# Quantum Gate Cost Estimations for Shor's Algorithm on ECDLP

**Huzeifah Mohammed**
BSc in Mathematics and Computer Science


University of Birmingham
School of Computer Science
Christophe Petit
1567467
08/11/2017

# Quantum Gate Cost Estimations for Shor's Algorithm on ECDLP

## Abstract

This thesis provides an estimate for the total number quantum gates required for elliptic curve point addition and doubling. These estimates are then used to calculate first order estimates for the resources required to solve Shor's Elliptic Curve Discrete Logarithm Problem (ECDLP). This thesis focuses on Edward curves and builds upon the work by Roetteler et al. [23]. We created multiple algorithms, using different coordinate systems, to compute point addition and point doubling to see which would be the most efficient in a quantum environment. From our estimates we determine our point doubling algorithm, using projective coordinates, can be used to compute elliptic curve discrete logarithms using a total of $450n^3log_2n$ Toffoli gates. Our estimate is slightly more expensive than the Weierstrass curve estimate which required a total of $448n^3log_2n$ quantum gates.

**All software for this project can be found at:**

`https://git-teaching.cs.bham.ac.uk/mod-ug-proj-2017/hxm567.git`

**Acknowledgements**

I would like to thank Dr. Christophe Petit for his advice, supervision and guidance over this project.

# Contents

# Chapter 1

# Introduction

## 1.1 Background

The development of Quantum computers in recent years has shown how critical research in the area is. Scientists are edging closer towards a fully functional and useful quantum computer, however the engineering required is very complex. Currently, quantum computers with very few qubits exist so they are not very practical. Perhaps one of the biggest difficulties in creating a useful quantum computer is the effects of quantum decoherence. Quantum decoherence cannot be eliminated completely and causes the decay of quantum coherence, which leads to the quantum behaviour of the system being lost [21]. Possible sources of interference can come from magnetic/electric fields required to operate the machine. Research is taking place on how quantum decoherence can be reduced.

Algorithms that run on quantum computers are much faster at solving certain problems compared to classical algorithms, but there are no quantum computers large enough to run these algorithms. For example, the largest number factored on a quantum system is 56153. As vast amounts of research continues in this area, a quantum computer that can compute these algorithms may not be as far off as we think. Along with Shor's algorithm, other algorithm's such as Simon's and Grover's algorithms will be researched in this project.

## 1.2 Threat quantum algorithms pose to cryptanalysis

Quantum algorithms pose a great threat to public key cryptography as in certain circumstances, they provide an exponential speedup compared to classical algorithms. With the discovery of Shor's quantum algorithm, there was a massive surge of research in quantum computing. Shor's algorithm is able to perform prime factorization of very large numbers in polynomial time, a feat never thought possible with classical algorithms. This left the cryptographic community in a state of uncertainty, as integer factorization is core to public key cryptography. RSA, which is the most commonly used public key cryptosystem, relies on the fact that factoring large numbers is infeasible. However as explained above, with the introduction of quantum algorithms this is no longer the case. Shor's algorithm, using a large quantum computer, is able to crack the RSA cryptosystem in polynomial time. It

is also easily able to solve other popular algorithms used for security such as the discrete logarithm problem or the elliptic-curve discrete logarithm problem. Due to the advancements of quantum algorithms, the cryptographic community have looked into cryptographic algorithms that are thought to be secure from quantum attacks. This area of research is known as post-quantum cryptography.

## 1.3 Elliptic Curve Cryptography

The main focus area of this project is based on another cryptosystem called Elliptic Curve Cryptography (ECC). Elliptic curves have many applications such as encryption, digital signatures, Bitcoin, the Tor network and many more [23]. There are also many different curves with different sizes and forms, that have different levels of security. The general form of elliptic curves used in ECC is known as the Weierstrass curve, however this paper will focus on Edward curves. Edward curves have many advantages over Weierstrass curves such as its countermeasures against side attacks and more efficient algorithms for various operations.

The Elliptic Curve Discrete Logarithm Problem (ECDLP) is based on the fact that computing the discrete logarithm for an elliptic curve is very difficult. Currently the best algorithm to solve ECDLP is exponential in complexity. Elliptic curves are sometimes used over other schemes such as RSA, as it requires much shorter key sizes for the same level of security [20]. An example is given in [23] that states, "according to NIST recommendations from 2016, a 256-bit elliptic curve provides a similar resistance against classical attackers as an RSA modulus of size 3072 bits". Our project looked to apply Shor's algorithm to ECDLP and estimate the number of resources required to do so.

## 1.4 Methodology

A lot of research was required before we could tackle the problem. At first, we researched existing quantum algorithms and looked at how they worked. Three of the most significant algorithms can be found in Section 2 where we give descriptions of each algorithm. This gave us a good understanding of the whole area of quantum computing. We then researched more relevant work such as elliptic curve cryptography and existing work on solving Shor's ECDLP, which can be found in Section 3. After completing the research, there were a few different approaches we were considering to tackle the question. At first we thought of implementing a quantum circuit using quantum simulation software, such LIQU$i|\rangle$, that would be able to simulate Shor's ECDLP on elliptic curves, similar to [23]. However, as we had no prior knowledge on using quantum simulation software, implementing Shor's ECDLP would have been very time consuming and there was a fear we would be unable to implement it in time. Although implementing the quantum circuit would have made it easier to help locate where the resources could have been lowered, it would not have directly answered the question.

Another approach was to concentrate on Weierstrass curves and find methods to reduce the estimates found in [23]. We thought it would be more constructive to focus on another curve, that classically uses fewer resources and apply the curve to a quantum algorithm so we could then compare results with the Weierstrass

curve estimates and draw further conclusions. After choosing Edward curves (as classically, it uses fewer resources for most operations), we created an algorithm to solve Shor's ECDLP using modular arithmetic operations and calculated estimates for the cost of the algorithm. The modular arithmetic operation estimates from [23] were used to calculate the cost, as the estimates in [23] were accurate due to the quantum circuits for each operation being simulated in LIQU$i|\rangle$, which also calculated the cost. It would also provide a fair comparison when comparing our results with the results from [23], as the cost for the operations would be the same, so we would clearly be able to see which algorithm (and curve) is more efficient. We then looked to reduce our estimate, mainly by using different coordinate systems to represent the elliptic curve. We created different algorithms for each coordinate system and compared the results.

*SageMath* was used to test the algorithms and ensure the output for each algorithm was correct. It has a feature that allows the output to be printed in LaTeX. This was extremely helpful while writing up the algorithms in this thesis, as the output was simply copied and pasted into the thesis. Typing the output in LaTeXwould have been very tedious. Our working, algorithms and results can be found in Section 4. In Section 5, we conclude our thesis and provide some insight as to what future work could delve into.

# Chapter 2

# Quantum Algorithms

## 2.1 Quantum Computation

### 2.1.1 Difference between quantum and classical systems

Quantum systems use vastly different methods for computation compared to classical systems. One of these methods is known as the superposition principle of quantum mechanics, however this will be explained in the next subsection. Another method, known as quantum entanglement, is a fundamental trait of quantum systems. The phenomenon occurs when two distinct elements of a system cannot be described without taking the other part into consideration i.e. the quantum state of each particle cannot be described independently of the other. In a quantum system, the probability of observing a certain configuration of two qubits may depend on the probability of observing a different configuration of the two qubits. One interesting quality of entanglement is two particles can still be entangled irrespective of the distance between the two particles, even if they are separated by a huge distance [29]. Entanglement is crucial for many techniques such as superdense coding, which involves increasing the rate at which information can be sent through a quantum channel. The goal is to be able to send 2 bits of classical information using only one qubit [32].

One difference between classical and quantum systems is classical systems are deterministic, whereas quantum systems are probabilistic. A deterministic system is a system that involves no randomness. The output states are determined by the input states. On the contrary, the output of a probabilistic system cannot be perfectly predicted. With regards to a quantum system, probabilities for each output state are computed. The aim is to increase the probability of observing the correct state so we find an answer with a reasonable amount of certainty.

### 2.1.2 Qubits and superposition

Superposition is key to quantum computing. Classical systems utilize bits, which are in the states 0 or 1, whereas quantum systems utilize qubits. Instead of taking a distinct value, qubits exist in a superposition of all its possible values. Therefore, a quantum system is in all of its possible states at the same time. If we have $n$ qubits, we can simultaneously represent $2^n$ states. This allows quantum systems to solve some computations much faster than classical systems. When the superposition is

observed, it collapses into a classical state.

The superposition of a qubit $\psi$, in its uncollapsed state, can be written as a linear combination of the basis vectors $|0\rangle$ and $|1\rangle$.

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

Where $\alpha$ and $\beta$ are the complex scalar amplitudes of measuring $|0\rangle$ and $|1\rangle$ respectively. The amplitudes represent the probability that a specific state will be observed, when the superposition has collapsed [29]. In classical algorithms, the probabilities are represented by real numbers whereas in quantum systems they are represented by complex numbers. The probability of a qubit being in the state $|0\rangle$ is $|\alpha|^2$ and the probability of it being in the state $|1\rangle$ is $|\beta|^2$. As probabilities sum to 1, we get:

$$|\alpha|^2 + |\beta|^2 = 1$$

### 2.1.3 Quantum Fourier Transform

We will start this section stating the definition of a Discrete Fourier Transform. Let $f = (f_0, ..., f_{N-1})$ be a vector in $\mathbb{C}^2$. The Fourier Transform maps $f$ to the vector $y = (y_0, ..., y_{N-1})$ defined by:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \zeta^{kj} f_j$$

where $\zeta = exp(\frac{2\pi i}{N})$ is the $N^{\text{th}}$ root of unity [16].
The Fourier Transform allows us to find the periodic behaviour of a function, which is a necessity for Shor's algorithm and many more algorithms.
The quantum Fourier transform works in a similar manner. It will act on a quantum state $\sum_{i=0}^{N-1} x_i\,|i\rangle$ and will map it to a quantum state $\sum_{i=0}^{N-1} y_i\,|i\rangle$ using

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \zeta^{jk}$$

where $\zeta = exp(\frac{2\pi i}{N})$ is the $N^{\text{th}}$ root of unity [16].

