p.65–66].

### 4.2.5 Eigenvectors and eigenvalues

An *eigenvector* of a linear operator  $A$  on a vector space is a non-zero vector  $|v\rangle$  such that  $A|v\rangle = v|v\rangle$ , where  $v$  is a complex number known as the *eigenvalue* of  $A$  corresponding to  $|v\rangle$ . It will often be convenient to use the notation  $v$  both as a label for the eigenvector and to represent the eigenvalue. The *characteristic function* is used for finding eigenvalues and eigenvectors and is defined to be  $c(\lambda) \equiv \det|A - \lambda I|$ , where  $\det$  is the *determinant* function for matrices. The solutions of the *characteristic equation*  $c(\lambda) = 0$  are the eigenvalues of the operator  $A$ . By the fundamental theorem of algebra, every polynomial has at least one complex root, so every operator  $A$  has at least one eigenvalue, and a corresponding eigenvector. The *eigenspace* corresponding to an eigenvalue  $v$  is the set of vectors which have eigenvalue  $v$ . It is a vector subspace of the vector space on which  $A$  acts [29, p.68–69].

A *diagonal representation* for an operator  $A$  on a vector space  $V$  is a representation  $A = \sum_i \lambda_i |i\rangle\langle i|$ , where the vectors  $|i\rangle$  form an orthogonal set of eigenvectors for  $A$ , with corresponding eigenvalues  $\lambda_i$ . An operator is said to be *diagonalizable* if it has a diagonal representation. Diagonal representations may also be called *orthonormal decompositions* [29, p.69].

### 4.2.6 Tensor products

The *tensor product* is a way of putting vector spaces together to form larger vector spaces. This construction is crucial to understanding the quantum mechanics of multiparticle systems. By definition the tensor product satisfies the following basic properties [29, p.72–73]:

1. For an arbitrary scalar  $x$  and elements  $|v\rangle$  of  $V$  and  $|w\rangle$  of  $W$ ,

$$z(|v\rangle \otimes |w\rangle) = (z|v\rangle) \otimes |w\rangle = |v\rangle \otimes (z|w\rangle). \quad (4.15)$$

2. For arbitrary  $|v_1\rangle$  and  $|v_2\rangle$  in  $V$  and  $|w\rangle$  in  $W$ ,

$$(|v_1\rangle + |v_2\rangle) \otimes |w\rangle = |v_1\rangle \otimes |w\rangle + |v_2\rangle \otimes |w\rangle. \quad (4.16)$$

3. For arbitrary  $|v\rangle$  in  $V$  and  $|w_1\rangle$   $|w_2\rangle$  in  $W$ ,

$$|v\rangle \otimes (|w_1\rangle + |w_2\rangle) = |v\rangle \otimes |w_1\rangle + |v\rangle \otimes |w_2\rangle. \quad (4.17)$$

A linear operator  $A \otimes B$  on  $V \otimes W$  can be defined by the equation

$$(A \otimes B)(|v\rangle \otimes |w\rangle) \equiv A|v\rangle \otimes B|w\rangle. \quad (4.18)$$

The definition of  $A \otimes B$  is then extended to all elements of  $V \otimes W$  in the natural way to ensure linearity of  $A \otimes B$ . Finally I want to mention the useful notation  $|\psi\rangle^{\otimes k}$ , which means  $|\psi\rangle$  tensored with itself  $k$  times [29, p.73–74].

### 4.3 Entanglement

The deep ways that quantum information differs from classical information involve the properties, implications, and uses of *quantum entanglement*. Entangled states are interesting because they exhibit correlations that have no classical analog. For example, the *maximally entangled* state of two qubits (or EPR *pair*) is defined as [32, p.4]:

$$|\phi^+\rangle_{AB} = \frac{1}{\sqrt{2}}(|00\rangle_{AB} + |11\rangle_{AB}). \quad (4.19)$$

”Maximally entangled” means that when we trace over qubit  $B$  to find the density operator  $\rho_A$  of qubit  $A$ , we obtain a multiple of the identity operator:

$$\rho_A = \text{tr}_B(|\phi^+\rangle\langle\phi^+|) = \frac{1}{2}\mathbf{I}_A, \quad (4.20)$$

(and similarly  $\rho_B = \frac{1}{2}\mathbf{I}_B$ ). This means that if we measure spin  $A$  along *any* axis, the result is completely random - we find spin up with probability 1/2 and spin down with probability 1/2. Therefore, if we perform local measurement of  $A$  or  $B$ , we acquire no information about the preparation of the state, instead we merely generate a random bit. This situation contrasts sharply with case of a single qubit in a pure state; there we can store a bit by preparing, say, either  $|\uparrow_{\hat{n}}\rangle$  or  $|\downarrow_{\hat{n}}\rangle$ , and we can recover that bit reliably by measuring along the  $\hat{n}$  - *axis*. With two qubits, we ought to be able to store two bits, but in the state  $|\phi^+\rangle_{AB}$  this information is *hidden*; at least, we can’t acquire it by measuring  $A$  or  $B$  [32, p.4–5].

In fact,  $|\phi^+\rangle_{AB}$  is one member of a basis of four mutually orthogonal states for the two qubits, all of which are maximally entangled - the basis:

$$|\phi^\pm\rangle = \frac{1}{\sqrt{2}}(|00\rangle \pm |11\rangle), \quad (4.21)$$

$$|\psi^\pm\rangle = \frac{1}{\sqrt{2}}(|01\rangle \pm |10\rangle). \quad (4.22)$$

Imagine that Alice and Bob play a game with Charlie. Charlie prepares one of these four states, thus encoding two bits in the state of the two-qubit system. One bit is the *parity* bit ( $|\phi\rangle$  or  $|\psi\rangle$ ): are the two spins aligned or antialigned? The other is the *phase* bit (+ or -): what superposition was chosen of the two states of like parity. Then Charlie sends qubit  $A$  to Alice and qubit  $B$  to Bob. To win the game, Alice (or Bob) has to identify which of the four states Charlie prepared [32, p.5].

Of course, if Alice and Bob bring their qubits together, they can identify the state by performing an orthogonal measurement that projects onto the  $\{|\phi^+\rangle, |\phi^-\rangle, |\psi^+\rangle, |\psi^-\rangle\}$  basis. But suppose that Alice and Bob are in different cities, and that they are unable to communicate at all. Acting locally, neither Alice nor Bob can collect any information about the identity of the state [32, p.5].

What they *can* do locally is *manipulate* this information. Alice may apply  $\sigma_3$  to qubit  $A$ , flipping the relative phase of  $|0\rangle_A$  and  $|1\rangle_A$ . This action flips the phase bit stored in the entangled state:

$$|\phi^+\rangle \leftrightarrow |\phi^-\rangle, \quad (4.23)$$

$$|\psi^+\rangle \leftrightarrow |\psi^-\rangle. \quad (4.24)$$

On the other hand, she can apply  $\sigma_1$ , which flips her spin ( $|0\rangle_A \leftrightarrow |1\rangle_A$ ), and also flips the parity bit of the entangled state:

$$|\phi^+\rangle \leftrightarrow |\psi^+\rangle, \quad (4.25)$$

$$|\phi^-\rangle \leftrightarrow -|\psi^-\rangle. \quad (4.26)$$

Bob can manipulate the entangled state similarly. In fact, either Alice or Bob can perform a local unitary transformation that changes one maximally entangled state to any other maximally entangled state. What their local unitary transformations *cannot* do is alter



$\rho_A = \rho_B = \frac{1}{2}\mathbf{I}$  - the information they are manipulating is information that neither one can read [32, p.5–6].

But suppose that Alice and Bob are able to exchange (classical) messages about their measurement outcomes; together, then, they can learn about how their measurements are correlated. The entangled basis states are conveniently characterized as the simultaneous eigenstates of two commuting observables:

$$\sigma_1^{(A)} \otimes \sigma_1^{(B)}, \quad (4.27)$$

$$\sigma_3^{(A)} \otimes \sigma_3^{(B)}; \quad (4.28)$$

the eigenvalue of  $\sigma_3^{(A)} \otimes \sigma_3^{(B)}$  is the parity bit, and the eigenvalue of  $\sigma_1^{(A)} \otimes \sigma_1^{(B)}$ , is the phase bit. Since these operators commute, they can in principle be measured simultaneously. But they cannot be measured simultaneously if Alice and Bob perform localized measurements. Alice and Bob could both choose to measure their spins along the  $z$ -axis, preparing a simultaneous eigenstate of  $\sigma_3^{(A)}$  and  $\sigma_3^{(B)}$ . Since  $\sigma_3^{(A)}$  and  $\sigma_3^{(B)}$  both commute with the parity operator  $\sigma_3^{(A)} \otimes \sigma_3^{(B)}$ , their orthogonal measurements do not disturb the parity bit, and they can combine their results to infer the parity bit. But  $\sigma_3^{(A)}$  and  $\sigma_3^{(B)}$  do *not* commute with phase operator  $\sigma_1^{(A)} \otimes \sigma_1^{(B)}$ , so their measurement disturbs the phase bit. On the other hand, they could both choose to measure their spins along the  $x$ -axis; then they would learn the phase bit at the cost of disturbing the parity bit. But they can't have it both ways. To have hope of acquiring the parity bit without disturbing the phase but, they would need to learn about the product  $\sigma_3^{(A)} \otimes \sigma_3^{(B)}$  without finding out anything about  $\sigma_3^{(A)}$  and  $\sigma_3^{(B)}$  separately. That cannot be done locally [32, p.6].

