AGH University
Of Science and Technology in Kraków

Faculty of Computer Science, Electronics
and Telecommunications

Institute of Computer Science

Master of Science Thesis

Review, analysis and simulation of quantum algorithms in cryptography

Bartłomiej Patrzyk

Supervisor:
dr inż. Katarzyna Rycerz

Kraków 2014

OŚWIADCZENIE AUTORA PRACY

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie, i nie korzystałem ze źródeł innych niż wymienione w pracy.

. . . .

PODPIS

Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Informatyki, Elektroniki i Telekomunikacji

Katedra Informatyki

Praca Magisterska

Przegląd, analiza i symulacja algorytmów kwantowych w kryptografii

Bartłomiej Patrzyk

Opiekun:
dr inż. Katarzyna Rycerz

Kraków 2014

Acknowledgements

I would like to express the deepest appreciation to my supervisor, dr Katarzyna Rycerz, for her invaluable help and support. I am grateful for the valuable discussions and suggestions.
I would like to thank dr Marian Bubak for inspiring me to conduct the research in the field of quantum computation, as well as for his valuable advices regarding the technical aspects of writing the thesis.
I gratefully acknowledge the support of dr Maciej Malawski and dr Włodzimierz Funika. Their comments on using the Shor's Algorithm during classes helped me improve my implementations.
I would like to thank Joanna Patrzyk for her support, suggestions and also for creating the QuIDE simulator which made it possible to conveniently simulate Shor's Algorithm.
Last, but not least, I acknowledge the help of Mr. Marcin Kliś and Mr. Christopher Majda who read the proof of this thesis and corrected the language mistakes.

Abstract

Quantum Computer Science is becoming an important field of science, as well as a significant branch of industry. One of its applications is the cryptology. There are quantum mechanical systems for secure cryptographic key distribution. Quantum computers can also be used for compromising widely used asymmetric cryptography applications.

Shor's Factoring Algorithm makes it possible to factor numbers in polynomial time on quantum computer. The difficulty of factoring into primes is the basis of the cryptographic strength of the RSA cryptosystem. Currently, there are no quantum computers capable of executing Shor's Algorithm. Nevertheless, there is extensive scientific research on the optimization possibilities of this algorithm.

In this thesis we analyze the optimization variants of Shor's Factoring Algorithm. We describe and compare the quantum circuits. We also simulate significant variants of Shor's Algorithm in the QuIDE quantum computer simulator. The results of simulations are compared in terms of computational complexity, memory complexity and the success rate.

The thesis is organized as follows: Chapter 1 introduces the quantum computer science, states the problem discussed in this thesis and presents the goals to be achieved. In Chapter 2 we describe the notation used throughout the thesis, the quantum bits and their properties as well as the quantum gates and circuits. Chapter 3 presents quantum key distribution and quantum commitment protocols. This Chapter also explains how Shor's Algorithm leads to breaking the RSA cryptosystem. In Chapter 4 we describe Shor's Factoring Algorithm in detail. We present and compare different optimization variants of quantum circuits. Chapter 5 presents the results of the Shor's Algorithm simulations. In Chapter 6 we discuss the achievement of thesis goals, summarize the results of the research and outline the future directions. Appendix A presents papers related to this thesis.

Contents

Chapter 1

Introduction

This Chapter introduces the scope of the thesis, namely quantum computation and its applications in cryptology. It describes the motivation of the thesis, its goals and the outline of the problem it solves. Section 1.1 describes the connection between quantum physics and computer science. It briefly presents the development in the fields of Quantum Computation and Quantum Communication. Section 1.2 summarizes the problem which is to be solved in this thesis. In Section 1.3, the goals which should be achieved in the thesis are presented. Section 1.4 refers to the contribution of other authors to this thesis. In Section 1.5, the outline of the following Chapters is presented

1.1 Quantum Physics in Computer Science

Soon after the discovery of elementary particles, it has been noticed that they can be treated as information carriers. Subsequently, in 1980 it has been observed that quantum mechanics can be used to perform computations [1]. Later, Richard Feynman has shown that quantum computation may actually be more powerful than digital computers. Over time, more and more ideas for exploiting quantum mechanics in computer science have been proposed. Consequently, a new field of science has been developed, sometimes referred to as Quantum Computer Science. It can be further divided into Quantum Computation and Quantum Communication.

1.1.1 Quantum Computation

Richard Feynman is believed to be the first to state that quantum computation may be more powerful than the Turing machine [2]. He also gave an explanation as to why the simulation of a quantum computer on a classical computer is very difficult computationally. He stated that performing quantum computations may be possible by the computer based on the laws of quantum mechanics. David Deutsch was the first to ask the question implicitly stated by Feynman, whether quantum computation leads to solving problems quicker than on the classical computer [3].

First attempts to prove that quantum computation may actually be faster than classical were carried out by Deutsch and Jozsa [4] as well as by Berthiaume and Brassard [5]. They have not shown any improvement in the computational complexity using a quantum computer. However, they did show problems, for which the quantum computers find exact solution in polynomial time. The same problems can be solved in polynomial time by classical computers only with some probability.

Problems for which computational complexity is much better on the quantum computer than on a classical computer were discussed by Bernstein and Vazirani [6] and Simon [7]. Both the problems involve finding a constant value programmed into a subroutine in which the internal structure is not known. In each case, there is significant speedup when quantum computation is concerned. The Bernstein and Vazirani problem can be solved by applying the subroutine a number of times which grows linearly on classical computer while on quantum computer it has to be applied only once. Even better optimization is achieved in Simon's problem - exponential complexity on a classical computer is reduced to linear complexity using a quantum computer.

Simon's problem was an inspiration for the notable Shor's Factoring Algorithm [8]. It provides a means to effectively factor large numbers into primes. It is of significant importance because the difficulty of factoring into primes is the basis for widely used RSA cryptosystem [9]. Due to the significance of the algorithm, many implementations and optimizations were proposed. They are discussed in Chapter 4.

Experiments and Realizations

In the recent years there were several attempts to build a physical circuit implementing Shor's Factoring Algorithm. The first experiment was demonstrated by IBM Almaden Research Center [10]. The number 15 was factored into 3 and 5 using 7 qubits by the means of Nuclear Magnetic Resonance. However, the experiment has been criticized for not being a real demonstration of the Shor's algorithm because no entanglement was observed [11]. Entanglement means that the state of one particle is dependent on the state of other particle.

In 2012, the number 15 was factored with success at the University of California [12]. In the experiment, a compiled version of the algorithm was implemented. It was proven that the circuit was capable of creating Bell states and three-qubit entanglement. In the same year the number 21 was successfully factored using a two-photon compiled version of the algorithm [13].

There is also a progress in the development of a general purpose quantum computer. The first commercially available machine, The D-Wave One™, was presented in 2010 by the D-Wave Systems Inc [14]. The latest model, The D-Wave Two™, was introduced in 2013. It has a 512 qubit register and it is able to solve optimization problems. The D-Wave Two™ computers are intended to be used to help design new medicines, debug software code, improve algorithms for optimization tasks and build more accurate models for many applications such as speech recognition and web search.

1.1.2 Quantum Communication

At the same time, a lot of work has been done to investigate how quantum mechanics can be useful for communication. It turns out that it can provide a means to safely exchange random cryptographic keys between communicating parties.

The first work in this field was by Bennett and Brassard [15], resulting in protocol a that enables communicating parties to safely agree upon a secret key. Protocols for secret key agreement are usually referred to as Quantum Key Distribution (QKD) protocols.

Most QKD protocols are based on the properties of preparing and measuring quantum states. A different approach was proposed by Ekert [16]. In order to distribute a secret key between parties it uses entangled particles, unlike pure state particles used by the previously discussed protocols. Ekert's protocol exploits famous Einstein-Podolsky-Rosen paradox [17][18] and generalized Bell's theorem [19][20] to ensure safe key agreement between parties.

On the other hand, there is the Quantum Commitment protocol, which enables parties to exchange a decision. This protocol ensures, that after committing a decision by one party it can not be changed before revealing it to the other party.

We describe QKD and Quantum Commitment protocols in Chapter 3.

Experiments and Realizations

Several experimental Quantum Key Distribution networks are deployed. The Wroclaw Quantum Network is a project of Wroclaw University of Technology [21]. Three buildings about 5 kilometers apart, are connected with an optical Quantum Key Distribution network.

The Tokyo QKD Network consists of several nodes, ranging from 1 to 90 kilometers apart [22]. It incorporates devices from several different vendors and research departments The network was successfully used to demonstrate quantum key distribution application to secure video conferencing and telephone calls [23] .

While both the Tokyo QKD Network and the Wroclaw Quantum Network are based on point-to-point connections between nodes, the Los Alamos National Laboratory quantum network uses a different approach [24]. It has hub-and-spoke topology, where endpoints are connected to a central hub. The feature of such a design is that the endpoint nodes use less expensive small form factor devices. Only the central node needs expensive and large photon detectors.

There are several companies which offer commercial Quantum Key Distribution appliances. Their products are used by public institutions, governments, industry and research centers. Most notable vendors are idQuantique (Switzerland) [25], MagiQ Technologies (United States of America) [26], QuintessenceLabs (Australia) [27] and SeQureNet (France) [28].

There are also Quantum Key Distribution simulators [29]. They can be used to help develop and verify new Quantum Key Distribution protocols [30]. Such simulators also enable the verification of QKD network parameters such as Quantum Bit Error Rate (QBER).

1.2 Problem Outline

Quantum Computation is becoming a more and more practical scientific field. Nowadays the first quantum computers are available, such as The D-Wave Two™. They are not yet universal computing devices, but are capable of solving specific problems such as minimization.

However, the research in the field began with theoretical searching for problems which can be solved more efficiently on quantum computer rather than on classical devices. Most of the very first algorithms were rather artificial, however the development led to discoveries of significantly practical meaning.

In 1994 Peter Shor discovered that the quantum computer can factor numbers into primes in polynomial time [8]. In contrast, currently known factoring algorithms for classical computers work in exponential time. This computational complexity is the basis for widely used RSA cryptosystem. Shor's discovery thus led to the fact that a quantum computer can be used in cryptology.

It now seems important to investigate whether there are different approaches to use quantum mechanics in cryptological applications. This thesis attempts to summarize these approaches.

In terms of Quantum Communication, the most important cryptological applications are Quantum Key Distribution and Quantum Commitment. Quantum Key Distribution is a protocol which enables communicating parties to exchange a random secret key over an insecure communication channel. Quantum Commitment is an example of a protocol for passing a message in a secure way.

In turn, in Quantum Computation, Shor's Factoring Algorithm is most frequently cited and it is believed to be the most revolutionary. When implemented on a scalable quantum computer, it can lead to compromising the RSA cryptosystem. Current technology does not yet allow to build a quantum circuit for Shor's Algorithm capable of factoring numbers larger than 21. However, regardless of technological impossibility, a lot of work is done to provide the most efficient circuit implementation, based on assumptions regarding future quantum computer architectures.

In essence, a quantum computer consists of quantum bits (qubits). Qubit is a basic information unit, most often implemented with an elementary particle. It is now known that qubits are fragile and highly interfere with the environment. This forms fundamental obstacle for constructing a scalable quantum computer. Therefore, it is important to optimize quantum algorithms in terms of the number of used qubits. What is more, it is insufficient just to implement classical algorithms on quantum computers. It is inevitable to redesign them, exploiting quantum mechanics properties.

Shor's Factoring Algorithm is an example of a complex algorithm which parts can be implemented in various ways. This thesis intents to describe implementation variants and compare them. The emphasis is put on optimizing the number of qubits.

While there is still no technology allowing to execute Shor's algorithm in practice, quantum computer simulators have been implemented on classical computers. It has been proven that such simulators cannot execute quantum algorithms efficiently [2]. However, they are a great tool to test and analyze quantum algorithms. In this thesis several variants of Shor's Algorithm are discussed and simulated.

1.3 Goals of the Thesis

The main objective of this thesis is to review and simulate quantum algorithms in cryptology. It has been fulfilled by achieving the following goals:

Summary of Quantum Cryptology concepts

First of all, previously discussed algorithms and protocols, namely Shor's Factoring Algorithm, Quantum Key Distribution and Quantum Commitment should be introduced and briefly described.

Review of implementation variants of Shor's Algorithm

The Shor's Algorithm needs to be described in detail. We should enumerate the steps of the algorithm with respect to Classical Preprocessing, Quantum Order Finding and Classical Postprocessing. Different approaches to implement quantum circuits for Quantum Order Finding should be depicted and described. They they should be compared in terms of required quantum register lengths.

Simulation of Shor's Algorithm

The most significant examples of quantum circuits should be implemented in the quantum computer simulator. They need to be experimentally tested to be working correctly.

Simulation Results Analysis

We should present and analyze the results of the simulations. We should compare execution time, memory usage and the success rate of different implementation variants.

1.4 Contribution of Other Authors

This thesis requires implementation and simulation of optimization variants of Shor's Factoring Algorithm. We used QuIDE quantum computer simulation environment developed by Joanna Patrzyk in her M. Sc. thesis [31]. The QuIDE has been used to analyze the behavior of the algorithm as well as to compare the results of the optimization variants.

1.5 Thesis Outline

The thesis is organized as follows:

Chapter 2 introduces the terms of quantum computation theory. We present the notation used to describe quantum mechanical systems. We discuss the properties of quantum bits (qubits) and operations on qubits. At the end of the Chapter we give an introduction to quantum circuits.

In Chapter 3 the author discusses different quantum cryptology concepts. At the beginning Quantum Key Distribution is described. We give examples of Prepare and Measure protocols and Entanglement-based protocols. Subsequently, we describe Quantum Commitment. Afterwards, the RSA Cryptosystem and its relationship with Shor's Factoring Algorithm is described. At the end of the Chapter we summarize quantum cryptology and emphasize the most important concepts from the point of view of this thesis.

In Chapter 4 we describe Shor's Factoring Algorithm in detail, with the division into Classical Preprocessing, Quantum Order Finding and Classical Postprocessing. We present key parts of Quantum Order Finding and describe their meaning. Later in the Chapter we present the implementations and optimizations of circuits for Quantum Order Finding.

Chapter 5 presents the results of Shor's Factoring Algorithm simulations on the classical computer. We take two implementation variants of the algorithm into consideration. The results are compared in terms of computation time, memory usage and the success rate.

In Chapter 6 we discuss the achievement of the thesis goals. We summarize the results of the Shor's Algorithm simulations. Finally, we describe issues that should be considered in the future.

Appendix A presents the paper related to this thesis.

Chapter 2

Quantum Computation

In this Chapter we introduce the key terms of quantum computation theory. We present the concepts which are necessary to understand the thesis. More comprehensive description of the quantum computation can be found in referenced bibliography [33] [38].
In Section 2.1 we present the Dirac notation for describing quantum mechanical states. Section 2.2 describes the quantum bits and their features. In Section 2.3 we present the quantum gates. Section 2.4 introduces the quantum circuits. In Section 2.5 we summarize the Chapter.

2.1 Dirac Notation

The Dirac notation, also called a Braket notation, is used to describe the quantum mechanical systems. In Table 2.1 we summarize the notation used throughout this thesis.

TABLE 2.1: Summary of the Dirac notation.

Notation	Description
$\lvert\psi\rangle$	Vector. Also called a ket.
$\langle\psi\rvert$	Vector dual to $\lvert\psi\rangle$. Also called a bra.
$\langle\phi\lvert\psi\rangle$	Inner product between the vectors $\lvert\phi\rangle$ and $\lvert\psi\rangle$. Also called a braket.
$\lvert\phi\rangle \otimes \lvert\psi\rangle$	Tensor product of vectors $\lvert\phi\rangle$ and $\lvert\psi\rangle$.
$\lvert\phi\rangle \lvert\psi\rangle$	Abbreviated notation for tensor product of vectors $\lvert\phi\rangle$ and $\lvert\psi\rangle$.
z^*	Complex conjugate of the complex number z — $(1+i)^* = 1-i$
A^*	Complex conjugate of the A matrix.
A^T	Transpose of the A matrix.
$A^\dagger$	Hermitian conjugate of the A matrix — $A^\dagger = (A^T)^*$.

2.2 Fundamental properties of quantum bits

The quantum bit (qubit) is the quantum mechanical counterpart of bit in classical computations. In this Section we discuss the features of qubits and compare them to classical bits (cbits).

State of qubits

Both the classical and quantum bits have state. The state of cbit is always either a 0 or 1. The state of qubit can be a $|0\rangle$, $|1\rangle$ or the linear combination of states (superposition):

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

The α and β are complex numbers and $|\alpha|^2 + |\beta|^2 = 1$.

The states $|0\rangle$ and $|1\rangle$ are called computational basis states.

Multiple qubits

A group of n cbits has one of 2^n possible states. Any subset of such group also have a state. The state of n qubits can be expressed as follows:

$$\sum_{x=0}^{2^n-1} \alpha_x |x\rangle_n, \sum_{x=0}^{2^n-1} |\alpha_x|^2 = 1$$

The notation $|x\rangle_n$ means that there are n qubits representing the state x. For example, four qubit state $|0011\rangle_n$ can be expressed as $|3\rangle_n$. The squares of α_x coefficients represent the probability of measuring state $|x\rangle$, therefore they have to sum to 1.

The subsets of the groups of qubits have no states, unless all qubits are in the computational basis states. The groups of qubits are called quantum registers.

Quantum Parallelism

A quantum register can be in a superposition of states. If such register is applied as an input argument of a function, then the result of this function is a superposition of output values. In simple terms, a quantum computer calculates the results of a function for many input values in a single step. Quantum parallelism makes quantum computers very effective in resolving certain problems.

Measurement of quantum state

The state of a classical bit can be learned by examining it. However, it is impossible to determine the state of a qubit, that is the values of α and β. To get any useful information about the qubit, it has to be measured. The measurements yields the value 0 with probability $|\alpha|^2$ or the value 1 with probability $|\beta|^2$. After the measurement the qubit is left in a state corresponding to the result of measurement — either a $|0\rangle$ or $|1\rangle$.

The measurement of the whole quantum register yields one of the 2^n computational basis states $|x\rangle_n$ with probability $|\alpha_x|^2$. It is also possible to measure a subset of qubits. The two qubit quantum register has four computational basis states: $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$. The state of such register is described as follows:

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

For example, if measurement of the first qubit gives a value 0, then the state of the quantum register becomes:

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

Quantum entanglement

There are states of quantum registers for which the state of one qubit depends on the other — the qubits are entangled. An example for two qubit register is the state:

$$|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

The measurement of both qubits gives $|00\rangle$ or $|11\rangle$ with probabilities $\frac{1}{2}$. However, if only one of the qubits is measured it also sets the state of the second qubit to the same value, since there is no other possibility.

The state presented above is one of the four Bell states:

$$|\beta_{00}\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

$$|\beta_{01}\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}}$$

$$|\beta_{10}\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}}$$

$$|\beta_{11}\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}}$$

No-cloning theorem

The no-cloning theorem states that the copy of unknown quantum state cannot be created. That means there is no unitary transformation that can transform the state $|\psi\rangle|0\rangle$ into the state $|\psi\rangle|\psi\rangle$ for an arbitrary unknown $|\psi\rangle$.

2.3 Operations on quantum bits

The state of a qubit can be manipulated using quantum gates. The gate can be represented by two by two matrix. The only restriction on the matrix is that after the manipulation, the state of the qubit have to satisfy the normalization condition, that is $|\alpha|^2+|\beta|^2 = 1$. It is true when the gate matrix is unitary, that is $U^\dagger U = I$, where I is the identity matrix. All operations on a qubit, represented by the unitary transformation, are reversible. The hermitian conjugate $U^\dagger$ is the matrix of an operation opposite to U.

Application of a quantum gate is simply multiplying a vector by a matrix. Since a ket $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ is a vector, it can be represented in vector notation as:

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

The unitary matrix U is defined as follows:

$$U = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

The application of a gate U to a qubit in the state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ is then the multiplication:

$$U\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} a\alpha + b\beta \\ c\alpha + d\beta \end{bmatrix}$$

The state of the qubit after computation is $|\psi'\rangle = (a\alpha + b\beta)|0\rangle + (c\alpha + d\beta)|1\rangle$.

A quantum gate can also affect the state of a group of qubits. The gate matrix is then respectively larger, for example four by four for a 2-qubit register.

In the following Sections we introduce the most important quantum gates from the point of view of this thesis.

2.3.1 NOT Gate

A NOT is single qubit negation gate. It changes the state of a qubit to the opposite. The gate matrix is:

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

The graphical symbols of the gate is presented in Figure 2.1. The symbol in Figure (b) is more popular.

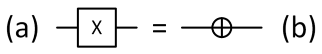


FIGURE 2.1: The symbols of NOT gate. The version in Figure (b) is more frequently used.

2.3.2 Controlled-NOT Gate

A Controlled-NOT (c-CNOT) gate operates on two qubits. One is the control qubit and the second is the target qubit. The target qubit's state is negated depending on the state of the control qubit. The state of a control qubit is left unchanged. The matrix of CNOT gate is:

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

Figure 2.2 shows the symbols of c-NOT gate. They can be used interchangeably, however the symbol in Figure (b) is more frequent.

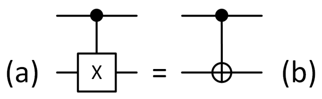


FIGURE 2.2: The symbols of Controlled-NOT gate. The version in Figure (b) is more frequently used.

2.3.3 Toffoli Gate

a Toffoli gate is a doubly controlled NOT. It acts on three qubits - two controls and one target. Only the state of a target qubit is flipped depending on the states of both control qubits. The gate matrix is:

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

Figure 2.3 presents the two equivalent symbols of Toffoli gate. The symbol in Figure (b) is used more often.

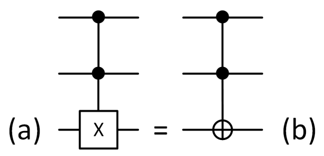


FIGURE 2.3: The symbols of Toffoli gate. The version in Figure (b) is more frequently used.

A NOT gate with any number of control qubits can be constructed out of Controlled-NOT and Toffoli gates.

2.3.4 Hadamard Gate

A Hadamard gate puts a qubit, originally in the state $|0\rangle$, into equally probable superposition of the $|0\rangle$ and $|1\rangle$ states. It is a very important gate, for example for constructing the Bell states or for the Quantum Fourier Transform. The gate matrix is:

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

The symbol of Hadamard gate is presented in Figure 2.4.

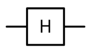


FIGURE 2.4: The symbols of Hadamard gate.

2.3.5 Walsh-Hadamard Gate

A Walsh-Hadamard is a multi-qubit gate which applies the Hadamard transform to each qubit. Figure 2.5 (a) presents the gate symbol of Walsh-Hadamard gate. Figure (b) shows the internal implementation of the gate.

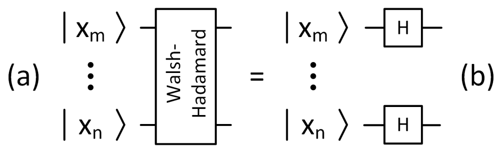


FIGURE 2.5: The symbol of Walsh-Hadamard gate. Figure (b) presents the internal implementation of the gate.

2.3.6 Phase Kick Gate

A phase kick gate acts on two qubits - a target and control. However, in this gate the target and control can be swapped without a change in behavior. The gate has the parameter k which can be interpreted as the distance between the target and control qubit in the quantum register, that is $k = m - n$. The matrix of phase kick gate is:

$$R_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{2\pi i/2^k} \end{bmatrix}$$

Figure 2.6 presents the symbol of a phase kick gate. The $|x_m\rangle$ is m-th qubit of quantum register, while $|x_n\rangle$ is n-th qubit. The parameter k is the distance between the qubits, $k = m - n$. Symbols on Figures (a) and (b) have the same behavior and can be used interchangeably.

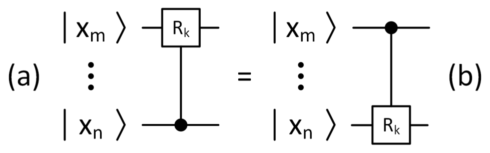


FIGURE 2.6: The symbols of Phase Kick gate. The symbols in Figures (a) and (b) have the same result.

2.3.7 Measurement Gate

A measurement gate performs the measurement of the qubit's state. It leaves the qubit in the state corresponding to the result. Figure 2.7 presents the measurement gate symbol.

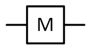


FIGURE 2.7: The symbols of Measurement gate.

2.4 Quantum circuits

Classical computers are built of the circuits with logic gates. By analogy quantum computations can be described by the circuits built of qubits and quantum gates.

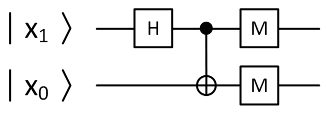


FIGURE 2.8: A circuit for the Bell state β_{00} followed by the measurement

Figure 2.8 shows circuit for one of the Bell states — β_{00} — followed by the measurement. The kets $|x_0\rangle$ and $|x_1\rangle$ on the left represent the initial states of the qubits. The horizontal lines are the "wires" of each qubit. Quantum circuits are read from left to right. Every quantum gate changes the state according to its matrix. All operation except the measurements are reversible, thus it is possible to step backward while evaluating the circuit. The state after the measurement gates is set to the result of the measurement.

2.5 Summary

In this Chapter we presented informations about the quantum computations essential to understand the thesis. We described the features and behavior of quantum bits. We introduced the notation used to describe the quantum states - the Dirac notation. We described the quantum gates used throughout the thesis and presented their symbols. At the end of the Chapter we presented how to construct and read quantum circuits.