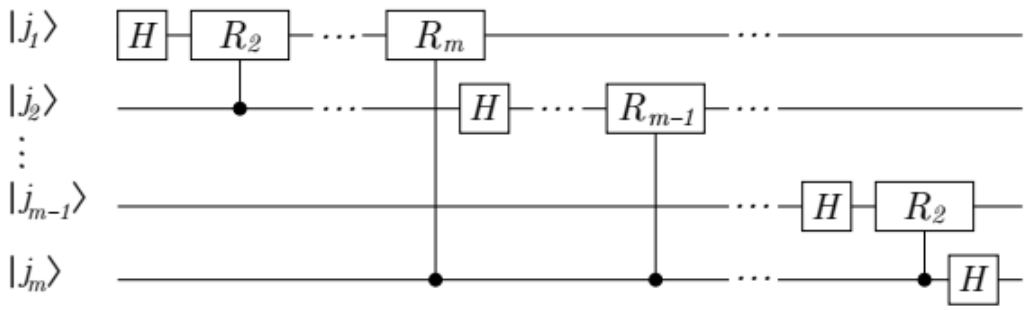


Figure 2.1: Circuit diagram of quantum Fourier transform [16]

## 2.2 Quantum gates

Quantum computing makes use of many gates. One gate in particular is known as the quantum CNOT gate, which is similar to the classical CNOT gate. The CNOT gate swaps the amplitudes of the $|0\rangle$ and $|1\rangle$ basis states of a qubit if the controlling qubit has the value $|1\rangle$ [29]. The CNOT gate is an example of a control operation, which are commonly used in quantum computing. Control operations are able to "change the state of a qubit based on the values of other qubits" [29]. The Toffoli gate is another example of a control operation and consists of two control qubits and a target qubit. "If both control qubits are set, then the amplitudes of the target qubit are flipped" [29]. The Toffoli gate can also compute operations such as AND, XOR and NOT, depending on the input, and can be used for classical and quantum circuits.

In [23], all the reversible computations are based on the Toffoli gate, $|x, y, z\rangle \rightarrow |x, y, z \oplus xy\rangle$, which is universal for reversible computing. The authors have stated two reasons for this. Firstly, the Toffoli gate can be implemented exactly over the Clifford+$T$ gate set. The Clifford+$T$ gate is universal and consists of a Hadamard gate $H$, a phase gate $P$, a CNOT gate along with a $T$ gate. Secondly, it is easier to test and debug Toffoli gate circuits. They can also be simulated using classical reversible simulators.

## 2.3 Simon's Algorithm

### 2.3.1 The Algorithm

Simon's algorithm [26], is a prime example of when quantum algorithms are exponentially faster than classical algorithms. It is given as follows:

Given a function

$$f : \{0,1\}^n \mapsto \{0,1\}^n$$

find s given

$$[f(x) = f(y)] \leftrightarrow [x \oplus y \in \{0^n, s\}]$$

Simon's algorithm does not have many uses in a real-life situation as the circumstances in which it can be applied are completely artificial. It was one of the first algorithms to show the advantages quantum algorithms have over classical algorithms. A classical algorithm requires $O(2^{n/2})$ queries to $f$ to find $s$. Simon's algorithm only requires $O(n)$ queries to find $s$ [24].

It is clear that this problem is very difficult to solve using a classical algorithm. This is because $s$ cannot be determined until it finds two inputs $x$ and $y$ such that $f(x) = f(y)$. Therefore, all it can do is input random $x$'s until it finds a match. The probability of this happening is very small indeed [27].

### 2.3.2 Algorithm Explained

We will now provide a description of how Simon's algorithm works using the information provided by [29] and [34]. The algorithm uses two registers with $n$ qubits. We start off by initializing the circuit to the state:

$$|0^n\rangle\,|0^n\rangle$$

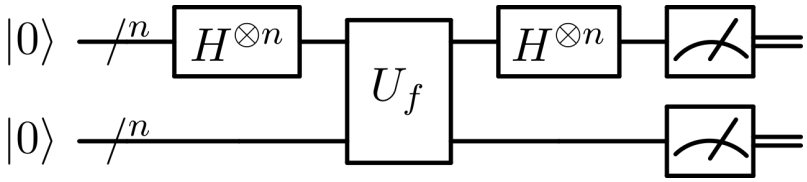


Figure 2.2: Circuit diagram of Simon's algorithm [28]

We then apply the Hadamard transform to the first register producing the superposition:

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

The unitary transformation $U_f$ performs the operation:

$$U_f |x\rangle |y\rangle = |x\rangle |f(x) \oplus y\rangle$$

which results in no change to the first register and $f(x)$ stored in the second register. This gives:

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |f(x)\rangle$$

We then measure the second register. It collapses to $|f(z)\rangle$ and the first register is now in the superposition:

$$\frac{1}{\sqrt{2}}(|z\rangle + |z \oplus s\rangle)$$

From this we must calculate $s$, which is difficult because we cannot measure the state as the superposition will collapse [9]. We now apply the Hadamard Transform again so we can gain information about $s$:

$$H^{\otimes n} \left[ \frac{1}{\sqrt{2}} |z\rangle + \frac{1}{\sqrt{2}} |z \oplus s\rangle \right]$$

$$\frac{1}{\sqrt{2}} \left[ \frac{1}{2^{n/2}} \sum_{y \in \{0,1\}^n} (-1)^{z.y} |y\rangle \right] + \frac{1}{\sqrt{2}} \left[ \frac{1}{2^{n/2}} \sum_{y \in \{0,1\}^n} (-1)^{(z \oplus s).y} |y\rangle \right]$$

$$\frac{1}{\sqrt{2}} \frac{1}{2^{n/2}} \sum_{y \in \{0,1\}^n} (-1)^{z.y} |y\rangle + \frac{1}{\sqrt{2}} \frac{1}{2^{n/2}} \sum_{y \in \{0,1\}^n} (-1)^{(z \oplus s).y} |y\rangle$$

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{y \in \{0,1\}^n} \left[ (-1)^{z.y} + (-1)^{(z \oplus s).y} \right] |y\rangle$$

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{y \in \{0,1\}^n} \left[ (-1)^{z.y} + (-1)^{(z.y) \oplus (s.y)} \right] |y\rangle$$

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{y \in \{0,1\}^n} (-1)^{z.y} \left[ 1 + (-1)^{(s.y)} \right] |y\rangle$$

This leaves two cases. Either $y \cdot s = 1$ and $y \cdot s = 0$. If $y \cdot s = 1$ then the equation simplifies to:

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{y \in \{0,1\}^n} (-1)^{z.y} \left[ 1 + (-1)^{1} \right] |y\rangle$$

$$= \frac{1}{\sqrt{2^{n+1}}} \sum_{y \in \{0,1\}^n} (-1)^{z.y} [0] |y\rangle = 0 |y\rangle$$

Therefore, the probability of measuring such a $y$ is 0. When we measure the first register, with certainty we will see a random $y$, such that $y \cdot s = 0$ and each $y$ has an equal probability of occuring. This simplifies the equation to:

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{y \in \{0,1\}^n} (-1)^{z.y} \left[1 + (-1)^0\right] |y\rangle$$

$$= \frac{1}{\sqrt{2^{n-1}}} \sum_{y \in \{0,1\}^n} (-1)^{z.y} |y\rangle$$

After computing the algorithm, we will get a random $y$ such that $s \cdot y = s_1 y_1 + ... + s_n y_n = 0$. To find $s$, we must repeat the algorithm $n-1$ times to obtain $n-1$ linearly independent equations. From here, we should be able to classically solve the system of equations to find $s$. A quantum computer would be able to solve this problem; however, it may be more beneficial if a classical computer was used to solve the linear system of equations. This is because, there are classical algorithms already available that can solve this problem in polynomial time, while using a quantum system may be quite expensive [29].

## 2.4 Shor's Algorithm

The importance of Shor's Algorithm has been mentioned previously. Shor's algorithm [25] can factor an integer $n$ in $O((logn)^2(loglogn)(logloglogn))$ [17], which is exponentially faster than any classical algorithm. The best classical algorithm is known as the number field sieve, which can factor an integer $n$ in $O(\exp(logn^{1/3}loglogn^{2/3}))$ [17]. Instead of looking for factors directly, Shor's algorithm uses a property of factoring known as period finding. The period is found with a quantum Fourier transform which provides the quantum speedup of the algorithm [5]. Our description of Shor's algorithm is split into two parts (the factoring problem and period finding), whilst using the information provided by [5], [17], [8], [31] and [12].

### 2.4.1 The Factoring Problem

The problem Shor's algorithm solves is, given an odd composite integer $n$ find its prime factors. The first part of Shor's algorithm is about reducing a factoring problem into a period finding problem.

