Resource Estimation of Fault Tolerant Quantum Information Set Decoding

Kyle Hutchings

A thesis presented for the degree of
Doctor of Philosophy

Department of Information Security
Royal Holloway, University of London
June 13, 2022

Declaration

These doctoral studies were conducted under the supervision of Prof. Rüdiger Schack.

The work presented in this thesis if the result of original research carried out by myself, whilst enrolled in the Department of Information Security as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university of educational establishment.

Kyle Hutchings
June 13, 2022

Acknowledgements

I started my academic journey over 7 years ago, venturing from the Isle of Man to the south-east of England. At the time, I had no idea how far I would come. The process has not been easy, with numerous ups and downs throughout, but I want to extend my thanks to those who made this entire endeavour possible.

First and foremost, I want to extend my deepest gratitudes to my supervisor Rüdiger Schack, whom without I would never have been given the opportunity to pursue a Ph.D. I truly cherish the all the discussion we had together, and learning all about quantum computing. Your guidance and wisdom over this journey was invaluable.

I've also had the pleasure of making numerous friends throughout my time in academia, to each and every one of you, thank you. You helped keep me sane, listened to my ramblings, and provided me with love, compassion, and support, for which I'm deeply grateful.

I would like to thank my incredible family: my parents Nick and Heather, my fantastic elder sisters Yasmin and Jade, and my partner Dominique. The endless love and support each of you gave me is what made this entire endeavour possible. I hope I have done all of you proud.

Finally, I want to dedicate this work to my cousin, Adam "Tizz" Shaw, and my Grandfather, Philip Hutchings, whom both passed away during the final stages of my studies. I miss both of you deeply, and want to thank you both for helping me to become the person I am today.

Abstract

With the ever-present threat of quantum computing looming over the world of cryptography, researchers have been investigating how best to replace existing cryptographic schemes with those that can withstand quantum attacks. Our research contributes to the area of resource estimation, a field concerned with analysing the amount of real-world resources (both temporal and spatial) required for a quantum computer to compromise a given cryptographic scheme using the best known current methods. We present a circuit to perform Prange's algorithm, a variant of quantum information set decoding. We embed our construction within an error-correction scheme in order to calculate the overhead costs incurred by fault-tolerance. Our analysis shows that current proposed parameters for code-based cryptography provide a much larger security margin than required for their specified security level, and as such could be reduced to improve performance whilst still ensuring quantum immunity.

Contents

Chapter 1

Introduction

1.1 The World of Quantum Computing

Quantum computing's first inception was in the 1980s. Following the groundbreaking work of Benioff [11] and Feynman [34], who demonstrated a quantum mechanical model of computing, and observed that efficient simulation of quantum systems was seemingly impossible for classical computers. In the years that followed, researchers were able to uncover various algorithms demonstrating the potential power that a quantum computer could be capable of [12, 13, 24, 85]. However, there were doubts about the practical uses of such algorithms and the feasibility of quantum computers, especially with a lack of any method to keep a quantum system error-corrected, rendering computations obsolete. But everything changed in 1994, when Peter Shor unveiled a quantum algorithm able to solve the integer factorization and discrete logarithm problems in polynomial time, a feat that classical computing has been unable to achieve [83]. The difficulty of solving integer factorization and discrete logarithms serve as the basis of security for many of the world's most popular cryptosystems, such as RSA [75] and ElGamal [30], the existence of an efficient algorithm to solve these problems has caused researchers to start seeking alternative security schemes based on alternative primitives. Needless to say, the invention of Shor's algorithm sparked huge interest in the field, which showed no signs of slowing in the years that followed, as Shor presented the first quantum error-correcting scheme [84]; additionally, Lov Grover demonstrated a quantum search algorithm offering a quadratic improvement versus its classical counterpart [48].

Ever since, the world of quantum computing has exploded. At the time of writing, quantum computing has become a multi-billion dollar industry worldwide with significant investments from large companies such as Google, IBM, Microsoft, and Amazon, as well as seeing the creation of quantum 'start-ups' like Rigetti, Cambridge Quantum, and Oxford Quantum Circuits. With all this investment, research has been chasing two major developments: quantum hardware, and quantum algorithms. Current progress within quantum hardware has been aimed at demonstrating quan-

tum supremacy; where a quantum computer is able to perform a computation that could not be performed on a classical computer in any feasible amount of time (regardless of the importance of the problem). Currently, due to device limitations and engineering difficulties, there has been no widely accepted demonstration of quantum advantage, although there has been signs of companies such as Google inching ever closer [92]. On the algorithmic side, research has been investigating what potential problems could be solved with quantum computing and how technology will be affected by the introduction of quantum computers to the world.

Within the industry of information security, research has mostly been centred around the dawn of post-quantum cryptography, how we can ensure security against quantum adversaries utilizing algorithms such as Shor's and Grover's. With the majority of modern cryptography at risk of quantum attacks, NIST (National Institute of Standards and Technology) issued a call-to-arms for submissions of quantum secure cryptography [22]. Following the NIST report, candidates have been shortlisted such that security standards can be adjusted to accommodate the ever-looming threat of quantum attacks.

1.2 Research Aims and Motivations

The aim of this thesis is to contribute to the field of resource estimation. In this context, resource estimation means to provide an approximation of the real-world costs, both spatial and temporal, of running a quantum algorithm on quantum hardware. The specific algorithms of interest to us are those that attack post-quantum cryptography schemes. The motivation behind this body of work is to provide those within the industry a more realistic assessment of the threat that a specific algorithm poses by factoring in the overhead costs required to run a large scale quantum algorithm (namely error-detection and error-correction) fault-tolerantly [42].

The idea behind resource estimation is a relatively new field, and as such, there have been a number of recent papers aiming to either improve or establish new estimates of the cost for quantum attacks against classical cryptographic schemes. Currently, this research can mostly be divided into two groups, those investigating how best to construct Shor's algorithm as a method of attacking RSA [38, 41, 44, 69] and elliptic curve based cryptosystems [41, 76], and those constructing Grover's oracles to attack a variety of symmetric-key cryptosystems such as AES [41, 47, 50, 54, 59] and SHA [4, 41, 50].

The primary contribution of our research is to investigate the advantage Grover's algorithm provides against code-based cryptography, a family of asymmetric cryptographic algorithms and one of the leading candidates for post-quantum security [22].

The inspiration behind this work is research detailing methods of how to perform the information set decoding (ISD) attack using a quantum computer [15, 55, 56], leading to recommendations of increased key sizes to ensure quantum immunity. However, this increase in key size comes at a significant cost, both in the spatial requirements for storing a large key, and the temporal cost of computation for encryption and decryption of large keys [80]. By extending previous research, we hope to find more realistic lower bounds for quantum ISD by factoring in overhead costs of implementation and error-correction required to run said algorithm fault-tolerantly, allowing us to provide new security guidelines for parameter selection of code-based cryptosystems, whilst still ensuring quantum immunity.

1.3 Thesis Structure

We begin this work in Chapter 2 by detailing the relevant background knowledge of quantum computing, resource estimation, linear codes, code-based cryptography, and the information set decoding algorithm. In Chapter 3, we detail our efforts in constructing a fault-tolerant quantum circuit to perform information set decoding and analyse the impact of our construction on current proposed post-quantum immune code-based cryptosystems. Finally, in Chapter 4 we compare our work with a recent publication by Perriello, Barenghi, and Pelosi [71] who also investigated the construction of quantum information set decoding circuits.

Chapter 2

Background

This chapter introduces the key background information and notation we will be using for the main body of this thesis. We will discuss the details of the quantum circuit model, Grover's algorithm, quantum error-correction, resource estimation, and finally, code-based cryptography and the information set decoding algorithm.

2.1 Quantum Bits & Circuits

Within classical computing, the fundamental unit of information is the *bit*; each bit is a binary integer and can either be 0 or 1. In quantum computing, we use the quantum bit, or *qubit* for short. While there are many physical interpretations of qubits, the underlying mathematical object remains the same; a single qubit is a two-state quantum system and similarly to classical computing, the two-states are $|0\rangle$ and $|1\rangle$. The symbol '$|\ \rangle$' is known as a *ket*, and forms part of the Bra-Ket notation used for denoting quantum states. Bits and qubits differ in that while a bit can only be 0 or 1, a qubit can be a state that is a complex linear combination of both $|0\rangle$ and $|1\rangle$. We refer to such combination states as being in *superposition*:

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle , \qquad \alpha_0, \alpha_1 \in \mathbb{C} \tag{2.1}$$

Another way to represent the state of a qubit is by using a two-dimensional vector. By using $|0\rangle$ and $|1\rangle$ to form an orthonormal basis we can again describe superposition states as a linear combination:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.2}$$

$$|\psi\rangle = \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} \tag{2.3}$$

The downside of quantum computing is that while it is possible for us to compute

with qubits in superposition, if we are ever to observe the state of a qubit, we are only able to receive either 0 or 1. We refer to this process of observing a qubit as a *measurement*. The output of a measurement is probabilistic, where the probability is dependent on the value of the coefficients (or amplitudes) α_0 and α_1. More formally, the probability of measuring a 0 is $|\alpha_0|^2$, and a 1 is $|\alpha_1|^2$, from this we can also infer that $|\alpha_0|^2 + |\alpha_1|^2 = 1$. Measurement will also alter the state of the qubit irreversibly, and cause any superposition state to collapse into either $|0\rangle$ or $|1\rangle$ correlated with the output we received.

Of course, we are also able to extend this definition to multi-qubit states. One such method is to simply represent multiple qubits as a tensor product ($\otimes$) of single qubit states, one example of such a state is $|10\rangle$:

$$|10\rangle = |1\rangle \otimes |0\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{2.4}$$

As with single qubit states, we are also able to have superposition states over two qubits, which can be represented as a linear combination of the four basis states, in this case, 00, 01, 10, and 11:

$$|\psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle = \begin{bmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{bmatrix} \tag{2.5}$$

More generally speaking, an n-qubit state can be represented as:

$$|\psi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x\,|x\rangle \tag{2.6}$$

Where $x \in \{0,1\}^n$ represents all unique n-length strings composed of 0 and 1, $\alpha_x \in \mathbb{C}$, and $\sum_{x \in \{0,1\}^n} |\alpha_x|^2 = 1$.

However, looking closely at the above definition, it is possible for us to have a state that cannot be expressed as a tensor product state (as in eq 2.4). For example if we have a state where $\alpha_{00} = \alpha_{11} = 1/\sqrt{2}$, and $\alpha_{01} = \alpha_{10} = 0$:

$$|\Phi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{2.7}$$

In this instance, it is impossible for us to describe this state as a tensor product of two separate qubit states, we refer to such states as being *entangled*. These states

exhibit uniquely quantum behaviour. If we were to measure the first qubit and find it in the state $|0\rangle$, which is true with a 50% probability, we would know with certainty that the second qubit will also be in the state $|0\rangle$, as the first measurement would collapse the overall state to $|00\rangle$.