Now let us bring Alice and Bob together, so that they can operate on their qubits jointly. How might they acquire both the parity bit and the phase bit of their pair? By applying an appropriate unitary transformation, they can rotate the entangled basis  $\{|\phi^\pm\rangle, |\psi^\pm\rangle\}$  to the unentangled basis  $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ . Then they can measure qubits  $A$  and  $B$  separately to acquire the bits they seek. But how is this transformation constructed [32, p.6]?

## 4.4 Conditions for quantum computation

Now I want to discuss the four basic requirements for quantum computation before we go on to the next chapter on quantum computation. These requirements are the abilities to [29, 279]:

1. Robustly represent quantum information
2. Perform a universal family of unitary transformations
3. Prepare a fiducial initial state
4. Measure output result

### 4.4.1 Representation of quantum information

Quantum information is based on transformation of quantum states. Quantum bits are two-level quantum systems, and as the simplest elementary building blocks for a quantum computer, they provide a convenient labeling for pairs of states and their physical realizations [29, 279].

For the purpose of computation, the crucial realization is that the set of accessible states should be *finite*. The position of  $x$  of a particle along a one-dimensional line is not generally a good set of states for computation, even though the particle may be in a quantum state  $|x\rangle$ , or even some superposition  $\sum_x c_x |x\rangle$ . This is because  $x$  has a continuous range of possibilities, and the Hilbert space has infinite size, so that in the absence of noise the information capacity is infinite. It is in fact generally desirable to have some aspect of symmetry dictate the finiteness of the state space, in order to minimize decoherence. If the choice of representation is poor, the decoherence will result [29, 279].

### 4.4.2 Performance of unitary transformations

Closed quantum systems evolve unitarily as determined by their Hamiltonians, but to perform quantum computation one must be able to control the Hamiltonians to effect an *arbitrary* selection from a universal family of unitary transformations. In fact, any unitary transform can be composed from single spin operations and controlled-NOT gates, and thus realization of those two kinds of quantum logic gates are natural goals for experimental quantum computation. However, implicitly required also is the ability to address individual

qubits, and to apply these gates to select qubits or pairs of qubits. This is not simple to accomplish in many physical systems [29, 281].

Unrecorded imperfections in unitary transforms can lead to decoherence. Similarly, the cumulative effect of systematic errors is decoherence, when the information needed to be able to reverse them is lost. Furthermore, the control parameters in the Hamiltonian are only approximately classical controls: in reality, the controlling system is just another quantum system, and the true Hamiltonian should include the back-action of the control system upon the quantum computer [29, 281].

Two important figures of merit for unitary transforms are the minimum achievable fidelity  $\mathcal{F}$  and the maximal time  $t_{op}$  required to perform elementary operations such as single spin rotations or a controlled-NOT gate [29, 281].

### 4.4.3 Preparations of fiducial initial states

One of the most important requirements for being able to perform a useful computation, even classically, is to be able to prepare the desired output. If one has a box which can perform perfect computations, what use is it if numbers cannot be input? With classical machines, establishing a definite input state is rarely a difficulty - one merely sets some switches in the desired configuration and that defines the input state. However, with quantum systems this can be very difficult, depending on the realization of qubits. Note that it is only necessary to be able to (repeatedly) produce one specific quantum state with high fidelity, since a unitary transform can turn it into any other desired input state [29, 281].

Input state preparation is a significant problem for most physical systems. Moreover, for physical systems in which ensembles of quantum computers are involved, extra concerns arise. Two figures of merit are relevant to input state preparation: the minimum fidelity with which the initial state can be prepared in a given state  $\rho_{in}$ , and the entropy of  $\rho_{in}$ . The entropy is important because, for example, it is very easy to prepare the state  $\rho_{in} = I/2^n$  with high fidelity, but that is a useless state for quantum computation, since it is invariant under unitary transforms. Ideally, the input state is a pure state, with zero entropy. Generally, input states with non-zero entropy reduce the accessibility of the answer from the output result [29, 282].

#### 4.4.4 Measurement of output result

What measurement capability is required for quantum computation? For the purpose of this discussion, let's think of measurement as a process of coupling one or more qubits to a classical system such that after some interval of time, the state of the qubits is indicated by the state of the classical system. For example, a qubit state  $a|0\rangle + b|1\rangle$ , represented by the ground and excited states of a two-level atom, might be measured by pumping the excited state and looking for fluorescence. If an electrometer indicates that fluorescence had been detected by a photomultiplier tube, then the qubit would collapse into the  $|1\rangle$  state, and this would happen with the probability  $|b|^2$ . Otherwise, the electrometer would detect no charge, and the qubit would collapse into the  $|0\rangle$  state [29, 282].

An important characteristic of the measurement process for quantum computation is the wave function collapse which describes what happens when a projective measurement is performed. The output of a good quantum algorithm is a superposition state which gives a useful answer with high probability when measured. For example, one step in Shor's quantum factoring algorithm is to find an integer  $r$  from measurement result, which is an integer close to  $qc/r$ , where  $q$  is the dimension of a Hilbert space. The output state is actually in a nearly uniform superposition of all possible values of  $c$ , but a measurement collapses this into a single, random integer, thus allowing  $r$  to be determined with high probability [29, 282].

Many difficulties with measurement can be imagined. Furthermore, projective measurements are often difficult to implement. They require that the coupling between the quantum and classical systems be large, and switchable. Measurements should not occur when not desired; otherwise they can be a decoherence process. Surprisingly, however, strong measurements are not necessary; weak measurements which are performed continuously and never switched off, are usable for quantum computation. This is made possible by completing the computation in time short compared with the measurement coupling, and by using large ensembles of quantum computers. These ensembles together give an aggregate signal which is macroscopically observable and indicative of the quantum state. Use of an ensemble introduces additional problems. For example, in the factoring algorithm, if the measurement output is  $q\langle c\rangle/r$ , the algorithm would fail because  $\langle c\rangle$ , the average value of  $c$ , is not necessarily an integer (and thus the continued fraction expansion would not be possible). Fortunately, it is possible to modify quantum algorithms to work with ensemble average readouts [29, 282–283].

A good figure of merit for measurement capacity is the signal to noise ratio (SNR).

This accounts for measurement inefficiency as well as inherent signal strength available from coupling a measurement apparatus to the quantum system [29, p.283].

# Chapter 5

## Quantum computation

Changes made to a quantum state can be described using the language of *quantum computation*. The same way a classical computer is built from an electrical circuit containing wires and logic gates, a quantum computer is built from a *quantum circuit* containing wires and elementary *quantum gates* to pass around the quantum information and manipulate it [29, p.17].

### 5.1 Quantum bits

The *bit* is the fundamental concept of classical computation and classical information. Quantum computation and quantum information is built upon a similar concept, the *quantum bit*, qubit for short. Here, properties for single and multiple qubits will be introduced and compared with properties of classical bits before we move on further in this chapter [29, p.13].

#### 5.1.1 Single qubits

So what is a qubit then? Just like a classical bit has a *state* - either 0 or 1 - a qubit also has a state. Two possible states for a qubit are the states  $|0\rangle$  and  $|1\rangle$ , which as you might guess correspond to the states 0 and 1 for a classical bit. Notation like  $|\ \rangle$  is called the *Dirac notation*, and this will be seen a lot as it is the standard notation for states in quantum mechanics. The difference between bits and qubits is that a qubit can be in a state *other* than  $|0\rangle$  or  $|1\rangle$ . It is also possible to form *linear combinations* of states, called *superpositions*

[29, p.13]:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle. \tag{5.1}$$

The numbers  $\alpha$  and  $\beta$  are complex numbers, although for many purposes not much is lost by thinking of them as real numbers. In another way, the state of a qubit is a vector in a two-dimensional complex vector space. The special states  $|0\rangle$  and  $|1\rangle$  are known as *computational basis states*, and form an orthonormal basis for this vector space [29, p.13].

We can examine a bit to determine whether it is in the state 0 or 1. For example, computers do this all the time when they retrieve the contents of their memory. Rather remarkably, we cannot examine a qubit to determine its quantum state, that is, the values of  $\alpha$  and  $\beta$ . Instead, quantum mechanics tells us that we can only acquire much more restricted information about the quantum state. When we measure a qubit we get either the result 0, with the probability  $|\alpha|^2$ , or the result 1, with the probability  $|\beta|^2$ . Naturally,  $|\alpha|^2 + |\beta|^2 = 1$ , since the probabilities must sum to 1. Geometrically, we can interpret this as the condition that the qubit's state be normalized to length 1. Thus, in general a qubit's state is a unit vector in a two-dimensional complex vector space [29, p.13].