We start off by checking if $n$ is even, prime or a prime power. If so Shor's algorithm will not be required to find its factors as there are efficient classical algorithms that can be used. We pick a random integer $x < n$ and calculate $\gcd(x, n)$. If $\gcd(x, n) \neq 1$ then we have found a factor of $n$ and no further steps are required. If $\gcd(x, n) = 1$, given the periodic function $f(x) = x^a \bmod n$ we must use a quantum computer to find the period $r$. We know $x^0 \bmod n = 1$, $x^r \bmod n = 1$, $x^{2r} \bmod n = 1$... as the function is periodic. We deduce

$$x^r \equiv 1 \bmod n$$

$$(x^{r/2})^2 = x^r \equiv 1 \bmod n$$

$$(x^{r/2})^2 - 1 \equiv 0 \bmod n$$

Now when $r$ is even, $(x^{r/2}-1)(x^{r/2}+1) \equiv 0 \bmod n$ and $(x^{r/2}-1)(x^{r/2}+1)$ is a multiple of $n$. Therefore if $x^{r/2} \neq \pm 1$ then either $(x^{r/2}-1)$ or $(x^{r/2}+1)$ must have a common factor with $n$. Then, if we were to calculate $\gcd(x^{r/2}-1, n)$ and $\gcd(x^{r/2}+1, n)$ we obtain a factor of $n$ [12].

Shor's algorithm looks to find the period $r$ of $x^a \bmod n$, where $x$ is coprime to $n$. We choose $a$'s to be integers from 0 to $q-1$. $q$ is a power of two such that $n^2 \leq q < 2n^2$. Now we must use the period finding routine to find the period.

### 2.4.2 Period Finding

The second part of Shor's algorithm involves using a quantum Fourier transform to the find the period. We begin by initializing two qubit registers. The first register should have enough qubits to represent integers as large as $q-1$, while the second register should have enough qubits to represent integers as large as $n-1$.

$$|\psi\rangle = |0\rangle\,|0\rangle$$

Now we apply the quantum Fourier transform to the first register. Recall that the quantum Fourier transform is as follows:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \zeta^{jk}$$

where $\zeta = exp(\frac{2\pi i}{N})$. The first register holds all integers $0, 1, 2, ...., q-1$. The current state of the system is:

$$|\psi\rangle = \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a, 0\rangle$$

We now apply the unitary operator $U_f$ to the registers. $U_f$ applies the function $f(a) = x^a \bmod n$. After calculating $f(a)$ for each integer in the first register, the results are stored in the second register:

$$U_f\,|a\rangle\,|0\rangle = |a\rangle\,|f(a)\rangle$$

The state of the system is now:

$$\frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a, 0\rangle \xrightarrow{U_f} \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a, x^a \bmod n\rangle$$

Now we measure the second register which gives $f(a) = u$, and collapses to the state $|u\rangle$. The first register also collapses into a superposition of each $a$ such that $f(a) = u$. We will call this set of values $A$. The state of the first register is now:

$$\frac{1}{\sqrt{A}} \sum |a\rangle$$

As $f$ is periodic, the system can be written as:

$$|\psi\rangle = \frac{1}{\sqrt{A}} \sum_{k=0}^{A-1} |a_0 + k \cdot r\rangle \, |u\rangle$$

Now we apply the quantum Fourier transform giving:

$$|\psi\rangle = \frac{1}{\sqrt{A}} \sum_{k=0}^{A-1} \frac{1}{\sqrt{q}} \sum_{y=0}^{q-1} exp\left(\frac{2\pi i(a_0 + k \cdot r)y}{q}\right) |y\rangle$$

$$= \frac{1}{\sqrt{Aq}} \sum_{y=0}^{q-1} exp\left(\frac{2\pi i a_0 y}{q}\right) \sum_{k=0}^{A-1} exp\left(\frac{2\pi i k r y}{q}\right) |y\rangle$$

We can now measure the first register, getting the integer $\lambda$, which has a high probability of being a multiple of $q/r$. The period $r$ can now be calculated classically, hence the Euclidean algorithm can be used to find the factor $z$ where $z = gcd(x^{r/2} - 1, n)$.

### 2.4.3 Summary of Shor's Factoring Algorithm

1. Check if $n$ is even, prime or a prime power using a classical computer. If it is, Shor's algorithm is not required as an efficient classical algorithm can be used. Choose a random integer $x < n$. Use the Euclidean algorithm to calculate $\gcd(x, n)$. If $\gcd(x, n) \neq 1$ then we have found a factor of $n$ and can exit. Else go to 2.
2. Choose $q$ to be the smallest power of 2 such that $n^2 < q < 2n^2$. Find the period $r$ of $x^a \bmod n$.
3. If $r$ is odd, then go back to 1. and try again. Else go to 4.
4. If $x^{r/2} + 1 = 0 \bmod n$ then go back to 1. If $x^{r/2} + 1 \neq 0 \bmod n$ then go to 5.
5. Use the euclidean algorithm to find $z = \gcd(x^{r/2} - 1, n)$, where $z$ is the solution.

### 2.4.4 Shor's Discrete Logarithm Algorithm

In his paper [25], Shor has also introduced an algorithm for solving discrete logarithms in finite fields. [20] has described the method in which the algorithm can be applied to elliptic curves. This will be discussed in Section 3.

## 2.5 Grover's Algorithm

### 2.5.1 Time complexity

Searching algorithms are hugely important in computer science and have many applications, such as searching through a database. Unfortunately, classically the only way to perform a search is to systematically search through every element until the solution has been found. Therefore if there are $N$ elements, in the worst-case the time taken to complete a search is $O(N)$ and on average $O(N/2)$. Grover's algorithm provides a quadratic speedup and works in time $O(\sqrt{N})$ [10]. This is possible as it takes advantage of the principle of superposition.

### 2.5.2 The Quantum Oracle

Suppose we have a list with $N$ elements. Each element has index $x$ and is configured as follows: $0, 1, ..., N-1$. We assume $N = 2^n$ so the index can be stored in $n$ qubits. Our search problem has $M$ solutions, $1 \leq M \leq N$. Now we introduce a quantum oracle. The quantum oracle is a black-box function that is able to recognise the solutions to the search problem. We define some function $f(x)$ as follows [22]:

$$f(x) = \begin{cases} 0 & \text{if } x \text{ is not a solution} \\ 1 & \text{if } x \text{ is a solution} \end{cases}$$

The quantum oracle takes in an index value $x$ and an "Oracle qubit" $|q\rangle$. The oracle $O$ sets $f(x) = 1$ if a solution was indexed and sets $f(x) = 0$ otherwise. If $f(x) = 1$, the oracle flips the state of $|q\rangle$. This can be written as:

$$|x\rangle |q\rangle \xrightarrow{O} |x\rangle |q \oplus f(x)\rangle$$

If we initialise $|q\rangle$ as $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ then the equation can be written as

$$|x\rangle \left[\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right] \xrightarrow{O} (-1)^{f(x)} |x\rangle \left[\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right]$$

Recall, $(-1)^{f(x)}$ flips the qubit if $f(x) = 1$ and remains unchanged if $f(x) = 0$. As the oracle qubit never changes, it can be omitted. Therefore oracle can be written as:

$$|x\rangle \xrightarrow{O} (-1)^{f(x)} |x\rangle$$

### 2.5.3 The Grover Iteration

To describe the Grover Iteration, we have used information from [30], [19] and [29]. We begin with the initial state:

$$|0\rangle^{\otimes n} = |0\rangle$$

We then apply the Hadamard Transform to put the system into an equal superposition of states:

$$|\psi\rangle = H^{\otimes n} |0\rangle^{\otimes n} = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$

We now refer to the *diffusion transform*, "which performs inversion about the average, transforming the amplitude". The diffusion transform applies a Hadamard Transform. It then applies a conditional phase shift that shifts every state, except for $|0\rangle$, by $-1$. The phase shift is represented by the operator:

$$2 |0\rangle \langle 0| - I$$

which has two possibilities. Either:

$$2 |0\rangle \langle 0| - I |0\rangle = 2 |0\rangle \langle 0|0\rangle - I = |0\rangle$$

Or:

$$2 |0\rangle \langle 0| - I |x\rangle = 2 |0\rangle \langle 0|x\rangle - I = - |x\rangle$$

After the phase shift we apply another Hadamard transform. The entire process is as follows:

$$H^{\otimes n}(2\left|0\right\rangle\left\langle 0\right| - I)H^{\otimes n} = 2H^{\otimes n}\left|0\right\rangle\left\langle 0\right|H^{\otimes n} - I = 2\left|\psi\right\rangle\left\langle\psi\right| - I$$

The entire Grover Iteration is given as follows:

$$(2\left|\psi\right\rangle\left\langle\psi\right| - I)O$$

### 2.5.4 Summary of Grover's Algorithm

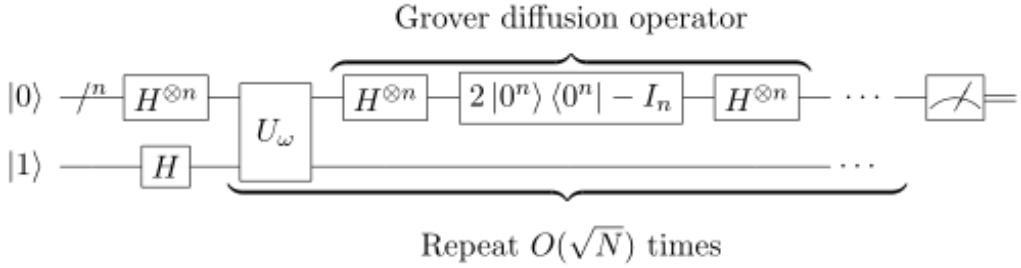


Figure 2.3: Circuit diagram of Grover's Algorithm [11], [1]

Grover's Algorithm has been summarized in [19] as follows:
Input:

- A quantum oracle which performs the operation $\left|x\right\rangle\left|q\right\rangle \xrightarrow{O} \left|x\right\rangle\left|q \oplus f(x)\right\rangle$
- $n$ qubits initialised to the state $\left|0\right\rangle$

Output: $x_0$

1. Initial state: $\left|0\right\rangle^{\otimes n}$
2. Apply Hadamard Transform: $\left|\psi\right\rangle = H^{\otimes n}\left|0\right\rangle^{\otimes n} = \frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}\left|x\right\rangle$
3. Apply Grover Iteration, (repeat as many times as required): $(2\left|\psi\right\rangle\left\langle\psi\right| - I)O$
4. Measure the register $x_0$

# Chapter 3

# Key Material from Previous Work

This chapter looks at previous work related to ECDLP in a quantum environment. We will be looking at two papers in particular, **Quantum Resource Estimates for Computing Elliptic Curve Discrete Logarithms** [23] by Martin Roetteler, Michael Naehrig, Krysta M. Svore, and Kristin Lauter and **Shor's discrete logarithm quantum algorithm for elliptic curves** [20] by John Proos and Christof Zalka. The paper of Roetteler et al. focuses on Weierstrass curves, our paper looks to build upon their work and extend the estimates to Edward curves.