Chapter 3

Quantum Cryptology

This Chapter introduces quantum protocols and algorithms in cryptography. Section 3.1 describes Quantum Key Distribution protocols for secure distribution of secret cryptographic keys. Section 3.2 introduces the Quantum Commitment protocol which enables to take the binary decision but reveal it later. In Section 3.3 we describe RSA cryptosystem and show how Shor's Factoring Algorithm makes it easily breakable. Section 3.4 summarizes and concludes concepts presented in this Chapter.

3.1 Quantum Key Distribution

Symmetric cryptography is nowadays the most widespread way of ensuring privacy of communication over insecure channels [30]. One of the common problems of symmetric cryptography is the distribution of the secret key. In order to establish secure communication channel, communicating parties first have to agree upon the key, which in symmetric cryptography is the same for encrypting and decrypting. In a real-world environment it is usually not possible to exchange a key over a classical public channel in a completely secure way. Nevertheless by exploiting some of the quantum mechanics principles it is possible to establish a secure key distribution channel. Quantum Key Distribution is the only known physically secure method for exchanging a key between two distant communicating parties in the presence of an eavesdropper [32].

Quantum key distribution protocols can be divided into two groups - Prepare and Measure Protocols and Entanglement-based Protocols.

3.1.1 Prepare and Measure Protocols (Single-photon)

Prepare and Measure Protocols use single qubits in pure states. Most commercial applications use photons as information carriers. Typically, photons are transmitted over optical fiber. The most popular Prepare and Measure protocol is BB84.

BB84 Protocol

BB84 is the first Quantum Key Distribution Protocol proposed in 1984 by Charles H. Bennet and Gilles Brassard [15]. It exploits the uncertainty principle and no-cloning theorem to ensure that the transmission of the key have not been eavesdropped or altered.

The quantum channel does not convey encrypted data. It transfers random bits which form the secret key. The data is encrypted with some classical algorithm, such as one-time pad, using this key and sent over a classical public channel.

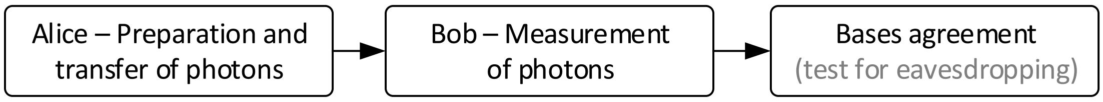

Alice – Preparation and transfer of photons
Bob – Measurement of photons
Bases agreement (test for eavesdropping)

Figure 3.1: Outline of key agreement procedure in BB84 protocol.

The establishment of a secure key in BB84 protocol can be divided into three parts summarized in Figure 3.1. At first, Alice prepares photons and sends them to Bob. Next, Bob measures photons and store measurement results. At the end there is bases agreement procedure and optional test for eavesdropping.

Table 3.1: Photon's spin orientation angles in BB84 protocol

		Base			
		✛	Rectilinear	✕	Diagonal
Value	0	↔	0°	↗	45°
	1	↕	90°	↖	135°

Figure 3.2 depicts the process of preparation, transfer and measurement of photons. At the beginning, Alice chooses random values (0 or 1) and random bases (rectilinear or diagonal). Then she prepares photons with spin orientation according to randomly chosen values and bases. Table 3.1 shows photon's spin orientation angles according

as chosen base (rectilinear or diagonal) and value (0 or 1). Arrows shows graphical representation of spin's angle.

Photons are sent to Bob over a quantum communication channel. When Bob receives a photon, he randomly chooses the measurement basis. He measures the photon in the basis of his choice and stores the result along with the base used for measurement.

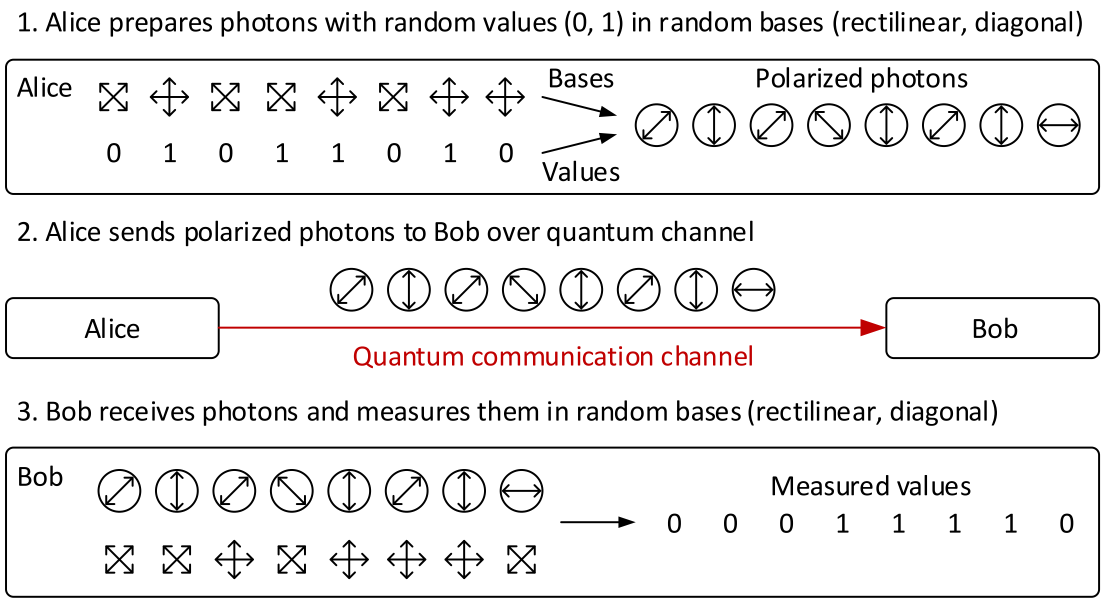

1. Alice prepares photons with random values (0, 1) in random bases (rectilinear, diagonal)
Alice
0 1 0 1 1 0 1 0
Bases
Values
Polarized photons
2. Alice sends polarized photons to Bob over quantum channel
Alice
Bob
Quantum communication channel
3. Bob receives photons and measures them in random bases (rectilinear, diagonal)
Bob
Measured values
0 0 0 1 1 1 1 0

FIGURE 3.2: Preparation, transfer and measurement of photons in BB84 protocol.

After sufficient number of values have been transfered, the phase of bases agreement begins. It is shown in figure 3.3. The discussion of bases is held over an insecure classical communication channel. Bob reports to Alice the base he chose for measuring each photon. He also states which photons were lost during transmission. Alice informs Bob which bases were correct. They both remove the values for which bases were not the same. At this point Alice and Bob have agreed which bits are known to both of them - these bits are the candidate for the key.

Table 3.2 shows example key agreement procedure where eight photons are sent. The Table is divided in three Sections. The first depicts the preparation of photons. Alice randomly chooses bases and values. Each random value is encoded in the photon's spin in respective basis, as show in Table 3.1. Photons with such spin orientation are sent to Bob. Second Section illustrates what Bob does to extract values from photons. First of all, he chooses random base for measuring each photon. Then Bob measures the photon in this random basis. After getting the value of each photon, there is a bases agreement procedure. Alice and Bob exchange information about their bases for each photon and discard values for which bases did not comply. After such procedure Alice and Bob have the same value of the key, that is 0111.

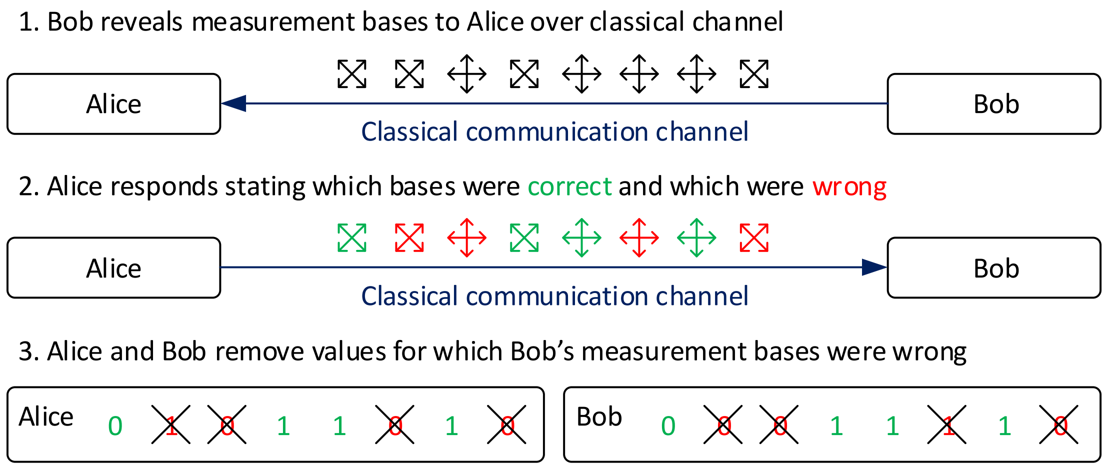

1. Bob reveals measurement bases to Alice over classical channel
Alice
Bob
Classical communication channel
2. Alice responds stating which bases were correct and which were wrong
Alice
Bob
Classical communication channel
3. Alice and Bob remove values for which Bob's measurement bases were wrong
Alice 0 1 0 1 1 0 1 0
Bob 0 0 0 1 1 1 1 0

FIGURE 3.3: Bases agreement procedure in BB84 protocol.

TABLE 3.2: Example key agreement procedure in BB84 protocol - no eavesdropping

Alice – preparation of photons								
Random basis	⤧	✛	⤧	⤧	✛	⤧	✛	✛
Random value	0	1	0	1	1	0	1	0
Photon's spin orientation	⤢	↕	⤢	⤡	↕	⤢	↕	↔
Bob – measurement of photons								
Random basis	⤧	⤧	✛	⤧	✛	✛	✛	⤧
Measured value	0	0	0	1	1	1	1	0
Bases agreement procedure								
Compliant bases	⤧	⤧	✛	⤧	✛	✛	✛	⤧
Agreed key	0			1	1		1	

Since photons were sent over an insecure channel they may have been eavesdropped or manipulated. To check for eavesdropping Bob choose random subset of key bits (usually one third of them is enough) and reveals them to Alice. Alice confirms whether she has the same values. If any of the bits vary the transmission may have been eavesdropped or altered, therefore it needs to be repeated. If all the test bits are confirmed, the remaining bits can be used as the key.

Despite the fact that all communication takes place over channels prone to eavesdropping, the protocol is still secure. Due to the no-cloning theorem photons cannot be copied in order to measure the copy and leave the original photon intact. If eavesdropper Eve wants to reveal useful information from the photon she has to measure it in the basis of her choice. If she happens to choose the correct base there is no way Alice and Bob will notice. But if she chooses an incorrect basis (which happens in half the cases) Alice and Bob will not agree upon bit value, knowing the transmission was eavesdropped. This feature of quantum key distribution ensures communicating parties that the secret key was not compromised. The public discussion of the measurement basis also does not compromise the key because knowledge of the basis after all the photons were measured is not useful to Eve.

Nevertheless, technological imperfection can compromise the protocol. In currently available physical realizations usually a weak laser pulse is used as a photon source. As such a source does not deterministically emit one photon per pulse, the protocol is prone to Photon Number Splitting attacks [32]. Under certain conditions, Eve is able to block single photon pulses and save one photon from multi photon pulses in her quantum memory. Since all photons from one pulse are polarized in the same basis, Eve can wait until basis agreement between Bob and Alice and then measure her memorized photons in correct basis, revealing the key.

3.1.2 Entanglement-based Protocols

Entanglement-based Protocols use pairs of entangled qubits. These protocols are more sophisticated but they are of less practical importance, since it is not yet possible to transmit entangled particles [33]. The E91 protocol best demonstrates the features of Entanglement-based protocols.

E91 Protocol

E91 is a protocol proposed by Artur Ekert in 1991 [16]. It is similar to BB84 with the difference that it assumes the existence of the entangled particles source. This source can be in possession of Alice, Bob or any other trusted party.

Figure 3.4 depicts the process of key agreement in the E91 protocol. Entangled particles are emitted regularly. One of the coupled particles is transmitted to Alice and the other one to Bob. After both particles reach their destination, Alice and Bob independently and randomly choose the basis and perform measurement. They store the results of the measurements. After reaching a sufficient number of measurements they announce the basis of every measurement to each other. They also exchange the measurements results in which their random bases did not match. They use these results and exploit the generalized Bell' s theorem to ensure the transmission was not eavesdropped or altered [16]. This procedure leads them to agree upon a secret key formed with the measurement results they took in the same basis.

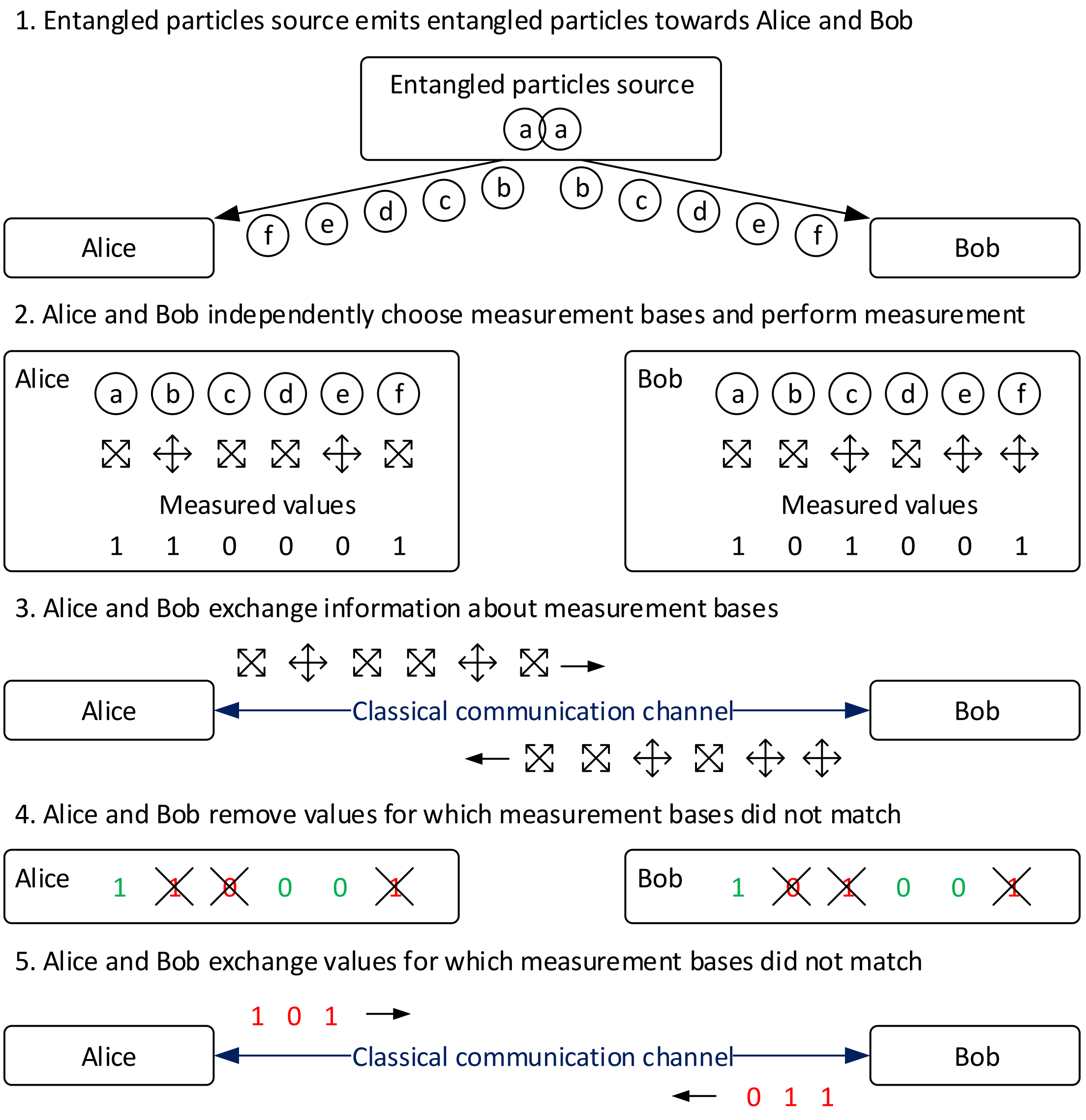

1. Entangled particles source emits entangled particles towards Alice and Bob
Entangled particles source
a a
b c d e f
Alice
Bob
2. Alice and Bob independently choose measurement bases and perform measurement
Alice a b c d e f
Measured values
1 1 0 0 0 1
Bob a b c d e f
Measured values
1 0 1 0 0 1
3. Alice and Bob exchange information about measurement bases
Classical communication channel
4. Alice and Bob remove values for which measurement bases did not match
Alice 1 1 0 0 0 1
Bob 1 0 1 0 0 1
5. Alice and Bob exchange values for which measurement bases did not match
1 0 1
Classical communication channel
0 1 1

FIGURE 3.4: Secret key agreement procedure in BB84 protocol.

Entanglement based protocols are considered secure. Eavesdropper Eve cannot obtain any useful information from the transmitted particle because it does not carry any meaningful information. The information appears after the measurements by legitimate parties are taken. Eve may also try to substitute the entangled particles source. But since she knows nothing about the measurement bases there is no way she can escape being detected.

3.2 Quantum Commitment

In certain situations there is a need for a mechanism which enables one party to make a decision, without revealing it before specific time, in a way that it is impossible to change the decision. It can be compared to sending locked box with a message and sending the key later. While it is possible to break into the box, quantum commitment secures the message from being revealed prematurely.

Quantum commitment protocol was first implicitly proposed by Charles H. Bennet and Gilles Brassard in 1984 [15] and it was redefined in 1993 by Gilles Brassard, Claude Crépeau, Richard Jozsa and Denis Langlois [34].

Suppose one party, Bob, wants another party, Alice to make some binary decision before a specific date. However the decision should not be revealed before some later date. Bob wants to be sure that the decision has been taken and that it has not been altered before it was revealed.

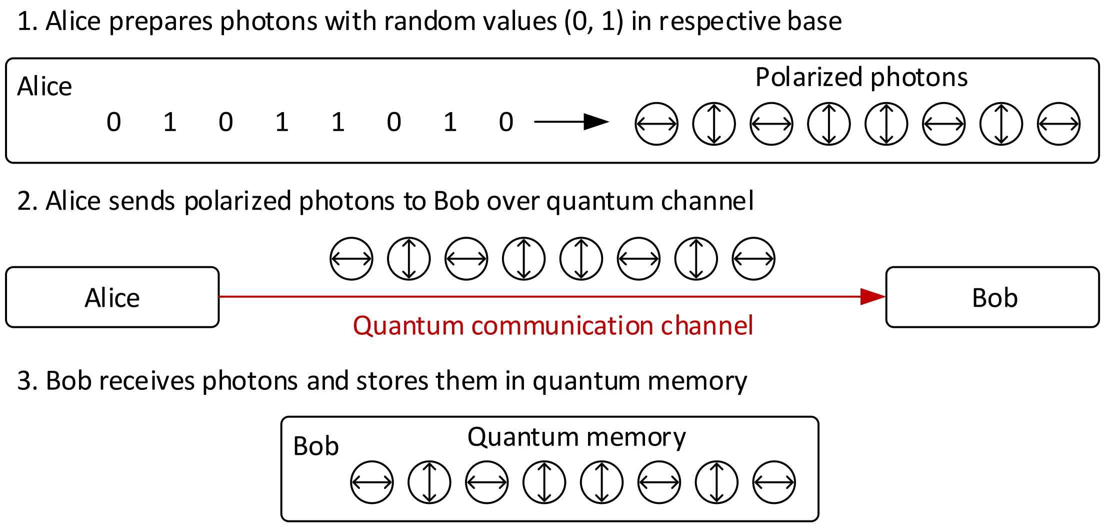

1. Alice prepares photons with random values (0, 1) in respective base
Alice
Polarized photons
0 1 0 1 1 0 1 0
2. Alice sends polarized photons to Bob over quantum channel
Alice
Bob
Quantum communication channel
3. Bob receives photons and stores them in quantum memory
Bob
Quantum memory

FIGURE 3.5: The example of procedure for committing the decision in Quantum Commitment protocol for answer YES.

Figure 3.5 shows the process of committing a decision for answer YES. For completeness, photon spins for both answers, YES and NO, are presented in Table 3.3. To commit the decision, Alice prepares a large number of photons in random state (0 or 1) in rectilinear basis if her answer is YES or in diagonal basis if her answer is NO. The angles of spins in both bases are presented in Table 3.1. Then she transfers photons to Bob. Bob stores them in quantum memory.

The revealing phase of the protocol is depicted in Figure 3.6. Bob has to wait for Alice to announce the basis she chose and her random values. Then he measures photons in

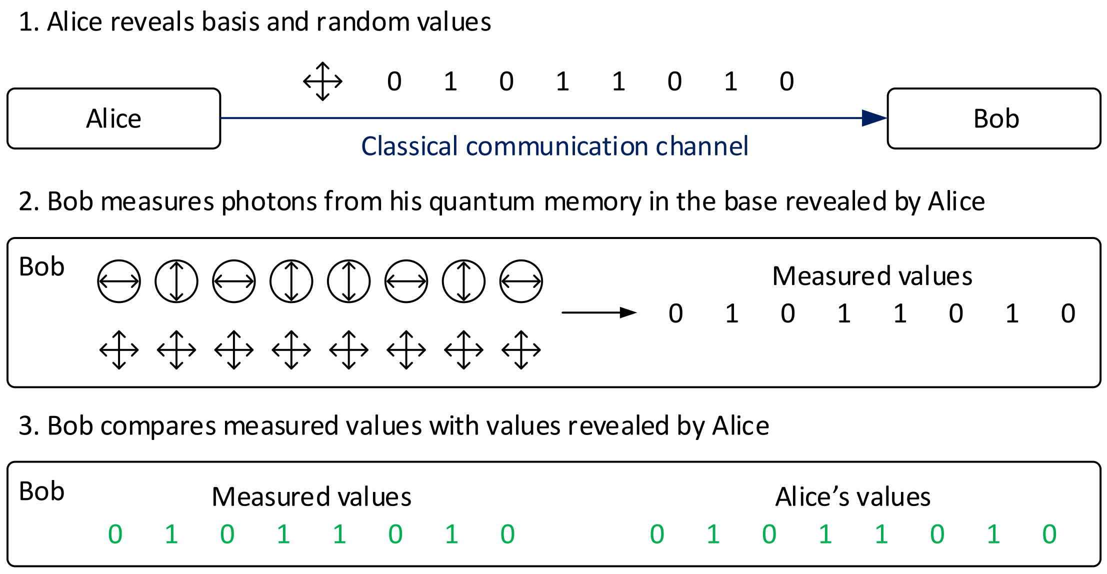

1. Alice reveals basis and random values
0 1 0 1 1 0 1 0
Alice
Bob
Classical communication channel
2. Bob measures photons from his quantum memory in the base revealed by Alice
Bob
Measured values
0 1 0 1 1 0 1 0
3. Bob compares measured values with values revealed by Alice
Bob
Measured values
0 1 0 1 1 0 1 0
Alice's values
0 1 0 1 1 0 1 0

FIGURE 3.6: The example of procedure for revealing the decision in Quantum Commitment protocol for answer YES.

the correct basis and compares the results with Alice's values. If these values comply, Bob can be sure that the decision was not altered.

If Bob tries to reveal the decision prematurely he can do no better than to randomly choose the basis and perform measurement. But it does not provide him with any useful information because such a measurement yields a random string of 0's and 1's. Furthermore, if Bob chose the wrong basis his values would not comply with Alice's encoded values.

Quantum Commitment was however proved to be unsure [35]. Alice can use entangled

TABLE 3.3: Example photons spins for answer YES and answer NO in Quantum Commitment protocol.

Answer YES – rectilinear basis								
Random value	0	1	0	1	1	0	1	0
Photon's spin orientation	↔	↕	↔	↕	↕	↔	↕	↔

Answer NO – diagonal basis								
Random value	0	1	0	1	1	0	1	0
Photon's spin orientation	⤢	⤡	⤢	⤡	⤡	⤢	⤡	⤢

particles and store one of the coupled particles for herself and send the other to Bob. If she wants to change her decision, she performs a measurement on her particles with the basis of her choice. She then reports this basis to Bob along with the results of the measurements convincing him that it was her original decision.

3.3 Breaking RSA cryptosystem

In Section 3.1 we discussed how quantum mechanics can help exchange a secret key for symmetric cryptography. In this Section we show the opposite - how quantum computation can help to break public-key cryptography. First we introduce the RSA cryptosystem and then show how Shor's Factoring Algorithm can be used to break it [9][8][33].

RSA cryptosystem is widely used for securing data transmission. It is public-key cryptosystem, which means that there are two keys - a public key and a private key. The public key is used for encryption and is given to all parties who want to encrypt messages. The private key is used for decryption and it has to be kept secret by the legitimate recipient of encrypted messages.

Figure 3.7 presents the process of encrypting, transferring and decrypting a message in RSA cryptosystem. If Alice wants to send a message to Bob, she has to obtain the public key from Bob. Bob can either use a previously generated key pair or generate a new one. He sends the public key to Alice, but he keeps the private key to himself. Alice encodes her message with Bob's public key and sends her encrypted message to Bob. Bob uses his private key to decode Alice's message. The public key can be reused by Alice to encrypt further messages for Bob.