Equipped with the knowledge of qubit states, superposition, entanglement, and measurement, we will now look at how we can manipulate qubit states in order to perform computations. In classical computing, the fundamental operations are Boolean functions often represented by logic gates. Rather unsurprisingly, the quantum equivalent is the *quantum logic gate*. However, rather than being a representation of a Boolean function, quantum gates are unitary operators and are usually described as *unitary matrices*. A matrix, U, is considered unitary if and only if $U^\dagger U = UU^\dagger = \mathbb{1}$, where $U^\dagger$ is the complex transpose of U and $\mathbb{1}$ is the identity matrix. Unitary matrices come with two rather important properties: the first is that quantum gates preserve the probability amplitudes of quantum states, ensuring that our measurement outcomes still sum to 1, and the second is that quantum gates are, by definition, reversible. To demonstrate an example of a quantum gate, let's delve into the *Hadamard gate*:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2.8}$$

In order to evaluate the action of a gate on a qubit, we multiply the matrix of the gate with the state vector, showing how the application of the gate will evolve our quantum system:

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle \tag{2.9}$$

$$H|1\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle \tag{2.10}$$

Looking at the behaviour of the Hadamard gate, we can see that if used on a basis state, the output is in superposition; often, you will find these states labelled as $|+\rangle$ and $|-\rangle$, as they frequently they occur in quantum computing. But what if we applied our gate to a superposition state? As unitary gates, such as the Hadamard gate, are linear, we can simply apply the operation distributively on all parts of the state :

$$U(\alpha_0|0\rangle + \alpha_1|1\rangle) = \alpha_0 U|0\rangle + \alpha_1 U|1\rangle \tag{2.11}$$

To demonstrate this, we shall take a look at how applying a second Hadamard gate will affect the output from the above equations (2.9 & 2.10):

$$
\begin{aligned}
H|+\rangle &= H(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)) \\
&= \frac{1}{\sqrt{2}}(H|0\rangle + H|1\rangle) \\
&= \frac{1}{\sqrt{2}}(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) + \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)) \\
&= \frac{1}{\sqrt{2}}(\sqrt{2}|0\rangle) \\
&= |0\rangle
\end{aligned}
\tag{2.12}
$$

$$
\begin{aligned}
H|-\rangle &= H(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)) \\
&= \frac{1}{\sqrt{2}}(H|0\rangle - H|1\rangle) \\
&= \frac{1}{\sqrt{2}}(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) - \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)) \\
&= \frac{1}{\sqrt{2}}(\sqrt{2}|1\rangle) \\
&= |1\rangle
\end{aligned}
\tag{2.13}
$$

One additional feature we learn from applying the Hadamard gate to the states $|+\rangle$ and $|-\rangle$ is that the Hadamard gate is self-inverse i.e., $H = H^\dagger$; a more formal way to show this is that $HH = \mathbb{1}$. Hadamard is an example of a single-qubit gate, however, it is also possible to define multi-qubit gates. One common example of such gates are the *controlled-gates*. Given a single-qubit unitary operator U, we can construct a controlled-U gate, CU:

$$
\begin{aligned}
U &= \begin{bmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{bmatrix} \\
CU &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{bmatrix}
\end{aligned}
\tag{2.14}
$$

When CU is applied to a two-qubit system, the behaviour can be quite extraordinary:

$$
\begin{aligned}
&|00\rangle \mapsto |00\rangle \\
&|01\rangle \mapsto |01\rangle \\
&|10\rangle \mapsto |1\rangle \otimes U|0\rangle = |1\rangle \otimes (u_{00}|0\rangle + u_{10}|1\rangle) \\
&|11\rangle \mapsto |1\rangle \otimes U|1\rangle = |1\rangle \otimes (u_{10}|0\rangle + u_{11}|1\rangle)
\end{aligned}
\tag{2.15}
$$

The first qubit is unchanged by the operation, however, the second qubit will have U applied to it, if and only if the first qubit is in the state $|1\rangle$. It is for this reason that

when looking at controlled gates we refer to the first qubit as the *control* and the second qubit as the *target*. To give an example of a such a gate, let us look at one of the most important gates in quantum computing, the *controlled-not* gate. This gate is the controlled form of the *Pauli* X gate, which is sometimes referred to as a NOT gate, as it acts as a bit flip operation when used on a qubit in a computational basis state:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \begin{aligned} X|0\rangle &= |1\rangle \\ X|1\rangle &= |0\rangle \end{aligned} \tag{2.16}$$

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad \begin{aligned} CNOT|00\rangle &= |00\rangle \\ CNOT|01\rangle &= |01\rangle \\ CNOT|10\rangle &= |11\rangle \\ CNOT|11\rangle &= |10\rangle \end{aligned} \tag{2.17}$$

The $CNOT$ gate resembles the behaviour of the classical XOR gate, and as such, has numerous uses in computation. It is also worth noting that it is also possible to construct gates with more than a single control or target qubit. A frequently encountered gate in this category is the *Toffoli* gate, which is simply a $CNOT$ gate with two controls and a single target. By using a Toffoli gate on a target qubit fixed in the state $|0\rangle$, it is analogous to the classical AND gate acting on the target qubits.

Over the course of this body of work, we will be constructing quantum algorithms using circuit diagrams. Armed with the basic knowledge of qubits and gates, we can now discuss the notation used in such diagrams. In this model, the horizontal axis represents time, moving from left to right. Qubits are denoted by single lines known as wires and often have labels to show their input and output states. Qubits can be collated together to form a *quantum register* and are represented with a diagonal line across the wire with a number representing their size. Any operations on a qubit are visualized by a square box placed over the qubit wire(s) it operates on with a label to identify the gate. While this notation is the standard for arbitrary gates, there are some commonly used gates that deviate from this norm e.g., $CNOT$ gates. A controlled-gate places a 'dot' on the wire that is the control and links to the gate it is controlling with a vertical line. Table 2.1 contains some of the most commonly encountered symbols used in quantum circuit notation, and to show an example of a full quantum algorithm in circuit notation, we have included the teleportation circuit [13] in figure 2.1.

Symbol	Description
$\|0\rangle$ —— $\|0\rangle$	Single qubit wire in the state $\|0\rangle$.
$\|0\rangle$ —/^{n} $\|0\rangle$	Quantum register containing n qubits.
$\|0\rangle$ —[H]— $\|+\rangle$	Quantum gate, H, acting on a single qubit.
$\|1\rangle$ —•— $\|1\rangle$ $\|0\rangle$ —⊕— $\|1\rangle$	$CNOT$ gate, the first qubit is the control and the second is the target.
$\|0\rangle$ —[↗]= 0	Measurement gate, the wire transforms to a classical wire after measurement.

Table 2.1: Common symbols and their description used in quantum circuit notation.

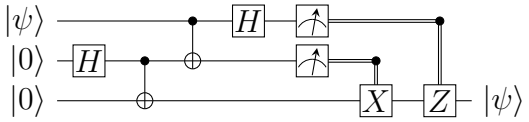


Figure 2.1: Quantum circuit for the teleportation algorithm [13]. This algorithm contains Hadamard gates, H, controlled-not gates, measurements and classically controlled quantum gates.

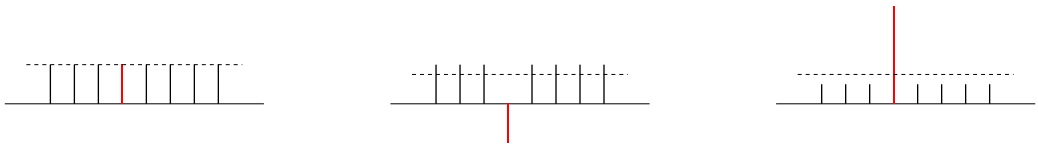
Now that we have covered the basics of notation around quantum computing and quantum circuits, we will move on to looking at Grover's algorithm. This algorithm is central to the resource estimation that we will be performing later in this work.

2.2 Grover's Algorithm

Assume there is some collection of unsorted data and you wish to find one specific entry in that collection. Classically, there is only really one way to approach this problem; you evaluate each item until you find the one you were looking for. If the number of items you have to search through is N, this 'method' would require $O(N)$ evaluations to find your desired entry. Originally designed by Lov Grover in 1996 [48], Grover's algorithm demonstrates an approach whereby it is possible to find a solution in $O(\sqrt{N})$ evaluations. We shall begin by demonstrating this algorithm visually to gain a better understanding of how it operates.

Assume we have a collection of 8 entries and our goal is to try to find one specific entry. We will begin by having a quantum register that holds these 8 entries as states in superposition, each with the same amplitude. To represent this, we use a bar chart, where the length is the amplitude of each state, the red bar is our desired state (4th from the left), and the dotted horizontal line represents the mean of our amplitudes. After the system has been prepared in uniform superposition of the search space, the

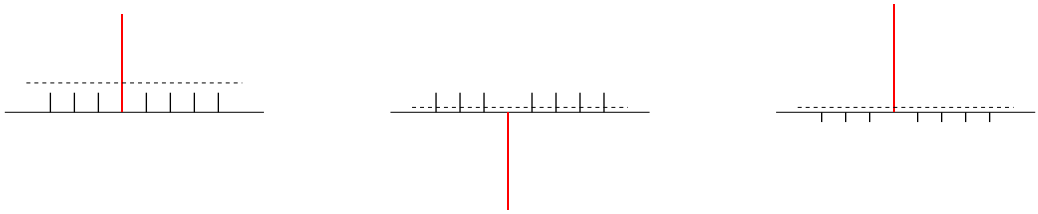
next step in Grover's algorithm is to apply an operator that will invert the phase of our solution state. Due to measurement probability being equal to $|\alpha_x|^2$, where α_x is the amplitude of state x, this step has not changed the probability of measuring our goal state. However, it has altered the mean, such that the amplitude of the non-goal states are now greater than the mean and the goal state is now less than the mean. From here, the third step of Grover's algorithm is to apply an inversion about the mean, where we apply an operation to each state to change their amplitude from α to $\alpha' = \mu + (\mu - \alpha)$ or equivalently $\alpha' = 2\mu - \alpha$. This results in the amplitude of the goal state increasing, whilst decreasing the amplitude of the non-goal states.



(a) Input state (b) Phase inversion (c) Inversion about the mean

Figure 2.2: Effects of applying both steps of Grover's algorithm to a state in equal superposition.

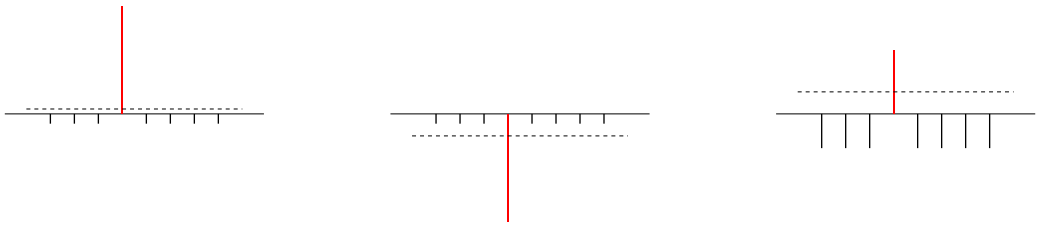
At this stage, we could measure our register and with a decent probability could find our goal state. However, if we were to apply both steps again we found that the amplitude of our goal state increases even more.



(a) Input state (b) Phase inversion (c) Inversion about the mean

Figure 2.3: Applying a second iteration of Grover's algorithm to the out from figure 2.2, resulting in the amplitude of the goal state increasing further.

What if we were to try to apply the same process a third time? The output from the second iteration differs from the first in that the non-goal states already have a negative phase. This means that once we apply phase inversion, all states have a negative amplitude, and the inversion about the mean causes the amplitude of the goal state to decrease (fig. 2.4).



(a) Input state (b) Phase inversion (c) Inversion about the mean

Figure 2.4: Result from applying a third iteration of Grover's algorithm to the out from figure 2.3, causing the amplitude of the goal state to decrease.