## 3.1 Shor's Quantum Algorithm for solving ECDLP

As mentioned in Chapter 2, along with his factoring algorithm, Shor has also provided an algorithm that is able to solve discrete logarithm problems in polynomial time. In [20], Proos and Zalka have shown how to implement this algorithm for elliptic curves. Using the information from [23] and [20], the method is given as follows:

We are given an elliptic curve $E$ over $GF(p)$ where $p$ is a large prime. The base of the logarithm is a point $P \in E$, whose order is $\text{ord}(P) = r$, which is typically a large prime. Let $Q \in \langle P \rangle$ be an element in the sub group generated by $P$. Our goal is to find $m \in \{1, ..., r\}$ such that $Q = [m]P$. We initialize two registers of length $n + 1$ qubits, in the state $|0\rangle$. We then implement the following transformation

$$\frac{1}{2^{n+1}} \sum_{k,l=0}^{2^{n+1}-1} |k, l\rangle \rightarrow \frac{1}{2^{n+1}} \sum_{k,l=0}^{2^{n+1}-1} |k, l\rangle \, |[k]P + [l]Q\rangle$$

A quantum Fourier transform is then applied to the two registers and the states are measured. From the result, the discrete logarithm can be calculated from classical computations, as shown in [25].

## 3.2 Reversible Modular Arithmetic Operations

ECDLP requires scalar multiplications, which consists of point additions and point doublings. Both of which require modular operations such as, modular addition, modular multiplication and modular inversion. Integer addition and bit shift operations are required to implement these operations. Integer addition is performed

using the circuit from [33], where two registers hold the input integers. The first register is of size $n$ and the second is of size $n+1$. The contents of the second register are then replaced to hold the sum, whilst being able to store a carry if required, due to the additional qubit [23]. The modular arithmetic operations we have used for our calculations have been taken from [23]. We will state a few of them below:

`add_modp` performs a modular addition of two integers $x$ and $y$ held in $n$-qubit quantum registers $|x\rangle$ and $|y\rangle$, modulo the constant integer modulus $p$. The second register is replaced with the result: $|x\rangle\,|y\rangle \rightarrow |x\rangle\,|(x+y) \bmod p\rangle$

`mul_modp` performs Montgomery modular multiplication [18], with the result stored in a register $|0\rangle$ which holds $n$ logical qubits: $|x\rangle\,|y\rangle\,|0\rangle \rightarrow$
$|x\rangle\,|y\rangle\,|z = x \cdot y \cdot R^{-1} \bmod p\rangle$.

`inv_modp` is performed using a circuit that implements Kaliski's algorithm [14], for inverting a number given in Montgomery form. The algorithm only uses elementary reversible operations, whose sequence of instructions does not depend on the given input $x2^n \bmod p$ and whose output is in Montgomery form [23].

`squ_modp` performs a modular squaring, using modular doublings and controlled modular addition: $|x\rangle\,|0\rangle \rightarrow |x\rangle\,|x^2 \bmod p\rangle$.

The number of Toffoli gates used for each of the modular arithmetic algorithms was calculated by determining how many constant incrementors occur. These estimates are then used to calculate the cost of Shor's ECDLP. For elliptic curve point addition, the estimate is multiplied by $2n$ as Shor's algorithm is incremented twice.

## 3.3 Cost Estimates

In [23], the cost and resources required for computing Shor's ECDLP, have been calculated by implementing the required algorithms using the quantum computing software LIQU$i\,|\rangle$. They were then able to obtain precise counts for the number of qubits, Toffoli gates and gate depth. The counts can be found below in Table 3.1. We have used their estimates for modular arithmetic operations to calculate the cost of our own algorithms. The total number of qubits required for Shor's ECDLP depends on the modular inversion circuit, as it requires the most qubits from the modular operations. The other modular arithmetic algorithms will not have an effect on the total number of qubits, as they can use the ancilla qubits provided by the inversion circuit. Due to this, [23] opted to use the Montgomery multiplication algorithm instead of the double and add algorithm, as although the Montgomery algorithm uses more qubits, it uses much less gates, lowering the cost for the entire algorithm.

### 3.3.1 Scalar Multiplication

Scalar multiplication is key to solving ECDLP. A lot of research is going into how scalar multiplication can be made more efficient via various methods. [23] have followed the approach given by Proos and Zalka [20], by classically precomputing all $n$ 2-power multiples of $P$. This allows a lot of the operations to be computed classically, while the quantum circuit will only be required for the point addition.

### 3.3.2 Elliptic curves vs RSA

Proos and Zalka have calculated estimates for the number of resources required to compute Shor's ECDLP. They have provided a table in [20] which compares their results with the resource estimates for Shor's factoring algorithm. From this they conclude that computing the factoring algorithm is much harder than computing elliptic curves for similar levels of security. The results provided by [23] also support this conclusion.

| Modular arithmetic circuit | # of qubits | | # Toffoli gates |
|---|---|---|---|
| | total | ancillas | |
| `add_const_modp`, `sub_const_modp` | $2n$ | $n$ | $16n\log_2(n) - 26.9n$ |
| `ctrl_add_const_modp`, `ctrl_sub_const_modp` | $2n+1$ | $n$ | $16n\log_2(n) - 26.9n$ |
| `ctrl_sub_modp` | $2n+4$ | 3 | $16n\log_2(n) - 23.8n$ |
| `ctrl_neg_modp` | $n+3$ | 2 | $8n\log_2(n) - 14.5n$ |
| `mul_modp` (dbl/add) | $3n+2$ | 2 | $32n^2\log_2(n) - 59.4n^2$ |
| `mul_modp` (Montgomery) | $5n+4$ | $2n+4$ | $16n^2\log_2(n) - 26.3n^2$ |
| `squ_modp` (dbl/add) | $2n+3$ | 3 | $32n^2\log_2(n) - 59.4n^2$ |
| `squ_modp` (Montgomery) | $4n+5$ | $2n+5$ | $16n^2\log_2(n) - 26.3n^2$ |
| `inv_modp` | $7n + 2\lceil\log_2(n)\rceil + 9$ | $5n + 2\lceil\log_2(n)\rceil + 9$ | $32n^2\log_2(n)$ |

Table 3.1: Number of qubits and Toffoli gates required for each operation used in the algorithms [23].

# Chapter 4

# Research Findings

## 4.1 Edwards Curves

### 4.1.1 Definition

Short Weierstrass curves and Montgomery curves are two of the most commonly used forms of elliptic curves. Edwards curve was discovered in 2007 and showed that for ECC, it required fewer multiplications compared to other curves [4].

The original equation for Edwards curve is $x^2 + y^2 = c^2(1 + x^2y^2)$ [7]. However, we will use the parameter $d$, as this allows us to cover more curves [15]. We give the equation for Edwards curve as follows:

$$E : x^2 + y^2 = c^2(1 + dx^2y^2)$$

with $d \notin \{0, 1\}$ and $(0, c)$ as the neutral element. For the rest of the paper we will assume $d$ is non-square and choose $c = 1$.

### 4.1.2 Point Addition

The equation to calculate the sum of two points $(x_1, y_1)$ and $(x_2, y_2)$ is given:

$$(x_1, y_1) + (x_2, y_2) = \left( \frac{x_1y_2 + y_1x_2}{c(1 + dx_1x_2y_1y_2)}, \frac{y_1y_2 - x_1x_2}{c(1 - dx_1x_2y_1y_2)} \right)$$

As $d$ is non-square, the denominators $1 \pm dx_1x_2y_1y_2 \neq 0$, so there are no exceptional cases. Proofs are given in [3] that show that "the Edwards addition law produces points on the curve, that the Edwards addition law corresponds to the standard addition law on a birationally equivalent elliptic curve, and that the Edwards addition law is complete when $d$ is not a square."

We have implemented 3 reversible point addition algorithms. For each algorithm, we have assumed $P_1 = (x_1, y_1)$ , $P_2 = (x_2, y_2)$ and $x_1 \neq x_2$ therefore $P_1 \neq \pm P_2$. So $P_1 + P_2 = P_3$ where $P_3 = (x_3, y_3)$. (For **Algorithm 2, 3**, we have represented the coordinates $(x, y)$ as $(X, Y)$ to differentiate between affine and projective coordinates). We assume $P_2$ is a constant point as it has been precomputed, therefore any operations involving $x_2$ and $y_2$ have been implemented as constant operations [23]. Each algorithm uses a variety of temporary variables and takes $P_1$ and $P_2$ as inputs. The output is stored in $x_3$ and $y_3$ ($X_3$ and $Y_3$ for the projective coordinate

algorithms). Each temporary variable is then uncomputed. At each step of all the algorithms, we show what is currently being held by the target variable. The cost is calculated by totalling the number of multiplications, inversions and squarings and multiplying each total by the number of gates required for each operation.