### 5.1.2 Multiple qubits

Suppose we have two qubits. If these were classical bits, then there would be four possible states, 00, 01, 10, and 11. Correspondingly a two qubit system has four *computational basis states* denoted  $|00\rangle$ ,  $|01\rangle$ ,  $|10\rangle$ ,  $|11\rangle$ . A pair of qubits can also exist in superpositions of these four states, so the quantum state of two qubits involves associating a complex coefficient - sometimes called an *amplitude* - with each computational basis state, such that the state vector describing the two qubits is [29, p.16]:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle. \tag{5.2}$$

Similar to the case for a single qubit, the measurement result  $x=(00,01,10,11)$  occurs with probability  $|a_x|^2$ , with the state of the qubits after the measurement being  $|x\rangle$ . The condition that probabilities sum to one is therefore expressed by the *normalization* condition that  $\sum_{x \in \{0,1\}^2} |a_x|^2 = 1$ , where the notation ' $\{0,1\}^2$ ' means 'the set of strings of length two with each letter being zero or one'. For a two qubit system, we could measure just a subset of the qubits, say the first qubit: measuring the first qubit alone gives 0 with the probability

$|\alpha_{00}|^2 + |\alpha_{01}|^2$ , leaving the post measurement state [29, p.16]:

$$|\psi'\rangle = \frac{\alpha_{00}|00\rangle + \alpha_{01}|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}} \quad (5.3)$$

Note how the post-measurement state is *re-normalized* by the factor  $\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}$  so that it still satisfies the normalization condition, just as we expect for a legitimate quantum state [29, p.16].

A important two qubit state is the *Bell state* or *EPR pair*,

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}}. \quad (5.4)$$

This innocuous-looking state is responsible for many surprises in quantum computation and quantum information. It is the key ingredient in quantum teleportation and superdense coding and prototype for many other interesting quantum states. The Bell state has the property that upon measuring the first qubit, one obtains two possible results: 0 with probability 1/2, leaving post-measurement state  $|\psi'\rangle = |00\rangle$ , and 1 with probability 1/2, leaving  $|\psi'\rangle = |11\rangle$ . As a result, a measurement of the second qubit always gives the same result as the measurement of the first qubit. That is, the measurement outcomes are *correlated* [29, p.17].

More generally, we may consider a system of  $n$  qubits. The computational basis states of this system are of the form  $|x_1x_1 \dots x_n\rangle$ , and so a quantum state of such a system is specified by  $2^n$  amplitudes. For  $n = 500$  this number is larger than the estimated number of atoms in the universe. Trying to store all these complex numbers would not be possible on any conceivable classical computer. Hilbert space is indeed a big place. In principle, however, nature manipulates such enormous quantities of data, even for systems containing only a few hundred atoms. It is as if nature were keeping  $2^{500}$  hidden pieces of scratch paper on the side, on which she performs her calculations as the system evolves [29, p.17].

## 5.2 Quantum gates and circuits

Classical computers are built of wires and logic gates. The wires are used to carry information around the circuit, while logic gates perform manipulations of the information, converting it from one form to another. And then together, the wires and logic gates form circuits.

Now I will introduce some notation that will be used going forward. Qubits are represented with horizontal lines, and the single-qubit unitary transformation  $U$  is denoted:



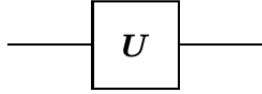


Figure 5.1: Illustration of unitary transformation [32, p.6].

A particular single-qubit unitary we will find useful is the *Hadamard transform*

$$\mathbf{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{\sqrt{2}}(\boldsymbol{\sigma}_1 + \boldsymbol{\sigma}_3), \quad (5.5)$$

which has the properties

$$\mathbf{H}^2 = \mathbf{I} \quad (5.6)$$

and

$$\mathbf{H}\boldsymbol{\sigma}_1\mathbf{H} = \boldsymbol{\sigma}_3, \quad (5.7)$$

$$\mathbf{H}\boldsymbol{\sigma}_3\mathbf{H} = \boldsymbol{\sigma}_1. \quad (5.8)$$

Also useful is the two-qubit transformation known as the reversible XOR or controlled-NOT transformation; it acts as

$$\mathbf{CNOT} : |a, b\rangle \rightarrow |a, a \oplus b\rangle, \quad (5.9)$$

on the basis states  $a, b = 0, 1$ , where  $a \oplus b$  denotes addition modulo 2. The **CNOT** is denoted:

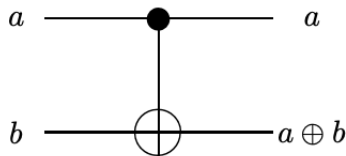


Figure 5.2: Illustration of the **CNOT** transformation [32, p.7].

Thus this transformation flips the second bit if the first is 1, and acts trivially if the first bit is 0; it has the property

$$(\mathbf{CNOT})^2 = \mathbf{I} \otimes \mathbf{I} \quad (5.10)$$

In addition to these gates, there is also some notation used in quantum circuits that should be introduced:

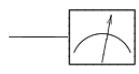


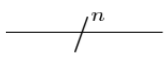
Measurement		Projection onto $ 0\rangle$ and $ 1\rangle$
Qubit		Wire carrying a single qubit(left to right)
Classical bit		Wire carrying a single classical bit(left to right)
$n$ qubits		Wire carrying $n$ qubits

Table 5.1: Additional notation for quantum circuits [29].

### 5.3 The quantum Fourier transform

In this section I will provide the development of the *quantum Fourier transform*, which is the key ingredient for quantum factoring and many other quantum algorithms. The quantum Fourier Transform is an efficient quantum algorithm for performing a Fourier transform of quantum mechanical amplitudes. It does *not* speed up the classical task of computing Fourier transforms of classical data. But one important task which it does enable is *phase estimation*, the approximation of the eigenvalues of a unitary operator under certain circumstances. This allows us to solve several other interesting problems, including the *order-finding problem* and the *factoring problem* [29, p.216].

One of the most useful ways of solving a problem in mathematics or computer science is to *transform* it into some other problem for which a solution is known. A great discovery of quantum computation has been that some such transformations can be computed much faster on a quantum computer than on a classical computer, a discovery which has enabled the construction of fast algorithms for quantum computers [29, p.217].

One such transformation is the *discrete Fourier transform*. In the usual mathematical notation, the discrete Fourier transform takes as input a vector of complex numbers,  $x_0, \dots, x_{N-1}$  where the length  $N$  of the vector is a fixed parameter. It outputs the transformed data, a vector of complex numbers  $y_0, \dots, y_{N-1}$ , defined by [29, p.217]:

$$y_k \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i j k / N}. \quad (5.11)$$

The *quantum Fourier transform* is exactly the same transformation, although the con-

ventional notation for the quantum Fourier transform is somewhat different. The quantum Fourier transform on an orthonormal basis  $|0\rangle, \dots, |N-1\rangle$  is defined to be a linear operator with the following action on the basis states [29, p.217],

$$|j\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i j k / N} |k\rangle. \quad (5.12)$$

Equivalently, the action on an arbitrary state may be written

$$\sum_{j=0}^{N-1} x_j |j\rangle \rightarrow \sum_{k=0}^{N-1} y_k |k\rangle, \quad (5.13)$$

where the amplitudes  $y_k$  are the discrete Fourier transform of the amplitudes  $x_j$ . it is not obvious from the definition, but this transformation is a unitary transformation, and thus can be implemented as the dynamics for a quantum computer. The unitary of the Fourier transform will be demonstrated by construction a manifestly unitary quantum circuit computing the Fourier transform [29, p.217].

In the following, we take  $N = 2^n$ , where  $n$  is some integer, and the basis  $|0\rangle, \dots, |2^n - 1\rangle$  is the computational basis for an  $n$  qubit quantum computer. it is helpful to write the state  $|j\rangle$  using the binary representation  $j = j_1 j_2 \dots j_n$ . More formally,  $j = j_1 2^{n-1} + j_2 2^{n-2} + \dots + j_n 2^0$ . It is also convenient to adopt the notation  $0.j_l j_{l+1} \dots j_m$  to represent the *binary fraction*  $j_l/2 + j_{l+1}/4 + \dots + j_m/2^{m-l+1}$  [29, p.217–218].

With a little algebra the quantum Fourier transform can be given the following useful *product representation*:

$$|j_1, \dots, j_n\rangle \rightarrow \frac{(|0\rangle + e^{2\pi i 0.j_n} |1\rangle) (|0\rangle + e^{2\pi i 0.j_{n-1} j_n} |1\rangle) \dots (|0\rangle + e^{2\pi i 0.j_1 j_2 \dots j_n} |1\rangle)}{2^{n/2}}. \quad (5.14)$$

This product representation is so useful that one may even wish to consider this to be the *definition* of the quantum Fourier transform. As explained shortly this representation allows us to construct an efficient quantum circuit computing the Fourier transform, a proof that the quantum Fourier transform is unitary, and provides insight into algorithms based upon the quantum Fourier transform [29, p.218].