This behaviour is why it is important for us to be careful of the number of iterations of Grover's algorithm. If we repeat the process either too few or too many times, we end up with a lower than expected probability of measuring the goal state, which in turn will lead to a decrease in probability of measuring the goal state. Thankfully, the number of iterations can be calculated easily and is equal to $\lfloor \frac{\pi}{4}\sqrt{N} \rfloor$, where N is the total size of our search space. In the example above $N = 8$, which implies the optimal number of iterations is $\lfloor \sqrt{8} \rfloor = 2$, which was demonstrated when the third iteration caused the amplitude of our goal state to decrease.

From this, we can define Grover's algorithm:

1. Prepare the system in uniform superposition over all N states
2. Apply phase inversion to the goal state
3. Apply inversion about the mean over all states
4. Repeat steps previous two steps $\lfloor \frac{\pi}{4}\sqrt{N} \rfloor$ times

Now we shall discuss how we can construct a circuit to perform Grover's algorithm. In order to prepare the system in uniform superposition of all N states, we apply Hadamard gates to $n = \lfloor \log_2 N \rfloor$ qubits in the state $|0\rangle$:

$$H^{\otimes n} |0\rangle^{\otimes n} = \frac{1}{\sqrt{n}} \sum_{x \in \{0,1\}^n} |x\rangle \,, \tag{2.18}$$

where $\{0,1\}^n$ is the set of all n-bit strings

It should be noted at this point, if N is not a power of 2, then we will have more than N states in our superposition, and as such, the search space will increase. This increase in search space requires an increase in the number of Grover's iterations we need to perform. Next, to apply phase inversion, we will utilize a technique known as *phase kickback*. This technique leverages the fact that applying an X gate to the state $|-\rangle$ results in the state having a negative phase applied to it:

$$X\left|-\right\rangle = -\left|-\right\rangle \tag{2.19}$$

If, we instead use a $CNOT$ gate targeted on the state $|-\rangle$, we find that the phase affects the whole state:

$$\begin{aligned} CNOT\left|0-\right\rangle &= \left|0-\right\rangle \\ CNOT\left|1-\right\rangle &= \left|1\right\rangle \otimes X\left|-\right\rangle \\ &= \left|1\right\rangle \otimes -\left|-\right\rangle \\ &= -\left|1-\right\rangle \end{aligned} \tag{2.20}$$

If we were to place the control qubit in superposition, we find that only the component of the state where the control is $|1\rangle$ is affected by the phase:

$$\begin{aligned} CNOT\left|+-\right\rangle &= \frac{1}{\sqrt{2}}(CNOT\left|0-\right\rangle + CNOT(\left|1-\right\rangle)) \\ &= \frac{1}{\sqrt{2}}(\left|0-\right\rangle + CNOT(\left|1-\right\rangle)) \\ &= \frac{1}{\sqrt{2}}(\left|0-\right\rangle - \left|1-\right\rangle) \end{aligned} \tag{2.21}$$

Now, assume we have an oracle function f, where $f(x) = 1$ if, and only if, $x = a$ where a is our goal state and $f(x) = 0$ otherwise. If we are to implement f as a quantum gate $U_f : |x\rangle\,|y\rangle \mapsto |x\rangle\,|f(x) \oplus y\rangle$, then by having $y = |-\rangle$ we will only apply the negative phase to the goal state:

$$U_f\left[\frac{\left|x,0\right\rangle - \left|x,1\right\rangle}{\sqrt{2}}\right] = \left[\frac{\left|x,f(x)\right\rangle - \left|x,f(x)\oplus 1\right\rangle}{\sqrt{2}}\right] = \left|x\right\rangle\left[\frac{\left|f(x)\right\rangle - \left|\overline{f(x)}\right\rangle}{\sqrt{2}}\right] \tag{2.22}$$

By rewriting this state, we can see how this will perform our phase inversion of the goal state:

$$(-1)^{f(x)}\left|x\right\rangle\left|-\right\rangle = \begin{cases} -1\left|x\right\rangle\left|-\right\rangle, & \text{if } x = a \\ +1\left|x\right\rangle\left|-\right\rangle, & \text{if } x \neq a \end{cases} \tag{2.23}$$

Finally, we need to perform our inversion about the mean. The operator used in this stage is *Grover's diffusion operator* and is usually expressed as $2\left|\psi\right\rangle\left\langle\psi\right| - \mathbb{1}$, where $|\psi\rangle = \frac{1}{\sqrt{n}}\sum_{x\in\{0,1\}^n}|x\rangle$ and $\langle|\psi| = |\psi\rangle^\dagger$. To construct this operator all we need to perform is $H^{\otimes n}U_0H^{\otimes n}$, where:

$$U_0\left|\psi\right\rangle = \begin{cases} -\left|\psi\right\rangle, & \text{if } \left|\psi\right\rangle = \left|\mathbf{0}\right\rangle \\ \left|\psi\right\rangle, & \text{if } \left|\psi\right\rangle \neq \left|\mathbf{0}\right\rangle \end{cases} \tag{2.24}$$

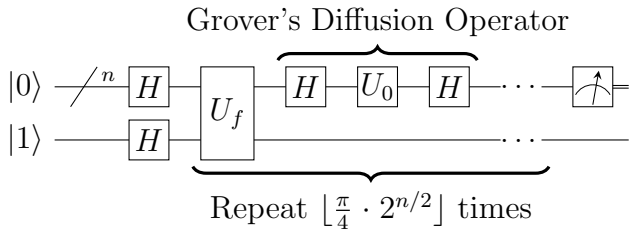


Figure 2.5: Quantum Circuit for Grover's Algorithm, involving phase kickback and Grover's diffusion operator which are both to repeated $O(\sqrt{N})$ times before measuring the top n qubits.

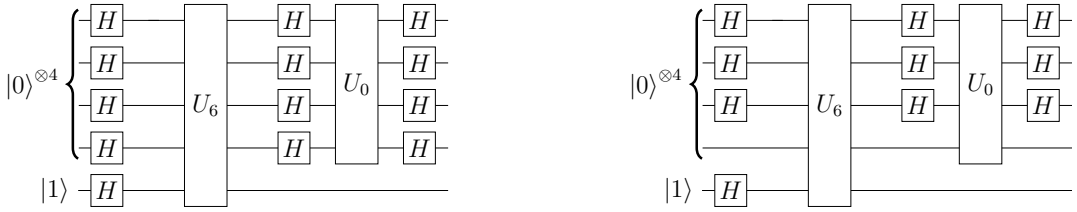
Using this we can now draw up the quantum circuit to perform Grover's algorithm (fig 2.5).
Grover's algorithm can also be generalized to apply to problems where there is more than one goal state, often known as amplitude amplification [18]. To do so we simply need to modify the phase inversion step such that our oracle function f will give $f(x) = 1$ if and only if $x \in A$, where A is a set of all goal states, and $f(x) = 0$ otherwise.

$$(-1)^{f(x)} |x\rangle |-\rangle = \begin{cases} -1 |x\rangle |-\rangle , & \text{if } x \in A \\ +1 |x\rangle |-\rangle , & \text{if } x \notin A \end{cases} \tag{2.25}$$

Using this new kickback operation, we can apply the algorithm exactly as we did previously. Depending on the number of iterations, all goal states with equally high amplitude, as well as all non-goal states with equally low amplitude would be in a state of superposition, which we can then either perform further computations with, or measure to sample one goal state. Once again, we need to be careful with the number of iterations we perform to have the best probability of measuring a goal state. Assuming one knows the total number of goal states, we need to iterate $\lfloor \frac{\pi}{4} \sqrt{N \,/\, |A|} \rfloor$ times, where $|A|$ is the size of the set A. It is also of importance to note that in the case where the size of A is unknown, there exists the quantum counting algorithm, which can be run first to calculate the number of solutions before moving on to perform Grover's search [19].

Another modification that can be made to Grover's is adjusting the search space of the algorithm. In the previous example, we established a uniform superposition over all states of 3 qubits by applying Hadamard to all the qubits; we then applied the same operations before and after the U_0 gate of our diffusion operator. If we were instead to apply an operator V that establishes a superposition $\sum_{x \in X} |x\rangle$ over some pre-determined set X, and then apply V and $V^\dagger$ before and after the U_0 respectively, we have in effect adjusted the search space of our Grover's search. The result of this modification allows us to further alter the formula for number of iterations required to $\lfloor \frac{\pi}{4} \sqrt{|X| \,/\, |A|} \rfloor$, where $|X|$ is the size of the set containing all states we

are searching over, and $|A|$ once again is the size of set containing all our goal states. As an example, let us investigate searching for the state $|0110\rangle$ over two different spaces: all possible 4 qubit states vs. all possible 4 qubit states where the last qubit is $|0\rangle$ (another way to think of this is searching for the number 6 in the integers 0-15 vs. even integers in 0-15).



(a) Searching over all possible states.

(b) Searching over even states

Figure 2.6: Two variations of Grover's algorithm to search for the number 6. By removing the Hadamard from the last qubit we fix its value at $|0\rangle$ and so our superposition state only contains even states.

Iteration	Fig. 2.6a	Fig. 2.6b
1	.473	.781
2	.908	.945
3	.961	.330

Table 2.2: Probability of measuring a goal state for each of the variants of Grover's algorithm (fig. 2.6) after each iteration.

From table 2.2, it is clear to see that by reducing the search space we have reduced the required number of iterations from 3 to 2 at the detriment of only a 2% loss in probability. Of course, this example is highly trivial as we could simplify the circuit to not include the fourth qubit and rephrase the problem to finding the state $|0110\rangle$ over four qubits to finding the state $|011\rangle$ over three qubits, if we knew the last qubit was fixed at $|0\rangle$. However, what this example is trying to demonstrate is that in order to limit the search space of Grover's, we must first prepare our state using some unitary operator V and that we must modify our diffusion such that we apply V and $V^\dagger$ before and after U_0 (see fig. 2.7), allowing us to reduce the overall number of iterations.

2.3 Resource Estimation

As mentioned in our introduction, the goal of resource estimation is to try approximate costs of running a quantum algorithm. The motivation behind this research stems from the threat that quantum algorithms such as Shor's and Grover's pose to currently deployed cryptographic systems. While large-scale quantum computing is

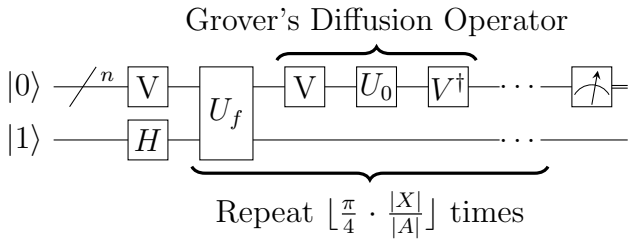


Figure 2.7: Quantum Circuit for a Grover's Algorithm when searching over a set of elements, X.

still very much in the distant future, reliably migrating to new forms of cryptography is a slow process; due to the complexities of establishing new security standards. Resource estimation allows us to more accurately ascertain the requirements necessary for a quantum computer to be capable of breaking current cryptographic systems. Simultaneously, it also provides insight into the development of large-scale quantum algorithms, including circuit and gate synthesis, and fault-tolerance.