In order to understand why quantum computation can help break the cryptosystem, we have to analyze what the private and public key consist of. Figure 3.8 illustrates the process of key pair generation. When Bob wants to generate the public-private key pair, he has to choose two large prime numbers p and q and calculate their product $N = pq$. Then Bob chooses a coding number c that has no factors in common with $(p-1)(q-1)$. The pair (N, c) forms the public key. To generate the private key, Bob has to compute a decoding number d which is the multiplicative inverse of $c \bmod (p-1)(q-1)$, that is $cd \equiv 1 \bmod (p-1)(q-1)$. The pair (N, d) is the private key. Figure 3.9 shows the contents of the private and the public key.

Equations 3.1 and 3.2 describe the encryption using the public key (N, c) and decryption using the private key (N, d). To encrypt the message, Alice represents it as number a

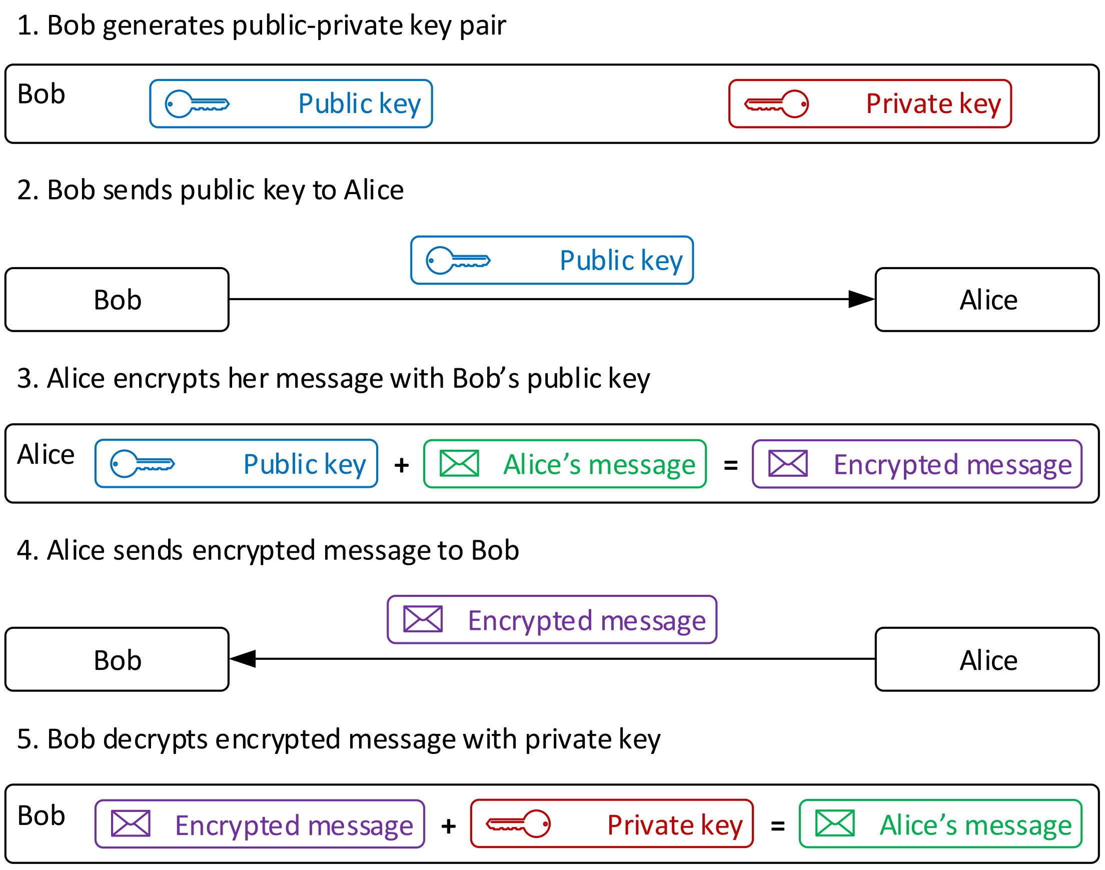

1. Bob generates public-private key pair
Bob
Public key
Private key
2. Bob sends public key to Alice
Public key
Bob
Alice
3. Alice encrypts her message with Bob's public key
Alice
Public key
+
Alice's message
=
Encrypted message
4. Alice sends encrypted message to Bob
Encrypted message
Bob
Alice
5. Bob decrypts encrypted message with private key
Bob
Encrypted message
+
Private key
=
Alice's message

FIGURE 3.7: The process of exchanging encrypted message in RSA cryptosystem.

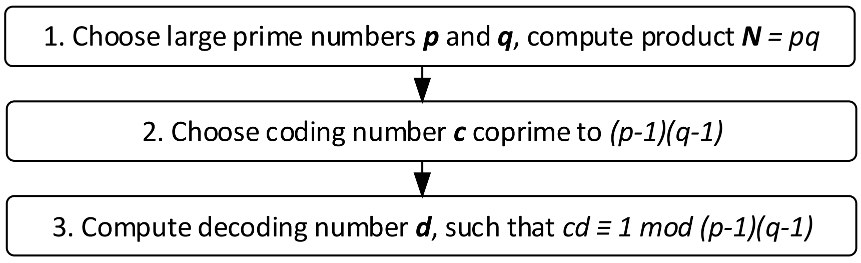

1. Choose large prime numbers ***p*** and ***q***, compute product ***N*** = *pq*
2. Choose coding number ***c*** coprime to *(p-1)(q-1)*
3. Compute decoding number ***d***, such that *cd ≡ 1 mod (p-1)(q-1)*

FIGURE 3.8: The sequence of private-public key pair generation.

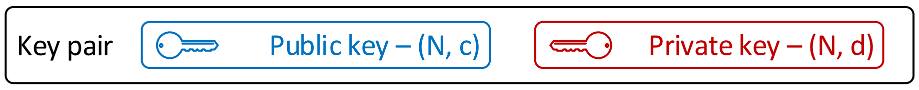


FIGURE 3.9: The contents of the key pair with respect to private and public key.

less than N. If her message is bigger than N she has to split it in the pieces less than N and encrypt them separately. To obtain the encrypted message b she computes:

$$b = a^c \bmod N \tag{3.1}$$

To decrypt the message b, Bob exploits his knowledge of private key and computes:

$$a = b^d \bmod N \tag{3.2}$$

At this point it is important to summarize which values have to be kept secret and which can be made public. The public key consists of N and c so these are not secret values. Decoding number d is the part of the private key so it has to be kept secret. Since $cd \equiv 1 \bmod (p-1)(q-1)$, then having c, p and q it is possible to compute d. Therefore values of p and q also have to be secret. Table 3.4 sums up secret and non-secret values.

Table 3.4: Secret and non-secret values in RSA cryptosystem.

Secret values	Non-secret values
• Prime numbers *p* and *q* • Decoding number *d*	• *N = pq* • Coding number *c*

As stated before, having separate values of p and q enables one to easily compute the secret decoding number d. The reason why $N = pq$ can be made public is that RSA cryptosystem assumes computational difficulty of factoring into primes. It is true when classical computers are taken into consideration. However, quantum computers with Shor's Factoring Algorithm may be able to significantly speed up factoring.

In Figure 3.10 we present why this poses a threat of compromising RSA cryptosystem. If eavesdropper Eve learns public key (N, c) and she is in possession of a quantum computer, she can use Shor's algorithm to factor N into separate values of p and q. Knowing p, q, and c she can compute d the same way Bob does when generating the key pair. That way, Eve obtains private key (N, d) which she can use to decrypt messages sent to Bob.

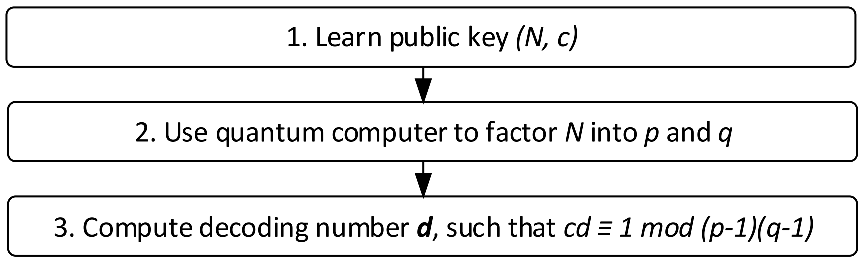

1. Learn public key *(N, c)*
2. Use quantum computer to factor *N* into *p* and *q*
3. Compute decoding number ***d***, such that *cd ≡ 1 mod (p-1)(q-1)*

Figure 3.10: The sequence of generating private key from public key using quantum computer.

In this Section we showed how a quantum computer with Shor's Algorithm can lead to breaking RSA cryptosystem. The quantum algorithm for factoring is described in detail in Chapter 4.

3.4 Summary

In this Chapter we have discussed several quantum cryptology concepts. We described how Quantum Key Distribution can be used to exchange a secret key for symmetric cryptography algorithms. We also showed how Quantum Commitment enables to commit and reveal binary decisions. At the end we presented why quantum computation can pose a threat to RSA cryptosystem.

Quantum Key Distribution and Quantum Commitment are protocols, which are of significant importance in the field of Telecommunication. On the other hand, Shor's Factoring Algorithm is interesting from the computational point of view. In this thesis we focus on Shor's Algorithm and its relationship with breaking RSA cryptosystem.

In Chapter 4 we present a detailed description of Shor's Algorithm. We also discuss different quantum circuit implementations. In Chapter 5 we present simulation results of two implementation variants of Shor's Algorithm.

Chapter 4

Shor's Factoring Algorithm

This Chapter describes Shor's Factoring Algorithm and its implementation variants. In Section 4.1 we introduce the fundamental parts of the algorithm - Classical Preprocessing, Quantum Order Finding and Classical Postprocessing. Section 4.2 describes the Quantum Order Finding and introduces its parts. In Section 4.3 we provide standard quantum circuits for Quantum Order Finding. Section 4.4 describes optimization variants of these circuits. In Section 4.5 we summarize the circuits variants presented in this Chapter.

4.1 Outline of the Algorithm

In Chapter 3 we presented why efficient factoring can lead to breaking RSA cryptosystem. As a reminder, the public key consist of $N = pq$, where p and q are large primes, and coding number c coprime to $(p-1)(q-1)$. The private key is the pair $N = pq$ and decoding number d, such that $cd \equiv 1 \bmod (p-1)(q-1)$. Values of N and c can be announced in public, however d, p and q have to be kept secret.

Knowing N and c does not lead to compute d easily, because to do so one has to factor N into prime numbers p and q. It turns out that factoring into primes is computationally hard on classical computers [36][37].

However, in 1994 Peter W. Shor proposed a quantum algorithm for prime factorization [8]. It enables to one factor numbers into primes in polynomial time on a quantum computer.

The algorithm exploits mathematical theorems which reduce the problem of factorization to finding order r of an element x in the multiplicative group $\bmod N$ – the least integer such that $x^r \equiv 1 \bmod N$. Shor described an efficient subroutine, which enables one to find such an order in polynomial time on a quantum computer.

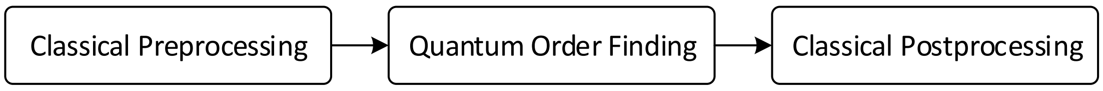


FIGURE 4.1: The Shor's Factoring Algorithm consist of three parts: Classical Preprocessing, Quantum Order Finding and Classical Postprocessing.

Shor's Algorithm consist of three parts, classical preprocessing, quantum order finding and classical postprocessing – classical parts are executed on a digital computer [8][33][38]. This is shown in Figure 4.1. The motivation for such division is that classical parts can be executed on classical computers more efficiently.

Although Shor's algorithm can be used to factor numbers in general, in this thesis we discuss the case relevant to RSA cryptosystem, that is factoring N into prime numbers p and q.

4.1.1 Classical Preprocessing

Classical preprocessing prepares some values for Quantum Order Finding. First of all, random element x of multiplicative group $\bmod N$ is chosen. Also, L, the number of bits of N is computed.

It is described in following steps:

1. Choose random positive number x less than N
2. Check using the Euclidean algorithm whether x is coprime to N. If they are not coprime then x is either p or q.
3. Compute the number of bits L necessary to store N, $L = \lceil \log_2 N \rceil$

4.1.2 Quantum Order Finding

The Quantum Order Finding subroutine is the only part of Shor's Algorithm executed on a quantum computer. Order finding does not compute the exact value of r. It yields value y, from which the order is extracted in postprocessing. It consists of several steps - Register Preparation, Quantum Modular Exponentiation and Quantum Fourier Transform (QFT). Quantum Order Finding is thoroughly described in Section 4.2.

4.1.3 Classical Postprocessing

Order finding does not compute the exact value of the order r. It has to be extracted using the Continued Fractions algorithm. This is done efficiently on a classical computer. If r happens to be the correct value of an order it is possible to compute the values of p and q.

Classical Postprocessing can be described as follows:

1. Reverse the bit order of y. Quantum Fourier Transform reverses bit order so it has to be brought back to its initial order.
2. Apply continued fractions algorithm to $y/2^{2L}$ in order to extract period candidate r_0.
3. Check whether r_0 is the correct period by verifying if $x^{r_0} \equiv 1 \bmod N$. If it does not, try several low integer multiples of r_0, namely $2r_0, 3r_0, 4r_0, \ldots$. If no order is found restart the algorithm from the beginning of classical preprocessing.
4. Check whether order r is even and whether $x^{r/2} + 1 \not\equiv 0 \bmod N$. If not restart the algorithm.
5. Compute the greatest common divisors $\gcd(x^{r/2} - 1, N)$ and $\gcd(x^{r/2} + 1, N)$. These are the values of p and q

4.2 Quantum Order Finding

Quantum Order Finding is the only quantum computational part of Shor's Algorithm. It is also the most important part since it enables to factor numbers into primes in polynomial time.

In contrast to its name, order finding does not find the exact value of an order. It yields the value from which order can be computed in classical postprocessing.

Order finding, followed by postprocessing, enables to find order r of an element x in the multiplicative group $\bmod N$ – the least integer such that $x^r \equiv 1 \bmod N$. Such order can be interpreted as the period of the periodic function $x^a \bmod N$, where a is an integer. This function is called Modular Exponentiation. Figure 4.2 shows an example graph of function $4^a \bmod 55$. It can be noticed that this function has period $r = 10$.

First step for finding the order is to put the input register a in equally probable superposition of all states this register is able to keep. Let's illustrate it on an example.

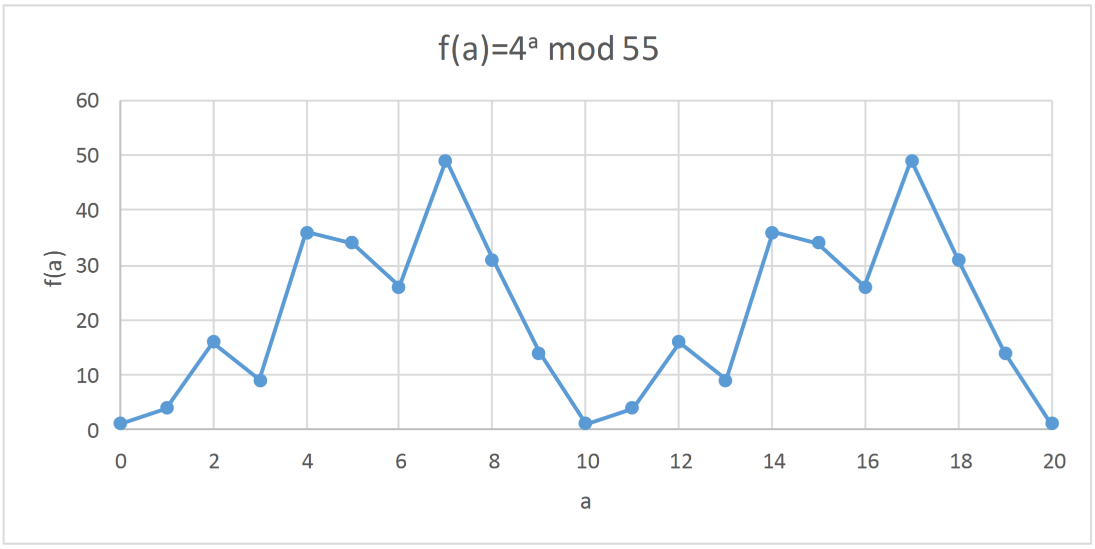


FIGURE 4.2: An example graph of a function x^a mod N, where $x = 4$ and $N = 55$.

Table 4.1 presents 2-qubit register in superposition of states. It can be imagined that this register is in all possible states – 0, 1, 2, 3 – at the same time. The probability 1/4 of each state means that performing measurement on that register would yield one of its states with such probability.

TABLE 4.1: The superposition of states in a 2-qubit quantum register.

Value		
Decimal	Binary	Probability
0	00	¼
1	01	¼
2	10	¼
3	11	¼

If such superposition of states is applied as an exponent a in expression x^a mod N, then we immediately get expression values for every a – in our example 0, 1, 2, 3. It is depicted in Table 4.2. Please note that it is not relevant to breaking RSA since $N = 3$. Its aim is to demonstrate quantum parallelism. When classical value $(x = 2)$ is raised to the power represented by the quantum register in equally probable states $(a = 0, 1, 2, 3)$ then the output register is set to equally probable states $(x^a = 1, 2, 4, 8)$. If we further compute the remainders of division mod$N$ then the register ends up in the state $(1, 2, 1, 2)$. A remark regarding the state of the output register: In Table 4.2 it seems that the probability of states (1,2,1,2) in the output register is 1/4. But it is so only if we take both – input and output – registers into consideration at the same time. If we look only at output register, the state is (1,2) with probabilities 1/2.

TABLE 4.2: An example of modular exponentiation in quantum parallelism. The input register size is 2, N=3, x=2.

a	$x^a=2^a$	x^a mod N	Probability
0	1	1	¼
1	2	2	¼
2	4	1	¼
3	8	2	¼

In this example we can see that the order $r = 2$ because:

$$2^0 \bmod 3 = 2^{0+2} \bmod 3 = 2^{0+r} \bmod 3$$

Moreover, for $r = 2$, $x^r = 2^2 \equiv 1 \bmod 3$.

In Table 4.2 we can see the period of $x^a \bmod N$ with the naked eye. In real quantum computation it is not that obvious. Performing a measurement on the output register would yield one of the values – in our example either 0 or 1 – with equal probability. it does not provide any information about the period. Due to the no-cloning theorem it is also impossible to copy the register state and perform two measurements.

To extract any information about the period it is necessary to apply Quantum Fourier Transform to the register. It enables to obtain a powerful clue about the period with a single measurement of the output register. Together with postprocessing it makes it possible to compute order r.

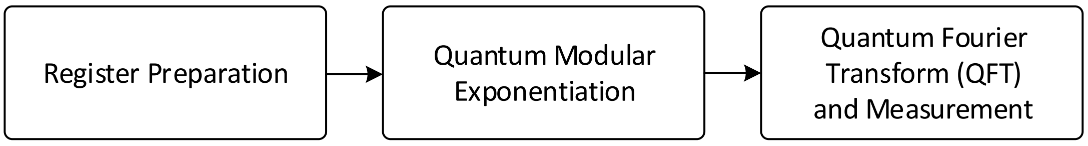

Register Preparation
Quantum Modular Exponentiation
Quantum Fourier Transform (QFT) and Measurement

FIGURE 4.3: An outline of the phases of the Quantum Order Finding subroutine.

Figure 4.3 summarizes the phases of the Order Finding subroutine - register preparation, Quantum Modular Exponentiation and Quantum Fourier Transform with register measurement. These phases are presented in more detail in Figure 4.4.

Figure 4.4 presents fundamental steps of Quantum Order Finding. Computation begins with two registers - input register (INREG) of size $2L$ and output register (OUTREG)

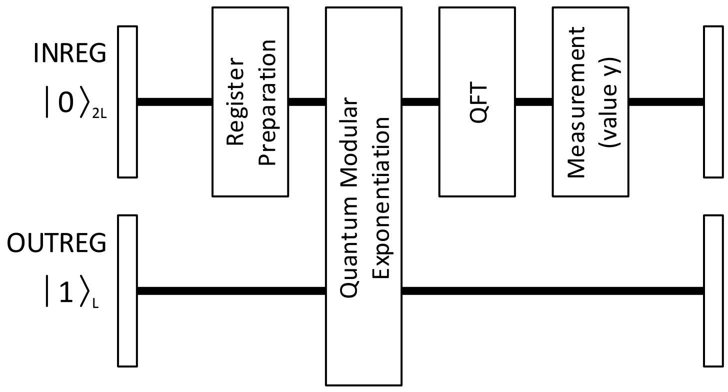


FIGURE 4.4: An overview of the circuit for a Quantum Order Finding

of size L. The first step is to prepare the input register into an equally probable superposition of states. This is done by performing Walsh-Hadamard transform on the register. It is described in Section 4.3.1.

Quantum Modular Exponentiation sets the state of the output register to $x^{|a\rangle} \bmod N$, where $|a\rangle$ is the state of the input register. The actual state of output register is not used in following computations, but computing modular exponentiation entangles it with the input register. Circuit diagrams for modular exponentiation are presented in Section 4.3.2.

The last part of order finding is Quantum Fourier Transform followed by the measurement of the input register. The Fourier transform enables to extract period characteristics form the state of input register. We show circuits for Quantum Fourier Transform and measurement in Sections 4.3.3 and 4.4.2.

4.3 Standard Quantum Circuits Implementations

In this Section we present basic quantum circuits for performing Shor's Algorithm. First of all, we show how to prepare quantum registers. Later we describe standard circuits for Quantum Modular Exponentiation and Quantum Fourier Transform.

4.3.1 Register Preparation

In order to ensure the reversibility of computation, two registers are needed - input register and output register. First of all it is necessary to determine the size of the registers (i.e. the number of qubits). Both of the registers have to be able to store the value of N, that is they must be at least $L = \lceil \log_2 N \rceil$ qubits long. Notwithstanding, the input register has to be $2L$ qubits long to ensure that after computing $x^a \bmod N$, the output register contains at least N full periods [33]. Computation begins with input and output registers respectively in the states:

$$|0\rangle_{2L}\,|1\rangle_{L}$$

The first operation to perform is to evaluate the state of the first register into the superposition of all non-negative integer values less than N. This is done by applying Walsh-Hadamard transform on the input register, that is applying Hadamard gate to each of its qubits. This leaves the registers in the state:

$$\frac{1}{2^{2L/2}}\sum_{a=0}^{2^{2L}-1}|a\rangle\,|1\rangle$$

Registers in such states can now be used to compute modular exponentiation $x^a \mod N$.

Circuit diagram for Walsh-Hadamard transform is presented in Figure 4.5.

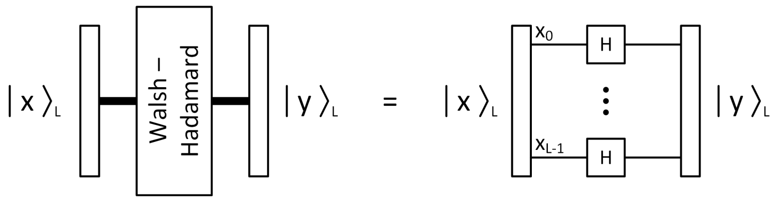


FIGURE 4.5: The circuit diagram for the Walsh-Hadamard transform. A Hadamard transform is applied to each qubit of the register.

4.3.2 Standard Circuit for Modular Exponentiation

Applying quantum modular exponentiation sets the state of the output register, but it retains the state of the input register, since the computation has to be reversible. However, after performing modular exponentiation, state of the input register is entangled with the state of the output register.

After applying modular exponentiation, the state of registers is:

$$\frac{1}{2^{2L/2}}\sum_{a=0}^{2^{2L}-1}|a\rangle\,|x^a \bmod N\rangle$$

By exploiting the quantum parallelism, the output register now holds the value of the modular exponent $x^a \bmod N$ for every a in the input register. Doubled input register size ensures that the output register stores at least N full periods.

Shor has not presented detailed circuits for modular exponentiation [8]. However, he has shown a general idea of computing modular exponent. The first thing important to notice is that we want to compute the exponent of form $x^a \bmod N$ where a is superposition of states in the input register, but x and N are classical values. This means that values of x and N can be built into the structure of the circuit.

In order to efficiently compute modular exponentiation with large exponent, it can be split into the product of smaller exponentiations:

$$x^a \bmod N = x^{\sum a_i 2^i} \bmod N = \prod_i (x^{a_i 2^i} \bmod N) \bmod N$$

where a_i is the i-th bit of binary representation of a. Such a product can be further substituted by the series of modular multiplications. Modular multiplications can be simplified by the series of modular additions. Moreover, modular additions can be built of plain additions.

In this Section we present a circuit which is based on classical adder implemented in a reversible way. This circuit implementation was proposed by Vedral, Barenco and Ekert in 1996 [39]. It implements the modular exponentiation operator $U_{x,N}$, where x and N are predefined classical parameters. It is based on a classical adder implemented in a reversible way. The only quantum gates used in the circuit are NOT, controlled-NOT and Toffoli gates.