The equivalence of the product representation (5.14) and the definition (5.12) follows from some elementary algebra [29, p.218]:

$$|j\rangle \rightarrow \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e^{2\pi i j k / 2^n} \quad (5.15)$$

$$= \frac{1}{2^{n/2}} \sum_{k_1=0}^1 \cdots \sum_{k_n=0}^1 e^{2\pi i j (\sum_{l=1}^n k_l 2^{-l})} |k_1 \dots k_n\rangle \quad (5.16)$$

$$= \frac{1}{2^{n/2}} \sum_{k_1=0}^1 \cdots \sum_{k_n=0}^1 \bigotimes_{l=1}^n e^{2\pi i j k_l 2^{-l}} |k_l\rangle \quad (5.17)$$

$$= \frac{1}{2^{n/2}} \bigotimes_{l=1}^n \left[ \sum_{k_l=0}^1 e^{2\pi i j k_l 2^{-l}} |k_l\rangle \right] \quad (5.18)$$

$$= \frac{1}{2^{n/2}} \bigotimes_{l=1}^n \left[ |0\rangle + e^{2\pi i j 2^{-l}} |1\rangle \right] \quad (5.19)$$

$$= \frac{(|0\rangle + e^{2\pi i 0 \cdot j_n} |1\rangle)(|0\rangle + e^{2\pi i 0 \cdot j_{n-1} j_n} |1\rangle) \cdots (|0\rangle + e^{2\pi i 0 \cdot j_1 j_2 \cdots j_n} |1\rangle)}{2^{n/2}}. \quad (5.20)$$

The product representation (5.14) makes it easy to derive an efficient circuit for the quantum Fourier transform. Such a circuit is shown in Fig. 5.3. The gate  $R_k$  denotes the unitary transformation

$$R_k \equiv \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i / 2^k} \end{bmatrix}. \quad (5.21)$$

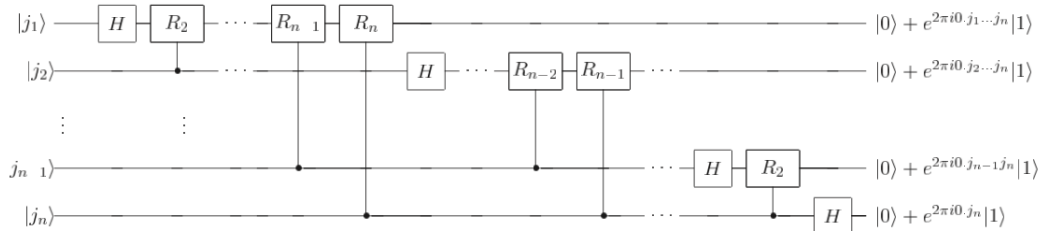


Figure 5.3: Efficient circuit for the quantum Fourier transform. This circuit is easily derived from the product representation (5.14) for the quantum Fourier transform. Not shown are the swap gates at the end of the circuit which reverse the order of the qubits, or normalization factors of  $1/\sqrt{2}$  in the output [29, p.219].

To see that the pictured circuit computes the quantum Fourier transform, consider what happens when the state  $|j_1 \dots j_n\rangle$  is input. Applying the Hadamard gate to the first bit

produces the state

$$\frac{1}{2^{1/2}}(|0\rangle + e^{2\pi i 0 \cdot j_1}|1\rangle)|j_2 \dots j_n\rangle, \quad (5.22)$$

since  $e^{2\pi i 0 \cdot j_1} = -1$  when  $j_1 = 1$ , and is  $+1$  otherwise. Applying the controlled- $R_2$  gate produces the state

$$\frac{1}{2^{1/2}}(|0\rangle + e^{2\pi i 0 \cdot j_1 j_2}|1\rangle)|j_2 \dots j_n\rangle. \quad (5.23)$$

We continue applying the controlled- $R_3, R_4$  through  $R_n$  gates, each of which adds an extra bit to the phase of the co-efficient of the first  $|1\rangle$ . At the end of this procedure we have the state

$$\frac{1}{2^{1/2}}(|0\rangle + e^{2\pi i 0 \cdot j_1 j_2 \dots j_n}|1\rangle)|j_2 \dots j_n\rangle. \quad (5.24)$$

Next, we perform a similar procedure on the second qubit. The Hadamard gate puts us in the state

$$\frac{1}{2^{2/2}}(|0\rangle + e^{2\pi i 0 \cdot j_1 j_2 \dots j_n}|1\rangle)(|0\rangle + e^{2\pi i 0 \cdot j_2}|1\rangle)|j_3 \dots j_n\rangle, \quad (5.25)$$

and the controlled- $R_2$  through  $R_{n-1}$  gates yield the state

$$\frac{1}{2^{2/2}}(|0\rangle + e^{2\pi i 0 \cdot j_1 j_2 \dots j_n}|1\rangle)(|0\rangle + e^{2\pi i 0 \cdot j_2 \dots j_n}|1\rangle)|j_3 \dots j_n\rangle. \quad (5.26)$$

We continue this way for each qubit, giving a final state

$$\frac{1}{2^{n/2}}(|0\rangle + e^{2\pi i 0 \cdot j_1 j_2 \dots j_n}|1\rangle)(|0\rangle + e^{2\pi i 0 \cdot j_2 \dots j_n}|1\rangle) \dots (|0\rangle + e^{2\pi i 0 \cdot j_n}|1\rangle). \quad (5.27)$$

Swap operations, omitted from Figure 6.1 for clarity, are then used to reverse the order of the qubits. After the swap operations, the state of the qubits is

$$\frac{1}{2^{n/2}}(|0\rangle + e^{2\pi i 0 \cdot j_n}|1\rangle)(|0\rangle + e^{2\pi i 0 \cdot j_{n-1} j_n}|1\rangle) \dots (|0\rangle + e^{2\pi i 0 \cdot j_1 j_2 \dots j_n}|1\rangle). \quad (5.28)$$

Comparing with Eq. 5.14 we can see that this is the desired output from the quantum Fourier transform. This construction also proves that the quantum Fourier transform is unitary, since each gate in the circuit is unitary [29, p.218–219].

But how many gates does this circuit use? We start by doing a Hadamard gate and  $n - 1$  conditional rotations on the first qubit, a total of  $n$  gates. This is followed by a Hadamard

gate and  $n - 2$  conditional rotations on the second qubit, for a total of  $n + (n - 1)$  gates. Continuing this way, we see that  $n + (n - 1) + \dots + 1 = n(n + 1)/2$  gates are needed, plus the gates in the swaps. At most  $n/2$  swaps are needed, and each swap can be accomplished using three controlled-NOT gates. Therefore, this circuit provides a  $\Theta(n^2)$  algorithm for performing the quantum Fourier transform[29, p.219–220].

In contrast, the best classical algorithms for computing the discrete Fourier transform on  $2^n$  elements are algorithms such as the *Fast Fourier Transform (FFT)*, which compute the discrete Fourier transform using  $\Theta(n^2)$  gates. That is, it requires exponentially more operations to compute the Fourier transform on a classical computer than it does to implement the quantum Fourier transform on a quantum computer [29, p.220].

At face value this sounds terrific, since the Fourier transform is a crucial step in so many real-world data processing applications. For example, in computer speech recognition, the first step in phoneme recognition is to Fourier transform the digitized sound. Can the quantum Fourier transform be used to speed up the computation of these Fourier transforms? Unfortunately, the answer is that there is no way to do this. The problem is that the amplitudes in a quantum computer cannot be directly accessed by measurement. Thus, there is no way of determining the Fourier transformed amplitudes of the original state. Worse still, there is in general no way to efficiently prepare the original state to be Fourier transformed. Thus, finding uses for the quantum Fourier transform is more subtle than first hoped. Further I will introduce the use of the quantum Fourier transform in phase estimation, which is used in order-finding and factoring problems[29, p.220].

## 5.4 Phase estimation

The Fourier transform is the key to the general procedure *phase estimation*, which in turn is the key for many quantum algorithms. Suppose a unitary operator  $U$  has an eigenvector  $|u\rangle$  with eigenvalue  $e^{2\pi i\varphi}$ , where the value of  $\varphi$  is unknown. The goal for the phase estimation algorithm is to estimate  $\varphi$ . To perform the estimation we assume that we have available *black boxes* (sometimes known as *oracles*) capable of preparing the state  $|u\rangle$  and performing the controlled- $U^{2^j}$  operation, for suitable non-negative integers  $j$ . The use of black boxes indicates that the phase estimation procedure is not a complete quantum algorithm in its own right. Rather, you should think of phase estimation as a kind of "subroutine" or "module" that, when combined with other subroutines, can be used to perform interesting computational tasks. In specific applications of the phase estimation procedure we shall do

exactly this, describing how these black box operations are to be performed, and combining them with the phase estimation procedure to do useful tasks. Continuing forward, we will continue imagining them as black boxes [29, p.221].

The quantum phase estimation procedure uses two registers. The first register contains  $t$  qubits initially in the state  $|0\rangle$ . How  $t$  is chosen depends on two things: the number of digits of accuracy we wish to have in the estimate for  $\varphi$ , and with what probability we wish the phase estimation procedure to be successful. The dependence of  $t$  on these quantities emerges naturally from the following analysis [29, p.221].