For our analysis, we will be following the framework established by Amy et al. [4] (which was further explored by Gheorghiu and Mosca [41]). The methodology splits our approach into four layers (see figure 2.8), starting from a broad algorithmic level, all the way down to individual qubits. The reason for this choice of framework is due to its modular approach; as quantum computing lies in its infancy, there is a constant stream of new research impacting various subsections of the field. Modularity affords us the ability to observe which layers have been impacted by new research, and subsequently, update pre-existing estimates with relative ease. This ensures up-to-date estimates whilst also highlighting the impact of new research. Whilst Chapter 3 will discuss more of the specifics of each of these layers, we will now provide a high-level overview of the processes undertaken in each and at the end discuss the final metric of cost that we are aiming to derive in this analysis.

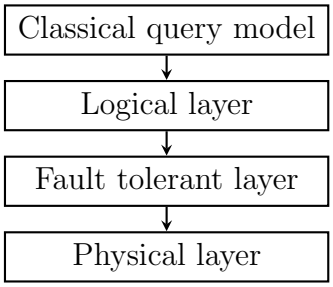


Figure 2.8: The Resource estimation framework first introduced in Amy et al. [4].

2.3.1 Classical Query Model

In the first layer, we evaluate the cost of an attack based on how many times we need to run or query a 'black box'. A black box is a system defined purely by its

inputs and outputs rather than its internal workings; this definition can be quite vague as it is dependent on the specific attack being evaluated. One of the most common interpretations of this layer is to evaluate an attack based on how many Grover iterations we need to make. For example, in the case of [4] when evaluating the number of Grover iterations to make against SHA-256, the size of the search space is 2^{256} and so the total number of queries, or Grover iterations, is 2^{128}. While this can be a useful metric, it assumes that all queries are equally expensive. In the next layer, we go deeper into the workings of this black box in order to decipher how expensive these queries are.

2.3.2 Logical Layer

During the logical layer, we construct and analyse quantum circuits required to implement our queries. Here, we will compute cost based on the *circuit depth*, *gate count*, and total number of *logical qubits*, including *ancillae*.

Definition 2.3.1 (Circuit Depth)**.** If every gate costs a unit time-step, the depth of a quantum circuit is the fewest number of time steps required to perform said quantum circuit. Gates which act on no common qubits can be performed in the same time step.

Definition 2.3.2 (Gate Count)**.** The total number of gates included in a quantum circuit from input to output.

Definition 2.3.3 (Logical Qubits)**.** A logical qubit specifies how a single qubit should behave in a quantum computation and is not subject to error.

Definition 2.3.4 (Ancillae Qubits)**.** An auxiliary qubit, often used for the temporary storage of intermediary values that will be uncomputed later.

Typically, during this stage, we will synthesize down to a universal gate set; a set of gates that we can use to express any unitary operation. Technically, such an endeavour is impossible with anything less than an uncountable set of gates, as the number of possible quantum gates is equally uncountable. To solve this problem, we instead deem a gate set universal if it is able to approximate any quantum operation [67]. A universal gate set aids heavily in the implementation of quantum algorithms as quantum hardware need only be capable of performing the gates included in such a set, and any gates outside the set can be converted using a compiler.

The most common method for creating a universal gate set is to use the Clifford set (H, S, and $CNOT$) and extend it to include one additional non-Clifford gate, usually the T-gate. We refer to this set as the Clifford+T gate set, and define the gates as follows:

$$H = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, \quad CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad T = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{bmatrix}$$

The reasoning behind this approach is due to the fact that Clifford gates are easy to keep fault-tolerant using stabilizer codes [45], the basis of most current quantum error correcting codes, however, alone do not form a universal set, and in fact can be efficiently simulated classically [46].

As quantum error correction is one of the major contributing factors to large-scale quantum algorithms, it is often important at this stage to look ahead to later layers and make optimizations that will influence our final cost metric. For example when working with Clifford+T we will look to minimize the depth and count of our T-gates specifically, as they are often vastly more expensive to construct on quantum hardware than the Clifford gates. We will discuss the details of why this is the case in the next section.

2.3.3 Error Correction and the Fault-Tolerant Layer

Up until this point, we have only considered quantum computations that behave ideally; the unfortunate truth is that physical quantum systems are subject to a considerable amount of noise. When building physical quantum computers, the qubits often take the form of individual atomic-scale physical systems, such as atoms, photons, or trapped ions. Despite the best efforts of quantum engineers to isolate these systems, there is still unwanted environmental factors that are able to become entangled with our qubits and generate noise. This noise can cause decoherence of our qubits, which will ultimately render our computations obsolete. As the size of our quantum computations increase, so too does the impact of noise due to the increase in requirements of both qubits and time.

In the early years of quantum computing, the idea of error-correction seemed impossible; whilst classical error-correction was a well established practice, it relied upon observing the state of our data in order to correct it, a process that when applied to qubits, would collapse our states and ruin our quantum computation. However, Shor [84] and Steane [87] both managed to derive a solution to this problem by spreading the information of one qubit onto several entangled qubits. We are able to make specifically-crafted measurements to retrieve information about an error (if one has occurred) without disturbing our quantum state. In the years that have followed since, there has been numerous improvements to quantum error-correction schemes,

however, they still operate similarly; first we encode multiple physical qubits to form logical ones, then we can perform gate operations periodically performing our measurements to detect and correct any errors, and finally decode our qubits back to their original form.

During the fault-tolerant layer, we choose an error-correction scheme and compute the resources necessary to keep our logical circuit fault-tolerant. Fault-tolerance in this setting refers to being able to correct errors faster than they are created to avoid decoherence. Current error-correction schemes seem to place the desired overall error probability of an algorithm to be around 1% in order to be considered fault-tolerant [37].

The current best known quantum error-correcting code is the *surface code*[39]. We omit the specifics of this scheme and instead refer interested parties to read [39]. However, at a high-level, the surface code encodes logical qubits using a two-dimensional lattice of physical qubits. The distance, d of our surface code refers to size of these lattices, a code with distance d can correct $\lfloor (d-1)/2 \rfloor$ errors. Physical qubits are divided into *data* qubits and *measure* qubits. After performing a logical operation on our system, we perform a *surface code cycle*, where the surrounding measure qubits will perform a stabilizer measurement of the data qubit in order to detect and correct errors.

The cost of error-correction is often linked with the overall number of T-gates used in a computation. The reason for this is that every T gate requires us to prepare a *magic state*:

$$|A_L\rangle = |0\rangle + e^{\frac{i\pi}{4}} |1\rangle \tag{2.26}$$

We prepare our magic states inside *distilleries* or *factories*, specialized circuits independent of the rest of our computation and embedded within their own surface code. Each time we wish to perform a T-gate within our circuit we must distil a magic state, that cannot be reused. Furthermore, magic state distillation is not a perfect process and each magic state produced is done so with some error. The amount of error can be reduced by increasing the sizes of our distilleries, however this inevitably comes at the cost of requiring more qubits, and time. We detail the process of constructing suitable distilleries in section 3.3, for now we note that the main objective of this layer is to construct suitably sized distilleries, and determine the code distance required to perform our computation fault tolerantly.

2.3.4 Physical Layer

In the final layer, we need to compute the total number of physical resources required to run our fault-tolerant algorithm. The resources will be defined based on the error

correction schemes, and parameters we use. By the end of this layer, we will have our final cost represented as number of physical qubits and the total amount of time required to run our algorithm. From this we can derive a single number that represents our cost:

$$\text{Cost} = \log_2(\#\text{ of physical qubits} \times \#\text{ of seconds}) \tag{2.27}$$

While there are other possible representations of cost, such as $\log_2(\#$ logical qubits $\times$ $\#$ surface code cycles) used in [4], we have chosen to represent costs based on physical qubits and seconds. The intention behind this is two-fold: first to allow for a clear comparison between classical cryptographic techniques that often report cost similarly, usually referred to as bits of security, and secondly to allow for comparison between our results, and any future methods that may not rely on surface code techniques.

2.4 Code Based Cryptography

Having now discussed the foundations of quantum algorithms and the methods behind resource estimation, we will now look at the foundations of the cryptosystem that we will be estimating an attack against. This section follows the literature [52, 72, 91] to cover the core concepts behind classical linear error-correcting codes and how they can be used as a building block for asymmetric cryptography. We then go on to cover information sets and how they can be used in information set decoding, the best known generic attack against code-based cryptosystems.

2.4.1 Linear Codes

Linear codes are another form of error correcting code. Not too dissimilar from the previously discussed quantum error correcting codes, the goal is to be able to recover a message that has been distorted by error introduced from a noisy communication channel [82]. The core idea remains the same; we *encode* messages by adding redundant information such that if any error were to occur during transmission, the message is still able to be *decoded* such that the original message can be obtained free from any error. The methodology for encoding and decoding, and the number of errors that can be corrected is dependent on the method used, as well as the size of the error-correcting code employed. Error-correction schemes that correct more errors are typically more expensive as they require more redundancy, reducing the ratio of message bits to redundant bits. To put it simply, a linear code is one where any linear combination of codewords will also result in a codeword. More formally, we can define a linear code as a vector subspace:

Definition 2.4.1. (Binary Linear Code) An $[n, k, d]$ linear code C is a k-dimensional subspace of the n length vector space $\mathbb{F}_2^n$, where $\mathbb{F}$ represents a finite field. The

vectors belonging to C are referred to as codewords. The minimum Hamming distance between two distinct codewords is d.

By taking the basis vectors of this vector space and using them as rows of a matrix, G, we are able to encode messages by multiplying them with G. It is for this reason that we refer to this as a *generator* matrix of the code. Generator matrices can also be represented in systematic (or standard) form — a systematic code is one where we append the redundant information to the end (or start) of the original message.

Definition 2.4.2. (Generator Matrix) The row space of a generator matrix, $G \in \mathbb{F}_2^{k\times n}$, forms the codewords of a linear code, C. To generate a codeword, x, for a message, m, $x = mG$. The systematic (or standard) form of a generator matrix is $G = [\mathbb{1}_k|P]$ where $\mathbb{1}_k$ is the $k \times k$ identity matrix and $P \in \mathbb{F}_2^{k\times(n-k)}$.

Once we have sent our encoded message over our channel, the receiver is able to verify if a codeword is valid by multiplying it with a corresponding *parity-check* matrix. The result of this multiplication will give you the *syndrome*. If the syndrome is $\mathbf{0}$, then the codeword is valid and no error has occurred. We can find the parity-check matrix for any code if we possess a generator matrix for the code in systematic form.

Definition 2.4.3. (Parity-Check Matrix) Given a linear code, C, any matrix $H \in \mathbb{F}_2^{(n-k)\times n}$ with $Hc^T = \mathbf{0}$, $\forall c \in C$ and $\text{RANK}(H) = n - k$, is a parity-check matrix of the code. Given a generator matrix in systematic form $G = [\mathbb{1}_k|P]$, the parity-check matrix is $H = [P^T|\mathbb{1}_{n-k}]$.

Definition 2.4.4. (Syndrome) Given a parity-check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$ of a code C and a vector $x \in \mathbb{F}_2^n$, we refer to the output of $s = Hx^T$, as the syndrome of x. A syndrome of $\mathbf{0}$ indicates $x \in C$.

But what if we do not possess a generator matrix in systematic form? Thankfully, it is possible for us to convert any generator matrix of a code to systematic form using a sequence of elementary row operations and column permutations, detailed in algorithm 1 [52]. If the first k columns of the generator G are linearly independent then we do not require the use of column permutations, which results in G' (our systematic form generator) producing an *equal* code to G, whereas if we require the use of column permutations the codes are only *equivalent*. Two codes are equal if given the same message, both will produce the same codeword, and equivalent if one code can be derived from the other by a permutation.