In our paper, we have focused on Edward curves due to its speedup for certain operations, compared to other curves. This can be found in [3], where the authors have comprised data for a variety of curves and tabularized the results. They conclude that Edward curves are the fastest for multi-scalar multiplication and is competitive with Montgomery curves for single-scalar multiplication. The authors have also reduced cost by rewriting certain operations. For example, $c(1 + dx_1^2y_1^2)$ can be written as $(x_1^2 + y_1^2)/c$ and $c(1 - dx_1^2y_1^2)$ can be written as $(2c^2 - (x_1^2 + y_1^2))/c$. These operations have also been implemented into our algorithms to reduce costs significantly.

One main advantage Weierstrass curves have over Edward curves, when trying to reversibly compute the sum of two points has been explained in [20]. When computing $P_1 + P_2 = P_3$, the slope $\lambda$ can be re-computed from the result $P_3$, using the equation

$$\frac{y_1 - y_2}{x_1 - x_2} = -\frac{y_3 + y_2}{x_3 - x_2}$$

[23]. This gives an advantage as it allows variables to be reused, saving a lot of operations and reducing the cost and number of variables to be computed/uncomputed. Further research can look to explore whether the same method can be applied to Edward curves.

### 4.1.3 Point Doubling

The equation to calculate the point doubling $2(x_1, y_1) = (x_1, y_1) + (x_1, y_1)$ is given as follows:

$$2(x_1, y_1) = \left(\frac{2x_1y_1}{c(1 + dx_1^2y_1^2)}, \frac{y_1^2 - x_1^2}{c(1 - dx_1^2y_1^2)}\right) = \left(\frac{2x_1y_1c}{x_1^2 + y_1^2}, \frac{(y_1^2 - x_1^2)c}{2c^2 - (x_1^2 + y_1^2)}\right)$$

Doubling occurs when $(x_1, y_1)$ and $(x_2, y_2)$ are equal. The point doubling algorithms calculate $2P_1 = 2(x_1, y_1) = P_2$ where $P_2 = (x_2, y_2)$. We have rewritten $c(1 + dx_1^2y_1^2)$ as $(x_1^2 + y_1^2)/c$ and $c(1 - dx_1^2y_1^2)$ as $2c^2 - (x_1^2 + y_1^2)$ [3]. This saves a lot of operations. **Algorithm 4,5** takes $P_1$ as input and outputs $P_2$. $P_2$ is stored in $x_2$ and $y_2$ for **Algorithm 4** while for **Algorithm 5** the result is stored in $X_2$ and $Y_2$. Similar to the point addition algorithms, each temporary variable is uncomputed so the algorithms are reversible, and the costs are computed.

## 4.2 Cost and Resource Estimates

Different coordinate systems have been provided on [2], so we created algorithms and calculated cost estimates for different coordinate systems and compared the results, to see which coordinate system is best suited in a quantum environment. Each algorithm was implemented in *SageMath* to automate calculating the cost and to check if the output for each algorithm is correct. This was done by manually comparing the output for each algorithm to the original formulae. *SageMath* was

used as it is openly available and we could easily implement the algorithms by representing the variables as a polynomial ring. This made it very easy to calculate the costs.

Below there are 5 reversible algorithms for elliptic curve point addition and doubling on Edwards curve with the cost estimate for each algorithm. The cost has been calculated using the estimates for each operation given by [23], the table of which can be found in Section 3. Each algorithm has been created using the modular arithmetic operations provided by [23] where circuits for each operation can also be found. Note some operations given in [23] make use of a control qubit. For our estimates, we are calculating the leading order coefficient for each algorithm. As mentioned previously, this is calculated by totalling the number of multiplications, inversions and squarings, none of which make use of the control qubit. Thus, the control qubit will not affect our estimates, so for simplicity we have not illustrated the outcome of operations involving the control qubit in our algorithms.

From the three point addition algorithms, **Algorithm 3** uses the least amount of Toffoli gates with $336n^2log_2(n)$ gates. It uses 2 less multiplications compared to its inverse (**Algorithm 2**). Projective coordinates were used as it massively reduces the number of modular inversions required, while increasing the number of multiplications and additions. Typically, inversions are very costly and are usually avoided if possible. In our situation, a modular inversion is twice as costly as a modular multiplication.

We have two algorithms for point doubling, one using affine coordinates whilst the other uses projective coordinates. **Algorithm 5** uses the least number of gates with $240n^3log_2n$ Toffoli gates. **Algorithm 4** uses much more, with an extra $80n^2log_2n$ quantum gates. This was due to **Algorithm 5** requiring no inversions, whereas **Algorithm 4** used 4 inversions.

The total number of gates required for Shor's ECDLP is calculated by multiplying the estimates for the point addition algorithms by $2n$. This is because the point addition is iterated $2n$ times. For our point doubling algorithms, we multiply our estimates by $3n/2$, as the scalar multiple will be calculated using the double and add algorithm, which requires $n$ doublings and $n/2$ additions. We also have to factor in the difference in cost between addition and doubling (16%) to get an accurate estimate. Out of the 5 algorithms, the projective point doubling algorithm (**Algorithm 5**) required the least number of gates with an estimate of $450n^3log_2n$, whilst the inverse projective point addition algorithm (**Algorithm 2**) used the most with an estimate of $768n^3log_2n$.

### 4.2.1 Comparison

In a classical environment, using projective coordinates rather than affine coordinates, would usually result in a much more cost-effective algorithm due to the reasons mentioned above. However, in a quantum environment, a major obstacle is having to compute with a lot of temporary variables, which must then also be uncomputed. This resulted in a lot more modular multiplication, and thus we did not see a big cost difference between using affine or projective coordinates. **Algorithm 2** actually required more gates compared to **Algorithm 1**, even though no inversions were required whereas **Algorithm 1** used 4 inversions. Note, one inversion will be required when computing the scalar multiple for the projective coordinate

algorithms. This is to revert to affine coordinates. As the inversion is only applied once all the computations for scalar multiplication have been computed, the cost of the inversion has been omitted from the algorithms, but is included in the total cost for scalar multiplication and Shor's ECDLP. Below we have presented a table comprising of our results.

Although **Algorithm 3** uses less Toffoli gates than **Algorithm 1**, when estimating the total number of gates for Shor's ECDLP they both require the same number of gates. Although with **Algorithm 3** only one inversion is required (compared with 4 for **Algorithm 1**) to compute the total for Shor's ECDLP, we use many more variables and compute more multiplications. As suggested in the next section, if the cost for modular multiplication can be reduced, the total would be significantly lower.

The advantages of using projective coordinates rather than affine coordinates can be seen when comparing the point doubling algorithms. We were also able to reduce the number of multiplications in **Algorithm 5** by computing many additions/subtractions instead of multiplications. We were unable to replicate this for the point addition algorithms, which is why using projective coordinates didn't reduce the cost by much. In fact the cost for **Algorithm 2** increased. Future work could look at avoiding modular multiplications and computing modular additions/subtractions instead.

| Algorithm | # Toffoli gates | # Gates required for Shor's ECDLP |
|---|---|---|
| **Algorithm 1** | $352n^2log_2n$ | $704n^3log_2n$ |
| **Algorithm 2** | $368n^2log_2n$ | $768n^3log_2n$ |
| **Algorithm 3** | $336n^2log_2n$ | $704n^3log_2n$ |
| **Algorithm 4** | $320n^2log_2n$ | $557n^3log_2n$ |
| **Algorithm 5** | $240n^2log_2n$ | $450n^3log_2n$ |

Table 4.1: Summary of total number of gates required for each algorithm with total required for Shor's Algorithm

**Algorithm 1** Point addition - Affine coordinates (4I + 14M = $352n^2log_2(n)$). Uses affine coordinates with a total 4 inverters and 14 multipliers. The leading order coefficient for a single point addition is $4 \cdot 32 + 14 \cdot 16 = 352n^2log_2n$.

| | | |
|---|---|---|
| 1: | `mul_modp` $x_1$ $x_2$ $a$; | // $a \leftarrow x_1x_2$ |
| 2: | `mul_modp` $y_1$ $y_2$ $b$; | // $b \leftarrow y_1y_2$ |
| 3: | `mul_modp` $a$ $b$ $c$; | // $c \leftarrow x_1x_2y_1y_2$ |
| 4: | `mul_modp` $c$ $d$ $h$; | // $h \leftarrow x_1x_2y_1y_2d$ |
| 5: | `mul_modp` $c$ $d$ $i$; | // $i \leftarrow x_1x_2y_1y_2d$ |
| 6: | `add_modp` $x_1$ $y_1$; | // $x_1 \leftarrow x_1 + y_1$ |
| 7: | `add_const_modp` $x_2$ $y_2$; | // $x_2 \leftarrow x_2 + y_2$ |
| 8: | `mul_modp` $x_1$ $x_2$ $e$; | // $e \leftarrow x_1x_2 + x_2y_1 + x_1y_2 + y_1y_2$ |
| 9: | `sub_modp` $x_1$ $y_1$; | // $x_1 \leftarrow x_1$ |
| 10: | `sub_const_modp` $x_2$ $y_2$; | // $x_2 \leftarrow x_2$ |
| 11: | `sub_modp` $e$ $a$; | // $e \leftarrow x_2y_1 + x_1y_2 + y_1y_2$ |
| 12: | `sub_modp` $e$ $b$; | // $e \leftarrow x_2y_1 + x_1y_2$ |
| 13: | `add_const_modp` 1 $h$; | // $h \leftarrow x_1x_2y_1y_2d + 1$ |
| 14: | `sub_const_modp` 1 $i$; | // $i \leftarrow -x_1x_2y_1y_2d + 1$ |
| 15: | `inv_modp` $h$ $t$; | // $t \leftarrow \frac{1}{x_1x_2y_1y_2d+1}$ |
| 16: | `inv_modp` $i$ $u$; | // $u \leftarrow \frac{1}{-x_1x_2y_1y_2d+1}$ |
| 17: | `sub_modp` $b$ $a$; | // $b \leftarrow -x_1x_2 + y_1y_2$ |
| 18: | `mul_modp` $t$ $e$ $x_3$; | // $x_3 \leftarrow$ `Answer` |
| 19: | `mul_modp` $u$ $b$ $y_3$; | // $y_3 \leftarrow$ `Answer` |
| 20: | | // Reverse states |
| 21: | `add_modp` $b$ $a$; | // $b \leftarrow b$ |
| 22: | `inv_modp` $i$ $u$; | // $u \leftarrow 0$ |
| 23: | `inv_modp` $h$ $t$; | // $t \leftarrow 0$ |
| 24: | `add_const_modp` 1 $i$; | // $i \leftarrow i$ |
| 25: | `mul_modp` $d$ $c$ $i$; | // $i \leftarrow 0$ |
| 26: | `sub_const_modp` 1 $h$; | // $h \leftarrow h$ |
| 27: | `mul_modp` $d$ $c$ $h$; | // $h \leftarrow 0$ |
| 28: | `add_modp` $e$ $b$; | // $e \leftarrow e - a$ |
| 29: | `add_modp` $e$ $a$; | // $e \leftarrow e$ |
| 30: | `add_modp` $x_1$ $y_1$; | // $x_1 \leftarrow x_1 + y_1$ |
| 31: | `add_const_modp` $x_2$ $y_2$; | // $x_2 \leftarrow x_2 + y_2$ |
| 32: | `mul_modp` $x_2$ $x_1$ $e$; | // $e \leftarrow 0$ |
| 33: | `sub_modp` $x_1$ $y_1$; | // $x_1 \leftarrow x_1$ |
| 34: | `sub_const_modp` $x_2$ $y_2$; | // $x_2 \leftarrow x_2$ |
| 35: | `mul_modp` $b$ $a$ $c$; | // $c \leftarrow 0$ |
| 36: | `add_modp` $b$ $a$; | // $b \leftarrow 0$ |
| 37: | `mul_modp` $y_2$ $y_1$ $b$; | // $a \leftarrow 0$ |
| 38: | `mul_modp` $x_2$ $x_1$ $a$; | |