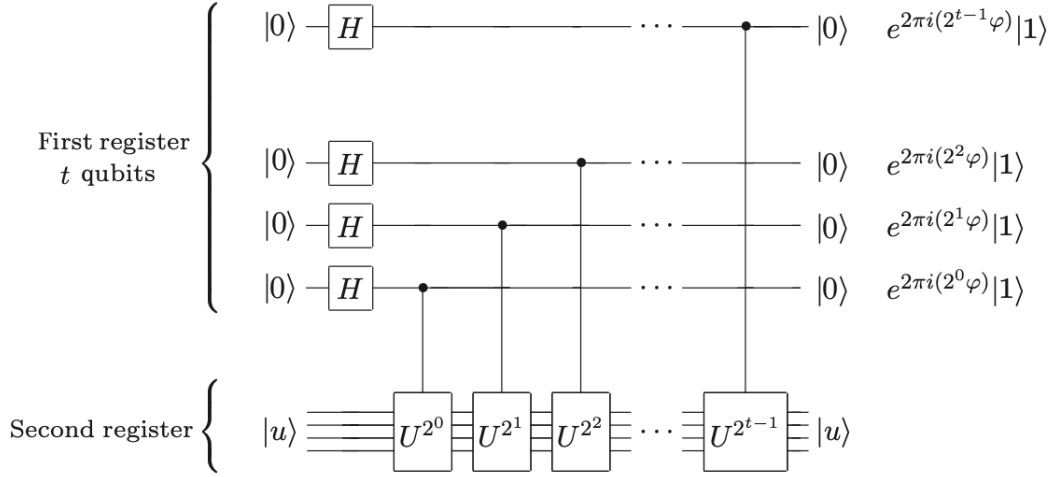


Figure 5.4: The first stage of the phase estimation procedure. Normalization factors of  $1/\sqrt{2}$  have been omitted, on the right [29, p.222].

The second register begins in the state  $|u\rangle$ , and contains as many qubits as is necessary to store  $|u\rangle$ . Phase estimation is performed in two stages. First, we apply the circuit shown in Fig. 5.4. The circuit begins by applying a Hadamard transform to the first register, followed by application of controlled- $U$  operations on the second register, with  $U$  raised to successive powers of two. The final state of the first register is easily seen to be:

$$\begin{aligned} \frac{1}{2^{t/2}} (|0\rangle + e^{2\pi i 2^{t-1}\varphi} |1\rangle) (|0\rangle + e^{2\pi i 2^{t-2}\varphi} |1\rangle) \dots (|0\rangle + e^{2\pi i 2^0\varphi} |1\rangle) \\ = \frac{1}{2^{t/2}} \sum_{K=0}^{2^t-1} e^{2\pi i \varphi K} |K\rangle. \end{aligned} \quad (5.29)$$

We omit the second register from this description, since it stays in the state  $|u\rangle$  throughout the computation [29, p.221–222].

The second stage of the phase estimation is to apply the *inverse* quantum Fourier transform on the first register. This is obtained by reversing the circuit for the quantum Fourier transform, and that can be done in  $\Theta(t^2)$  steps. The third and final stage of phase estimation is to read out the state of the first register by doing a measurement in the computational basis. It will now be shown that this provides a good estimate for  $\varphi$ . An overall schematic of the algorithm is shown in Fig. 5.5 [29, p.222].

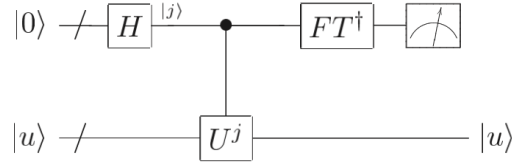


Figure 5.5: Schematic of the overall phase estimation procedure. The top  $t$  qubits (the ‘\’ denotes a bundle of wires, as usual) are the first register, and the bottom qubits are the second register, numbering as many as required to perform  $U$ .  $|u\rangle$  is an eigenstate of  $U$  with eigenvalue  $e^{2\pi i\varphi}$ . The output of the measurement is an approximation to  $\varphi$  accurate to  $t - \lceil \log(2 + \frac{1}{2\epsilon}) \rceil$  bits, with probability of success at least  $1 - \epsilon$  [29, p.223].

To see why phase estimation works, suppose  $\varphi$  may be expressed exactly in  $t$  bits, as  $\varphi = 0.\varphi_1 \dots \varphi_t$ . Then the state (6.19) resulting from the first stage of phase estimation may be rewritten

$$\frac{1}{2^{t/2}} (|0\rangle + e^{2\pi i 0.\varphi_t} |1\rangle) (|0\rangle + e^{2\pi i 0.\varphi_{t-1}\varphi_t} |1\rangle) \dots (|0\rangle + e^{2\pi i 0.\varphi_1\varphi_2 \dots \varphi_t} |1\rangle). \quad (5.30)$$

Summarizing, the phase estimation algorithm allows one to estimate the phase  $\varphi$  of an eigenvalue of a unitary operator  $U$ , given the corresponding eigenvector  $|u\rangle$ . An essential feature at the heart of this procedure is the ability of the inverse Fourier transform to perform the transformation

$$\frac{1}{2^{t/2}} \sum_{j=0}^{2^t-1} e^{2\pi i \varphi j} |j\rangle |u\rangle \rightarrow |\tilde{\varphi}\rangle |u\rangle, \quad (5.31)$$

where the  $|\tilde{\varphi}\rangle$  denotes a state which is a good estimator for  $\varphi$  when measured [29, p.223].



## 5.5 Order-finding and factoring

The phase estimation procedure can be applied to solve a variety of problems. Here, I am going to describe two of the more interesting problems: the *order-finding problem* and the *factoring problem*. These two problems are equivalent to each other and after explaining how the order finding problem works, I will explain how it further applies the ability to factor as well.

### 5.5.1 Order-finding

For positive integers  $x$  and  $N$ ,  $x > N$ , with no common factors, the *order* of  $x$  modulo  $N$  is defined to be the least positive integer,  $r$ , such that  $x^r = 1 \pmod{N}$ . The order-finding problem is to determine the order for some specified  $x$  and  $N$ . Order-finding is believed to be a hard problem on a classical computer, in the sense that no algorithm is known to solve the problem using resources polynomial in the  $O(L)$  bits needed to specify the problem, where  $L \equiv \lceil \log(N) \rceil$  is the number of bits needed to specify  $N$ . It will be explained in this section how phase estimation may be used to obtain an efficient quantum algorithm for order-finding.

The quantum algorithm for order-finding is just the phase estimation algorithm applied to the unitary operator.

$$U|y\rangle \equiv |xy \pmod{N}\rangle, \quad (5.32)$$

with  $y \in \{0, 1\}^L$ . (Note that here and below. When  $N \leq y \leq 2^L - 1$ , we use the convention that  $xy \pmod{N}$  is just  $y$  again. That is,  $U$  only acts non-trivially when  $0 \leq y \leq N - 1$ ). A simple calculation shows that the states defined by

$$|u_s\rangle \equiv \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp\left[\frac{-2\pi i s k}{r}\right] |x^k \pmod{N}\rangle, \quad (5.33)$$

for integer  $0 \leq s \leq r - 1$  are eigenstates of  $U$ , since

$$U|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp\left[\frac{-2\pi i s k}{r}\right] |x^{k+1} \bmod N\rangle \quad (5.34)$$

$$= \exp\left[\frac{2\pi i s}{r}\right] |u_s\rangle. \quad (5.35)$$

Using the phase estimation procedure allows us to obtain, with high accuracy, the corresponding eigenvalues  $\exp(2\pi i s/r)$ , from which we can obtain the order  $r$  with a little bit more work.

There are two important requirements for us to be able to use the phase estimation procedure: we must have efficient procedures to implement a controlled- $U^{2^j}$  operation for any integer  $j$ , and we must be able to efficiently prepare an eigenstate  $|u_s\rangle$  with a nontrivial eigenvalue, or at least a superposition of such eigenstates. The first requirement is satisfied by using a procedure known as *modular exponentiation*, with which we can implement the entire sequence of controlled- $U^{2^j}$  operations applied by the phase estimation procedure using  $O(L^3)$  gates.

The second requirement is a little trickier: preparing  $|u_s\rangle$  requires that we know  $r$ , so this is out of the question. Fortunately, there is a clever observation which allows us to circumvent the problem of preparing  $|u_s\rangle$ , which is that

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle = |1\rangle. \quad (5.36)$$

In performing the phase estimation procedure, if we use  $t = 2L + 1 + \lceil \log(2 + \frac{1}{2\varepsilon}) \rceil$  qubits in the first register, and prepare the second register in the state  $|1\rangle$  - which is trivial to construct - it follows that for each  $s$  in the range 0 through  $r - 1$ , we will obtain an estimate of the phase  $\varphi \approx s/r$  accurate to  $2L + 1$  bits., with probability at least  $(1 - \varepsilon)/r$ . The order-finding algorithm is schematically depicted in Fig. (5.4).

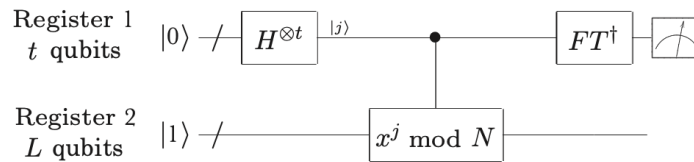


Figure 5.6: Quantum circuit for the order-finding algorithm. This circuit can also be used for factoring, using the reduction given in the next section [29, p.229].