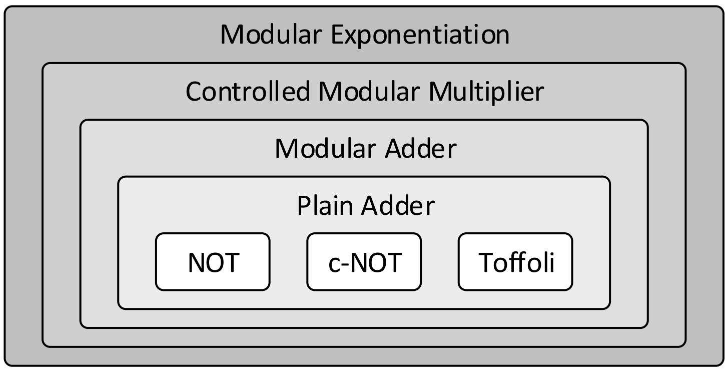


FIGURE 4.6: The hierarchy of composite gates in the circuit for a Modular Exponentiation.

The circuit for $U_{x,N}$ operator follows the idea described by Shor - the substitution with more fundamental operations. Figure 4.6 presents the hierarchy of complex gates. We describe the gates in bottom-up fashion. We begin with describing the basic Plain Adder circuit which uses only NOT, controlled-NOT and Toffoli gates. Then we show more sophisticated circuits for modular addition, controlled modular multiplication and modular exponentiation. Each of them consist of previously defined complex gates. We end up by defining the modular exponentiation operator $U_{x,N}$.

Plain Adder

Let $L = \max\left(\lceil \log_2 a \rceil, \lceil \log_2 b \rceil\right)$ be the number of bits needed to store the operands a, b of the addition $a + b$. The addition of numbers a, b stored in registers $|a\rangle_L \, |b\rangle_{L+1}$ can be expressed as follows:

$$|a\rangle_L \, |b\rangle_{L+1} \longrightarrow |a\rangle_L \, |a + b\rangle_{L+1}$$

The register $|b\rangle$ must be $L+1$ bits long to be able to store the result of $a+b$. To compute the most significant bit of $a + b$, L-qubit helper register is needed to store carry bits. It has to be set to 0 and it is reset to 0 after computation so it can be reused.

$$|a\rangle_L \, |b\rangle_{L+1} \, |0\rangle_L \longrightarrow |a\rangle_L \, |a + b\rangle_{L+1} \, |0\rangle_L$$

Figure 4.7 presents a circuit diagram for a plain adder. At first, the most significant bit b_L of the sum $a + b$ is computed. This is done by computing all carries c_i using the relation:

$$c_i \leftarrow (a_i \text{ AND } b_i) \text{ OR } (b_i \text{ AND } c_i) \text{ OR } (a_i \text{ AND } C_i)$$

The last of such an operation stores the carry bit directly in b_L.

Afterwards, all temporary carry values c_i are uncomputed to restore register c to state $|0\rangle$. This is done using inverse-carry, which is the carry operation executed in reversed order. During this process, all bits b_i of the sum, except b_L, are computed. This is done using relation:

$$b_i \leftarrow (a_i \text{ XOR } c_i) \text{ XOR } b_i$$

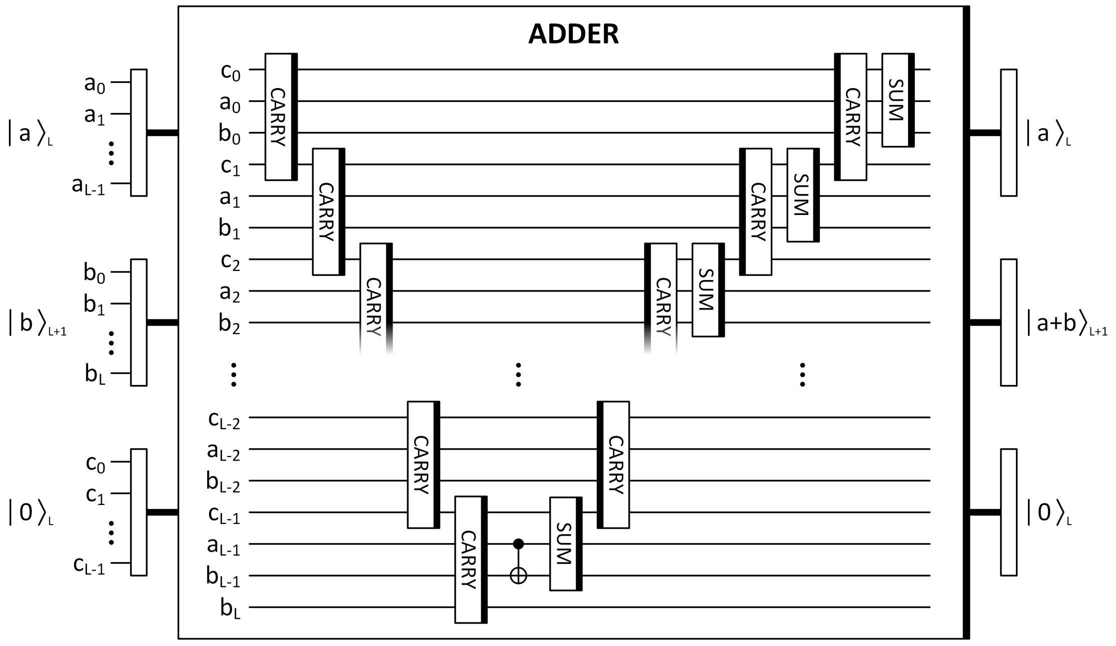


FIGURE 4.7: The circuit representing a Plain Adder.

The circuit diagram for Carry and Sum operations are presented in Figure 4.8. Inverse-Carry is implemented by reversing the order of elementary gates in Carry circuit.

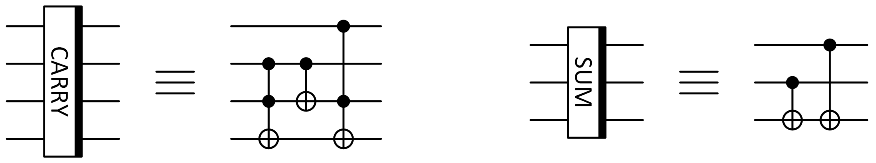


FIGURE 4.8: The circuits for the Carry and Sum operations used in a Plain Adder. The thick bar on the right means that the elementary gates are executed in presented order. If it is on the left it represents reversed order of the circuit.

Figure 4.9 presents the gate symbols for a Plain Adder. In Figure (a) all three registers $|a\rangle_L$, $|b\rangle_{L+1}$ and $|c\rangle_L = |0\rangle_L$ are shown. In Figure (b) the carry register is omitted, but it is still implicitly used during computation. This register can be reused.

If the adder is applied backwards it becomes a subtractor. This is depicted in Figure (c). If the input is $|a\rangle_L |b\rangle_{L+1}$ then output is $|a\rangle_L |a-b\rangle_{L+1}$ for $a \geq b$. If $a < b$, then the result is $|a\rangle_L |2^{L+1} - (b-a)\rangle_{L+1}$. In this case, the most significant bit b_L always contains 1. This means that the reverse-adder can be used not only to subtract two numbers, but also to compare them. This feature will be used in the design of a modular adder.

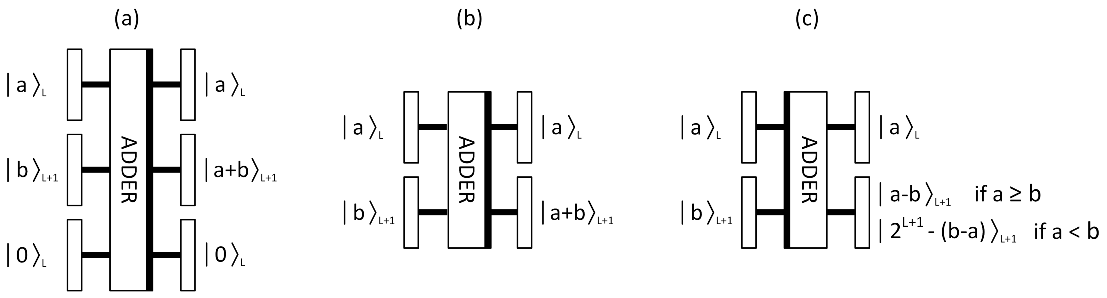


FIGURE 4.9: The gate symbols for a Plain Adder. In (a) the carry register $|0\rangle_L$ is explicitly provided. In (b) the carry register is omitted for readability, however it is still used in the circuit. The (c) presents the adder circuit used in reversed order, which is a subtractor.

Modular Adder

We can express modular addition of numbers a, b modulo N as:

$$|a\rangle_L |b\rangle_{L+1} \longrightarrow |a\rangle_L |a+b \bmod N\rangle_{L+1}$$

for $0 \leq a, b < N$. Now each register has to be able to contain N, therefore $L = \lceil \log_2 N \rceil$

Figure 4.10 presents a circuit for the Modular Adder. At first, $|a\rangle_L$ is added to $|b\rangle_{L+1}$ resulting in state $|a\rangle_L |a+b\rangle_{L+1}$. In the next step N is subtracted from $a+b$. It enables to compare $a+b$ and N. If the most significant qubit of the result is 0 then $a+b \geq N$, if it is 1 then $a+b<N$. Now we have to discuss the behavior of the circuit in this these two cases.

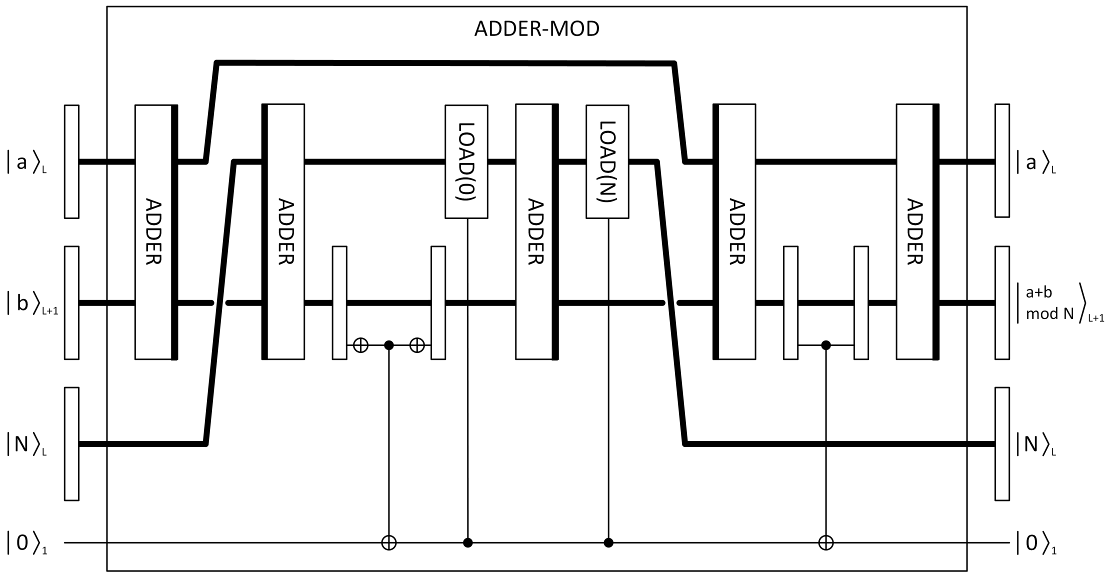


FIGURE 4.10: The circuit for a Modular Adder.

If $a+b \geq N$, then no overflow occurred during the subtraction. It means that the result of subtraction is $|N\rangle_L\,|a+b-N\rangle = |N\rangle_L\,|a+b \bmod N\rangle$. Since the most significant bit of the result is 0, then c-NOT gate sets the state of temporary register $|t\rangle_1 = |0\rangle_1$ to $|1\rangle_1$. Now, controlled-load gate loads value 0 which replaces N in the register. This can be achieved using c-NOTs, since the value of N is known, classical value and the register is in pure state. Value 0 is now added to the result, but it has no influence on the second register. Value N is restored in the first register. Further steps are done to restore the state of temporary register $|t\rangle_1$ to $|0\rangle_1$. Value a is subtracted from the result, resulting in state $|a\rangle_L\,|(a+b \bmod N)-a\rangle_{L+1}$. The most significant qubit of the result can be now used to restore value of $|t\rangle_1$. At the end a is added back to restore the state $|a\rangle_L\,|a+b \bmod N\rangle_{L+1}$

When $a+b<N$, then, due to overflow, the most significant qubit of the result is 1. This means that value of temporary register $|t\rangle_1 = |0\rangle_1$ is not affected in this case. The state is at this point $|N\rangle_L\,|2^{L+1}-(a+b-N)\rangle_{L+1}$. Since controlled-load gates have the control qubit set to 0, value N is added back to the result. The result is now $|a+b\rangle_{L+1} = |a+b \bmod N\rangle_{L+1}$. At the end a is subtracted and added back, but it has no effect on the state.

In Figure 4.11 we present the gate symbols for a modular adder. Figure (a) includes register storing N and temporary register $|t\rangle_1$. In Figure (b) this details are hidden, however they are still used in computation. Since states of these registers are restored, the can be recycled in subsequent executions of modular adder.

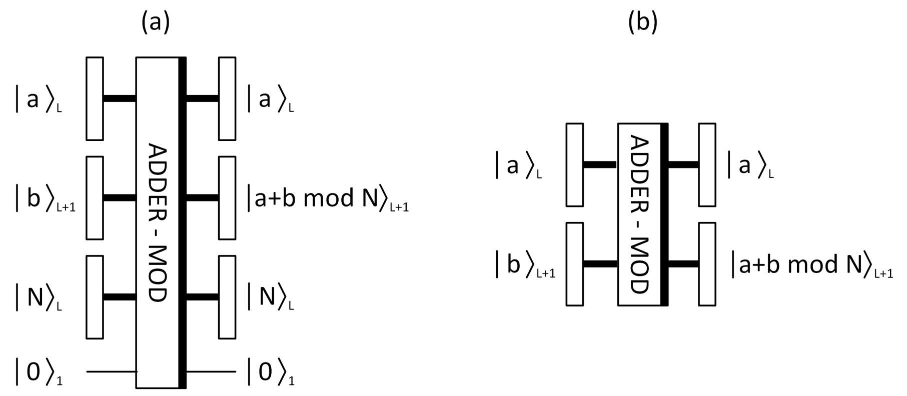


FIGURE 4.11: The gate symbols for a modular adder. In Figure (a) the register storing N and the temporary register $|t\rangle_1 = |0\rangle_1$ are shown. In Figure (b) they are hidden, however they are still used in the circuit.

Controlled Modular Multiplier

Controlled modular multiplier multiplies the state of quantum register by some classical value a if value of control qubit $|c\rangle_1$ is $|1\rangle_1$.

$$|c\rangle_1 |a\rangle_L |0\rangle_{L+1} \longrightarrow \begin{cases} |c\rangle_1 |a\rangle_L |xa \bmod N\rangle_{L+1}, & \text{if } |c\rangle_1 = |1\rangle_1 \\ |c\rangle_1 |a\rangle_L |a\rangle_{L+1}, & \text{if } |c\rangle_1 = |0\rangle_1 \end{cases}$$

This can be easily achieved using modular adders, because $xa = 2^0 xa_0 + 2^1 xa_1 + \ldots + 2^{n-1} xa_{n-1}$, where a_l is the l-th bit of a.

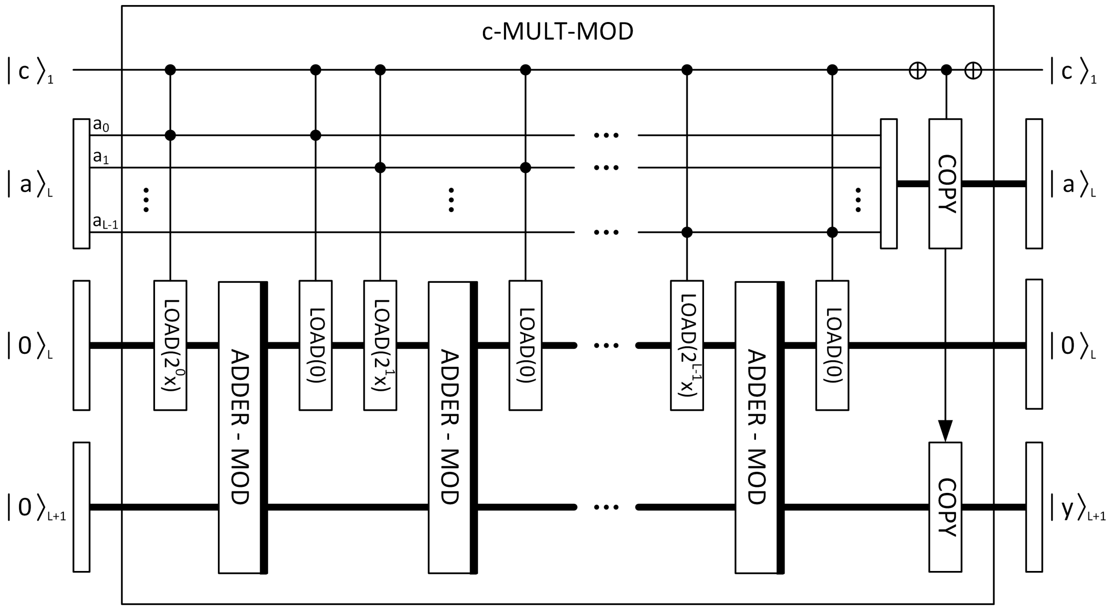


FIGURE 4.12: The circuit for a controlled modular multiplier.

Figure 4.12 presents the circuit for controlled modular multiplication. It consist of L modular adders. Before every adder there is a doubly controlled-load gate which loads value $2^l x$ to first register if the control bit $|c\rangle$ and l-th bit of $|a\rangle$ have value $|1\rangle$. After each adder there is a doubly controlled-load gate which loads value 0. Doubly controlled-load gates can be made out of Toffoli gates acting on respective qubits. The conditional-copy gate at the end of the circuit ensures that if the control qubit $|c\rangle_1$ state is $|0\rangle_1$ then value $|a\rangle$ is copied to the result register. It can be implemented using Toffoli gates. The result $|y\rangle_{L+1}$ is defined as follows:

$$|y\rangle_{L+1} = \begin{cases} |xa \bmod N\rangle_{L+1}, & \text{if } |c\rangle_1 = |1\rangle_1 \\ |a\rangle_{L+1}, & \text{if } |c\rangle_1 = |0\rangle_1 \end{cases}$$

Figure 4.13 shows gate symbols for controlled modular multiplier. Figure (a) presents gate symbol with temporary register $|0\rangle_L$. It is omitted in figures (b) and (c), since it can be recycled during computation. Figure (c) depicts inversion of controlled modular multiplier.

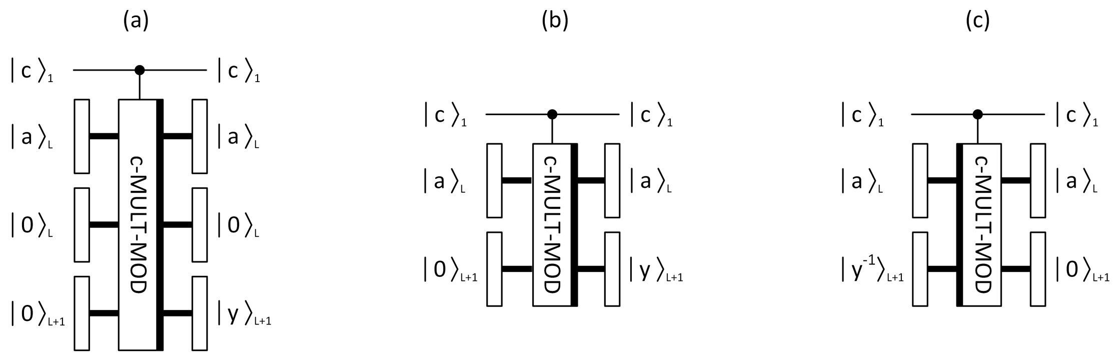


FIGURE 4.13: The gate symbols for a controlled modular multiplier. Figure (b) omits the temporary register $|0\rangle_L$ from Figure (a). Figure (c) presents the inverse controlled modular multiplier.

Modular Exponentiation

Finally, we describe the circuit for modular exponentiation, which implements unitary operator $U_{x,N}$. It raises classical value x to the power of value stored in quantum register $|a\rangle_L$ and then divides it mod$N$:

$$U_{x,N} |a\rangle_{2L} |1\rangle_{L+1} = |a\rangle_{2L} |x^a \bmod N\rangle_{L+1}$$

This is done by dividing exponentiation into a series of multiplications:

$$x^a = x^{2^0 a_0} \times x^{2^1 a_1} \times \ldots \times x^{2^{2L-1} a_{2L-1}}$$

where a_i is the i-th bit of a.

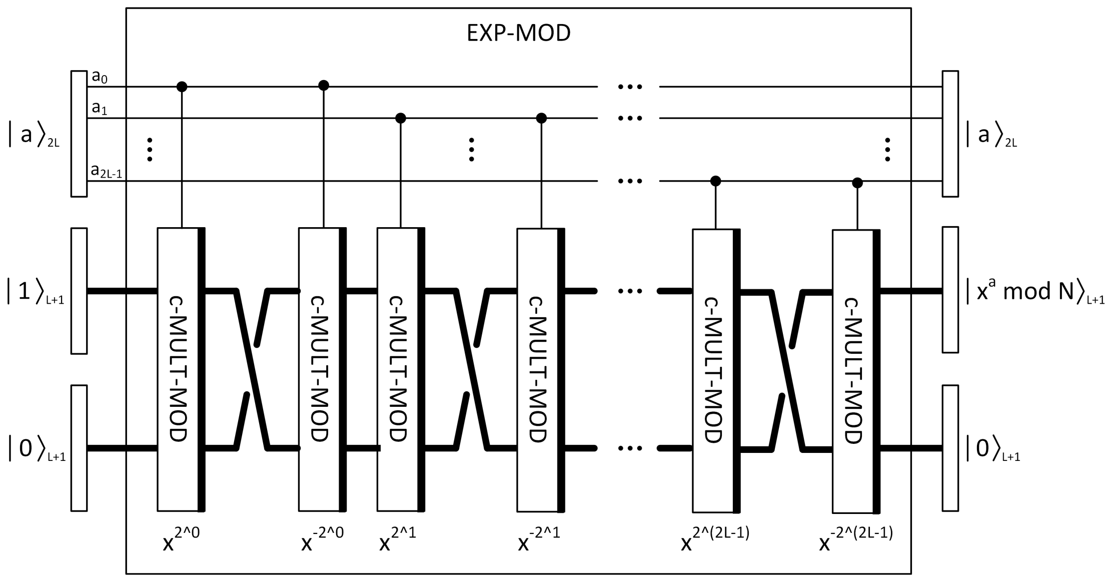


FIGURE 4.14: The circuit for a modular exponentiation

Figure 4.14 shows the circuit for modular exponentiation. It is divided into m stages. In i-th stage there is controlled modular multiplication which takes classical value x^{2^i} as an argument. Then there is the swapping of registers, which can be done using c-NOT gates. After swapping there is another controlled modular multiplier which takes value x^{-2^i}. This is done to restore the temporary register to its initial state.

4.3.3 Basic Implementation of Quantum Fourier Transform

Quantum Fourier Transform is the key part of Quantum Order Finding. It makes it possible to extract the period from the superposition of states in the input register entangled with the output register.

The L-qubit unitary Quantum Fourier Transform U_{QFT} is defined as follows:

$$U_{QFT}\,|x\rangle_L = \frac{1}{2^{L/2}} \sum_{y=0}^{2^L-1} e^{2\pi ixy/2^L}\,|y\rangle_L \tag{4.1}$$

The circuit for the Quantum Fourier Transform has been described by Peter Shor [8]. A quantum circuit performing a unitary Quantum Fourier Transform U_{QFT} can be built of only 1-qubit Hadamard gates H_i and 2-qubit controlled phase kick gates $R_{j,k}$:

$$U_{QFT} = H_{L-1}R_{L-2,L-1}H_{L-2}R_{L-3,L-1}R_{L-3,L-2}H_{L-3}\dots H_1R_{0,L-1}R_{0,L-2}\dots R_{0,2}R_{0,1}H_0$$

To make the definition of U_{QFT} more legible it can be divided into the series of L unitary transformations S_l.

$$U_{QFT} = S_{L-1}S_{L-2}\dots S_1S_0$$

Transformations S_l can be defined as:

$$S_l = R_{l,l+1}R_{l,l+2}\dots R_{l,L-2}R_{l,L-1}H_l$$

Index l denotes the qubit to which Hadamard transform is applied.

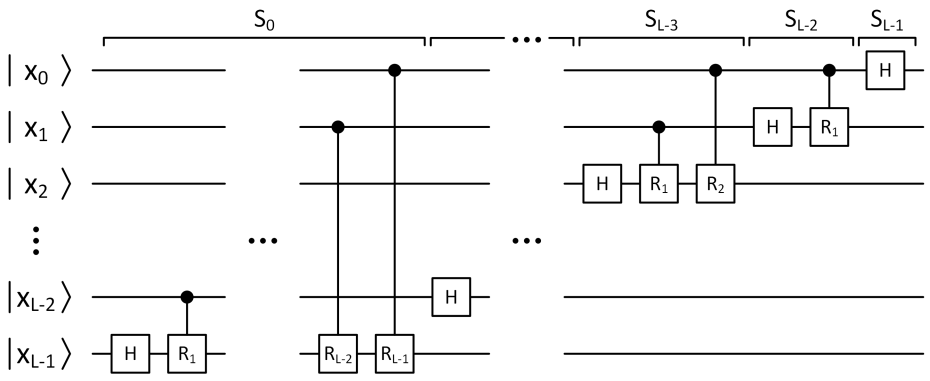


FIGURE 4.15: The Quantum Fourier Transform circuit. H are the Hadamard gates. R are the controlled phase kick gates with parameter l, which is the "distance" between target and control qubit.