Algorithm 1 systematicForm

Input: Generator matrix, $G \in \mathbb{F}_2^{k \times n}$

Output: Generator matrix in systematic form, $G' = [\mathbb{1}_k | P]$

for all $j \in \{1, \ldots, k\}$ **do**
  if $G_{jj} = 0$ **then**
    if $\exists\, i > j$ where $G_{ij} \neq 0$ **then**
      swap rows i and j
    else if $\exists\, h > j$ where $G_{jh} \neq 0$ **then**
      swap columns j and h
    end if
  end if
  for all $i \in \{1, \ldots, k\}$, where $i \neq j$ **and** $G_{ij} \neq 1$ **do**
    add row j to row i
  end for
end for

To summarize, when sending a message the sender will first encode their message (as a binary string of length k) by multiplying it with a generator matrix to receive their length n codeword. After transmission, the receiver can then multiply the received codeword with the corresponding parity-check matrix. If the resulting syndrome $x \neq \mathbf{0}$ then we can deduce that an error has occurred and use our corresponding decoding algorithm to correct up to t-many errors, where $t \leq \lceil \frac{d-1}{2} \rceil$. If more than t errors occur, our decoding algorithm will incorrectly decode to a different codeword than was originally sent[1]. If instead our syndrome $x = \mathbf{0}$, then we can determine that either no error has occurred or that enough error has occurred that the original codeword has been distorted into a different codeword (the likelihood of this outcome relies entirely on how many errors your code is capable of detecting and how noisy the channel is). The specifics of decoding strategies is dependent on the exact error-correcting code you are using; we omit these details from this work as they are not necessary, and instead refer interested parties to [52]. In the next section, we will demonstrate an example of how it is possible to use a linear code as the basis of a public-key encryption scheme.

2.4.2 McEliece Cryptosystem

The first and perhaps most well-known code based cryptosystem was developed in 1978 by Robert McEliece [65]. This algorithm is an asymmetric or public-key encryption algorithm, where there exists two keys: a public-key and a private-key.

[1]There does exist algorithms that attempt to correct more than $\frac{d-1}{2}$ errors, known as list-decoding algorithms, that will output a list of close codewords. While these have use in code-based cryptosystems for this analysis we will assume that $t \leq \lceil \frac{d-1}{2} \rceil$.

The public key is openly available information that is used to encrypt a message that can be sent to an intended party. The receiver can then use the corresponding private key in order to decrypt and obtain the original message.

We will now explain the three core details behind McEliece, how we generate our pair of keys, how to encrypt a message, and how to decrypt the message. To aid our explanation we will be using the Alice and Bob notation introduced in [75], in this scenario Bob is attempting to send a message to Alice securely.

Key generation In order to create a public and private key, Alice does the following:

1. Alice selects a binary $[n, k]$ linear code C able to correct t errors with an efficient decoding algorithm A, this code will give her a $k \times n$ generator matrix G
2. Alice selects a random $n \times n$ permutation matrix P
3. Alice selects a random $k \times k$ binary invertible matrix S
4. Alice then scrambles G by computing $G' = SGP$
5. (G', t) is the public key and (S, P, A) is her private key

Encryption
If Bob wishes to send a k-length message m to Alice using her public key (G', t):

1. Bob computes the codeword $c = mG'$
2. Bob selects a random n-length vector e with exactly t ones i.e., $\text{wt}(e) = t$
3. Bob then adds the error to the codeword to obtain his ciphertext $y = c + e$

Decryption
Alice receives the ciphertext y from Bob and then can decrypt using her private key (G', S, P) to obtain the original message, m:

1. Alice multiplies the ciphertext y with the inverse of P to obtain $c' = yP^{-1}$
2. Alice uses her decoding algorithm A to decode c' into m'
3. Alice recovers the original message using the inverse of S, $m = m'S$

The security of this algorithm is reliant upon the NP-hard *linear decoding problem*, that states that it is hard to decode a **general** linear code[14]:

Problem 2.4.1. (Linear Decoding Problem) Given a random generator matrix $G \in \mathbb{F}_2^{k \times n}$ of a linear code C and a vector $y = mG + e$, $e \in \mathbb{F}_2^n$, $\text{wt}(e) = t$, recover the message, m.

However, this problem frequently is rephrased as a different, yet equivalent problem, the *syndrome decoding problem* [35]:

Problem 2.4.2. (Syndrome Decoding Problem) Given a random parity-check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$, and a syndrome $s \in \mathbb{F}_2^n$, find the error vector $e \in \mathbb{F}_2^n$ with $\mathrm{wt}(e) = t$, such that $He^T = s$.

The main idea behind McEliece, is that while the linear code C has an efficient decoding algorithm, the code C' (the code generated by G') is a **general** code and therefore is NP-hard to decode, assuming that it has been sufficiently obfuscated.

The McEliece cryptosystem has so far managed to stand the test of time and resisted cryptanalysis for over 40 years which have, for the most part, only required minor adjustments to the scheme overall [10, 16, 29, 64, 88][2]. Despite this, initially the system failed to gain much popularity due to the large key sizes, especially when compared to the vastly more popular RSA encryption scheme [75]. However, with the threat of quantum computing (namely Shor's algorithm [83]) condemning the future of RSA, the search for post-quantum cryptography has caused McEliece and other code-based cryptosystems to become a key area of research, with 'Classic McEliece' managing to reach round 3 of the NIST post-quantum submissions. [1, 17].

2.4.3 Information Sets and Information Set Decoding

In this section, we will cover the details of information sets, and the information set decoding (ISD) algorithm first presented by Prange [73]. The ISD algorithm is a generic, message recovery attack against code based cryptosystems. This means that if we are able to acquire a ciphertext, we can use ISD to recover the original message, irrespective of what code-based system it was encrypted with. ISD exploits the presence of information sets which exist due to the inherent redundancy within linear error-correcting codes; the amount of redundant bits for an $[n, k, d]$ linear code is $r = n - k$. Because of this redundancy, if we assume we have a codeword, c, with no errors $s = Hc^T = \mathbf{0}$, then we are able to discard up to r-many bits and still be able to recover the original message. We define an information set as any set that contains the remaining $n - r = k$ bits of the codeword, from which we are able to use to recover the message.

Definition 2.4.5. (Information Set) Given an $[n, k, d]$ linear code, C with a corresponding generator matrix $G \in \mathbb{F}_2^{k\times n}$. An information set, I, is any k-combination of the index set for columns of G i.e., $I \subseteq \{1 \ldots, n\}$, that gives rise to a submatrix $G_I \in \mathbb{F}_2^{k\times k}$ with $\textsc{rank}(G_I) = |I| = k$.

[2]McEliece as described above is not chosen-ciphertext or "IND-CCA2" secure, there are techniques for ensuring IND-CCA2 security, one such method can be found in [31].

Definition 2.4.6. (Combination) A k-combination of a set S is a subset of k distinct elements of S. The total number of k-combinations for a size n set is equal to the binomial coefficient $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

Using this property of information sets, we are able to decode a message if we assume that the rows of the error vector, indexed by our information set, are equal to zero [53][3]:

Theorem 1. *Given a vector $y = c + e$ and an information set I where the rows indexed by I of the error e gives rise to $e_I = \mathbf{0}$, it is possible to decode c to recover the original message as $m = y_I G_I^{-1}$.*

Proof. G_I is a square matrix with full rank, therefore invertible. For any codeword $c = mG$, we have $y_I G_I^{-1} = (c + e)_I G_I^{-1}$. As $e_I = \mathbf{0}$, we find $c_I G_I^{-1} = (mG)_I G_I^{-1}$. Using the observation that the jth row of c is only computed from linear combinations of the entries in the jth row vector of G, we can replace $c_I = (mG)_I = mG_I$ to get $mG_I G_I^{-1} = m$. □

Before we continue with the full definition of ISD, we will first rephrase definition 2.4.5 in order to define information sets in terms of parity-check matrices. The reason for doing so is purely to keep in line with the standard notation of ISD algorithms. Section 2.4.1 shows how we can construct a corresponding parity-check matrix, in the case that the public key we have access to is a generator matrix (as was the case with McEliece shown in section 2.4.2).

Definition 2.4.7. (Information Set of a Parity Check Matrix) Given an $[n, k, d]$ linear code, C with $r = n - k$ redundant bits and a corresponding parity check matrix $H \in \mathbb{F}_2^{r \times n}$. An information set I, is any k-combination of the index set for columns of H, i.e., $I \subseteq \{1 \ldots, n\}$ with complement $I^* := \{1, \ldots, n\} \setminus I$, that gives rise to a submatrix $H_{I^*} \in \mathbb{F}_2^{r \times r}$ with $\text{RANK}(H_{I^*}) = |I^*| = r$.

We can now define the information set decoding algorithm as originally stated by Prange [73]. At its core, Prange's algorithm is a brute-force searching algorithm that attempts to find the error vector, e, in order to solve the syndrome decoding problem. The algorithm will permute the columns of the parity-check matrix, H, which will also permute the positions of 1s in e. The goal is to find a permutation π, such that the weight of the first k positions of $\pi(e)$ are error-free, allowing us to decode. More formally:

1. Choose uniformly at random an information set I
2. Select a permutation matrix $\pi \in \mathbb{F}_2^{n \times n}$ that corresponds to permuting the I-indexed columns to the right-hand side of H, giving $H_\pi = H\pi$

[3]This assumption is valid so long as $n - k \geq t$, as it is possible for all 1s in the error vector to be contained in the redundant $n - k$ bits.

3. We now have a new (unknown) error $\epsilon = e\pi^{-1}$ obtained by permuting the I-indexed columns of e to the left-hand side

4. Use elementary row operations R to bring H_π to systematic form $\hat{H} = RH_\pi = [Q|\mathbb{1}_r]$

5. Apply the same operations R to the syndrome $\hat{s} = sR$

6. We now have $\epsilon \cdot \hat{H}^T = \hat{s}$ which we can rewrite as $\left[\, \epsilon_I \mid \epsilon_{I^*} \,\right] \left[\frac{Q^T}{\mathbb{1}_r}\right] = Q^T\epsilon_I + \epsilon_{I^*} = \hat{s}$

At this stage, we now need to check whether our criterion $\epsilon_I = \mathbf{0}$ has been satisfied. To check this criterion, we observe that if it were satisfied, then the Hamming weight[4] of $\hat{s}$ would be equal to t as $\mathrm{wt}(\epsilon) = \mathrm{wt}(\epsilon_{I^*}) = \mathrm{wt}(\hat{s}) = t$. If we find that $\hat{s}$ satisfies this weight condition then we can recover the original error vector as $e = \epsilon\pi$ [53]. If our criterion has not been met, then we repeat the entire process with a different information set. We present a fully detailed version of Prange in algorithm 2.