**Algorithm 2** Point addition - Projective coordinates (21M + 2S = $368n^2log_2(n)$). We represent $x$ as $X/Z$ and $y$ as $Y/Z$. This allows us to avoid using any inverters, until we need to revert back to affine coordinates which will require 1 inversion. In total this algorithm uses 21 multiplications and 2 squarings. So an estimate to the total number of gates is $21 \cdot 16 + 2 \cdot 16 = 368n^2log_2 n$.

| | | |
|---|---|---|
| 1: `mul_modp` $Z_1$ $Z_2$ $A$; | // | $A \leftarrow Z_1 Z_2$ |
| 2: `squ_modp` $A$ $B$; | // | $B \leftarrow Z_1^2 Z_2^2$ |
| 3: `mul_modp` $X_1$ $X_2$ $C$; | // | $C \leftarrow X_1 X_2$ |
| 4: `mul_modp` $Y_1$ $Y_2$ $D$; | // | $D \leftarrow Y_1 Y_2$ |
| 5: `mul_modp` $C$ $D$ $E$; | // | $E \leftarrow X_1 X_2 Y_1 Y_2$ |
| 6: `mul_modp` $d$ $E$ $F$; | // | $F \leftarrow X_1 X_2 Y_1 Y_2 d$ |
| 7: `mul_modp` $d$ $E$ $F_2$; | // | $F_2 \leftarrow X_1 X_2 Y_1 Y_2 d$ |
| 8: `sub_modp` $F_2$ $B$; | // | $F_2 \leftarrow -X_1 X_2 Y_1 Y_2 d + Z_1^2 Z_2^2$ |
| 9: `add_modp` $F$ $B$; | // | $F \leftarrow X_1 X_2 Y_1 Y_2 d + Z_1^2 Z_2^2$ |
| 10: `add_modp` $X_1$ $Y_1$; | // | $X_1 \leftarrow X_1 + Y_1$ |
| 11: `add_const_modp` $X_2$ $Y_2$; | // | $X_2 \leftarrow X_2 + Y_2$ |
| 12: `mul_modp` $X_1$ $X_2$ $G$; | // | $G \leftarrow X_1 X_2 + X_2 Y_1 + X_1 Y_2 + Y_1 Y_2$ |
| 13: `sub_modp` $X_1$ $Y_1$; | // | $X_1 \leftarrow X_1$ |
| 14: `sub_const_modp` $X_2$ $Y_2$; | // | $X_2 \leftarrow X_2$ |
| 15: `sub_modp` $G$ $C$; | // | $G \leftarrow X_2 Y_1 + X_1 Y_2 + Y_1 Y_2$ |
| 16: `sub_modp` $G$ $D$; | // | $G \leftarrow X_2 Y_1 + X_1 Y_2$ |
| 17: `mul_modp` $A$ $F_2$ $H$; | // | $H \leftarrow -X_1 X_2 Y_1 Y_2 Z_1 Z_2 d + Z_1^3 Z_2^3$ |
| 18: `mul_modp` $H$ $G$ $X_3$; | // | $X_3 \leftarrow$ `Answer` |
| 19: `sub_modp` $D$ $C$; | // | $D \leftarrow -X_1 X_2 + Y_1 Y_2$ |
| 20: `mul_modp` $A$ $F$ $I$; | // | $I \leftarrow X_1 X_2 Y_1 Y_2 Z_1 Z_2 d + Z_1^3 Z_2^3$ |
| 21: `mul_modp` $I$ $D$ $Y_3$; | // | $Y_3 \leftarrow$ `Answer` |
| 22: `mul_modp` $F$ $F_2$ $Z_3$; | // | $Z_3 \leftarrow$ `Answer` |
| 23: | // | Reverse states |
| 24: `mul_modp` $F$ $A$ $I$; | // | $I \leftarrow 0$ |
| 25: `mul_modp` $F_2$ $A$ $H$; | // | $H \leftarrow 0$ |
| 26: `add_modp` $G$ $D$; | // | $G \leftarrow (X_1 + Y_1)(X_2 + Y_2) - C$ |
| 27: `add_modp` $G$ $C$; | // | $G \leftarrow G$ |
| 28: `mul_modp` $X_2$ $X_1$ $G$; | // | $G \leftarrow 0$ |
| 29: `sub_modp` $F$ $B$; | // | $F \leftarrow F$ |
| 30: `mul_modp` $E$ $d$ $F$; | // | $F \leftarrow 0$ |
| 31: `add_modp` $F_2$ $B$; | // | $F_2 \leftarrow F_2$ |
| 32: `mul_modp` $E$ $d$ $F_2$; | // | $F_2 \leftarrow 0$ |
| 33: `mul_modp` $D$ $C$ $E$; | // | $E \leftarrow 0$ |
| 34: `add_modp` $D$ $C$; | // | $D \leftarrow D$ |
| 35: `mul_modp` $Y_2$ $Y_1$ $D$; | // | $D \leftarrow 0$ |
| 36: `mul_modp` $X_2$ $X_1$ $C$; | // | $C \leftarrow 0$ |
| 37: `squ_modp` $A$ $B$; | // | $B \leftarrow 0$ |
| 38: `mul_modp` $Z_2$ $Z_1$ $A$; | // | $A \leftarrow 0$ |

**Algorithm 3** Point addition - Inverted Projective coordinates (19M + 2S = $336n^2log_2(n)$). For this algorithm, we represent $x$ as $Z/X$ and $y$ as $Z/Y$. Like the previous algoroithm, projective coordinates allows us to avoid inversions. This algorithm uses 19 multiplications and 2 squarings, so the estimated number of gates is $19 \cdot 16 + 2 \cdot 16 = 336n^2log_2n$. From the 3 point addition algorithm, this algorithm requires the least amount of quantum gates.