Figure 4.15 depicts a Quantum Fourier Transform circuit. For the sake of simplicity the phase kick gates on the diagram accept only one parameter $l = k - j$. Parameters k and j can be treated as the numbers of control and target qubits, but in the case of the phase kick gate it does not matter which one is which. Parameter l can be understood as the "distance" between target and control qubit.

We can now analyze how the circuit works [40]. Let $e(t) = e^{2\pi it}$. We can rewrite (4.1) as:

$$|\phi(a)\rangle_L = U_{QFT}\,|x\rangle_L = \frac{1}{2^{L/2}}\sum_{y=0}^{2^L-1} e(xy/2^L)\,|y\rangle_L \tag{4.2}$$

Since the state in (4.2) is unentangled, we can present it as a tensor product of individual qubits:

$$|\phi(x)\rangle_L = |\phi_{L-1}(x)\rangle \otimes \dots \otimes |\phi_1(x)\rangle \otimes |\phi_0(x)\rangle \tag{4.3}$$

Where $|\phi_k(x)\rangle_1$ is the k-th qubit of $|\phi(x)\rangle_L$:

$$|\phi_k(x)\rangle_1 = \frac{1}{\sqrt{2}}(|0\rangle + e(x/2^k)\,|1\rangle) \tag{4.4}$$

Since $e(1+x) = e^{2\pi i(1+x)} = e^{2\pi i}e^{2\pi i x} = e^{2\pi i}e(x) = e(x)$, we can notice that $e(a/2^k) = e(0.a_k \dots a_1 a_0)$ where $0.a_k \dots a_1 a_0$ is a binary fraction. Then we express (4.4) as:

$$|\phi_k(a)\rangle_1 = \frac{1}{\sqrt{2}}(|0\rangle + e(0.a_k \dots a_1 a_0)\,|1\rangle) \tag{4.5}$$

The state in (4.5) results form the quantum gates applied to qubit. After the Hadamard transform the state is:

$$|a_k\rangle_1 \xrightarrow{H} \frac{1}{\sqrt{2}}(|0\rangle + e(0.a_k)\,|1\rangle) \tag{4.6}$$

Phase kick gates following the Hadamard transform modify the state as follows:

$$\begin{aligned}\frac{1}{\sqrt{2}}(|0\rangle + e(0.a_k)\,|1\rangle) \xrightarrow{R_1} \frac{1}{\sqrt{2}}(|0\rangle + e(0.a_k a_{k-1})\,|1\rangle) \xrightarrow{R_2} \dots \xrightarrow{R_{L-k-1}} \\ \xrightarrow{R_{L-k-1}} \frac{1}{\sqrt{2}}(|0\rangle + e(0.a_k a_{k-1} \dots a_0)\,|1\rangle)\end{aligned} \tag{4.7}$$

4.4 Circuits Variants and Optimization

In Section 4.3 we presented standard circuits for Modular Exponentiation and Quantum Fourier Transform. In this Section we show how these circuits can be optimized. We focus on optimizations in terms of the number of qubits. Additionally, in Section 4.4.4 we briefly describe some of other approaches to the optimization of the Order Finding circuit.

In Figure 4.16 we show an overview of implementation variants described in this Section. On the top of the Figure, the three steps of Shor's Algorithm are presented - Register Preparation, Quantum Modular Exponentiation and Quantum Fourier Transform. These steps are introduced in Section 4.2. In the bottom part of the Figure there are implementation variants below the corresponding step.

Register Preparation is a simple step and there is no place for any improvement, therefore there is only one version of it, described in Section 4.3.1. There are two variants of Quantum Modular Exponentiation. One of them, with the Classical Adder is described in Section 4.3.2, while the other, with QFT Adder is introduced in Section 4.4.1.

The Quantum Fourier Transform also has two implementations. The standard QFT is described in Section 4.3.3. The semiclassical QFT, which uses only classically controlled single-qubit gates, is presented in Section 4.4.2. The Quantum Modular Exponentiation with QFT Adder and Semiclassical QFT with a single control qubit combines the concepts of the QFT Adder and Semiclassical QFT to significantly reduce the number of qubits needed to perform computations. It is described in Section 4.4.3.

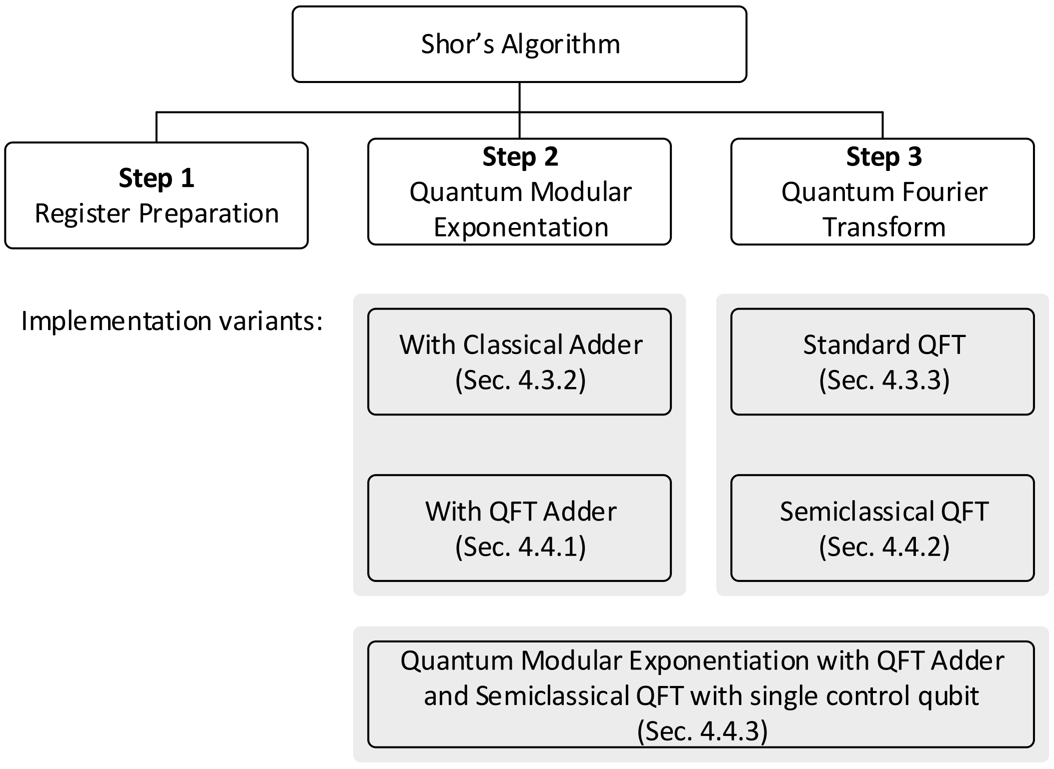

Shor's Algorithm
Step 1
Register Preparation
Step 2
Quantum Modular Exponentation
Step 3
Quantum Fourier Transform
Implementation variants:
With Classical Adder (Sec. 4.3.2)
With QFT Adder (Sec. 4.4.1)
Standard QFT (Sec. 4.3.3)
Semiclassical QFT (Sec. 4.4.2)
Quantum Modular Exponentiation with QFT Adder and Semiclassical QFT with single control qubit (Sec. 4.4.3)

FIGURE 4.16: An overview of optimization variants of the Order Finding circuits.

4.4.1 Modular Exponentiation with Quantum Fourier Transform Adder

In Section 4.3.2 we described a circuit for modular exponentiation based on the classical adder implemented in reversible way. Thomas Draper has proposed a different approach - to implement an adder using the Quantum Fourier Transform and phase kick gates [40]. Stéphane Beauregard developed this idea into a complete circuit for modular exponentiation [41].

It follows the same complex gate hierarchy as the circuit with the classical adder. Modular exponentiation is built out of controlled modular multipliers. These are substituted with modular adders, which consist of plain adders.

Plain Adder

Plain Adder is built from controlled phase kick gates R_l. It operates in Fourier space, which means that the Quantum Fourier Transform has to be applied to one of the registers before computation, and then the inverse Quantum Fourier Transform has to be applied after computation.

Figure 4.17 presents an adder circuit which adds values a and b stored in quantum registers. Notation $|\phi(b)\rangle$ means that the register stores value b and that Quantum

Fourier Transform has been applied to this register. In order to extract value from such register, inverse Quantum Fourier Transform has to be applied.

This circuit works similarly to the QFT circuit described in Section 4.3.3. The series of conditional phase kick gates is applied to k-th qubit $|\phi_k(b)\rangle_1$ resulting in state $|\phi_k(a+b)\rangle_1$:

$$\begin{aligned} |\phi_k(b)\rangle_1 = \frac{1}{\sqrt{2}}(|0\rangle + e(0.b_k \dots b_1 b_0)\,|1\rangle) \xrightarrow{R_0} \frac{1}{\sqrt{2}}(|0\rangle + e(0.b_k \dots b_1 b_0 + 0.a_k)\,|1\rangle) \\ \xrightarrow{R_2} \dots \xrightarrow{R_{L-k-1}} \frac{1}{\sqrt{2}}(|0\rangle + e(0.b_k \dots b_1 b_0 + 0.a_k \dots a_1 a_0)\,|1\rangle) = |\phi_k(a+b)\rangle_1 \end{aligned} \tag{4.8}$$

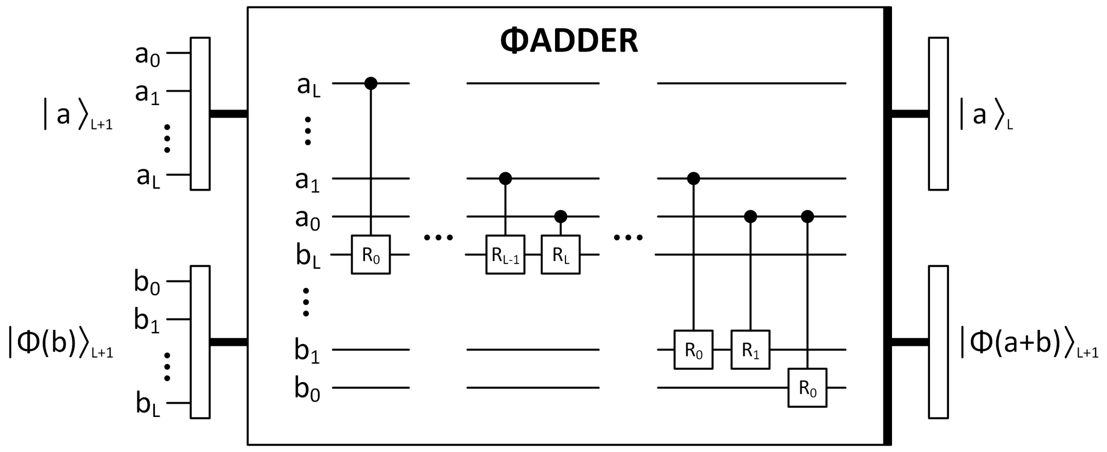


FIGURE 4.17: The circuit for adding values a and b stored in quantum registers. The addition is done in Fourier space.

Figure 4.18 shows a different version of the circuit from Figure 4.17. Here, classical value a is added to value b stored in a quantum register. Overall change of the phase of k-th qubit, from the controlled phase kick gates $R_0 \dots R_k$, is precomputed classically and applied by a single gate A_k.

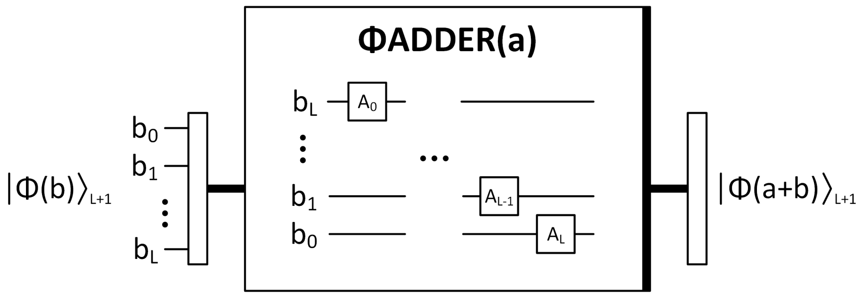


FIGURE 4.18: The circuit for adding classical value a to value b stored in quantum register. The addition is done in Fourier space.

In Figure 4.19 we show the gate symbol for an Adder in Fourier space. Figure (a) presents the adder, while Figure (b) shows the subtractor.

Figure 4.20 presents the usage of the Adder gate. Since the computation is performed in Fourier space, Quantum Fourier Transform have to be applied to register before addition.

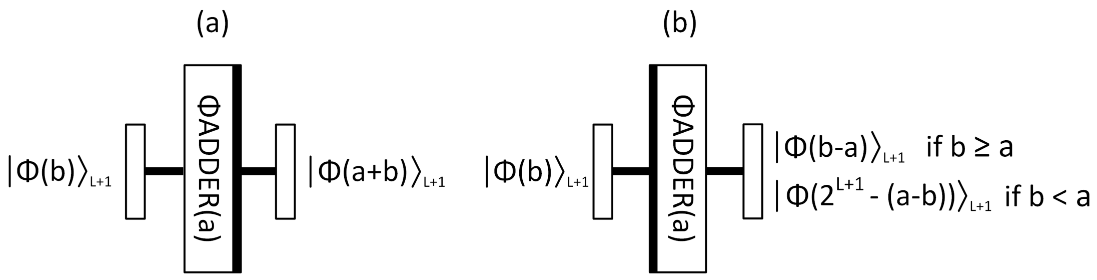


FIGURE 4.19: The gate symbol for the adder and subtractor in Fourier space. Figure (a) shows the gate which adds classical value a to the quantum register. Figure (b) presents inverse adder, that is a subtractor.

Also before retrieving the result of computation, inverse Quantum Fourier Transform have to be applied.

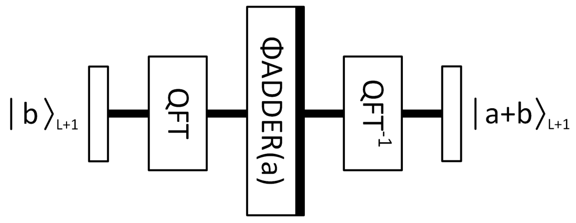


FIGURE 4.20: The usage of Adder gate in the Fourier space. The gate is preceded by Quantum Fourier Transform and followed by inverse Quantum Fourier Transform transform.

Controlled Modular Adder

Controlled Modular Adder is built out of Plain Adders. Figure 4.21 presents a circuit for modular addition. At the beginning, a classical value a is added to value b stored in a quantum register, which is already transformed into the Fourier space. Next, value N is subtracted to compare $(a + b)$ with N. Now the inverse Quantum Fourier Transform is computed to retrieve the value of the most significant qubit from the register. Value 1 means that an overflow occurred during subtraction, that is $N > (a + b)$. The value of the most significant qubit is copied using a c-NOT gate to the temporary register. Quantum Fourier Transformation is now applied to transform the state of the register back to the Fourier space. Now, depending on the state of the auxiliary register, value N is added back. Following computations do not have an effect on the value of the computation and are done only to restore the state of the auxiliary register to $|0\rangle$. The result of the circuit is:

$$|y\rangle_{L+1} = \begin{cases} |\phi((a+b) \bmod N)\rangle_{L+1}\,, & \text{if}\, |c_1\rangle_1 = |1\rangle_1 \,\text{and}\, |c_2\rangle_1 = |1\rangle_1 \\ |\phi(b)\rangle_{L+1}\,, & \text{if}\, |c_1\rangle_1 = |0\rangle_1 \,\text{or}\, |c_2\rangle_1 = |0\rangle_1 \end{cases}$$

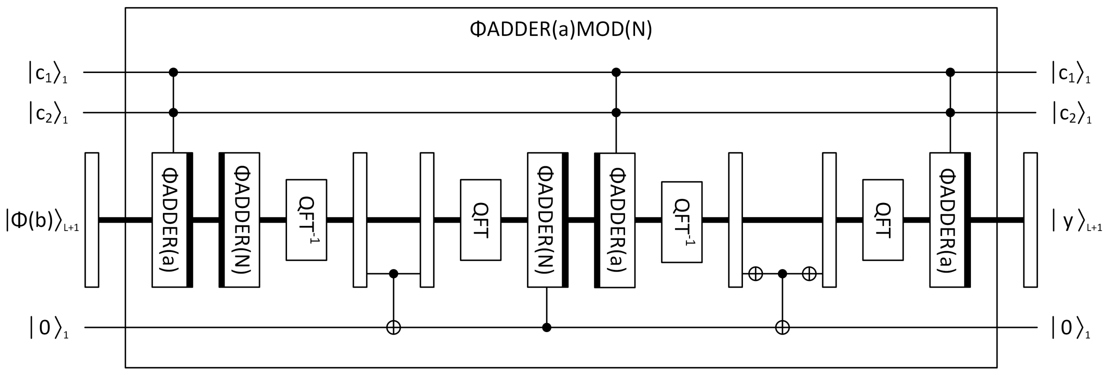


FIGURE 4.21: The circuit for a doubly controlled Modular Adder.

Figure 4.22 shows the gate symbol for a doubly controlled Modular Adder. The gate has two control qubits. Computation is performed in Fourier space.

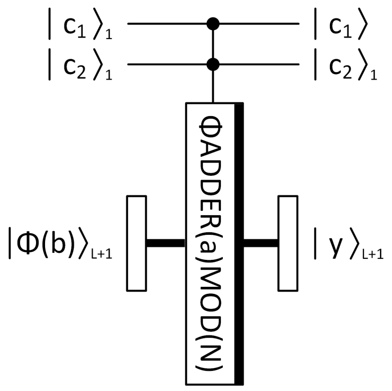


FIGURE 4.22: The gate symbol for a doubly controlled Modular Adder. Auxiliary qubit $|0\rangle_1$ is omitted in the gate symbol since it is reused during computation.

Controlled Modular Multiplier

The circuit for controlled Modular Multiplier is based on the identity:

$$ax \bmod N = (((2^0 xa_0) \bmod N + 2^1 xa_1) \bmod N + \ldots + 2^L xa_L) \bmod N$$

Figure 4.23 presents a circuit for modular multiplication. We can discriminate between the input register $|a\rangle_L$ and the output register $|b\rangle_{L+1}$. First of all QFT is applied to the output register to transform it to Fourier space. Next, the series of modular adder gates is applied to the output register. Each of them adds value $2^k x$ based on the state of k-th qubit of the input register $|a_k\rangle_1$. At the end, the inverse Quantum Fourier Transform is applied to transform the result into computational space. The value of the output register is:

$$|y\rangle_{L+1} = \begin{cases} |(b + ax) \bmod N\rangle_{L+1}, & \text{if } |c\rangle_1 = |1\rangle_1 \\ |b\rangle_{L+1}, & \text{if } |c\rangle_1 = |0\rangle_1 \end{cases}$$

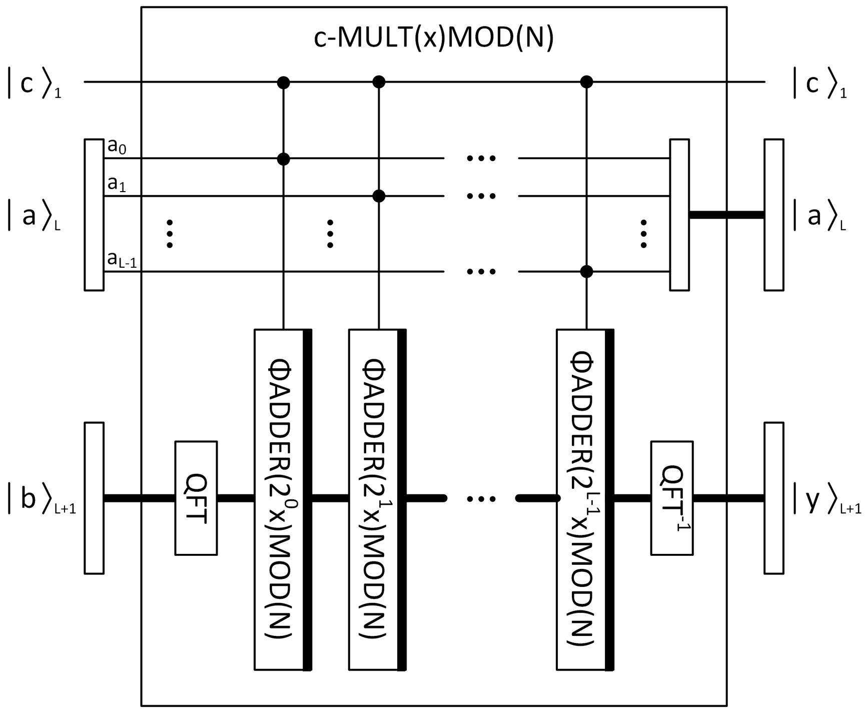


FIGURE 4.23: The circuit for a controlled Modular Multiplier

In Figure 4.24 we show the gate symbol for a controlled Modular Multiplier. The gate has one control qubit and it acts on two registers - input $|a\rangle_L$ and output $|b\rangle_{L+1}$.

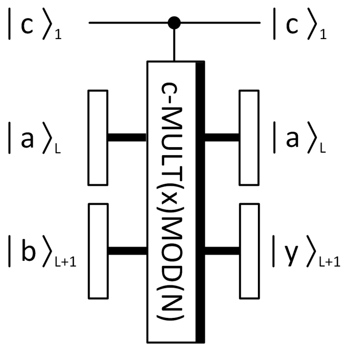


FIGURE 4.24: The gate symbol for a controlled Modular Multiplier.

Controlled U(x)

Controlled U(x) is a gate which actually transforms state $|a\rangle_L$ into state $|ax \bmod N\rangle_L$. We show the circuit in Figure 4.25. The gate operates on the input register $|a\rangle_L$ and the output register, initially in state $|0\rangle_{L+1}$. First, controlled modular multiplier is applied with parameter a. This sets the state of the output register to $|ax \bmod N\rangle_{L+1}$. Next, the L qubits of the input and output registers are swapped. The most significant qubit of the output register is always set to 0, therefore it is ignored. The controlled swap operation can be implemented using Toffoli gates. At the end, the inverse controlled modular multiplier is applied with argument a^{-1} – the modular multiplicative inverse of a. This restores the state of the output register to $|0\rangle_{L+1}$. The result of the gate is:

$$|y\rangle_L = \begin{cases} |(ax) \bmod N\rangle_L\,, & \text{if}\,|c\rangle_1 = |1\rangle_1 \\ |0\rangle_L\,, & \text{if}\,|c\rangle_1 = |0\rangle_1 \end{cases}$$

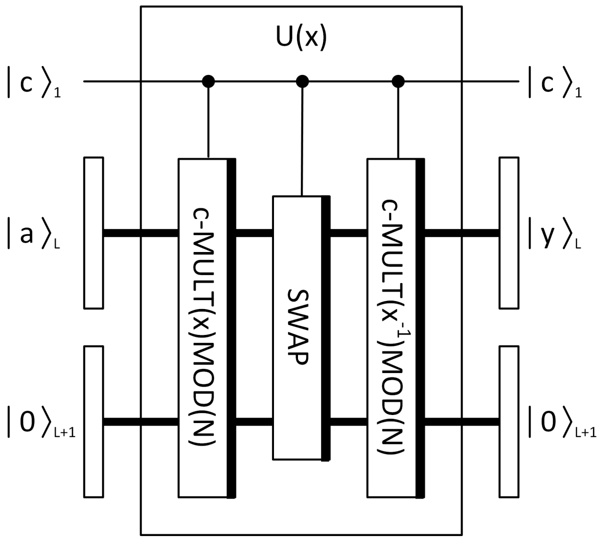


FIGURE 4.25: The circuit for a controlled U(x) gate.

Figure 4.26 presents gate symbol for a controlled U(x) gate. In Figure (a) there are both input and output registers. In Figure (b), the output register is hidden for readability. It is implicitly used during computation.

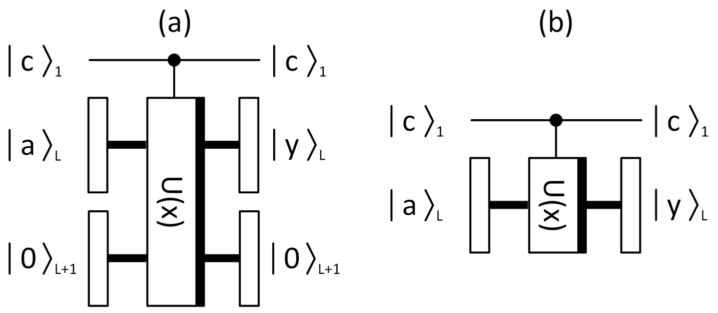


FIGURE 4.26: The gate symbol for controlled U(x). Figure (a) shows the input register $|a\rangle_L$ and the output register initiated to $|0\rangle_{L+1}$. In Figure (b) only the input register is shown because the output register is reused during computation.