Theorem 2. *Only the original error vector e satisfies $wt(e) = t$ as long as $t \leq \lfloor \frac{d-1}{2} \rfloor$.*

Proof. Let's assume there is another error vector $e' \neq e$ where $He'^T = s$, and $wt(e') \leq \lfloor \frac{d-1}{2} \rfloor$. The syndrome of our original error vector e is also $He^T = s$, therefore $He'^T = He^T$. Equivalently, this gives us $H(e' - e)^T = 0$. From definition 2.4.4 we find that $Hx^T = 0$ indicates $x \in C$, therefore we find $(e' - e) \in C$. However, $(e' - e)$ has weight, $wt(e' - e) \leq \lfloor \frac{d-1}{2} \rfloor + \lfloor \frac{d-1}{2} \rfloor \leq d - 1$, which is less than the minimum distance d of the code C and therefore cannot be a codeword of $C \rightarrow$ Contradiction. □

Finally, let us discuss how expensive this whole algorithm is. The main cause of cost is the number of guesses we need to make for π. The probability that a randomly chosen group of columns satisfies the criterion of $\epsilon_I = \mathbf{0}$ is $\binom{r}{t}/\binom{n}{t}$ [56], however we also need to factor in the probability that there is a chance that our systematic form operation could fail (due to requiring column permutations), which occurs roughly 29% of the time [7], which adjusts our probability to $Pr_{Prange} = (0.2887 \cdot \binom{r}{t})/\binom{n}{t}$ giving us an average of $Itr_{Prange} = (1/Pr_{Prange})$ iterations before we expect to find a solution. Over the years there have been many improvements to the ISD algorithm [10, 29, 60, 64, 88]; whilst we won't discuss the specific details of these variants, they essentially all function in the same manner to Prange, however, rely on different criteria to be met for ϵ_I, which increase the overall probability of finding a correct permutation at the cost of more expensive checks. Algorithm 2 can be adjusted to any of these variants by simply adjusting the search function.

[4]The Hamming weight, $wt(x)$, is equivalent to the Hamming distance between x and $\mathbf{0}$

Algorithm 2 isdPrange

Input:

Parity check matrix, $H \in \mathbb{F}_2^{r \times n}$
Cipher text, $y = c + e$
Error weight, $t = wt(e)$

Output:

Error vector, e

$s \leftarrow Hy^T$
while true **do**
    $\hat{H}, \pi, \hat{s} \leftarrow$ RANDOMIZE(H, s)
    $success, \epsilon \leftarrow$ SEARCH$(\hat{H}, \hat{s}, t)$
    if $success = true$ **then**
        $e \leftarrow \pi\epsilon$
        return e
    end if
end while

function RANDOMIZE(H, s)
    Choose a random permutation matrix $\pi \in \mathbb{F}_2^{n \times n}$
    $H_\pi \leftarrow H\pi$
    $\hat{H} \leftarrow$ SYSTEMATICFORM(H_π)
    // Return false if we require column operations, as our chosen permutation does not give rise to an information set
    // Store row operations as R
    // We now have $\hat{H} = RH_\pi = [Q \mid \mathbb{1}_r]$
    $\hat{s} = sR$
    return $\hat{H}, \pi, \hat{s}$
end function

function SEARCH$(\hat{H}, \hat{s}, t)$
    if $wt(\hat{s}) = t$ **then**
        $\epsilon_I \leftarrow \mathbf{0} \in \mathbb{F}_2^k$
        $\epsilon \leftarrow [\, \epsilon_I \mid \hat{s} \,]$
        return $true, \epsilon$
    else
        return $false, \mathbf{0}$
    end if
end function

Chapter 3

Resource Estimation of Quantum Information Set Decoding

In this chapter, we will be detailing our research into estimating the resources required to perform quantum information set decoding (QISD). We will be following the methodology set out in [4], detailed in section 2.3. The sections of this chapter follow the layers of this methodology as follows:

- Section 3.1 — Classical query model
- Section 3.2 — Logical layer
- Section 3.3 — Fault tolerant and physical layers
- Section 3.4 — Analysis of results

We will begin by demonstrating how to perform Prange's algorithm using a quantum computer, before constructing the circuit using our chosen gate set. After which, we will embed our circuit within a suitable error-correction code such that it is deemed fault-tolerant. Finally, we provide an analysis of the level of security that proposed code based parameters ensure against a quantum adversary.

3.1 Quantum Information Set Decoding

The first work published with a goal to creating a quantum variant of ISD to attack code-based cryptography was by Bernstein [15] in 2010. Bernstein's paper looked at the idea of using Grover's search to speed up the classic Prange ISD algorithm. As detailed in section 2.4.3, Prange's algorithm is a search algorithm that attempts to find a correct permutation, equal to selecting $r = n - k$ columns of the parity-check matrix and moving them to one side of the matrix. We deem a permutation correct if it gives rise to an information set, I, such that the criterion $\epsilon_I = \mathbf{0}$ is satisfied, which can be verified using $\mathrm{wt}(\epsilon) = t$. Bernstein observed that if we prepare a

Algorithm 3 quantumPrange

Input:

$|\pi\rangle$, n length register initialized to $|\mathbf{0}\rangle$
$|H\rangle$, $n \times r$ Parity-check matrix
$|s\rangle$, Syndrome
$|Aux\rangle$, auxiliary qubit initialized to $|-\rangle$

Output:

π, column permutation

1: Prepare uniform superposition of all combinations in register $|\pi\rangle$
2: Apply permutations $|\pi\rangle$ to $|H\rangle$
3: Perform row reduction to the last r columns of $|H\rangle$
4: Apply the same row operations from line 3 to $|s\rangle$
5: Calculate weight of $|s\rangle$
6: Apply oracle function $\text{wt}(|s\rangle) = t$ to $|Aux\rangle$
7: Reverse operations 2-5
8: Apply Grover's diffusion operator to $|\pi\rangle$
9: Repeat lines 2-8 $Itr_{qPrange} = \frac{\pi}{4}\sqrt{\binom{n}{t}/(0.2887 \cdot \binom{r}{t})}$ times
10: $\pi \leftarrow$ measure $|\pi\rangle$
11: **return** π

state containing all possible column selections, we can use Grover's search with an oracle function that tests if the resulting sub-matrix is full rank and produces an ϵ that satisfies our criterion, allowing us to reduce the expected run time of Prange's algorithm. Algorithm 3 shows the details of this quantum variant of Prange, equivalently figure 3.1 details a high-level quantum circuit of quantum Prange.

The goal of this section is to determine the number of queries we need to make, in our particular case this is equivalent to calculating the number of Grover's iterations. Noting that Grover's requires $\frac{\pi}{4}\sqrt{N/|A|}$ iterations, as N is the size of the search space and $|A|$ is the size of the set of solutions, it can equally be thought of as $1/Pr$, where Pr is the probability that a random $n \in N$ also satisfies $n \in A$. For Prange's algorithm, we know that the probability of finding a correct permutation is $Pr_{Prange} = (0.2887 \cdot \binom{r}{t})/\binom{n}{t}$, therefore, we can derive that the optimal number of Grover's iterations for Prange's algorithm is $Itr_{qPrange} = \frac{\pi}{4}\sqrt{1/Pr_{Prange}}$. Table 3.1 shows the number of Grover's queries evaluated for a number of parameter sets proposed by the Classic McEliece NIST submission [17]. It is worth to note here, that while we could simply stop our analysis at this point, the aim of our research is to find a lower bound of required resources for performing QISD. In doing so, we are able to help better inform researchers to the capability of QISD and the requirements necessary for optimally ensuring quantum-immunity.

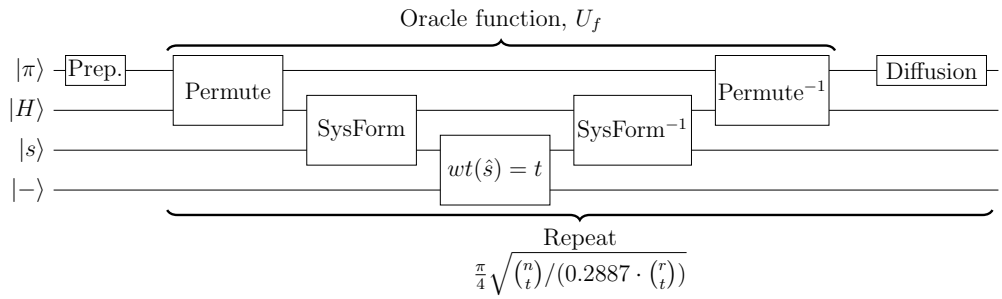


Figure 3.1: High-level quantum circuit for Performing Prange's algorithm

Parameters			Grover Iterations
n	k	t	
3488	2720	64	71.94
4608	3360	96	92.99
6688	5024	128	131.72
6960	5413	119	132.27
8192	6528	128	150.61

Table 3.1: Required number of Grover iterations to perform quantum Prange against proposed Classic McEliece parameter sizes [17]. We represent Grover iterations as $log_2(Itr)$.

3.2 Construction of Quantum Prange

With the first layer of our methodology complete, we can now begin to detail our attempts to construct quantum circuits to perform quantum Prange. We will begin this section with defining how we will be modelling cost during this layer, in order to explain what we are attempting to optimize for. After which, we will describe quantum circuits to perform the state preparation, permute, weight, and diffusion subroutines of quantum Prange (see figure 3.1). It should be noted at this point that we do not present a quantum circuit for the sysForm routine (sometimes referred to as row reduction or Gaussian elimination), however, we will show in our analysis in section 3.4 that in spite of this, we can still derive useful results due to how expensive the other routines are, namely state preparation.

3.2.1 Cost Model

In order to help our optimizations during this layer, we will first make the assumption that our computation will be running on a quantum computer that uses a surface code architecture in order to achieve fault tolerance. This is motivated by current research suggesting surface codes are the best-known method for error-correction [39].

Assumption 1. *The resource costs of a large scale quantum computation can be well approximated by the resource costs required to run that computation on a surface code based quantum computer.*

The impact of this assumption, whilst not obvious, helps give us a better cost metric for our overall analysis. The surface code (introduce in section 2.3.3) typically is implemented to work over the Clifford+T gate set, where it is able to perform fault-tolerant Clifford gates with relative ease. The downside of course is the heavy cost incurred when using T-gates, as these require the expensive processes of state injection and distillation in order to be produced at a low enough error rate for large scale quantum computations. Using this fact, we will assume that T-gates are what we intend to optimize for and will report as our costs at this stage.

Assumption 2. *The resource costs of a computation performed on a surface code based quantum computer can be well approximated by the resource costs required to perform only the T-gates of that computation.*

This gives us two important details to account for when constructing our circuits; the first is that we intend to report our costs using three values, number of logical qubits, T-gate count, and T-gate depth. The second detail extends from this, as it also implies that all Clifford gates in our constructions will be deemed 'free', as they will not contribute directly to any of these metrics.

Definition 3.2.1 (T-Gate Count)**.** The T-count is the total number of T-gates included in a quantum circuit from input to output. Let T_U^c denote the T-count for an arbitrary unitary operator U.

Definition 3.2.2 (T-Gate Depth)**.** Given a quantum circuit with a gate-depth, d. The T-depth of the circuit is the number of time-steps that contain one or more T gates. Let T_U^d denote the T-depth for an arbitrary unitary operator U.

3.2.2 Common Quantum Circuits

We will begin our construction by first looking at some common circuits that will be used in multiple places within our construction. To begin with it helps for us to have a definition of the three Pauli operators (often referred to as X, Y, and Z) as they are frequently used. We provide constructions for each of these gates in figure 3.2; it is important to note that all of these operators can be constructed using only single qubit Clifford gates and as such can be deemed free based on our cost model.
The next operator we will be in need of is the Toffoli gate. The Toffoli gate can be seen as an extension of the controlled-NOT gate, where it has two controls rather than one. This will cause the state of the target qubit to flip if, and only if, both control qubits are in the $|1\rangle$ state. This gate has many uses, most notably is when the target qubit is set to $|0\rangle$ it acts as a logical AND operator on the control qubits.