| | | |
|---|---|---|
| 1: | `mul_modp` $Z_1$ $Z_2$ $A$; | // $A \leftarrow Z_1Z_2$ |
| 2: | `squ_modp` $A$ $B$; | // $B \leftarrow Z_1^2Z_2^2$ |
| 3: | `mul_modp` $B$ $d$ $F$; | // $F \leftarrow Z_1^2Z_2^2d$ |
| 4: | `mul_modp` $X_1$ $X_2$ $C$; | // $C \leftarrow X_1X_2$ |
| 5: | `mul_modp` $Y_1$ $Y_2$ $D$; | // $D \leftarrow Y_1Y_2$ |
| 6: | `mul_modp` $C$ $D$ $E$; | // $E \leftarrow X_1X_2Y_1Y_2$ |
| 7: | `mul_modp` $X_1$ $X_2$ $H$; | // $H \leftarrow X_1X_2$ |
| 8: | `sub_modp` $H$ $D$; | // $H \leftarrow X_1X_2 - Y_1Y_2$ |
| 9: | `add_modp` $X_1$ $Y_1$; | // $X_1 \leftarrow X_1 + Y_1$ |
| 10: | `add_const_modp` $X_2$ $Y_2$; | // $X_2 \leftarrow X_2 + Y_2$ |
| 11: | `mul_modp` $X_1$ $X_2$ $I$; | // $I \leftarrow X_1X_2 + X_2Y_1 + X_1Y_2 + Y_1Y_2$ |
| 12: | `sub_modp` $I$ $C$; | // $I \leftarrow X_2Y_1 + X_1Y_2 + Y_1Y_2$ |
| 13: | `sub_modp` $I$ $D$; | // $I \leftarrow X_2Y_1 + X_1Y_2$ |
| 14: | `add_modp` $E$ $F$; | // $E \leftarrow Z_1^2Z_2^2d + X_1X_2Y_1Y_2$ |
| 15: | `mul_modp` $E$ $H$ $X_3$; | // $X_3 \leftarrow$ `Answer` |
| 16: | `sub_modp` $E$ $F$; | // $E \leftarrow X_1X_2Y_1Y_2$ |
| 17: | `sub_modp` $E$ $F$; | // $E \leftarrow -Z_1^2Z_2^2d + X_1X_2Y_1Y_2$ |
| 18: | `mul_modp` $E$ $I$ $Y_3$; | // $Y_3 \leftarrow$ `Answer` |
| 19: | `mul_modp` $A$ $H$ $G$; | // $G \leftarrow X_1X_2Z_1Z_2 - Y_1Y_2Z_1Z_2$ |
| 20: | `mul_modp` $G$ $I$ $Z_3$; | // $Z_3 \leftarrow$ `Answer` |
| 21: | | // Reverse states |
| 22: | `mul_modp` $H$ $A$ $G$; | // $G \leftarrow 0$ |
| 23: | `add_modp` $E$ $F$; | // $E \leftarrow E$ |
| 24: | `mul_modp` $D$ $C$ $E$; | // $E \leftarrow 0$ |
| 25: | `add_modp` $I$ $D$; | // $I \leftarrow I - C$ |
| 26: | `add_modp` $I$ $C$; | // $I \leftarrow I$ |
| 27: | `mul_modp` $X_2$ $X_1$ $I$; | // $I \leftarrow 0$ |
| 28: | `sub_const_modp` $X_2$ $Y_2$; | // $X_2 \leftarrow X_2$ |
| 29: | `sub_modp` $X_1$ $Y_1$; | // $X_1 \leftarrow X_1$ |
| 30: | `add_modp` $H$ $D$; | // $H \leftarrow H$ |
| 31: | `mul_modp` $X_2$ $X_1$ $H$; | // $H \leftarrow 0$ |
| 32: | `mul_modp` $Y_2$ $Y_1$ $D$; | // $D \leftarrow 0$ |
| 33: | `mul_modp` $X_2$ $X_1$ $C$; | // $C \leftarrow 0$ |
| 34: | `mul_modp` $d$ $B$ $F$; | // $F \leftarrow 0$ |
| 35: | `squ_modp` $A$ $B$; | // $B \leftarrow 0$ |
| 36: | `mul_modp` $Z_2$ $Z_1$ $A$; | // $A \leftarrow 0$ |

**Algorithm 4** Point Doubling - Affine coordinates (6M + 6S + 4I = $320n^2log_2(n)$). Uses affine coordinates with a total 4 inverters, 6 multipliers and 4 squarings. The leading order coefficient for a single point doubling is $4{\cdot}32{+}6{\cdot}16{+}4{\cdot}16 = 320n^2log_2 n$.

| | | |
|---|---|---|
| 1: | `mul_modp` $x_1$ $y_1$ $a$; | // $a \leftarrow x_1 y_1$ |
| 2: | `mul_modp` $a$ 2 $b$; | // $b \leftarrow 2x_1 y_1$ |
| 3: | `squ_modp` $x_1$ $c$; | // $c \leftarrow x_1^2$ |
| 4: | `squ_modp` $y_1$ $d$; | // $d \leftarrow y_1^2$ |
| 5: | `add_modp` $c$ $d$; | // $c \leftarrow x_1^2 + y_1^2$ |
| 6: | `inv_modp` $c$ $e$; | // $e \leftarrow 1/x_1^2 + y_1^2$ |
| 7: | `mul_modp` $b$ $e$ $x_2$; | // $x_2 \leftarrow$ `Answer` |
| 8: | `squ_modp` $x_1$ $c_2$; | // $c_2 \leftarrow x_1^2$ |
| 9: | `sub_modp` $d$ $c_2$; | // $d \leftarrow -x_1^2 + y_1^2$ |
| 10: | `neg_modp` $c$; | // $c \leftarrow -x_1^2 - y_1^2$ |
| 11: | `add_const_modp` $c$ 2; | // $c \leftarrow -x_1^2 - y_1^2 + 2$ |
| 12: | `inv_modp` $c$ $f$; | // $f \leftarrow 1/-x_1^2 - y_1^2 + 2$ |
| 13: | `mul_modp` $d$ $f$ $y_2$; | // $y_2 \leftarrow$ `Answer` |
| 14: | | // Reverse states |
| 15: | `inv_modp` $c$ $f$; | // $f \leftarrow 0$ |
| 16: | `sub_const_modp` $c$ 2; | // $c \leftarrow -(x_1^2 + y_1^2)$ |
| 17: | `neg_modp` $c$; | // $c \leftarrow x_1^2 + y_1^2$ |
| 18: | `add_modp` $d$ $c_2$; | // $d \leftarrow y_1^2$ |
| 19: | `squ_modp` $x_1$ $c_2$; | // $c_2 \leftarrow 0$ |
| 20: | `inv_modp` $c$ $e$; | // $e \leftarrow 0$ |
| 21: | `sub_modp` $c$ $d$; | // $c \leftarrow x_1^2$ |
| 22: | `squ_modp` $y_1$ $d$; | // $d \leftarrow 0$ |
| 23: | `squ_modp` $x_1$ $c$; | // $c \leftarrow 0$ |
| 24: | `mul_modp` 2 $a$ $b$; | // $b \leftarrow 0$ |
| 25: | `mul_modp` $y_1$ $x_1$ $a$; | // $a \leftarrow 0$ |

**Algorithm 5** Point Doubling - Projective coordinates (3M + 12S = $240n^2log_2(n)$). This algorithm requires no inversions, 3 multipliers and 12 squarings giving an estimate of $3 \cdot 16 + 12 \cdot 16 = 240n^2log_2(n)$ gates.

| | | |
|---|---|---|
| 1: | `squ_modp` $X_1$ $A$; | // $A \leftarrow X_1^2$ |
| 2: | `squ_modp` $Y_1$ $B$; | // $B \leftarrow Y_1^2$ |
| 3: | `squ_modp` $X_1$ $T_1$; | // $T_1 \leftarrow X_1^2$ |
| 4: | `squ_modp` $Y_1$ $T_2$; | // $T_2 \leftarrow Y_1^2$ |
| 5: | `add_modp` $A$ $B$; | // $A \leftarrow X_1^2 + Y_1^2$ |
| 6: | `add_modp` $T_1$ $T_2$; | // $T_1 \leftarrow X_1^2 + Y_1^2$ |
| 7: | `add_modp` $X_1$ $Y_1$; | // $X_1 \leftarrow X_1 + Y_1$ |
| 8: | `squ_modp` $X_1$ $C$; | // $C \leftarrow X_1^2 + 2X_1Y_1 + Y_1^2$ |
| 9: | `squ_modp` $Z_1$ $D$; | // $D \leftarrow Z_1^2$ |
| 10: | `sub_modp` $A$ $D$; | // $A \leftarrow X_1^2 + Y_1^2 - Z_1^2$ |
| 11: | `sub_modp` $A$ $D$; | // $A \leftarrow X_1^2 + Y_1^2 - 2Z_1^2$ |
| 12: | `sub_modp` $C$ $T_1$; | // $C \leftarrow 2X_1Y_1$ |
| 13: | `mul_modp` $C$ $A$ $X_3$; | // $X_3 \leftarrow$ `Answer` |
| 14: | `mul_modp` $T_1$ $A$ $Z_3$; | // $Z_3 \leftarrow$ `Answer` |
| 15: | `add_modp` $A$ $D$; | // $A \leftarrow X_1^2 + Y_1^2 - Z_1^2$ |
| 16: | `add_modp` $A$ $D$; | // $A \leftarrow X_1^2 + Y_1^2$ |
| 17: | `sub_modp` $A$ $B$; | // $A \leftarrow X_1^2$ |
| 18: | `sub_modp` $A$ $B$; | // $A \leftarrow X_1^2 - Y_1^2$ |
| 19: | `mul_modp` $A$ $T_1$ $Y_3$; | // $Y_3 \leftarrow$ `Answer` |
| 20: | | // Reverse states |
| 21: | `add_modp` $A$ $B$; | // $A \leftarrow X_1^2$ |
| 22: | `squ_modp` $X_1$ $A$; | // $A \leftarrow 0$ |
| 23: | `add_modp` $C$ $T_1$; | // $C \leftarrow (X_1 + Y_1)^2$ |
| 24: | `squ_modp` $X_1$ $C$; | // $C \leftarrow 0$ |
| 25: | `squ_modp` $Z_1$ $D$; | // $D \leftarrow 0$ |
| 26: | `sub_modp` $X_1$ $Y_1$; | // $X_1 \leftarrow X_1$ |
| 27: | `sub_modp` $T_1$ $T_2$; | // $T_1 \leftarrow T_1$ |
| 28: | `squ_modp` $Y_1$ $T_2$; | // $T_2 \leftarrow 0$ |
| 29: | `squ_modp` $X_1$ $T_1$; | // $T_1 \leftarrow 0$ |
| 30: | `squ_modp` $Y_1$ $B$; | // $B \leftarrow 0$ |

# Chapter 5

# Conclusion

When we compare our results with the results from [23], it is clear to see that our estimates show that similar levels of resources are required for Shor's ECDLP with Edward curves compared to Weierstrass curves. We had tried representing Edward curves with a variety of different coordinate systems, however none of our results used fewer resources than the Weierstrass estimate obtained in [23]. One possible reason for this has been explained in Section 4.1.2, where the slope of the Weierstrass curve can be re-computed from the result $P_3$, when computing the sum of two points. **Algorithm 5** was the closest to the Weierstrass curve estimates, using $240n^2log_2n$ Toffoli gates. This gave a first order estimate for the total number of quantum gates required for Shor's ECDLP as $450n^3log_2n$. To improve the accuracy of our result, we could implement a circuit for our algorithm using quantum simulation software. This would give us second order estimates, and we would be able to simulate our algorithm to test the number of resources required for different levels of security.