**Algorithm: Quantum order-finding** [29, p.232]

**Inputs:** (1) A black box  $U_{x,N}$  which performs the transformation  $|j\rangle|k\rangle \rightarrow |j\rangle|x^j k \bmod N$ , for  $x$  co-prime to the  $L$ -bit number  $N$ , (2)  $t = 2L + 1 + \lceil \log(2 + \frac{1}{2\varepsilon}) \rceil$  qubits initialized to  $|0\rangle$ , and (3)  $L$  qubits initialized to the state  $|1\rangle$ .

**Outputs:** The least integer  $r > 0$  such that  $x^r = 1 \pmod{N}$ .

**Runtime:**  $O(L^3)$  operations. Succeeds with probability  $O(1)$ .

**Procedure:**

1. initial state  

$$|0\rangle|1\rangle$$
2. create superposition  

$$\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j|1\rangle$$
3. apply  $U_{x, N}$   

$$\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j|x^j \bmod N\rangle$$

$$\approx \frac{1}{r\sqrt{2^t}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi i s j / r} |j|u_s\rangle$$
4. apply inverse Fourier transform to first register  

$$\rightarrow \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\tilde{s}/r\rangle|u_s\rangle$$
5. measure first register  

$$\rightarrow \tilde{s}/r$$
6. apply continued fractions algorithm  

$$\rightarrow r$$

## 5.5.2 Factoring

Given a composite number  $N$ , what prim numbers are multiplied together to get it? This *factoring problem* turns out to be equal to the order-finding problem above. This in the sense that a fast algorithm for order-finding can easily be turned into a fast algorithm for factoring. Here I will show the method used to reduce factoring to order-finding.

The reduction of factoring to order-finding goes on in two basic steps. The first step is to show that a factor can be computed of  $N$  if we can find a non-trivial solution  $x \not\equiv \pm 1 \pmod{N}$  to the equation  $x^2 \equiv 1 \pmod{N}$ . The second step is to show that a randomly chosen  $y$  co-prime to  $N$  is quite likely to have an order  $r$  which is even, and such that  $y^{r/2} \not\equiv \pm 1 \pmod{N}$ , and thus  $x \equiv y^{r/2} \pmod{N}$  is a non-trivial solution to  $x^2 \equiv 1 \pmod{N}$ . These two steps are embodied in the two following theorems[29, p.232–233].

**Theorem 5.5.1.** Suppose  $N$  is an  $L$  bit composite number, and  $x$  is a non-trivial solution to the equation  $x^2 \equiv 1 \pmod{N}$  in the range  $1 \leq x \leq N$ , that is, neither  $x \equiv 1 \pmod{N}$  nor  $x \equiv N - 1 \equiv -1 \pmod{N}$ . Then at least one of  $\gcd(x - 1, N)$  and  $\gcd(x + 1, N)$  is a non-trivial factor of  $N$  that can be computed using  $O(L^3)$  operations[29, p.233].

**Theorem 5.5.2.** Suppose  $N = p_1^{\alpha_1} \dots p_m^{\alpha_m}$  is the prime factorization of an odd composite positive integer. Let  $x$  be an integer chosen uniformly at random, subject to the requirements that  $1 \leq x \leq N - 1$  and  $x$  is co-prime to  $N$ . Let  $r$  be the order of  $x$  modulo  $N$ [29, p.233]. Then

$$p(r \text{ is even and } x^{r/2} \not\equiv -1 \pmod{N}) \geq 1 - \frac{1}{2^m}. \quad (5.37)$$

Theorems 5.5.1 and 5.5.2 can be combined to give an algorithm which, with high probability, returns a non-trivial factor of any composite  $N$ . All the steps in the algorithm can be performed efficiently on a classical computer except an order-finding subroutine. By repeating the procedure, a total prime factorization of  $N$  can be found. The algorithm is summarized below.

**Algorithm: Reduction of factoring to order-finding**[29, p.233–234]

**Inputs:** A composite number  $N$ .

**Outputs:** A non-trivial factor of  $N$ .

**Runtime:**  $O((\log N)^3)$  operations. Succeeds with probability  $O(1)$ .

**Procedure:**

1. If  $N$  is even, return the factor 2.
2. Determine whether  $N = a^b$  for integers  $a \geq 1$  and  $b \geq 2$ , and if so return the factor  $a$ .
3. Randomly choose  $x$  in the range 1 to  $N - 1$ . If  $\gcd(x, N) > 1$  then return the factor  $\gcd(x, N)$
4. Use the order-finding subroutine to find the order  $r$  of  $x \bmod N$ .
5. If  $r$  is even and  $x^{r/2} \not\equiv -1 \pmod{N}$  then compute  $\gcd(x^{r/2} - 1, N)$  and  $\gcd(x^{r/2} + 1, N)$ , and test to see if one of these is a non-trivial factor, returning that factor if so. Otherwise, the algorithm fails.

Steps **1** and **2** of the algorithm either return a factor, or else ensure that  $N$  is an odd integer with more than one prime factor. These steps may be performed using  $O(1)$  and  $O(L^3)$  operations, respectively. Step **3** either returns a factor, or produces a randomly chosen element  $x$  of  $0, 1, 2, \dots, N - 1$ . Step **4** calls the order-finding subroutine, computing the order  $r$  of  $x$  modulo  $N$ . Step **5** completes the algorithm, since Theorem (5.2) guarantees that with probability at least one-half  $r$  will be even and  $x^{r/2} \not\equiv -1 \pmod{N}$ , and then Theorem (5.1) guarantees that either  $\gcd(x^{r/2} - 1, N)$  or  $\gcd(x^{r/2} + 1, N)$  is a non-trivial factor of  $N$ [29, p.234].

## 5.6 How does it apply to RSA?

There are two ways of breaking RSA using the algorithms described in this chapter. One is based on order-finding and the other is based on factoring. Suppose Eve receives an encrypted message  $x^e \bmod n$ , and knows the public key  $(n, e)$  used to encrypt the message. Suppose she can find the order of the encrypted message, that is, she can find the smallest positive integer  $r$  such that  $(x^{er}) = 1 \bmod n$ . (Without loss of generality, we may suppose such an order exists, that is,  $x^e$  is co-prime to  $n$ . If it is not the case, then  $x^e \bmod n$  and  $n$  have a common factor that may be extracted by the Euclidean algorithm, which would allow us to break RSA, as in the second method described shortly.)  $r$  divides  $\Phi(n)$ , and since  $e$  is co-prime to  $\Phi(n)$  it must also be co-prime to  $r$ , and thus has a multiplicative inverse modulo  $r$ . Let  $d'$  be such a multiplicative inverse, so  $ed' = 1 + kr$  for some integer  $k$ . Then Eve can recover the original message  $x$  by raising the encrypted message to the  $d'$ th power [29, p.643]:

$$(x^e)^{d'} \pmod{n} = x^{1+kr} \pmod{n} \tag{5.38}$$

$$= x \cdot x^{kr} \pmod{n} \tag{5.39}$$

$$= x \pmod{n}. \tag{5.40}$$

The interesting thing is that Eve never actually learns the secret key  $(d, n)$ ; she only learns  $(d', n)$ . Of course,  $d'$  is closely related to  $d$ , since  $d'$  is the inverse of  $e$  modulo  $r$ ,  $d$  is the inverse of  $e$  modulo  $\Phi(n)$ , and  $r$  divides  $\Phi(n)$ . Nevertheless, this example shows that it is possible to break RSA without necessarily determining the exact value of the secret key. Of course, this method only works if Eve has an efficient method for order-finding, and no such method is known for a classical computer. On a quantum computer, however, order-finding (as you may recall from section 5.5.1) can be efficiently accomplished, and thus RSA can be broken [29, p.643].

A second method for breaking RSA (and the most implicit) allows one to determine the secret key completely using factoring. Suppose Eve could factor  $n = pq$ , extracting  $p$  and  $q$ , and thus giving a means for efficiently computing  $\Phi(n) = (p - 1)(q - 1)$ . It is then an easy matter for Eve to compute  $d$ , the inverse of  $e$  modulo  $\Phi(n)$ , and thus completely determine the secret key  $(d, n)$ . So, if factoring large numbers were easy then it would be easy to break RSA. As you may recall from section 5.5.2, this is possible and can be done efficiently on a quantum computer [29, p.643].

# Chapter 6

## Post-quantum cryptography

In the last three decades, public-key cryptography has become an indispensable part of global communication digital infrastructure. These networks support a massive amount of applications that are important to our economy, our security, our way of life, such as mobile phones, internet commerce, social networks and cloud computing. In the connected world we live in now, the ability of individuals, businesses and governments to communicate securely is of the utmost importance[16].

Many of our most crucial communication protocols rely on three core cryptographic functionalities: public-key encryption, digital signatures and key exchange. Currently, these functionalities are primarily implemented using Diffie-Hellman key exchange, the RSA cryptosystem, and elliptic curve cryptosystems. The security of these systems depends on the difficulty of number theoretic problems, such as Integer Factorization or the Discrete Log Problem over different groups [16].

In 1994, Peter Shor of Bell Laboratories showed that quantum computers, a new technology using the physical properties of matter and energy to perform calculations, can efficiently solve each of these number theoretic problems. Therefore all public-key cryptosystems based on these number theoretic problems are rendered impotent. That means a sufficiently powerful quantum computer will put many forms of modern communication, from key exchange to encryption to digital authentication, in jeopardy [16].