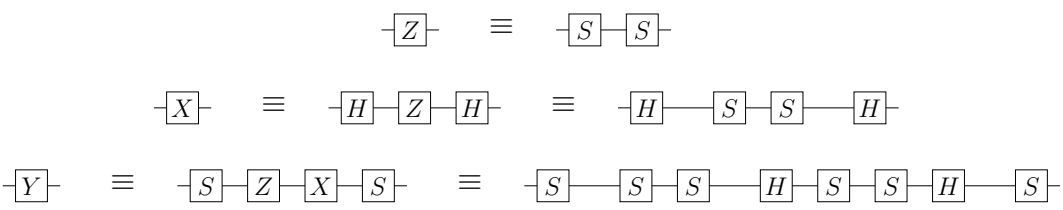


Figure 3.2: Construction of the Pauli operators using only Hadamard H and phase S gates belonging to the Clifford group.

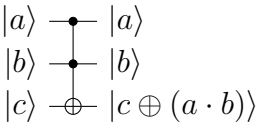


Figure 3.3: Quantum circuit notation for a Toffoli gate. The dots correspond to the control qubits, and the $\oplus$ denotes the target qubit of the gate. The output state on the qubit $|c\rangle$ is equal to $c\ XOR\ (a\ AND\ b)$.

There have been many attempts to construct minimal circuits for Toffoli gates, however, the current best known approach in the literature requires $T^c_{Tof} = 7$, and $T^d_{Tof} = 3$ [3]. There does exist a depth 1 construct presented by Selinger [79], however this construction comes at the cost of 4 ancillae per Toffoli gate (although these ancillae can be reused). The debate of using ancilla to reduce T-depth is one that will come up numerous times throughout our construction, for now we will simply show the available options and discuss the different approaches in later sections. There does also exist another optimization trick, where if one Toffoli gate is later uncomputed by another, then both gates in the pair can omit 3 T-gates from their cost [8].

Furthermore, it's possible for us to extend the Toffoli to include an arbitrary number of controls. We refer to this gate as a *mixed polarity multiple control Toffoli* (MPMCT) gate. The MPMCT is a Toffoli gate that contains $c > 2$ control or anti-control bits, an anti-control acts in the same manner as a control however will only apply the gate if the anti-control is in the $|0\rangle$ state. Di Matteo, Gheorghiu, and Mosca [25] demonstrate that by using $n - 1$ ancillae they are able to perform such a gate where $c \geq 4$ with a T-count of $T^c_{MPMCT} = 12c - 20$, and a T-depth of $T^d_{MPMCT} = 4c - 8$. Importantly, the $n - 1$ ancillae are returned to their original state after use and so can be reused if we need to perform multiple MPMCT gates in succession.

The final common constructions we will be presenting here are those for quantum arithmetic, namely addition, subtraction, and comparison. Currently, there is a multitude of methods for addition each with their own pros and cons depending on whether you wish to use more space (ancillae) or time (circuit depth) [23, 27, 28, 43, 90]. The most common method for performing addition is a quantum variant of the ripple carry adder, originally proposed by Vedral, Barenco, and Ekert [90]

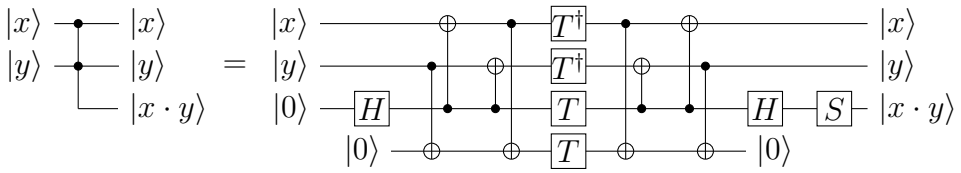


Figure 3.4: Circuit notation and construction of AND gate with $T^c_{AND} = 4$, $T^d_{AND} = 1$ and requiring 1 ancilla. Notation is from [43] and the construction is from [54].

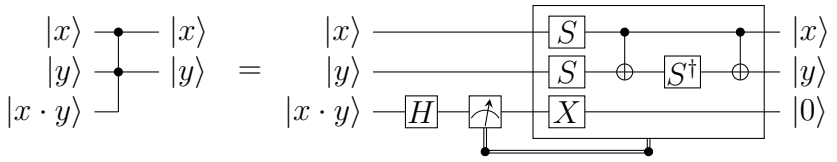


Figure 3.5: Circuit notation and construction of AND† gate requiring zero T gates. Notation is from [43] and the construction is from [54].

and further improved by Cuccaro et al. [23]. The ripple carry adder works by using Toffoli gates to calculate the carry for each set of bits recursively, after which the carry bits can then be uncomputed, and the sum can be calculated using $CNOT$ gates. This construction leads to using $2n - 1$ total Toffoli gates and a single ancilla to find the sum of two n-bit integers, however, we can form the Toffoli gates into $n - 1$ compute-uncompute pairs with one additional Toffoli. Using the values from above, each of these pairs have a T-count of 8, leading to a total T-count of $8n - 1$ with a depth of $2n + 1$.

Gidney [43] proposes a cheaper alternative to Toffoli gates that they refer to as *temporary logical-AND* gates. These logical-AND gates require the use of an ancilla qubit (to store the output) but only require $T^c_{AND} = 4$ and $T^d_{AND} = 1$, and require zero T-gates to uncompute (see figures 3.4 & 3.5). Utilizing these new gates, the authors propose a new scheme for quantum addition, requiring only a T-count of $4n - 4$ and a T-depth of $n - 1$ at the cost of an additional $n - 1$ ancillae. We report the costs of the adders from [23] and [43] in table 3.2.

Now we have established the costs required to perform addition, we can use the same constructions to also perform subtraction and comparisons by utilizing signed binary integers (either ones or twos complement). To find $a - b$ we can simply calculate $a + (-b)$, where $-b$ is the corresponding complement of b. To compare two signed

Method	T-Count	T-Depth	Ancilla
Cuccaro [23]	$8n - 1$	$2n + 1$	1
Gidney [43]	$4n - 4$	$n - 1$	$n - 1$

Table 3.2: Comparison of time vs. space for using Toffoli or AND gates to perform ripple carry addition.

integers, a and b, all we need to do is calculate the sign bit of the output from $a - b$; if the sign bit is 0, then $a \geq b$, or $a < b$ if the sign bit is one.

3.2.3 State Preparation

We can now start with the first stage of our circuit, state preparation. The goal of this portion is to create a superposition over our search space. In our use case, we are attempting to search over a subset of permutations that are equal to choosing r-many columns of the parity-check matrix H. As permutation matrices are rather large to store as well as difficult to prepare for our specific case, we instead propose to prepare a superposition of all possible combinations (see definition 2.4.6). We will represent combinations as binary strings of length n equal to the size of set, each with weight r, where the positions of the 1s correspond to selection of columns. As an example, if we had a set $S = \{1, 2, 3, 4, 5, 6\}$, where each entry corresponds to an index of a column in a matrix, the binary string 100110 would represent the combination $C \subset S = \{1, 4, 5\}$. Within the literature of quantum mechanics, there exists such a state that is a superposition of all n-length binary strings with fixed weight r, known as *Dicke states* [26].

Definition 3.2.3. (Dicke State, $|D_r^n\rangle$) A uniform superposition of all n-qubit states $|x\rangle$, with *Hamming weight*, $\text{wt}(x) = r$, i.e.,

$$|D_r^n\rangle = \binom{n}{r}^{-\frac{1}{2}} \sum_{x \in \{0,1\}^n, wt(x)=r} |x\rangle$$

Dicke states have a number of uses within quantum computing, such as quantum metrology [89], quantum game theory [70], quantum networking [74], and quantum optimization (QAOA) [33, 49]. Due to the variety of uses, the question of efficiently preparing Dicke states is one of great importance. Over this section, we will present two possible methods the first is our own design utilizing the *combinatorial number system*, and the second is an algorithm proposed by Bärtschi and Eidenbenz [9].

Combinatorial Number System

The combinatorial number system (CNS), is a bijective correspondence between natural numbers (including 0) and combinations. CNS has uses in classic computing as a means to directly generate a combination from a specific index [58]. CNS comes from an observation by Lehmer [61], who demonstrated that given a combination of choices $\{c_r, c_{r-1}, \ldots, c_1\}$ from the set $\{1, \ldots, n\}$ where $n > c_r > c_{r-1} > \cdots > c_1 \geq 0$ we can compute a value $N = \sum_{i=1}^{r} \binom{c_i}{i}$, this N has a unique value such that $0 \leq N < \binom{n}{r}$. This means that given any integer, N, such that, $0 \leq N < \binom{n}{r}$, we are able to find its corresponding r-combination of n elements. To transform a

combination to its corresponding integer in CNS we use a *ranking* algorithm, the inverse of this process (converting an integer to a combination) uses an *unranking* algorithm, we present the ranking and unranking algorithms for CNS in figures 4 and 5 respectively.

Algorithm 4 cnsRank

Input:
bit string, $x \in \{0,1\}^n$
Output:
integer N
1: $N \leftarrow 0$
2: $r \leftarrow 1$
3: **for all** $i \in \{0, \ldots, n-2, n-1\}$ **do**
4: **if** $x_i = 1$ **then**
5: $N \leftarrow N + \binom{i}{r}$
6: $r \leftarrow r + 1$
7: **end if**
8: **end for**
9: **return** N

Algorithm 5 cnsUnrank

Input:
integers r, n, N such that $0 \leq N < \binom{n}{r}$
Output:
bit string *out*, representing Nth rank r-combination of n elements
1: $out \leftarrow \{0\}^n$
2: **for all** $i \in \{n-1, n-2, \ldots, 0\}$ **do**
3: **if** $N \geq \binom{i}{r}$ **then**
4: $N \leftarrow N - \binom{i}{r}$
5: $out_i \leftarrow 1$
6: $r \leftarrow r - 1$
7: **end if**
8: **end for**
9: **return** *out*

We can construct a state preparation method as follows: we begin by preparing a register in superposition of all integers $0 \leq N < \binom{n}{r}$ using Hadamard gates, from which we use cnsUnrank on all integers in the register simultaneously in order to create our Dicke State. In order to construct such a circuit, we require a quantum implementation of cnsUnrank, which itself requires two key components:

1. Arithmetic (discussed in section 3.2.2)

2. Calculation of binomial coefficients

The biggest hurdle to overcome in performing cnsUnrank is the latter of those two, as we need to compare and subtract different binomial coefficients depending on

the value of r, which varies over the superposition. There exists efficient algorithms for classically computing such values in $O(r)$ time and $O(1)$ space, however, rely upon extensive use of both multiplication and division [40] which are both expensive processes to perform on a quantum computer. There is potential that with some fine-tuning, this arithmetic-based approach could be a faster method by constructing quantum circuits that multiply by a fixed number, due to the fact that it uses known multiplicands and divisors, however, we instead will be using an approach based on quantum RAM (lookup tables). qRAM is analogous to classical RAM by computing all required binomial coefficients ahead of time we can construct a circuit that can search for a value corresponding with a specific address:

$$\sum_j \alpha_j |j\rangle |0\rangle \xrightarrow{\text{qRAM}} \sum_j \alpha_j |j\rangle |b_j\rangle \tag{3.1}$$

Where $\sum_j \alpha_j |j\rangle$ is the address(es) of the query we are making to the qRAM and $|b_j\rangle$ is the jth memory location. How to efficiently construct such a circuit is a question of great importance to quantum computing as many algorithms require the existence of qRAM such as HHL [51]. Di Matteo, Gheorghiu, and Mosca [25] performed a resource estimation on two variants of qRAM: either loading classical data to be stored on quantum bits (known as bucket-brigade [6]) or constructing a network of n-bit MPMCTs that are controlled on the address bits and will set the relevant bits in the output register. In our implementation, we plan to use the MPMCT approach for two reasons: firstly the binomial coefficients to be calculated would require a large amount of space to store and query using bucket brigade methods and secondly as we know the contents of our memory we can employ optimization strategies [81] to reduce the overall cost. As an example of how to implement our qRAM, assume we have a known value n that we wish to compute $\binom{n}{r}$ for $0 \leq r \leq n/2$, choosing $n = 6$ we obtain the table 3.3.