Modular Exponentiation

Although Beauregard has not presented a circuit for computing modular exponentiation with a $2L$-qubit control register, we provide it for completeness. The circuit is presented in Figure 4.27. The series of controlled U(x) gates are applied to output register initially in state $|1\rangle_L$. It is based on following identity:

$$x^a \bmod N = (((x^{2^0 a_0}) \bmod N \times x^{2^1 a_1}) \bmod N \times \ldots \times x^{2^{2L-1} a_{2L-1}}) \bmod N$$

The k-th gate is controlled by the $|a_k\rangle_1$ qubit of the input register and it multiplies the output register by x^{2^k}. The result of computation is $|x^a \bmod N\rangle_L$.

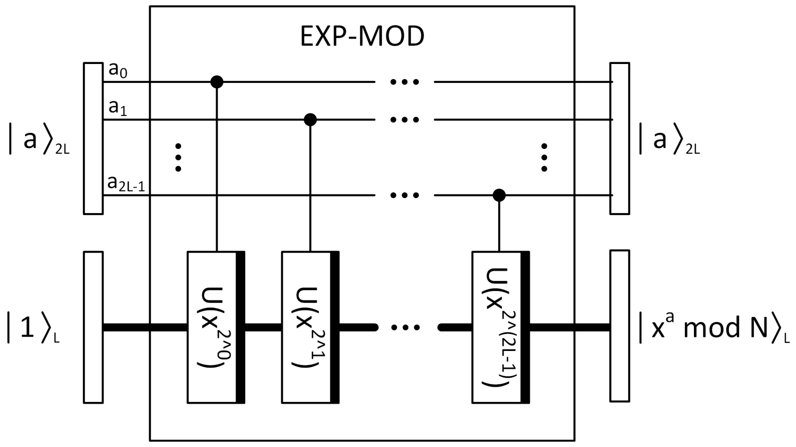


FIGURE 4.27: The circuit for Modular Exponentiation

4.4.2 Semiclassical Implementation of Quantum Fourier Transform

In 1995 Robert B. Griffiths and Chi-Sheng Niu described alternative way of performing Quantum Fourier Transform [42]. They assumed that two-qubit gates are going to be much more difficult to construct in physical implementations than one-qubit gates. They proposed a circuit which uses only one-qubit classically controlled gates.

In Shor's Algorithm, the Quantum Fourier Transform is directly followed by a measurement of the register. This makes it possible to interleave measurements of individual bits with the steps of Quantum Fourier Transform. Classical bit values of measurements can be then used to control subsequent quantum gates.

First of all, it is important to notice, that regarding controlled phase kick gates $R_{j,k}$, target and control qubits can be freely interchanged:

$$R_{j,k} = R_{k,j}$$

The graphical representation of such gates is presented in Figure 4.28.

In Figure 4.15 in Section 4.3.3 we presented the fundamental circuit for the Quantum Fourier Transform. It contains controlled phase kick gates of form (a) from Figure 4.28. In Figure 4.29 we show the same circuit, but using form (b) of the gates. For clarity, we

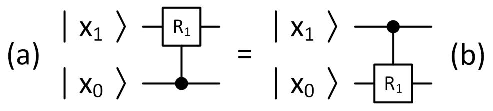


FIGURE 4.28: Controlled Phase Kick gate with interchanged target and control qubit.

also included subsequent measurements on the diagram. The y_l in measurement gates M represent l-th bit of value y which is the result of the order finding subroutine.

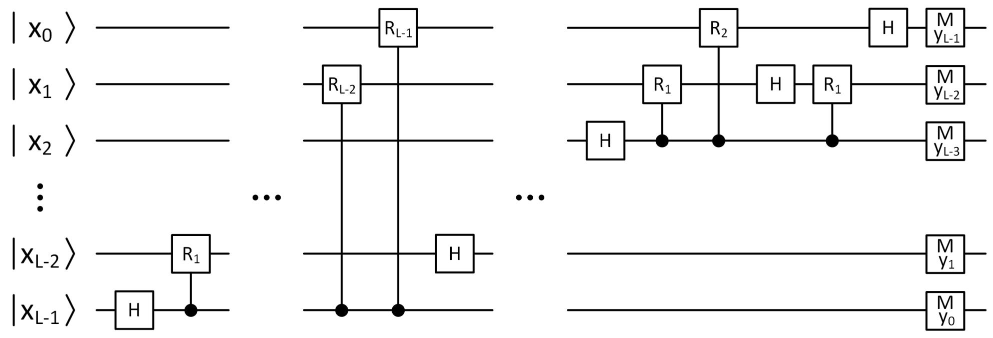


FIGURE 4.29: The circuit for Quantum Fourier Transform with interchanged target and control qubits of controlled phase kick gates. The circuit also shows measurement gates M. The y_l in measurement gates M represent l-th bit of value y which is the result of the order finding subroutine.

The most important observation from Figure 4.29 is that after applying the Hadamard gate on the l-th qubit there is no other gate modifying the state of that qubit before the measurement. Therefore it is possible to perform a measurement of the l-th qubit right after applying Hadamard gate. The result of measurement, which is a classical bit value, can then be used to classically control one-qubit phase kick gates.

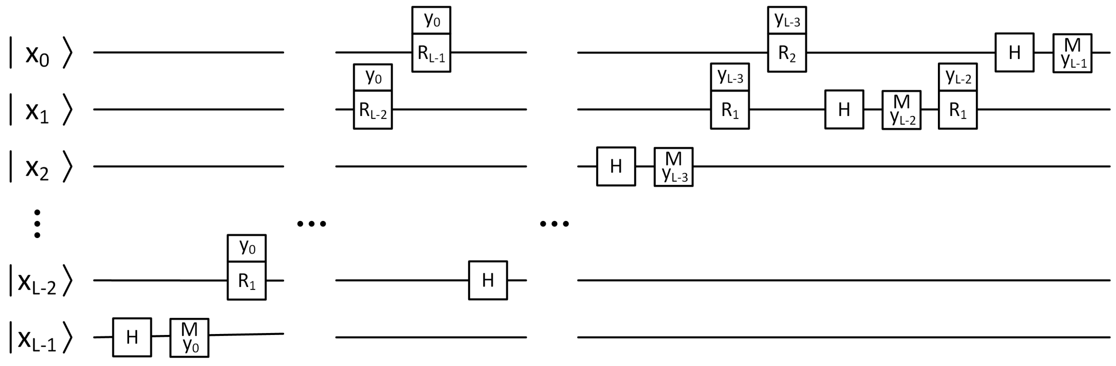


FIGURE 4.30: The circuit for Quantum Fourier Transform with semiclassically controlled phase kick gates. The y_l in measurement gates M represent l-th bit of value y which is the result of order finding subroutine. The y_l over phase kick gates R mean that gate is classically controlled by the value of bit y_l.

Figure 4.30 presents a circuit where the measurement of value y_l on l-th qubit is performed right after the Hadamard gate on that qubit. The value y_l is used to classically control subsequent phase kick gates acting on qubits from $l + 1$ to $L - 1$.

4.4.3 Quantum Modular Exponentiation with QFT Adder Semiclassical QFT with Single Control Qubit

The approach of using a single control qubit instead of $2L$ qubits to perform Quantum Modular Exponentiation and Quantum Fourier Transform has been proposed by Zalka [43]. Beauregard has developed a circuit which implements this idea [41].

In Section 4.4.1 we presented Quantum Modular Exponentiation circuit based on U(x) gates. Since all U(x) gates in this circuit commutes, they can be applied in any order. Together with Semiclassical Quantum Fourier Transform described in Section 4.4.2, it enables to implement the circuit for modular exponentiation and QFT using single control qubit.

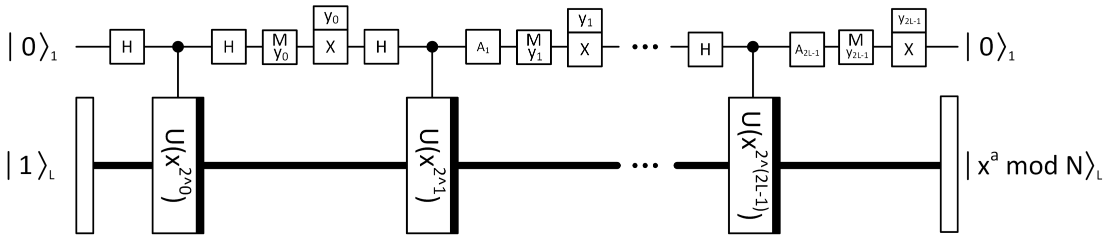


FIGURE 4.31: The circuit for semiclassical implementation of QFT with single control qubit.

In Figure 4.31 we show the circuit for Quantum Modular Exponentiation and Quantum Fourier Transform on a single qubit. It consist of $2L$ steps. At the beginning of each step, Hadamard transform is applied to achieve the equally probable superposition of states $|0\rangle$ and $|1\rangle$. Next, in the k-th step, controlled U(x) gate is applied with parameter x^{2^k}.

Subsequently, an A_k transform is applied. It is depicted in Figure 4.32. It consists of classically controlled phase kick gates, based on the results of previous measurements. Now the Hadamard transform is applied. These two transformations implement inverse QFT on k-th qubit. The computation of Fourier Transformation is followed by the measurement of the value y_k - the value of the k-th bit of value y returned by the Quantum Order Finding subroutine.

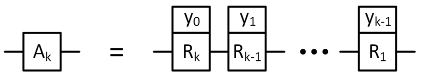


FIGURE 4.32: The A_k transform consisting of classically controlled phase kick gates.

4.4.4 Other Optimizations Approaches

Approximate Quantum Fourier Transform

Quantum Fourier Transform consist of a large number of controlled phase kick gates. For large gate's arguments k, the change in phase gets very small. Draper described that it is sufficient to apply only phase kick gates with argument k less than $\log_2 N$ [40]. It is accurate enough to be used in Shor's Factoring Algorithm [41]. The first advantage of this approximation is reducing the number of phase kick gates, since in QFT the k argument is between 1 and N. What is more, it significantly simplifies physical implementations of the circuit, because very small phase changes do not have to be applied to register.

Use of Mixed State Qubits

In some physical realizations of quantum circuits, such as Nuclear Magnetic Resonance, it is difficult to initialize a large amount of qubits in pure state. Parker and Plenio described that it is sufficient to use only one qubit in pure state and $L = \lceil \log_2 N \rceil$ qubits in maximally mixed states for Shor's Algorithm [44]. They exploit a single control qubit trick described in Section 4.4.3. They also assume existence of transformation $U_a \left|x\right\rangle_L = \left|ax \bmod N\right\rangle_L$ which multiplies quantum register $\left|x\right\rangle$ by classical value $\left|a\right\rangle$ using only L qubits. However, they do not describe how to implement such a function. Circuits described in Sections 4.3.2 and 4.4.1 require more than L qubits to perform multiplication in a reversible way.

4.5 Implementation Variants Summary

In this Chapter we presented Shor's Factoring Algorithm and its optimization variants. The algorithm consists of three parts - Classical Preprocessing, Quantum Order Finding and Classical Postprocessing. Quantum Order Finding is further divided into Register Preparation, Quantum Modular Exponentiation and Quantum Fourier Transform.

The optimizations apply to Quantum Modular Exponentiation and Quantum Fourier Transform. We focused on optimizing the registers length, that is the number of qubits. We described two versions of Quantum Modular Exponentiation - with Classical Adder and with QFT Adder. We also showed two variants of Quantum Fourier Transform - Standard QFT and Semiclassical QFT.

In the classical approach the Quantum Modular Exponentiation with Classical Adder is combined with the Standard QFT [39]. In this version the algorithm needs $7L + 3$

qubits, where L is the number of bits of factored number N. Modular exponentiation with Classical Adder can also be combined with semiclassical QFT, however it does not reduce the required register length.

Quantum Modular Exponentiation with QFT Adder is followed by Semiclassical QFT [41]. It needs $4L + 2$ qubits. This variant can be further optimized by exploiting the single control qubit trick. In this case the required registers length is $2L + 3$.

In Chapter 5 we discuss the simulation results of the two most distinct implementation variants. The first features Quantum Modular Exponentiation with Classical Adder and Standard QFT. The second has QFT Adder Modular Exponentiation and QFT Adder with single control qubit trick. Simulating other variants is also possible but it is out of the scope of this thesis.

Chapter 5

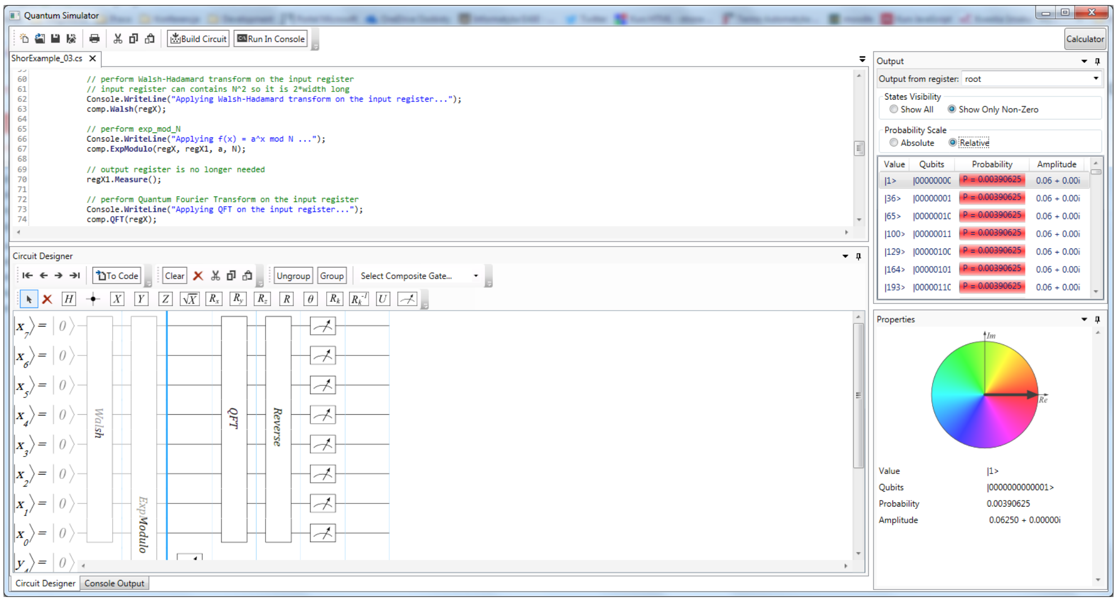
Simulations on the Classical Computer

In this Chapter we present the results of simulating Shor's Factoring Algorithm on the classical computer. In Section 5.1 we introduce the simulation environment. Section 5.2 describes the simulated variants of Shor's Algorithm. Section 5.3 presents the actual results of the simulations. At the end, in Section 5.4 we summarize and comment the simulation results.

5.1 Simulation Environment

For simulating the Shor's Algorithm we used QuIDE quantum computer simulator [31]. The fundamental module of the environment is the Simulation Library which emulates qubits and their behavior. Using the Library, QuIDE provides a development environment with the graphical circuit editor and the source code editor. The user can switch between the graphical and code representation at any time. The simulation environment of QuIDE enables the user to execute the circuit step by step or compute the final result. The environment provides the run-time preview of the internal quantum state. The Simulation Library can also be used as a standalone .NET library.

In Figure 5.1 we present the window of the QuIDE circuit editor and simulator. On top of the figure there is the source code editor. At the bottom of the figure there is the graphical circuit editor. The right side of the window shows the preview of the quantum state of the registers.



```
            // perform Walsh-Hadamard transform on the input register
            // input register can contains N^2 so it is 2*width long
            Console.WriteLine("Applying Walsh-Hadamard transform on the input register...");
            comp.Walsh(regX);

            // perform exp_mod_N
            Console.WriteLine("Applying f(x) = a^x mod N ...");
            comp.ExpModulo(regX, regX1, a, N);

            // output register is no longer needed
            regX1.Measure();

            // perform Quantum Fourier Transform on the input register
            Console.WriteLine("Applying QFT on the input register...");
            comp.QFT(regX);
```

Output from register: root

States Visibility: Show All [✓ Show Only Non-Zero]

Probability Scale: Absolute [✓ Relative]

Value	Qubits	Probability	Amplitude
|1>	|0000000C	P = 0.00390625	0.06 + 0.00i
|36>	|00000001	P = 0.00390625	0.06 + 0.00i
|65>	|0000001C	P = 0.00390625	0.06 + 0.00i
|100>	|00000011	P = 0.00390625	0.06 + 0.00i
|129>	|0000010C	P = 0.00390625	0.06 + 0.00i
|164>	|00000101	P = 0.00390625	0.06 + 0.00i
|193>	|0000011C	P = 0.00390625	0.06 + 0.00i

Properties

Value	|1>
Qubits	|0000000000001>
Probability	0.00390625
Amplitude	0.06250 + 0.00000i

FIGURE 5.1: The screenshot of QuIDE quantum computer simulator window.

QuIDE's simulation environment has been used to examine the behavior of internal parts of Shor's Algorithm. It helped understand the similarities and differences of implementation variants. The Simulation Library has been used for analyzing the execution time, memory usage and the success rate of implemented Shor's algorithm variants. A console program was developed to automate the execution of the simulation cases.

The simulations were executed on a notebook PC with the following configuration: Intel Core i5 M 450 2.4 GHz processor, 8 GB DDR3 memory, Windows 7 64-bit operating system.

5.2 Simulation Variants

Implementation variants of Shor's Algorithm have been thoroughly described in Section 4.4. In this Section we give a brief description of the variants that we implemented and executed in the simulation environment.

Standard Circuit - using 7L + 3 qubits

This variant implements a Standard Circuit for Modular Exponentiation described in Section 4.3.2. It also uses standard implementation of Quantum Fourier Transform introduced in Section 4.3.3. It needs a total register size of $7L + 3$ qubits, where L is the number of qubits of the factored number. We are going to refer to this variant as **7L + 3**.

Single Control Qubit Circuit - using 2L + 3 qubits

This implementation is based on Modular Exponentiation with the Quantum Fourier Transform adder described in Section 4.4.1. The Quantum Fourier Transform is implemented in the semiclassical way presented in Section 4.4.2. Modular Exponentiation and Quantum Fourier Transform are combined using the single control qubit trick described in Section 4.4.3. This variant needs a total register size of $2L + 3$ qubits, where L is the number of qubits of the factored number. We are going to refer to this variant as **2L + 3**.

5.3 Simulation Results

As previously described in Section 5.2 we simulate two variants of Shor's Algorithm - $7L + 3$ and $2L + 3$. We begin with comparing execution time of the variants in Section 5.3.1. Then, in Section 5.3.2, we focus on the memory requirements. In Section 5.3.3 we compare the success rates of the two simulated variants. At the end, in Section 5.3.4 we summarize the application of Shor's Algorithm implementation at the university classes.

5.3.1 Execution Time

In Figure 5.2 we show the relation between the number of bits of the factored number and the execution time. The Time axis is scaled in logarithmic scale. Dotted trend lines represent exponential functions.

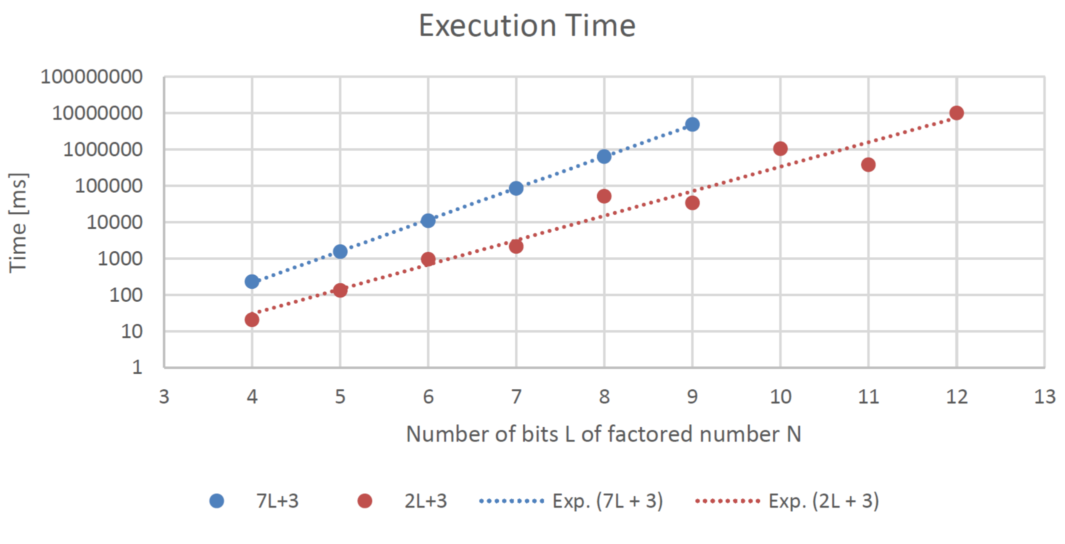


FIGURE 5.2: The chart presenting the relationship between the number of bits of factored number and the execution time. The Time axis is in logarithmic scale. Dotted lines are exponential function trend lines.

Due to the long execution time, the tests were executed a variable number of times. Table 5.1 presents the relationship between the number of bits L of factored number and the number of algorithm repetitions for both implementation variants.

TABLE 5.1: The relationship between the number of bits of factored number and the number of simulation repetitions for both implementation variants.

		Number of bits L of factored number N								
		4	5	6	7	8	9	10	11	12
Variant	7L + 3	10	10	10	10	10	5	-	-	-
	2L + 3	50	50	50	20	20	20	20	10	5

Both variants have exponential computational complexity. In the case of $7L + 3$ variant, all results of measurement lie exactly on the trend line. Measured execution times of the $2L + 3$ variant slightly differ from the trend line.

The standard deviation of the execution time for the $7L + 3$ variant is between 0.5% and 7.5% of the average value. In the case of the $2L + 3$ variant the standard deviation is much higher, ranging between 25% and 42% of the measured value. The high standard deviation is caused by large differences of execution times among the values of $\mathbf{x}$. This is further explained in the description of Figure 5.4.

The $2L + 3$ variant performs about an order of magnitude better than the $7L + 3$. In the $2L + 3$ Quantum Fourier Transform had to be computed multiple times, instead of only once as in the $7L + 3$. Nevertheless, a smaller number of qubits make operations on quantum registers much less time consuming.

Both simulated variants of Shor's algorithm have exponential computational complexity which complies with the theory as the quantum circuits simulated on classical computers would have at least exponential computational complexity [45] [46]. It has been proved that quantum algorithms can only be efficiently simulated using quantum systems [2].

Figures 5.3 and 5.4 present execution times for every number $\mathbf{x}$ coprime to $\mathbf{N} = 57$, respectively for the $7L + 3$ and $2L + 3$ variants. The number $\mathbf{x}$ is chosen at random in the classical preprocessing part of the algorithm. It is described in detail in Section 4.1. Number $\mathbf{N} = 57$ has been chosen to illustrate the case, but for other simulated numbers the results were similar. Normally, $\mathbf{x}$ is chosen at random at the beginning of the algorithm, but it is important to investigate what impact the chosen number has on the execution time. The dots on the charts present the average execution time of 100 measurements. Vertical lines span between minimum and maximum time. The times were taken into account regardless of the success of the iteration. There was no impact of the success or failure on the execution time of the algorithm.

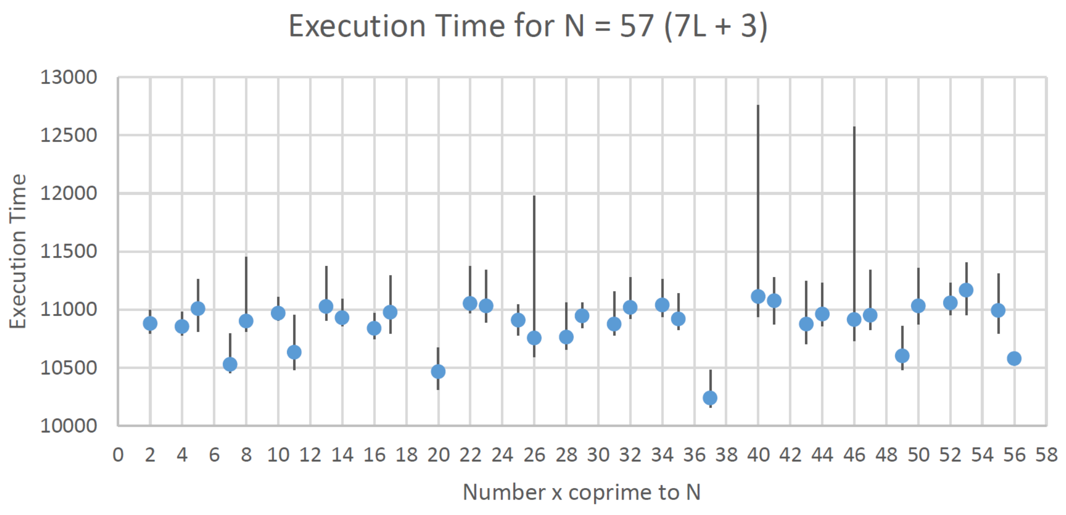


Figure 5.3: The chart showing execution time for every possible number x coprime to N = 57. The implementation variant is 7L + 3.

In the case of the $7L + 3$ variant, presented in Figure 5.3, most average times oscillate around 11000 ms. For most measurements the spread between average and extremal values does not exceed 5%. However, for numbers **x** such as 26, 40 and 46 the maximum execution time is significantly larger than the average time - even 13%. There are also some numbers **a** for which execution time is slightly lower - such as 7, 11, 20, 37, 47 and 56.