The discovery that quantum computers could potentially be used to solve certain problems faster than classical computers sparked great interest in quantum computing and raised a lot of questions that researchers are working on. For example is quantum complexity fundamentally different from classical complexity? Or when will large-scale quantum computers be built? Is there a way to resist both a quantum and a classical adversary [16]?

In the years since Shor's discovery, the theory of quantum algorithms has developed significantly. Quantum algorithms achieving exponential speed-up have been discovered for several problems relating to physics simulation, number theory and topology. Nevertheless, the list of problems admitting exponential speed-up by quantum computation remains relatively small. In contrast, more modest speed-ups have been developed for broad classes of problems related to searching, collision finding and evaluation of Boolean formulae. Particularly Grover's search algorithm proffers a quadratic speed-up on unstructured search problems. Even though such a speed-up does not render cryptographic techniques obsolete, it can have the effect of requiring larger key sizes, even in the symmetric key case. See table 6.1 and 6.2 for a summary of the impact of large-scale quantum computers on common cryptographic algorithms, such as RSA and AES. It is not known how far these quantum advantages can be pushed, in addition to how wide the gap is between feasibility in the classical and quantum models [16].

The question of when a large-scale quantum computer will be built is complicated and contentious. While in the past it was less clear that large quantum computers are a physical possibility, researchers now believe it to be merely a significant engineering challenge. Some experts even predict that within the next 20 years or so years, sufficiently large quantum computers will be built to break essentially all public-key cryptography schemes currently in use. It has taken almost 20 years to deploy our modern public-key cryptography infrastructure. It will take significant effort to ensure a smooth and secure transition from the current widely used cryptosystems to their quantum computing resistant counterparts. Therefore, regardless of whether we can estimate the exact time of the arrival of the quantum computing era, we must begin now to prepare our information security systems to be able to resist quantum computing [16].



<b>Cryptographic Algorithm</b>	<b>Type</b>	<b>Purpose</b>	<b>Impact from large-scale quantum computer</b>
AES	Symmetric key	Encryption	Larger key sizes needed
SHA-2, SHA-3	————	Hash functions	Larger output needed
RSA	Public-key	Signatures, key establishment	No longer secure
ECDSA, ECDH (Elliptic Curve Cryptography)	Public-key	Signatures, key establishment	No longer secure
DSA (Finite Field Cryptography)	Public-key	Signatures, key establishment	No longer secure

Table 6.1: Impact of quantum computing on common cryptographic algorithms [16, p.2]

In addition to the cryptographic algorithms taken into consideration by NIST, these algorithms has also been considered in the next table:

<b>Cryptographic Algorithm</b>	<b>Type</b>	<b>Purpose</b>	<b>Impact from large-scale quantum computer</b>
Diffie-Hellmann	Public-key	Key-exchange	No longer secure
Algebraically homomorphic	Public-key	Encryption	No longer secure
McEliece	Public-key	Encryption	Still secure
Lattice based cryptography	Public-key	Encryption	Still secure

Table 6.2: Continued: Impact of quantum computing on common cryptographic algorithms [1, p.24]

A large international community has emerged to address the issue of information security in a quantum computing future, in the hope that our public-key infrastructure may remain intact by utilizing new quantum-resistant primitives. In the academic world, this new area of science bears the name "Post-Quantum Cryptography". This is an active area of research, with its own conference series, PQCrypto, which started in 2006. It has received substantial support from national funding agencies, most notably in Europe and Japan, through the European Union(EU) projects PQCrypto and SAFEcrypto, and the CREST Crypto-Math project in Japan [16].

These contributions have led to advances in fundamental research, paving the way for the deployment of post-quantum cryptosystems in the real world. In the past few years, industry and standards organizations have started their own activities in this field: since 2013, the European Telecommunications Standards Institute(ETSI) has held three "Quantum-Safe Cryptography" workshops, and in 2015 NIST held a workshop on "Cybersecurity in a Post-Quantum World", which was attended by over 140 people from government, industry and academia [16].

## 6.1 Post-quantum primitives

The most important uses of public-key cryptography today are for digital signatures and key establishment. As mentioned above, the construction of a large-scale quantum computer would render many of these public-key cryptosystems insecure. In particular, this includes those based on integer factorization, such as RSA, in addition to the ones based on the hardness of the discrete log problem. In contrast, the impact on symmetric key systems will not be as drastic. Grover's algorithm provides a quadratic speed-up for quantum search algorithms in comparison with search algorithms on classical computers. It is not known whether Grover's algorithm will ever be practically relevant or not, but if it is, doubling the key size will be sufficient to preserve security. Furthermore, it has been shown that an exponential speed-up for search algorithms is impossible, suggesting that symmetric algorithms and hash functions should be usable in a quantum era [16].

Consequently, the search algorithms believed to be resistant to attacks from both classical and quantum computers has focused on public-key algorithms. In this section there will be given a brief overview of the main families for which post-quantum primitives have been proposed. These families include those based on lattices, codes and multivariate polynomials, in addition to a handful of others [16].

**Lattice-based cryptography** - Cryptosystems based on lattice problems have received renewed interest, for a few reasons. Exciting new applications (such as fully homomorphic encryption, code obfuscation and attribute-based encryption) have been made possible using lattice-based cryptography. Most lattice-based key establishment algorithms are relatively simple, efficient and highly parallelizable. Also, the security of some lattice-based systems are provably secure under a worst-case hardness assumption, rather than on the average case. On the other hand, it has proven difficult to give precise estimates of the security of

lattice schemes against even known cryptanalysis techniques [16].

**Code-based cryptography** - In 1978, the McEliece cryptosystem was first proposed, and has not been broken since. Since that time other systems based on error-correcting codes have been proposed. While quite fast, most code-based primitives suffer from having very large key sizes- Newer variants have introduced more structure into the codes in an attempt to reduce the key sizes, however the added structure has also led to successful attacks on some proposals. While there have been some proposals for code-based signatures, code-based cryptography has seen more success with encryption schemes [16].

**Multivariate polynomial cryptography** - These schemes are based on the difficulty of solving systems of multivariate polynomials over finite fields. Several multivariate cryptosystems have been proposed over the past few decades, with many having been broken. While there have been some proposals for multivariate encryption schemes, multivariate cryptography has historically been more successful as an approach to signatures [16].

**Hash-based signatures** - Hash-based signatures are digital signatures constructed using hash functions. Their security, even against quantum attacks, is well understood. Many of the more efficient hash-based signature schemes have the drawback that the signer must keep a record of the exact number of previously signed messages, and any error in this record will result in insecurity. Another drawback is that they can produce only a limited number of signatures. The number of signatures can be increased, even to the point of being effectively unlimited, but this also increases the signatures size [16].

**Other** - A variety of systems have been proposed which do not fall into the above families. One such proposal is based on evaluating isogenies on supersingular elliptic curves. While the discrete log problem on elliptic curves can be efficiently solved by Shor's algorithm on a quantum computer, the isogeny problem on supersingular curves has no similar quantum attack known. Like some other proposals, for example those based on the conjugacy search problem and related problems in braid groups , there has not been enough analysis to have much confidence in their security [16].

## 6.2 Post-quantum candidates

NIST (National Institute of Standards and Technology) launched a process in 2016 to find a new standard for one or more quantum-resistant public-key encryption and key-establishment algorithms. In addition to public-key and key-establishment algorithms, they also wanted find a new Digital Signature Standard.

In November 2017, there were submitted 82 candidates to NIST for consideration. 69 among these met the submission requirements and the minimum acceptance criteria. While the process is still ongoing, there are now 17 Second-Round candidates remaining public-key encryption and key-establishment candidates and 9 digital signature candidates. For simplicity public-key encryption and key establishment algorithms will be abbreviated to PKE and KEM and digital signature algorithms will be abbreviated to DS in the table when describing the type of the algorithms.

<b>Algorithm</b>	<b>Type</b>	<b>Primitive family</b>	<b>Reference to submission paper</b>
BIKE	PKE and KEM	Code-based	[5]
Classic McEliece	PKE and KEM	Code-based	[10]
CRYSTALS-Dilithium	DS	Lattice-based	[20]
CRYSTALS-KYBER	PKE and KEM	Lattice-based	[7]
Falcon	DS	Lattice-based	[22]
FodoKEM	PKE and KEM	Lattice-based	[3]
GeMSS	DS	Multivariate	[14]
HQC	PKE and KEM	Code-based	[23]
LAC	PKE and KEM	Lattice-based	[26]
LEDAcrypt	PKE and KEM	Code-based	[9]
LUOV	DS	Multivariate	[12]
MQDSS	DS	Multivariate	[17]
NewHope	PKE and KEM	Lattice-based	[4]
NTRU	PKE and KEM	Lattice-based	[15]
NTRU Prime	PKE and KEM	Lattice-based	[11]
NTS-KEM	PKE and KEM	Code-based	[2]
Picnic	DS	Other	[34]
qTESLA	DS	Lattice-based	[13]
Rainbow	DS	Multivariate	[19]
ROLLO	PKE and KEM	Code-based	[27]
Round5	PKE and KEM	Lattice-based	[8]
RQC	PKE and KEM	Code-based	[28]
SABER	PKE and KEM	Lattice-based	[18]
SIKE	PKE and KEM	Other	[25]
SPHINCS+	DS	Hash-based	[6]
ThreeBears	PKE and KEM	Lattice-Based	[24]

Table 6.3: Public-key encryption, key-establishment and digital signature algorithm candidates remaining in round 2 of the NIST competition [30].