As each of our algorithms require an inversion, they all use the same number of qubits - $9n + 2log_2(n) + 10$. The other operations (`mul_modp`, `squ_modp`) can be computed using the $5n + 2log_2(n) + 9$ ancilla qubits from the modular inversion operation, so no more qubits are required. Some ways to reduce the resources required could be to use a different algorithm for computing a modular inversion. We had tried using projective coordinates to minimize the number of inversions. The projective coordinate algorithms did not need to compute an inversion initially, however a single inversion was still required to revert to affine coordinates. Using projective coordinates also required more variables, hence more operations were required to compute them. A lot more modular multiplication was required so we had not observed the results we were expecting.

As **Algorithm 5** required the same number of qubits as the Weierestrass curve algorithm, and also required a similar number of Toffoli gates, our results provide additional evidence supporting the fact that computing Elliptic curves is much easier than computing RSA. This was first suggested by Proos and Zalka in [20] and their results were emphasized in [23]. Computing RSA requires much more qubits and quantum gates than computing elliptic curves with similar levels of security, making it much more secure than elliptic curves.

To conclude, we obtained resource estimates for 5 different algorithms that compute point addition and point multiplication on Edward curves. One of our algorithms was significantly lower than the others, which we used to calculate an estimate for the resources required for Shor's ECDLP. Our results show that elliptic

curves are less secure than RSA, as they can be computed much more easily. Below we have listed a few possible improvements that could reduce our estimates, which could be looked into for future work.

## 5.1 Possible improvements

### Reducing the cost of a modular inversion

As the modular inversion is the costliest operation required for Shor's ECDLP, if the cost for a modular inversion was reduced, this would reduce the cost substantially for algorithms that use multiple inversions. As **Algorithm 5** only uses one inversion, this change would not improve our result by much, however the affine coordinate algorithms use multiple inversions. A reduction in the cost of an inversion may reduce the cost massively for these algorithms. Perhaps future work could also look at avoiding inversions altogether when computing Shor's algorithm and use multiplications instead.

### Register sharing

Proos and Zalka [20], have suggested a way to reduce the number of qubits, required for modular inversion, by register sharing. The binary greatest common denominator algorithm [23], used in the modular inversion algorithm, works with 4 variables which requires $4n$ qubits. Proos and Zalka have shown that this can be shrunk down to $2n + 8\sqrt{n}$ qubits.

### Scalar Multiplication

Future work could perhaps look at using a different method for scalar multiplication. A windowing method could be used to reduce the number of point additions, while keeping the number of point

the same. Point additions are costlier than point doublings, so reducing the number of additions could reduce the cost of computing the scalar multiple.

### Point Compression

Point compression is an interesting method when computing elliptic curves. It involves using the $x$ (or the $y$) coordinate, to retrieve the corresponding $y$ (or $x$ coordinate). A bit is also required to identify the sign of the output. For example for Edward curves, if we are given the $x$ coordinate, the curve equation $x^2 + y^2 = c^2(1 + dx^2y^2)$ can be rearranged to give:

$$\pm y = \sqrt{\frac{c^2 - x^2}{1 - c^2dx^2}}$$

Already we can see from this method than an inversion and a square root is required. In [13], Justus has described two methods for point compression, one involving a square root and the other without. Future work could look at optimizing these methods, implementing them in a quantum environment or perhaps presenting a different approach where using an inversion could be avoided. Computing the square root, used in the square root method suggested, on a quantum computer could perhaps be avoided and instead post-processed classically.

### Simplifying modular multiplication

As Proos and Zalka mentioned in their paper [20], modular addition has been made simpler using a quantum Fourier transform of size $p$ as explained in [6]. If there was a way to implement modular multiplication in a similar fashion, it would help reduce costs and simplify the modular multiplication algorithms.

# Chapter 6

# Appendix

## 6.1 Instructions to run code

In the .zip folder there is only one file. It contains the code which was used to check if the algorithms were correct. It should be run as follows:

1. Open the file named **Algorithm Sage code**. Ensure it is opened using software that does not interfere with the formatting. (Notepad ruins formatting, but Notepad++ does not.)

2. This code was created using the online *SageMath* editor *CoCalc*. Create an account on the website `https://cocalc.com/`

3. Then create a new project and in the project create a new file. Select the file type as a Sage Worksheet. There should now be a blank Sage worksheet open.

4. From the **Algorithm Sage code** file, copy and paste all the code from the `Functions` section into the blank Sage worksheet. Then run the code, either by clicking the Run button or by pressing L-Shift and Enter.

5. Copy and paste the next section of the code (`Affine Point Addition`) underneath the output box for the `Functions` section and run it. The output box will appear below the code after running it. (The output box will be blank for the `Functions` section, however the rest of the sections have some output).

6. Repeat this for the remaining sections of the code, running each section separately, until all the code has been run on the Sage editor.

# Bibliography


[1] URL: https://commons.wikimedia.org/w/index.php?curid=13524800.

[2] Daniel J. Bernstein and Tanja Lange. *Edwards curves*. URL: https://www.hyperelliptic.org/EFD/g1p/auto-edwards.html.

[3] Daniel J Bernstein and Tanja Lange. "Faster addition and doubling on elliptic curves". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2007, pp. 29–50.

[4] Daniel Bernstein et al. "ECM using Edwards curves". In: *Mathematics of Computation* 82.282 (2013), pp. 1139–1179.

[5] Stephanie Blanda. *Shor's Algorithm – Breaking RSA Encryption*. 2014. URL: https://blogs.ams.org/mathgradblog/2014/04/30/shors-algorithm-breaking-rsa-encryption/.

[6] Thomas G Draper. "Addition on a quantum computer". In: *arXiv preprint quant-ph/0008033* (2000).

[7] Harold Edwards. "A normal form for elliptic curves". In: *Bulletin of the American Mathematical Society* 44.3 (2007), pp. 393–422.

[8] Stefan Tessarini Elisa Baumer Jan-Grimo Sobez. *Shor's Algorithm*. 2015. URL: http://www.qudev.ethz.ch/content/QSIT15/Shors%20Algorithm.pdf.

[9] *Fourier Sampling and Simon's Algorithm*. URL: https://courses.edx.org/c4x/BerkeleyX/CS191x/asset/chap4.pdf.

[10] Lov K Grover. "A fast quantum mechanical algorithm for database search". In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM. 1996, pp. 212–219.

[11] *Grover's Algorithm*. URL: https://en.wikipedia.org/wiki/Grover%27s_algorithm.

[12] Matthew Hayward. "Quantum computing and shor's algorithm". In: *Illinois Mathematics and Science Academy* 1 (2005), pp. 1–61.

[13] Benjamin Justus. "Point Compression and Coordinate Recovery for Edwards Curves over Finite Field". In: *Annals of West University of Timisoara-Mathematics* 52.2 (2014), pp. 111–125.

[14] Burton S Kaliski. "The Montgomery inverse and its applications". In: *IEEE transactions on computers* 44.8 (1995), pp. 1064–1065.

[15] Tanja Lange. *Edwards coordinates for elliptic curves, part 1*. URL: https://www.math.u-bordeaux.fr/emnt2007/talks/lange.pdf.

[16] Fang Xi Lin. "Shor's Algorithm and the Quantum Fourier Transform". In: *McGill University* (2014).

[17] SJ Lomonaco. "Shor's quantum factoring algorithm". In: *Proceedings of Symposia in Applied Mathematics*. Vol. 58. 2002, pp. 161–180.

[18] Peter L Montgomery. "Modular multiplication without trial division". In: *Mathematics of computation* 44.170 (1985), pp. 519–521.

[19] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[20] John Proos and Christof Zalka. "Shor's discrete logarithm quantum algorithm for elliptic curves". In: *arXiv preprint quant-ph/0301141* (2003).

[21] *Quantum Decoherence*. URL: https://en.wikipedia.org/wiki/Quantum_decoherence.

[22] *Quantum Search Algorithm*. URL: http://www.physics.udel.edu/~msafrono/650/Lecture%2010%20-%2011.pdf.

[23] Martin Roetteler et al. "Quantum resource estimates for computing elliptic curve discrete logarithms". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2017, pp. 241–270.

[24] Thomas Santoli and Christian Schaffner. "Using Simon's Algorithm to Attack Symmetric-Key Cryptographic Primitives". In: *arXiv preprint arXiv:1603.07856* (2016).

[25] Peter W Shor. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer". In: *SIAM review* 41.2 (1999), pp. 303–332.

[26] Daniel R Simon. "On the power of quantum computation". In: *SIAM journal on computing* 26.5 (1997), pp. 1474–1483.

[27] *Simon's Algorithm*. 2010. URL: http://www-inst.eecs.berkeley.edu/~cs191/sp12/notes/simon.pdf.

[28] *Simon's problem*. URL: https://en.wikipedia.org/wiki/Simon%27s_problem.

[29] Emma Strubell. "An introduction to quantum algorithms". In: *COS498 Chawathe Spring* 13 (2011), p. 19.

[30] Iain Styles. *Grover's Quantum Search Algorithm*. URL: http://www.cs.bham.ac.uk/internal/courses/intro-mqc/current/lecture08_handout.pdf.

[31] Iain Styles. *Shor's Factorisation Algorithm*. URL: http://www.cs.bham.ac.uk/internal/courses/intro-mqc/current/lecture07_handout.pdf.

[32] *Super-dense coding*. URL: https://www.quantiki.org/wiki/super-dense-coding.

[33] Yasuhiro Takahashi, Seiichiro Tani, and Noboru Kunihiro. "Quantum addition circuits and unbounded fan-out". In: *arXiv preprint arXiv:0910.2530* (2009).

[34] John Watrous. *Simon's algorithm*. 2006. URL: https://cs.uwaterloo.ca/~watrous/LectureNotes/CPSC519.Winter2006/06.pdf.