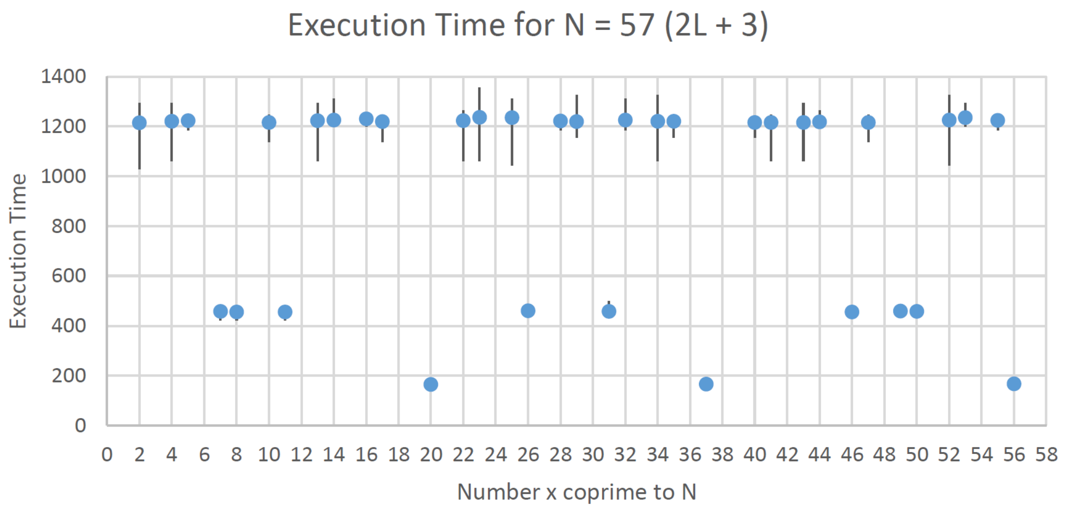


Figure 5.4: The chart showing execution time for every possible number x coprime to N = 57. The implementation variant is 2L + 3.

Taking the $2L + 3$ variant, presented in Figure 5.4, into consideration, it can be concluded that for most numbers **x** the execution time is 1200 ms. In contrast to the $7L + 3$ variant, maximal times do not significantly exceed the average. In some cases, the minimal values are about 16% lower than the average. What is more, for about 30% of measurements, the execution time is over 60% lower than for other 70% of samples. We can suppose that because of the single control qubit trick, the number of computations depend on the number of 1's in the binary representation of numbers used in computation.

To sum up, the $2L + 3$ variant performs about an order of magnitude better than $7L + 3$. What is more, the latter suffers from the values of **x** for which the execution time can be much worse than the average. In the case of the $2L + 3$ variant, maximal values do not the exceed average by much, and also there are a lot of numbers a for which the computation is performed much faster.

Regardless of chosen the implementation variant, it is very important for the performance of the algorithm to choose number **x** coprime to **N** at random before every iteration of the algorithm. This ensures equal probability to choose better or worse case.

5.3.2 Memory Usage

The chart in Figure 5.5 shows the memory usage according to the length of the factored number **N** in bits. The $7L + 3$ variant has exponential memory complexity. The blue dotted line represents the exponential trend function for the $7L + 3$ variant. In turn, the $2L + 3$ variant has linear memory complexity. The orange dotted line shows the linear trend function for this variant. The measurements were repeated 5 times in every case. The standard deviation in both cases is between 2.5% and 5%.

The better memory performance of the $2L + 3$ variant is caused by the internal implementation of the simulation environment as well as by the fewer needed qubits. The quantum states in the QuIDE simulator are represented by hash maps. Computing Quantum Fourier Transform for the quantum register of $2L$ size in the $7L + 3$ variant leads to the formation of a large number of quantum states. Due to the single control qubit trick and classically controlled QFT, the $2L + 3$ variant needs significantly less space to perform the same computations.

The L-qubit quantum register can have up to 2^L states. Therefore when such a register is simulated on a classical computer, the amount of memory increases exponentially with the length of register [45] [46]. Since the $7L + 3$ variant consist of purely quantum operations and because the Quantum Fourier Transform computes all states of the register,

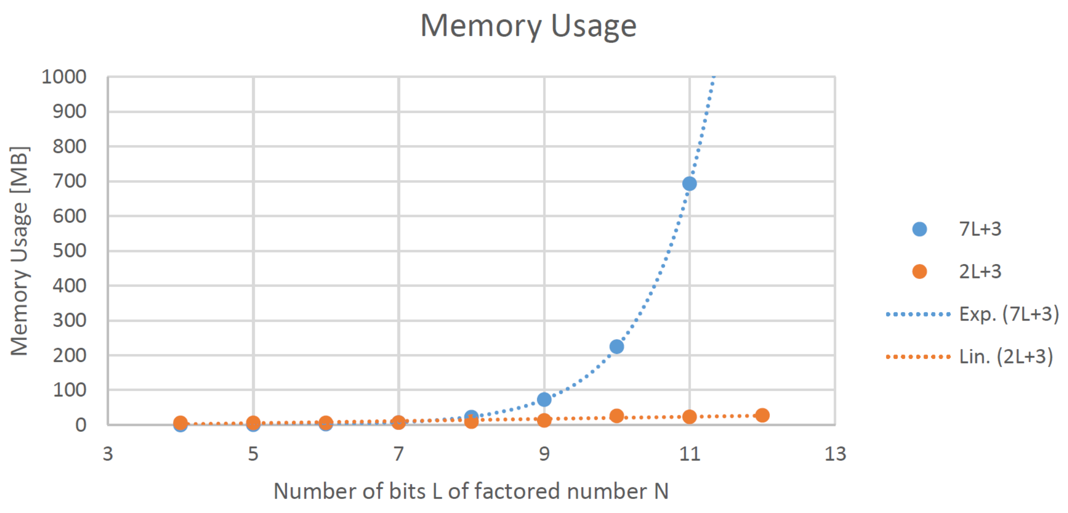


FIGURE 5.5: The chart presenting the relationship between the number of bits of factored number and memory usage. The dotted lines represent trend lines - exponential function for 7L + 3 and linear function for 2L + 3

the memory complexity of this variant is exponential. The semiclassical QFT and single control qubit in the the $2L + 3$ variant causes that all states of the quantum register do not have to be computed, because they are evaluated as classical values.

5.3.3 Success Rate

In this Section we analyze the success rate of both simulated variants, that is for how many executions the algorithm yields the correct period. The algorithm returns the correct period only with some probability [8]. In Section 4.1 we describe the features of the algorithm in more detail.

We begin by describing the success rate of the standard quantum algorithm in Section 5.3.3.1. Later on, in Section 5.3.3.2 we show how multiplying the order by a few small factors can enhance the effectiveness of order finding.

5.3.3.1 Standard Approach (without order multiplication)

In the standard approach the candidate for an order yield by the classical postprocessing is tested whether it is the correct order or not. If it fails to be an order, the Quantum Order Finding algorithm is executed one more time. If the value returned by the algorithm is not the correct order, the algorithm is immediately repeated without trying to multiply the value to get the correct order. In Section 5.3.3.2 we present a different approach, where the value is multiplied by a few small integers and checked whether it is an order.

In Figure 5.6 we show the success rates for all 6-bit numbers **N** being a product of two primes. For each number **N** it shows the average success rate for all possible numbers **x** coprime to **N**. The average is obtained as follows: we executed each implementation variant for every possible number **x** coprime to **N** for a hundred times. Then we computed the number of successful order findings for each number **x**. Finally, for each **N**, we computed the average success rate of all numbers **x** and we computed the standard deviation of the samples.

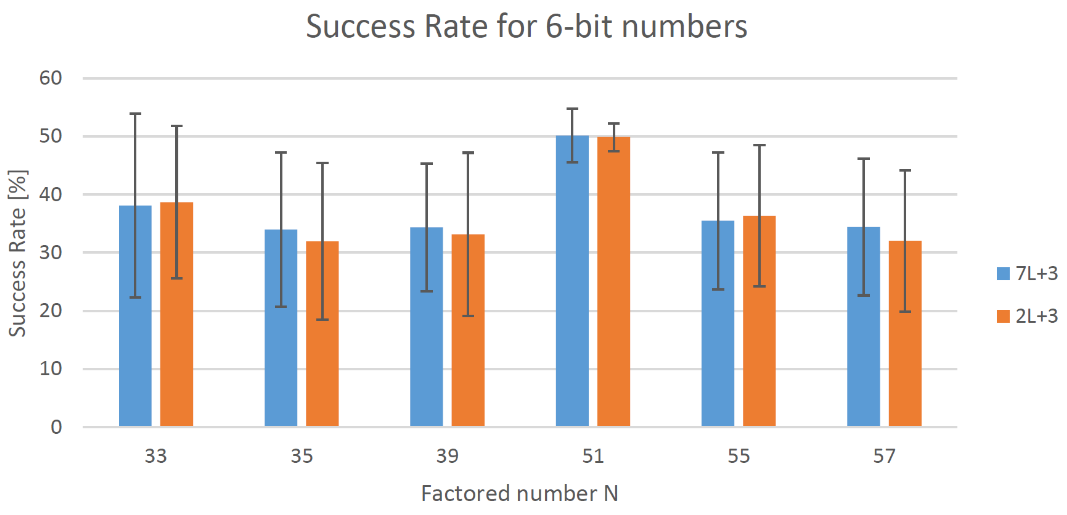


FIGURE 5.6: The chart presenting the success rate for all 6-bit numbers being a product of two primes. The bars represent the success rate of algorithms. The black lines denote the standard deviation of the results.

Both implementation variants performed similarly. The difference between the success rates does not exceed 2 percentage points. Also the standard deviations are on the similar level. For most factored numbers, the success rate is between 30% and 40%. The exception is **N** = 51 for which the rate is 50%. Taking all factored numbers into consideration, the mean success rate for both variants is 37%.

The success rate for **N** = 51 is slightly higher because of the form of the number. 51 = p * q, where p = 3, q = 17. Then (p-1) = 2 and (q-1) = 16. Both (p-1) and (q-1) are the powers of two. If **N** is of such special form it is more easily factored using Shor's Algorithm [33].

The theoretical success rate depends on the order **r**, which depends on the number **x** coprime to **N** [33]. It ranges between 20.27% and 40.53%. For all 6-bit factored numbers, except 51, the success rate is between 30% and 40%, thus it complies with the theory.

In Figures 5.7 - 5.12 we depict the relationship between the number **x** coprime to **N** and the success rate of order finding. We take the $7L + 3$ and $2L + 3$ variants into consideration.

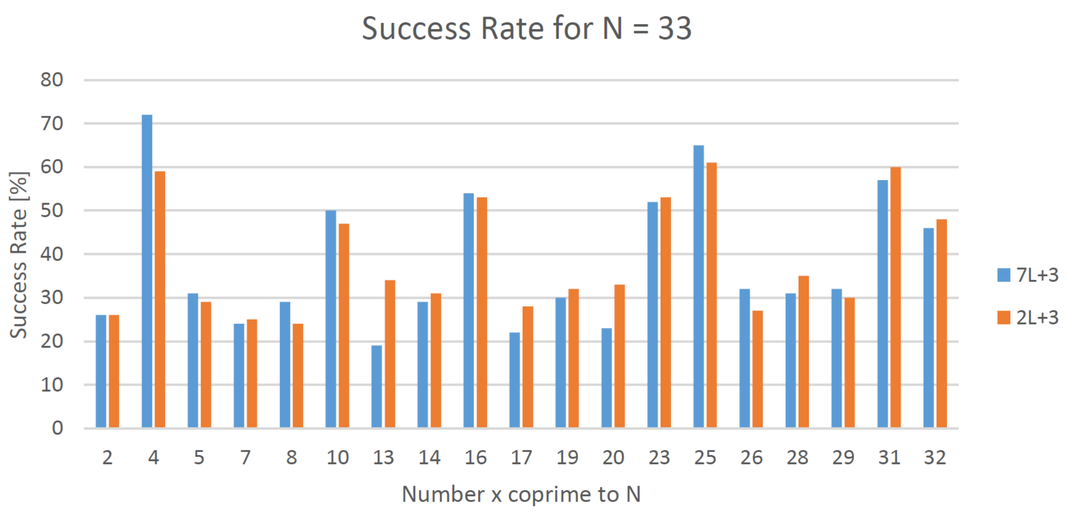


FIGURE 5.7: The chart presenting the relation between the number x coprime to N and the success rate of the Shor's Algorithm for N = 33.

The chart in Figure 5.7 presents the success rate in relation to number **x** coprime to **N** = 33. For most values of **x** the rate is about 30%. For some values of **x**, the success rate reaches or even exceeds 50%. For most measurements the difference between the $7L + 3$ and $2L + 3$ variants does not exceed 5 percentage points. However, for a few **x**'s the rate differs by more than 10 percentage points.

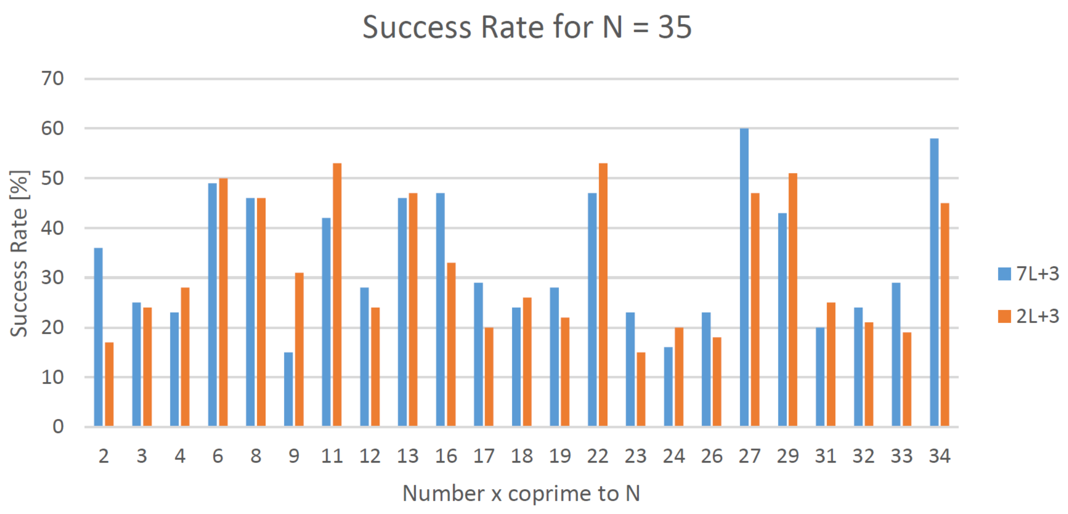


FIGURE 5.8: The chart presenting the relation between the number x coprime to N and the success rate of the Shor's Algorithm for N = 35.

The Figure 5.8 presents the relation between number **x** and the success rate for **N** = 35. For 35% of values of **x** the success rate is about 50%. For other values it is well below 30% and even below 20%. Nevertheless the mean success rate is still about 33%. Two implementation variants perform similarly in most cases. For some values the $7L + 3$ variant have better results, but for the other the $2L + 3$ is more accurate.

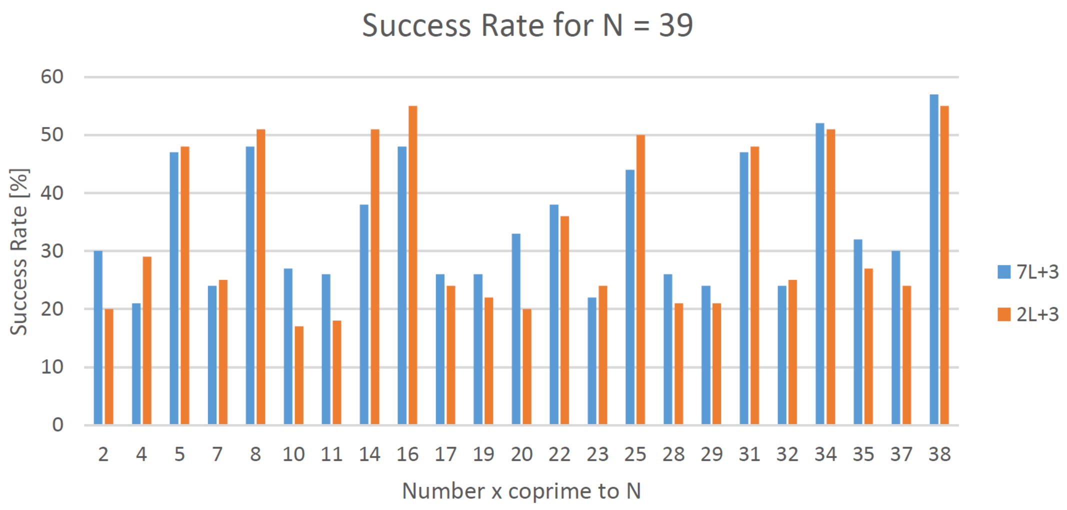


FIGURE 5.9: The chart presenting the relation between the number x coprime to N and the success rate of the Shor's Algorithm for N = 39.

In Figure 5.9 we present the success rates for **N** = 39. About 45% of values oscillate around 50% rate. Most of other values are between 20% and 30%. For a few of **x**'s values the difference between the implementation variants is about 10 percentage points.

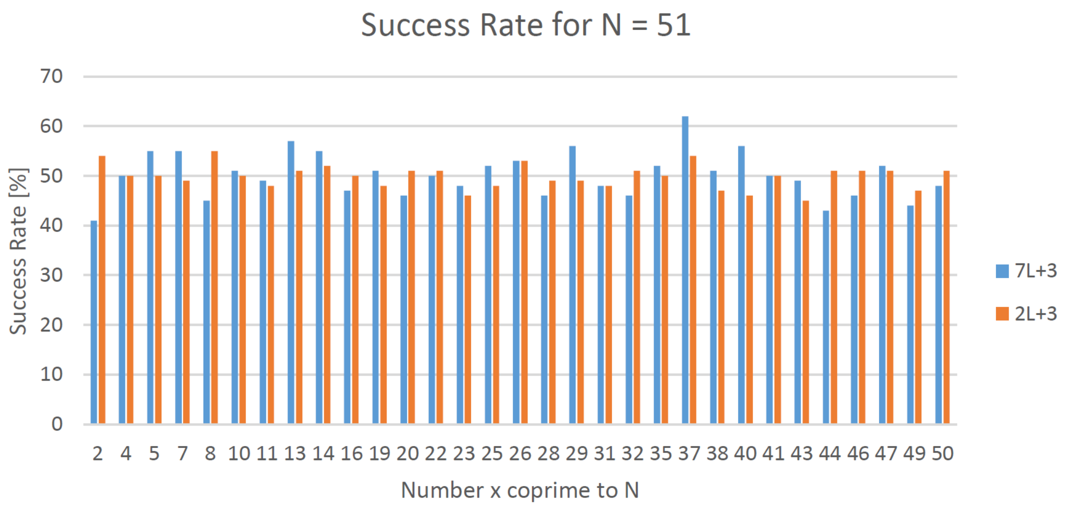


FIGURE 5.10: The chart presenting the relation between the number x coprime to N and the success rate of the Shor's Algorithm for N = 51.

Figure 5.10 presents the success rate chart for **N** = 51. In this case all success rate values are around 50%. Both variants have some values of **x** for which they outperform each other. For other values the differences are less than 5 percentage points. For this value of **N** the success rates for every **x** are more similar and the overall rate is much higher than for other presented 6-bit numbers. This is because of the special form of **N** = 51 described in Section 5.3.3.1.

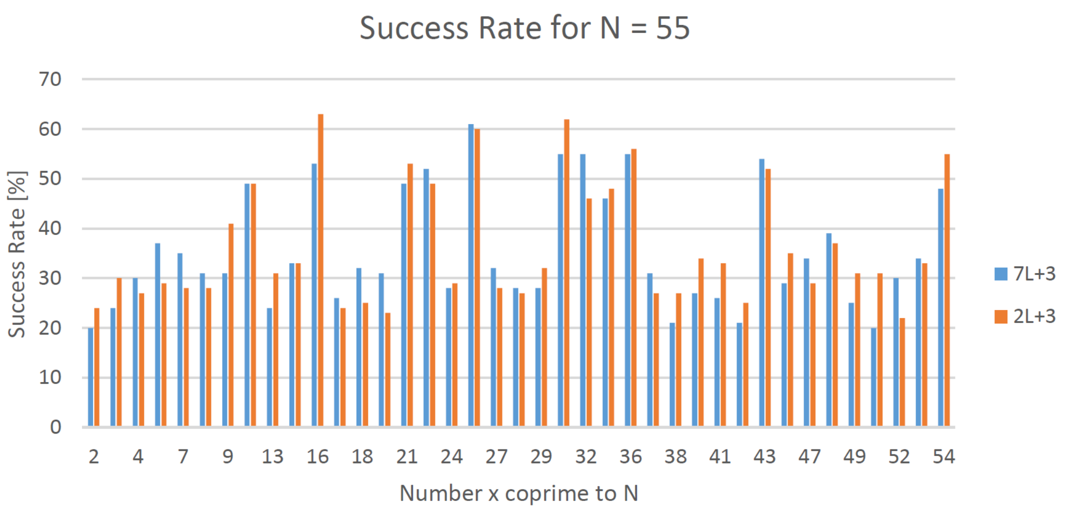


FIGURE 5.11: The chart presenting the relation between the number x coprime to N and the success rate of the Shor's Algorithm for N = 55.

In the Figure 5.11 the success rate values for **N** = 55 are presented. Most rates are about 30%, however some of them drop even to 20%. In 28% of the x values, the success rate reached or exceed 50%.

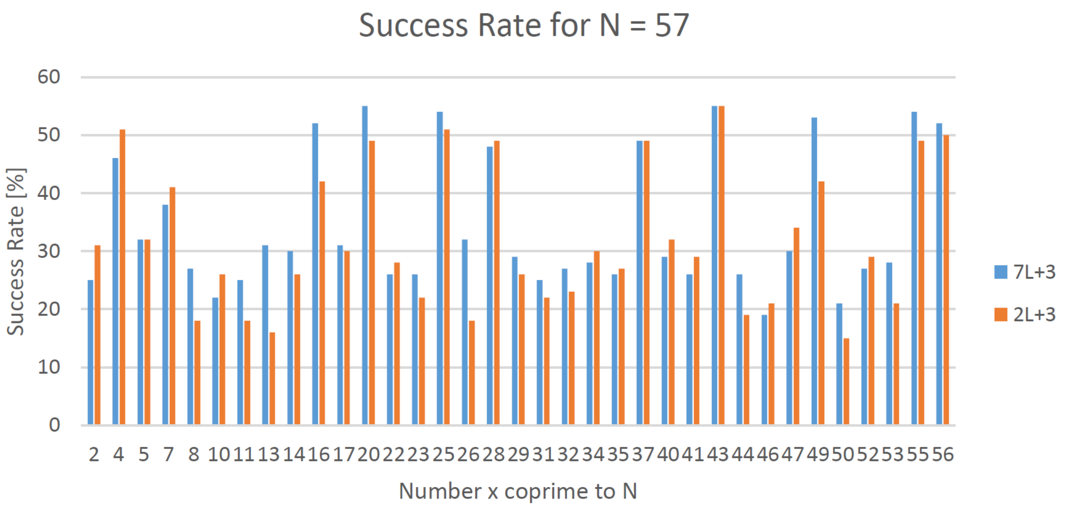


FIGURE 5.12: The chart presenting the relation between the number x coprime to N and the success rate of the Shor's Algorithm for N = 57.

In the Figure 5.12 the chart presenting the success rates for N = 57 is shown. Most of the rates are between 20% and 30%. About 28% of success rates reach 50%. In few cases the $7L + 3$ variant has significantly better rates than the $2L + 3$. The $2L + 3$ also have some better results but the difference is much smaller.

In conclusion, both implementation variants have similar success rates for the factored 6-bit numbers. This is correct because they are performing the same computation but implemented in two different ways. For some values of number $\mathbf{x}$, the rates differ, but since in the algorithm the value of $\mathbf{x}$ is chosen at random before each computation it does not have an effect on the effectiveness of algorithm. What is more, both implementation variants follow the same trends among the values of $\mathbf{x}$. Choosing between the $7L + 3$ and the $2L + 3$ variant have no significant impact on the success rate of the computation.

5.3.3.2 Enhanced Approach (with order multiplication)

The classical postprocessing, described in Section 4.1, includes a continued fraction algorithm which yields the candidate for order reduced by some divisors. If the returned value is not the correct order, it is reasonable to try multiplying it by a few small integers (i.e. $2, 3, 4, \ldots$) and check whether it is an order [33]. In this Section we present the success rate results for both variants of the algorithm with order multiplication.