# Chapter 7

## Conclusion

In my thesis I have shown how quantum computing with quantum algorithms can efficiently break public-key algorithms built upon a certain number theoretic problem, that is, factoring. Factoring has not yet been efficiently solved on a classical computer, and the RSA crypto scheme can be broken using either the quantum order-finding algorithm or the quantum factoring algorithm. Both of them are able to solve the factoring problem in polynomial time, which is a big difference from classical computing seeing that RSA has not yet been broken using classical measures.

One thing is for sure and that is the fact that a sufficiently large-scale quantum computer able to run the quantum algorithms discussed in my thesis will render widely used public-key algorithms based on factoring, like RSA, insecure. Even though quantum computers have existed for a while, they are not yet in commercial use seeing that they are expensive and exist at near zero Kelvin. This will not only affect the public-key schemes, but also the symmetric cryptography schemes that rely on the security of public-key algorithms to exchange keys in order to go through the process of encryption and decryption.

### 7.1 Path forward

Going forward one should carefully address the issue of quantum algorithms and quantum computers. The public-key algorithms needs to be replaced by post-quantum algorithms in the event that quantum computers are able to run the quantum algorithms discussed in my thesis. As to the matter of which algorithms that will replace the public-key algorithms in use now, this is not a recommendation that I am fit to give. But I am confident that the researchers in the post-quantum community will find a suitable fit that will provide

security in the foreseeable future. After looking at the candidates in the NIST post-quantum competition it seems like lattice-based cryptography is a popular option to explore going forward. What we also can expect is progress in classical computing and attacks on public-key algorithms. As shown in [35] progress is being made in classical terms. Those threats may be mitigated by increasing key sizes.





# Bibliography

- [1] *Post-Quantum Cryptography*. Lecture notes in computer science Post-quantum cryptography. Springer Berlin Heidelberg : Imprint: Springer, Berlin, Heidelberg, 1st ed. 2009. edition, 2009. ISBN 1-282-00091-8.
- [2] Martin Albrecht et al. Nts-kem. 2019.  
**URL:** <https://drive.google.com/file/d/1N3rv4HKCt9yU4xn6wuepsBUrfQW8cuFy/view>.
- [3] Erdem Alkim et al. Frodokem: Learning with errors key encapsulation. 2020.  
**URL:** <https://frodokem.org/files/FrodoKEM-specification-20200325.pdf>.
- [4] Erdem Alkim et al. Newhope. 2020.  
**URL:** [https://newhopecrypto.org/data/NewHope\\_2020\\_04\\_10.pdf](https://newhopecrypto.org/data/NewHope_2020_04_10.pdf).
- [5] Nicolas Aragon et al. Bike: Bit flipping key encapsulation. 2020.  
**URL:** [https://bikesuite.org/files/v4.0/BIKE\\_Spec.2020.05.03.1.pdf](https://bikesuite.org/files/v4.0/BIKE_Spec.2020.05.03.1.pdf).
- [6] Jean-Philippe Aumasson et al. Sphincs+ submission to the nist post-quantum project. 2019.  
**URL:** <https://sphincs.org/data/sphincs+-round2-specification.pdf>.
- [7] Roberto Avanzi et al. Crystals-kyber: Algorithm specifications and supporting documentation. 2019.  
**URL:** <https://pq-crystals.org/kyber/data/kyber-specification-round2.pdf>.
- [8] Hayo Baan et al. Round5: Kem and pke based on (ring) learning with rounding. 2020.  
**URL:** [https://round5.org/doc/Round5\\_Submission042020.pdf](https://round5.org/doc/Round5_Submission042020.pdf).
- [9] Marco Baldi et al. Ledacrypt. 2020.  
**URL:** [https://www.ledacrypt.org/documents/LEDACrypt\\_v3.pdf](https://www.ledacrypt.org/documents/LEDACrypt_v3.pdf).

- [10] Daniel J. Bernstein et al. Classic mceliece: conservative code-based cryptography. 2019.  
**URL:** <https://classic.mceliece.org/nist/mceliece-20190331.pdf>.
- [11] Daniel J. Bernstein et al. Ntru prime: reducing attack surface at low cost. 2019.  
**URL:** <https://ntruprime.cr.yp.to/ntruprime-20170816.pdf>.
- [12] Ward Beullens et al. Luov. 2019.  
**URL:** [https://github.com/WardBeullens/LUOV/blob/master/Supporting\\_Documentation/luov.pdf](https://github.com/WardBeullens/LUOV/blob/master/Supporting_Documentation/luov.pdf).
- [13] Nina Bindel et al. Submission to nist’s post-quantum project(2nd round): lattice-based digital signature scheme qtesla. 2020.  
**URL:** [https://qtesla.org/wp-content/uploads/2020/04/qTESLA\\_round2\\_14.04.2020.pdf](https://qtesla.org/wp-content/uploads/2020/04/qTESLA_round2_14.04.2020.pdf).
- [14] Antoine Casanova et al. Gemss: A great multivariate short signature. 2020.  
**URL:** <https://www-polsys.lip6.fr/Links/NIST/GeMSS.html>.
- [15] Chong Chen et al. Ntru. 2019.  
**URL:** <https://ntru.org/f/ntru-20190330.pdf>.
- [16] Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. Report on post-quantum cryptography. 2016.  
**URL:** <https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf>.
- [17] Ming-Shing Chen et al. Mqdss specifications. 2020.  
**URL:** <http://mqdss.org/files/mqdssVer2point1.pdf>.
- [18] Jan-Pieter D’Anvers et al. Saber: Mod-lwr based kem. 2020.  
**URL:** <https://www.esat.kuleuven.be/cosic/pqcrypto/saber/resources.html>.
- [19] Jintai Ding et al. Rainbow - algorithm specification and documentation. 2020.  
**URL:** <https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions>.
- [20] Léo Ducas et al. Crystals-dilithium. 2019.  
**URL:** <https://pq-crystals.org/dilithium/data/dilithium-specification-round2.pdf>.
- [21] Jean-Guillaume Dumas, Jean-Louis Roch, Éric Tannier, and Sébastien Varrette. In *Foundations of Coding*, pages 321–474. John Wiley Sons, Inc, 2015. ISBN 9781118881446.

- [22] Pierre-Alain Fouque et al. Falcon: Fast-fourier lattice-based compact signatures over ntru. 2019.  
**URL:** <https://falcon-sign.info/falcon.pdf>.
- [23] Philippe Gaborit et al. Hamming quasi-cyclic (hqc). 2020.  
**URL:** [http://pqc-hqc.org/doc/hqc-specification\\_2020-05-29.pdf](http://pqc-hqc.org/doc/hqc-specification_2020-05-29.pdf).
- [24] Mike Hamburg et al. Post-quantum cryptography proposal: Three bears. 2019.  
**URL:** <https://sourceforge.net/p/threebears/code/ci/master/tree/threebears-spec.pdf>.
- [25] David Jao et al. Supersingular isogeny key encapsulation. 2020.  
**URL:** <https://sike.org/files/SIDH-spec.pdf>.
- [26] Xianhui Lu et al. Lattice-based cryptosystems (lac). 2020.  
**URL:** <https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions>.
- [27] Carlos A. Melchor et al. Rollo - rank-ouroboros, lake locker. 2020.  
**URL:** [https://pqc-rollo.org/doc/rollo-specification\\_2020-04-21.pdf](https://pqc-rollo.org/doc/rollo-specification_2020-04-21.pdf).
- [28] Carlos A. Melchor et al. Rank quasi-cyclic (rqc). 2020.  
**URL:** [http://pqc-rqc.org/doc/rqc-specification\\_2020-04-21.pdf](http://pqc-rqc.org/doc/rqc-specification_2020-04-21.pdf).
- [29] M.A. Nielsen and I.L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. ISBN 9781139495486.  
**URL:** <https://books.google.no/books?id=-s4DEy7o-a0C>.
- [30] NIST. Post quantum cryptography, round 2 submissions, 2020.  
**URL:** <https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions>.
- [31] C. Paar and J. Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer Berlin Heidelberg, 2010. ISBN 9783642446498.
- [32] John Preskill. Lecture notes for ph219/cs219: Quantum information and computation, chapter 4. 2001.  
**URL:** [http://www.theory.caltech.edu/~preskill/ph229/notes/chap4\\_01.pdf](http://www.theory.caltech.edu/~preskill/ph229/notes/chap4_01.pdf).
- [33] David Sherill. Postulates of quantum mechanics, 2006.  
**URL:** <http://vergil.chemistry.gatech.edu/notes/quantrev/node20.html>.

- [34] Greg Zaverucha et al. The picnic signature algorithm. 2020.  
**URL:** <https://github.com/microsoft/Picnic/blob/master/spec/spec-v3.0.pdf>.
- [35] Paul Zimmermann. Factorization of rsa-250, 2020.  
**URL:** <https://lists.gforge.inria.fr/pipermail/cado-nfs-discuss/2020-February/001166.html>.