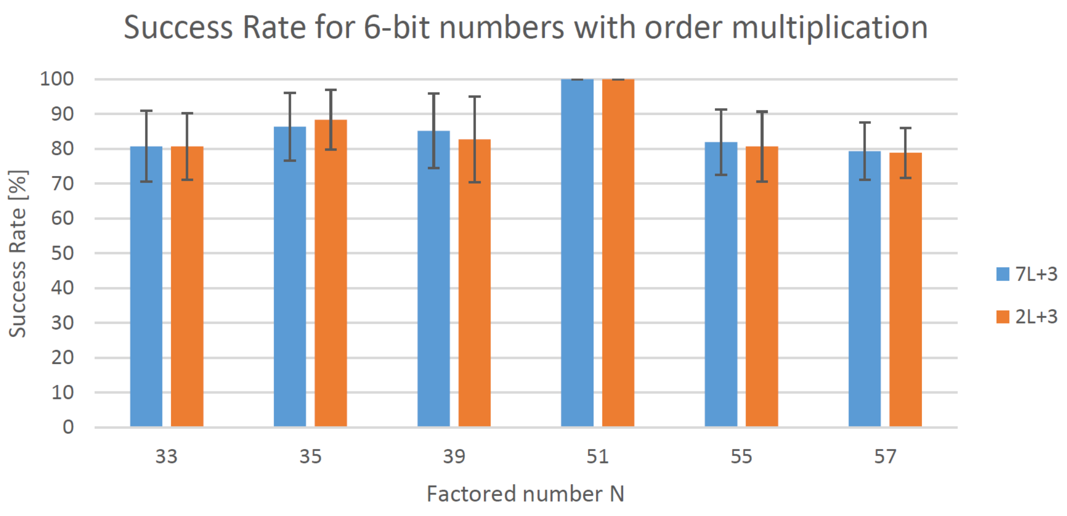


FIGURE 5.13: The chart presenting the success rates for all 6-bit products of two primes. If the result of an algorithm is not the order, it is multiplied by a few small numbers and checked for being an order. Vertical lines represent standard deviation of the results.

In Figure 5.13 we depict success rates for both implementation variants. For all factored numbers the success rate exceeds 80% regardless of the chosen implementation. The differences between the $7L + 3$ and $2L + 3$ implementation are less than 2 percentage points. The standard deviation is below 10 percentage points. The success rates are over two times better than in the standard approach, described in Section 5.3.3.1. The result for **N** = 51 is higher also in this case because of the special form of the number described in Section 5.3.3.1.

If **N** is of such form it is easier to be factored using Shor's Algorithm [33].

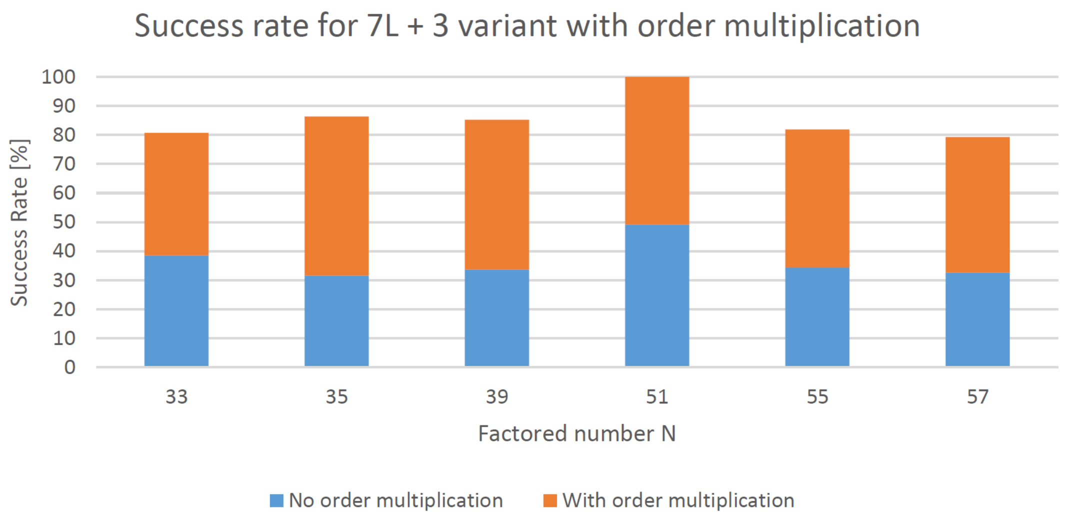


FIGURE 5.14: The success rate with and without order multiplication for 7L + 3 variant.

Figures 5.14 and 5.15 present the success rates for the $7L + 3$ variant and the $2L + 3$ variant respectively. Each bar is divided into two parts. The "No order multiplication" is the rate achieved by the quantum algorithm itself. The "With order multiplication" is the gain in the success rate achieved by multiplying incorrect candidates for order by a few small integers. In both cases the increase in the success rates exceeds 100%.

In Figure 5.16 we show the mean number of multiplication needed to compute the correct order. The average for factored numbers is between 2.5 and 3.5. The value between the two implementation variants does not vary more than 0.3. The standard deviation is between 0.8 and 1.5.

The order multiplication enhances the effectiveness by over 50 percentage points for 6-bit numbers. However, for such numbers there are only a few small orders. If we take factoring large numbers into consideration, we can suppose that the increase in the success rate will be significantly lower.

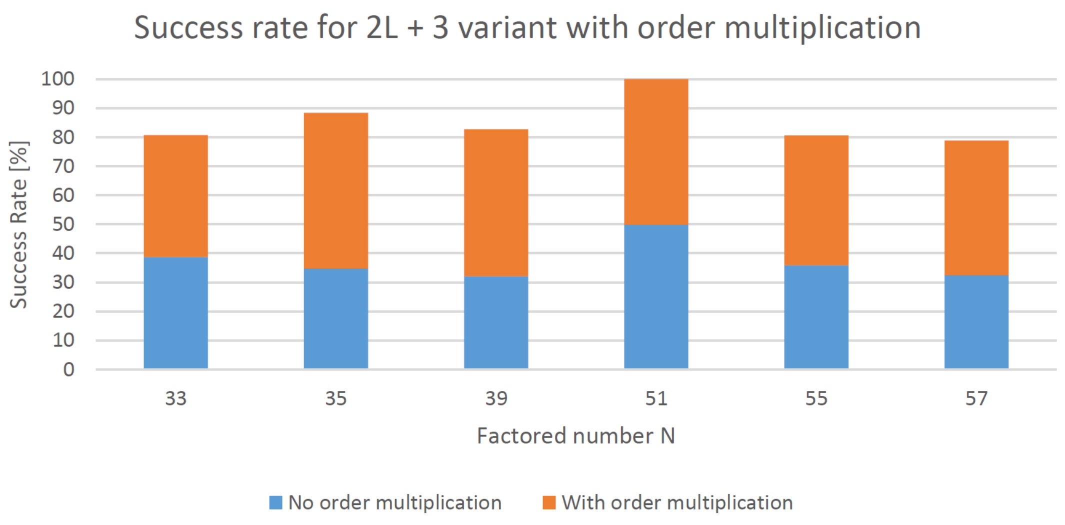


FIGURE 5.15: The success rate with and without order multiplication for 2L + 3 variant.

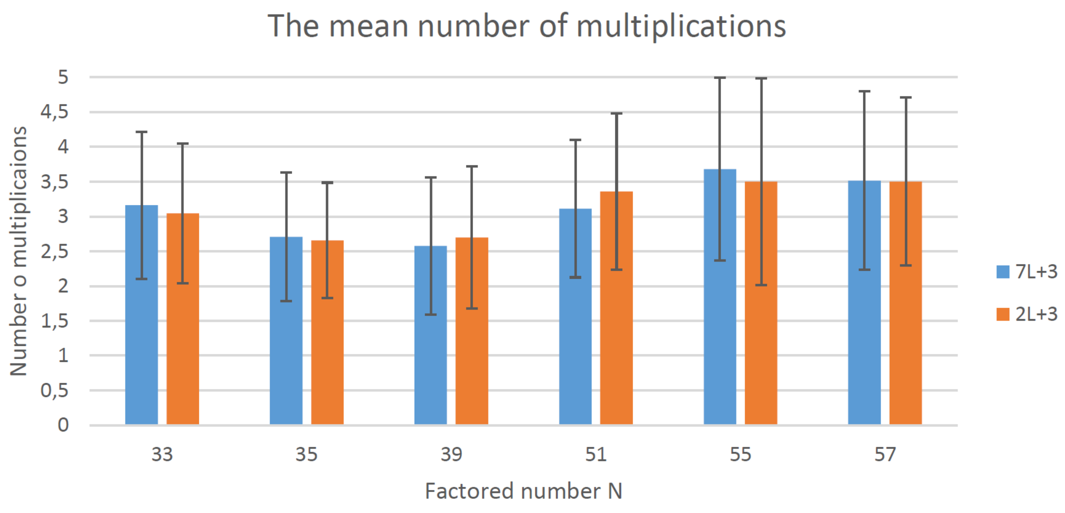


FIGURE 5.16: The chart presenting the mean number of multiplications needed to receive the correct order. The vertical lines depict the standard deviation.

5.3.4 Didactic Use

The Shor's Factoring Algorithm has been introduced during the *Matematyka w Informatyce Przyszłości*[1] (Quantum Computation) course. We presented the $7L + 3$ implementation variant in the QuIDE simulation environment. We used the simulator to describe how the respective parts of the algorithm function. During the classes, the students had to perform a few exercises regarding factoring a number into primes [2]. At the end of the course the students had to grade the usability of the QuIDE and the

[1] http://syllabuskrk.agh.edu.pl/2014-2015/en/magnesite/modules/15684
[2] http://icsr.agh.edu.pl/~kzajac/dydakt/matp/lab4/index.html

implementation of Shor's Algorithm. The grades were very good. The detailed results of the usability survey are described by Joanna Patrzyk in her M. Sc. thesis [31].

5.4 Conclusion

In this Chapter we have presented the simulation results for two variants of Shor's Factoring Algorithm. The first, $7L + 3$, consist of Quantum Modular Exponentiation with Classical Adder and Standard Quantum Fourier Transform. The latter, $2L + 3$, consist of QFT Adder Quantum Modular Exponentiation and Semiclassical QFT with the single control qubit trick. For simulations we used the QuIDE quantum computer simulation environment. We compared the implementation variants in terms of computational complexity, memory complexity and achieved success rate.

Both implementations have exponential computational complexity. Nevertheless, the $2L + 3$ have an order of magnitude lower execution times. For this variant there is also better chance to choose number $\mathbf{x}$ for which the execution time is significantly smaller.

The $2L + 3$ variant outperforms $7L + 3$ also in case of memory complexity. The $7L + 3$ has exponential memory complexity and it becomes impossible to factor numbers larger than 11-bit on standard PC. The linear complexity of $2L + 3$ makes it possible to factor much larger numbers.

The success rates for both implementation variants, the $7L + 3$ and $2L + 3$, are similar. They both reach the average of 37% of successfully computed orders. If we take the order multiplication technique into consideration, the average success rate increases to 85%. We have proved that their results does not significantly differ among values of $\mathbf{x}$ coprime to $\mathbf{N}$. This means that for the experimental purposes those algorithms can be used interchangeably.

To sum up, the $2L + 3$ variant gives the same results as $7L + 3$. However, its time and memory performance is significantly better. Therefore it should be an implementation of choice for simulating Shor's algorithm on a classical computer.

Chapter 6

Conclusion and Further Directions

In this Chapter we summarize the thesis. In Section 6.1 we discuss the achievement of the goals of this thesis. Section 6.2 presents the most important conclusions from the simulations of Shor's Algorithm. In Section 6.3 we present the further work.

6.1 Goals Achievement Discussion

The Goals of the thesis have been described in Section 1.3. In this Section we discuss how the goals have been achieved in the thesis.

Summary of Quantum Cryptology concepts

In Chapter 3 we have introduced the Quantum Cryptology. First of all, we described Quantum Key Distribution protocols. We took both Prepare and Measure protocols and Entanglement-based protocols into consideration. Later on we focused on the Quantum Commitment. We ended up by describing how Shor's Factoring Algorithm leads to breaking the RSA Cryptosystem.

Review of implementation variants of Shor's Algorithm

In Sections 4.1 and 4.2 we described the Shor's Algorithm in detail. We explained each part of the algorithm, namely Classical Preprocessing, Quantum Order Finding and Classical Postprocessing. We have also introduced elementary quantum subroutines needed to perform the Order Finding. We have described the standard implementation of Shor's Algorithm in Section 4.3. We provided circuits for Register Preparation, Quantum Modular Exponentiation and Quantum Fourier

Transform. In Section 4.4 we have presented the optimization variants of Quantum Modular Exponentiation and Quantum Fourier Transform. We have compared the implementation variants in terms of needed registers size.

Simulation of Shor's Algorithm

We have implemented the quantum circuits for Shor's Algorithm, described in Chapter 4, in the QuIDE simulation environment. We have tested whether the implementations are giving correct results according to theoretical assumptions. We have prepared and executed simulation cases in order to compare the implementation variants.

Simulation Results Analysis

In Chapter 5 we presented the outcomes of the simulations. We analyzed the performance parameters of implementations - the execution time and the memory usage. We have also compared the success rate of order finding achieved by algorithm variants. We concluded by stating which implementation is better suitable for experimental purposes.

6.2 Simulation Results Summary

We have simulated two implementation variants of Shor's Factoring Algorithm in the QuIDE quantum computer simulator. We compared the results in terms of computational complexity, memory complexity and the algorithm's success rate of order finding.

Both implementation variants have exponential computational complexity. This is correct according to the theoretical assumptions regarding simulating quantum computation on classical computers [45]. However, one of the variants performs about an order of magnitude better than the other.

The implementation variants differ in the case of memory complexity. One of the variants has exponential complexity, while the other have linear memory complexity. This is caused by the techniques used to reduce the required quantum register lengths.

The success rates of both implementation variants are on the same level. The average success rates for simulated cases range between 31% and 50%. It complies with the theoretical assumptions [33].

6.3 Further Work

In this thesis we have provided simulation results for two variants of Shor's Algorithm. There are also two other possibilities. Quantum Modular Exponentiation with Classical Adder can be combined with Semiclassical QFT and QFT Adder Quantum Modular exponentiation can be combined with Standard QFT. It would be worth to simulate those two variants and compare the results.

In Shor's Algorithm the number **x** coprime to **N** is chosen at random at the beginning of the algorithm. Classical computers provide only pseudo random numbers generators. It could be examined how using available physical quantum random number generators affects the simulation results.

In this thesis we focused on optimizing the required registers length. There are also variants of Shor's algorithm that optimize the number of operations or include parallel computations. These could also be implemented in the simulation environment.

Bibliography

[1] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of Statistical Physics*, 22(5):563–591, May 1980. doi: 10.1007/BF01011339. URL http://dx.doi.org/10.1007/BF01011339.

[2] Richard Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6-7):467âĂŞ488, 1982. doi: 10.1007/BF02650179. URL http://dx.doi.org/10.1007/BF02650179.

[3] David Deutsch. Quantum theory, the ChurchâĂŞTuring principle and the universal quantum computer. *Proceedings of Royal Society A*, 400:96–117, 1985. doi: 10.1098/rspa.1985.0070. URL http://dx.doi.org/10.1098/rspa.1985.0070.

[4] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of Royal Society A*, 439:553–558, 1992. doi: 10.1098/rspa.1992.0167. URL http://dx.doi.org/10.1098/rspa.1992.0167.

[5] André Berthiaume and Gilles Brassard. The quantum challenge to structural complexity theory. In *Proceedings of the Seventh Annual Structure in Complexity Theory Conference*, pages 132–137, Los Alamitos, CA, 1992. IEEE Computer Society Press. doi: 10.1109/SCT.1992.215388. URL http://dx.doi.org/10.1109/SCT.1992.215388.

[6] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 11–20, New York, 1993. ACM.

[7] Daniel R. Simon. On the power of quantum computation. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, volume 26, pages 116–123, Los Alamitos, CA, 1994. IEEE Computer Society Press. doi: 10.1109/SFCS.1994.365701. URL http://dx.doi.org/10.1109/SFCS.1994.365701.

[8] Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. In *Proceedings of the 35th Annual*

Symposium on Foundations of Computer Science, pages 124–134, 1994. doi: 10.1137/S0097539795293172. URL http://arxiv.org/abs/quant-ph/9508027.

[9] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. Cryptographic communications system and method, 1983.

[10] Lieven M.K. Vandersypen, Matthias Steffen, Gregory Breyta, Costantino S. Yannoni, Mark H. Sherwood, and Isaac L. Chuang. Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature*, 414:883–887, 2001. doi: 10.1038/414883a. URL http://dx.doi.org/10.1038/414883a.

[11] Samuel L. Braunstein, Carlton M. Caves, Richard Jozsa, Noah Linden, Sandu Popescu, and Ruediger Schack. Separability of very noisy mixed states and implications for NMR quantum computing. *Phys.Rev.Lett.*, 83:1054–1057, 1999. doi: 10.1103/PhysRevLett.83.1054. URL http://arxiv.org/abs/quant-ph/9811018.

[12] Erik Lucero, Rami Barends, Yu Chen, Julian Kelly, Matteo Mariantoni, Anthony Megrant, Peter O'Malley, Daniel Sank, Amit Vainsencher, James Wenner, Ted White, Yi Yin, Andrew N. Cleland, and John M. Martinis. Computing prime factors with a Josephson phase qubit quantum processor. *Nature Physics*, 8:1745–2473, 2012. doi: 10.1038/nphys2385. URL http://arxiv.org/abs/1202.5707.

[13] Enrique MartÃŋn-LÃşpez, Anthony Laing, Thomas Lawson, Roberto Alvarez, Xiao-Qi Zhou, and Jeremy L. O'Brien. Experimental realization of Shor's quantum factoring algorithm using qubit recycling. *Nature Photonics*, 6:773–776, 2012. doi: 10.1038/nphoton.2012.259. URL http://arxiv.org/abs/1111.4147.

[14] The D Wave Systems, 2014. URL http://www.dwavesys.com/. Accessed: 2014-05-19.

[15] Charles H. Bennet and Gilles Brassard. Quantum Cryptography: Public Key Distribution and Coin Tossing. In *Proceedings of IEEE International Conference on Computers, Systems and Signal Processing*, volume 175, page 8, 1984.

[16] Artur K. Ekert. Quantum Cryptography Based on Bell's Theorem. *Phys. Rev. Lett.*, 67:661–663, 1991. doi: 10.1103/PhysRevLett.67.661. URL http://link.aps.org/doi/10.1103/PhysRevLett.67.661.

[17] Albert Einstein, Borys Podolsky, and Nathan Rosen. Can Quantum-Mechanical Description of Physical Reality Be Considered Complete? *Phys. Rev.*, 47:777, 1935. doi: 10.1103/PhysRev.47.777. URL http://dx.doi.org/10.1103/PhysRev.47.777.

[18] David Bohm. *Quantum Theory*. Prentice Hall, Englewood Cliffs, NJ, 1951.

[19] John S. Bell. On the Einstein Podolsky Rosen paradox. *Physics*, 1:195, 1965.

[20] J. F. Clauser, M. A. Horne, A. Shimony, and R. A. Holt. Proposed Experiment to Test Local Hidden-Variable Theories. *Phys. Rev. Lett.*, 23:1804, 1969. doi: 10.1103/PhysRevLett.23.880. URL http://dx.doi.org/10.1103/PhysRevLett.23.880.

[21] M. Jacak, T. Martynkien, A. Janutka, J. Jacak, D. Melniczuk, W. Donderowicz, J. Gruber, and I. JÃşÅžwiak. Wroclaw Quantum Network - QKD deployment in a metropolitan network, 2014. Poster presented at The 5th LFPPI Symposium on Progress in Quantum Cryptography "seQre2014" âĂŞ 27th-28th January 2014, WrocÅĆaw, Poland.

[22] The Tokyo QKD Network, 2014. URL http://www.uqcc.org/QKDnetwork/. Accessed: 2014-05-19.

[23] M. Sasaki, M. Fujiwara, H. Ishizuka, W. Klaus, K. Wakui, M. Takeoka, A. Tanaka, K. Yoshino,, Y. Nambu, S. Takahashi, A. Tajima, A. Tomita, T. Domeki, T. Hasegawa, Y. Sakai, H. Kobayashi, T. Asai, K. Shimizu, T. Tokura, T. Tsurumaru, M. Matsui, T. Honjo, K. Tamaki, H. Takesue, Y. Tokura, J. F. Dynes, A. R. Dixon, A. W. Sharpe, Z. L. Yuan, A. J. Shields, S. Uchikoga, M. Legre, S. Robyr, P. Trinkler, L. Monat, J.-B. Page, G. Ribordy, A. Poppe, A. Allacher, O. Maurhart, T. Langer, M. Peev, and A. Zeilinger. Field test of quantum key distribution in the Tokyo QKD Network. *Optics Express*, 19:10387–10409, 2011. URL http://arxiv.org/abs/1103.3566.

[24] Richard J. Hughes, Jane E. Nordholt, Kevin P. McCabe, Raymond T. Newell, Charles G. Peterson, and Rolando D. Somma. Network-Centric Quantum Communications with Application to Critical Infrastructure Protection. Technical Report LA-UR-13-22718 (version 2), Los Alamos National Laboratory, 2013. URL http://arxiv.org/abs/1305.0305.

[25] The ID Quantique, 2014. URL http://www.idquantique.com/. Accessed: 2014-05-19.

[26] The MagiQ Technologies, 2014. URL http://www.magiqtech.com/. Accessed: 2014-05-19.

[27] The Quintessence Lab, 2014. URL http://www.quintessencelabs.com/. Accessed: 2014-05-19.

[28] The SeQureNet, 2014. URL http://www.sequrenet.com/. Accessed: 2014-05-19.

[29] Marcin Niemiec, Lukasz Romanski, and Marcin Swiety. Quantum Cryptography Protocol Simulator. *Multimedia Commun., Services and Security*, 149:286–92,

2011. doi: 10.1007/978-3-642-21512-4_34. URL http://dx.doi.org/10.1007/978-3-642-21512-4_34.

[30] Marcin Niemiec and Andrzej R. Pach. Management of security in quantum cryptography. *IEEE Communications Magazine*, 51(8), 2013. doi: 10.1109/MCOM.2013.6576336. URL http://dx.doi.org/10.1109/MCOM.2013.6576336.

[31] Joanna Patrzyk. Graphical and programming support for simulations of quantum computations. Master's thesis, AGH University of Science and Technology, Aleja Adama Mickiewicza 30, 30-059 KrakÃşw, 2014.

[32] Valerio Scarani, Antonio Acin, Gregoire Ribordy, and Nicolas Gisin. Quantum cryptography protocols robust against photon number splitting attacks for weak laser pulses implementations. *Phys. Rev. Lett.*, 92, 2004. URL http://arxiv.org/abs/quant-ph/0211131.

[33] David N. Mermin. *Quantum Computer Science. An Introduction.* Cambridge University Press, 2007.

[34] Gilles Brassard, Claude Crépeau, Richard Jozsa, and Denis Langlois. A Quantum Bit Commitment Scheme Provably Unbreakable by both Parties. In *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, pages 362–371, 1993. doi: 10.1109/SFCS.1993.366851. URL http://dx.doi.org/10.1109/SFCS.1993.366851.

[35] Dominic Mayers. The Trouble With Quantum Bit Commitment. Technical report, Computing Research Repository (CoRR), 1996. URL http://arxiv.org/abs/quant-ph/9603015.

[36] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen Lenstra, Emmanuel ThomÃľ, Joppe Bos, Pierrick Gaudry, Alexander Kruppa, Peter Montgomery, Dag Arne Osvik, Herman te Riele, Andrey Timofeev, and Paul Zimmermann. Factorization of a 768-bit RSA modulus. *Advances in Cryptology âĂŞ CRYPTO 2010*, pages 333–350, 2010. doi: 10.1007/978-3-642-14623-7_18. URL http://dx.doi.org/10.1007/978-3-642-14623-7_18.

[37] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms.* McGraw-Hill Higher Education, 2nd edition, 2001. ISBN 0070131511.

[38] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2000.

[39] Vlatko Vedral, Adriano Barenco, and Artur Ekert. Quantum networks for elementary arithmetic operations. *Phys. Rev. A*, 54:147–153, Jul 1996. doi: 10.1103/PhysRevA.54.147. URL http://link.aps.org/doi/10.1103/PhysRevA.54.147.

[40] Thomas G. Draper. Addition on a Quantum Computer. 2000. URL http://arxiv.org/abs/quant-ph/0008033.

[41] Stephane Beauregard. Circuit for Shor's Algorithm Using 2N+3 Qubits. *Quantum Info. Comput.*, 3(2):175–185, 2003. ISSN 1533-7146. URL http://arxiv.org/abs/quant-ph/0205095.

[42] Robert B. Griffiths and Chi S Niu. Semiclassical Fourier Transform for Quantum Computation. *Physical Review Letters*, 76(17):3228–3231, 1996. doi: 10.1103/PhysRevLett.76.3228. URL http://arxiv.org/abs/quant-ph/9511007.

[43] Christof Zalka. Fast version of Shor's quantum factoring algorithm. 1998. URL http://arxiv.org/abs/quant-ph/9806084.

[44] S. Parker and M.B. Plenio. Efficient factorization with a single pure qubit and logN mixed qubits. *Phys. Rev. Lett*, 85:3049–3052, 2000. doi: 10.1103/PhysRevLett.85.3049. URL http://arxiv.org/abs/quant-ph/0001066.

[45] Julia Wallace. Quantum Computer Simulators - A Review Version 2.1, 1999.

[46] H. De Raedt and K. Michielsen. Computational Methods for Simulating Quantum Computers. In M. Rieth and W. Schommers, editors, *Handbook of Theoretical and Computational Nanotechnology*, volume 3: Quantum and molecular computing, quantum simulations, chapter 1, page 248. American Scientific Publisher, 2006. URL http://arxiv.org/abs/quant-ph/0406210.

List of Figures

List of Tables

Appendix A

Papers

The results obtained in this thesis were published in the following paper. The author of this thesis is also co-author of the presented paper.

1. Joanna Patrzyk, Bartłomiej Patrzyk, Katarzyna Rycerz, Marian Bubak: A Novel Environment for Simulation of Quantum Computing, submitted to Cracow Grid Workshop (CGW) 2014.