Address	Output
00	00001
01	00110
10	01111
11	10100

Table 3.3: Memory table of $\binom{6}{r}$, where address is the value of r and output is the corresponding binomial coefficient.

Figure 3.6 shows an implementation of table 3.3 using MPMCT gates. Each MPMCT uses controls that check for a specific address and then sets the corresponding output bits. The cost of this implementation is dependent on the number of bits we need to set, as well as the cost of each individual MPMCT gate. Using the costs from section 3.2.2, the cost of an n-controlled MPMCT (where $n \geq 4$) is

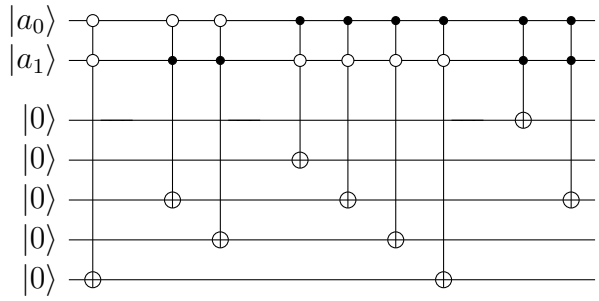


Figure 3.6: qRAM implementation of table 3.3 using MPMCT gates to set the output for its corresponding address.

$T^c_{MPMCT} = 12n - 20$, $T^d_{MPMCT} = 4n - 8$ requiring an additional $n - 1$ ancillae. The number of MPMCT gates required depends on the specifics of the data, from table 3.3 we have a total of 9 qubits in the output register set to 1, and 11 qubits set to 0. Therefore, it is cheaper for us to construct MPMCT gates to set the 1s of the output, if we had a table with a greater number of 0s then we would instead initialize the output to $|1\rangle^{\otimes n}$ using X gates and apply MPMCT gates to set relevant qubits to 0.

There is also a further adjustment that we must consider if we desire to trade space for time, if we were to fan out our address register in order to make copies we could perform more MPMCT gates in parallel at the cost of ancillae qubits. We are limited in the number of gates we can do in parallel based on the maximum number of operations on any one output qubit, for example in figure 3.6 the 3rd output qubit (5th from the top) is operated on 3 times therefore the lowest depth we can achieve is 3. Figure 3.7 demonstrates how we can construct a 3 depth MPMCT variant by creating two additional copies of the address register.

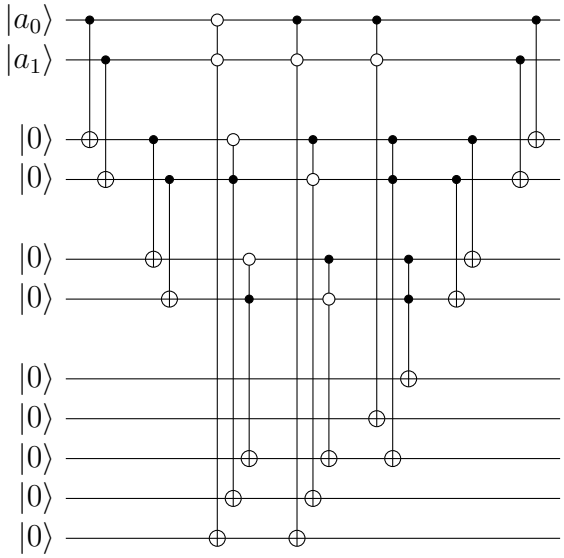


Figure 3.7: Parallel qRAM implementation of table 3.3 using control fan out to achieve an MPMCT depth of 3.

If we assume we have a memory table with addresses of length n corresponding to an m-length output with w many total bits to set (either 1s or 0s, whichever is fewer) and a maximum of d operations to be performed on any one output qubit, then we can calculate the cost of implementation using an MPMCT qRAM, we show these costs on table 3.4 including both the sequential and parallel variants. It is also important to note that there is further room for optimization of MPMCT qRAM circuits however, these techniques rely on abusing the specific structure of an MPMCT network for a given data set [66, 81].

Method	T-Count	T-Depth	Ancilla
Sequential	$w(12n-20)$	$w(4n-8)$	$n-1$
Parallel	$w(12n-20)$	$d(4n-8)$	$\lfloor \frac{w}{d} \rfloor n + \lceil \frac{w}{d} \rceil (n-1)$

Table 3.4: Cost estimate for implementation of a generic, non-optimised, qRAM using an MPMCT with $T^c = 12n - 20$ and $T^d = 4n - 8$ using $n - 1$ ancillae, not included in the table is the space required to store both the input and output qubit registers.

With the building blocks laid out for arithmetic and calculation of binomial coefficients, we can detail our construction of cnsUnrank and the costs to perform such a function. Looking once more at algorithm 5 the first step of each iteration is to perform a qRAM lookup on address r for our values of $\binom{i}{r}$. For each of our n iterations, we will require a different memory table t_i; the number of entries for a given table are dependent on the possible values of r, as i is fixed and decrements every iteration but, r varies for each input value N. For example t_{n-1} will always be of size 1 as during the first iteration the value of r is the same throughout, t_{n-2} will be of size 2 as the value of r will have decreased for some values of N in the superposition but not for others. This pattern increases until the rth iteration where $|t_{n-1-r}| = r+1$ however, from here there are no more possible values of r and as such the size of table shall remain at $r + 1$ until, $i < r$ at which point the size decreases by 1 each iteration until the final table t_0 will be of size 2. Table 3.5 demonstrates a full table for a smaller example of $\binom{12}{4}$, from this analysis we can calculate that the total number of qRAM queries we must make is $r(n - r) + n$ however as the first table is only of size 1 (and thus does not require a qRAM implementation) the total number of table entries is $r(n - r) + n - 1$ which can be simplified to $kr + n - 1$ as $r = n - k$. The algorithm begins with a register $|N\rangle$ containing the superposition of input integers to unrank, including a sign bit, $|r\rangle$ will originally be set to its relevant input value, we then require two registers initialized to $|\mathbf{0}\rangle$ one to hold the output of the qRAM lookup and the second to store the output bit string. We start by performing our qRAM lookup then subtracting its output from N, we then set the output bit based on the sign bit of N and then perform a controlled addition to

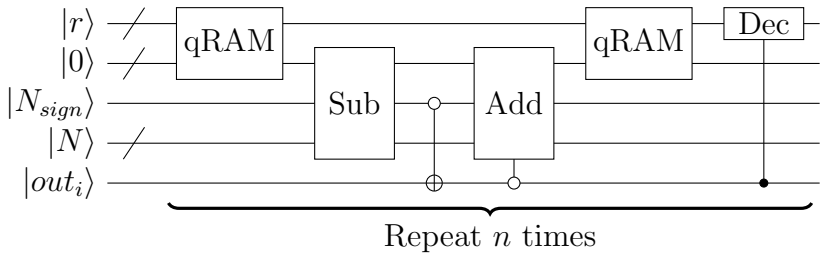


Figure 3.8: Black box implementation of algorithm 5 in order to create a superposition of r-combinations of n elements.

reverse the subtraction in the event that $N < \binom{i}{r}$, we then reverse our qRAM, and finally we decrement r only if the output has been set to 1. Figure 3.8 shows the process required to perform cnsUnrank and table 3.6 shows the T-count and T-depth cost of running the whole algorithm.

Table no.	Values
t_{11}	$\binom{11}{4}$
t_{10}	$\binom{10}{4},\binom{10}{3}$
t_9	$\binom{9}{4},\binom{9}{3},\binom{9}{2}$
t_8	$\binom{8}{4},\binom{8}{3},\binom{8}{2},\binom{8}{1}$
t_7	$\binom{7}{4},\binom{7}{3},\binom{7}{2},\binom{7}{1},\binom{7}{0}$
t_6	$\binom{6}{4},\binom{6}{3},\binom{6}{2},\binom{6}{1},\binom{6}{0}$
t_5	$\binom{5}{4},\binom{5}{3},\binom{5}{2},\binom{5}{1},\binom{5}{0}$
t_4	$\binom{4}{4},\binom{4}{3},\binom{4}{2},\binom{4}{1},\binom{4}{0}$
t_3	$\binom{3}{4},\binom{3}{3},\binom{3}{2},\binom{3}{1},\binom{3}{0}$
t_2	$\binom{2}{3},\binom{2}{2},\binom{2}{1},\binom{2}{0}$
t_1	$\binom{1}{2},\binom{1}{1},\binom{1}{0}$
t_0	$\binom{0}{1},\binom{0}{0}$

Table 3.5: Example table of values that need to be classically computed and implemented as qRAM for $n = 12, r = 4$.

Bärtschi and Eidenbenz Method

Bärtschi and Eidenbenz [9] present an alternative method to preparing Dicke states. This approach relies upon constructing a unitary $D_{n,r}$[1], that takes an input state $|0\rangle^{\otimes n-r}|1\rangle^{\otimes r}$ and will output the corresponding Dicke state $|D_r^n\rangle$. In the paper, they demonstrate that it is possible to construct such a unitary by using *split and cyclic shift* or $SCS_{n,r}$ gates recursively. These gates will not alter a state composed only of all-zero or all-one, however if acting on a state with zeroes appended with ones, will create a superposition where a zero has been permuted to the last qubit.

[1]In the original publication, the authors use the notation $U_{n,r}$. To prevent confusion with our oracle functions, U_n, we have re-labelled this operator.

Process	T-count	T-depth
qRAM	$w_{t_i}(12R-20)$	$d_{t_i}(4R-8)$
Subtract	$4N-4$	$N-1$
ctrl-Add	$8N-8$	$9N+6$
Decrement	$4R-4$	$R-1$
1 round	$w_{t_i}(12R-20)+12N+4R-16$	$d_{t_i}(4R-8)+10N+R+4$
n rounds	$2W(12R-20)+12Nn+4Rn-16n$	$2D(4R-8)+10Nn+Rn+8n$

Table 3.6: Total costs to run cnsUnrank in order to obtain a register in uniform superposition of r-combinations of n elements using qRAM (parallel approach) to load pre-computed binomial coefficients, where $R=\lfloor \log_2 r \rfloor + 2$, $N=\lfloor \log_2 \binom{n}{r} \rfloor + 2$, w_{t_i} = no. of bits to set in t_i and $W=\sum_{i=0}^{n-1} w_{t_i}$, d_{t_i} = depth of bits to set in t_i and $D=\sum_{i=0}^{n-1} w_{t_i}$. This algorithm requires $2N+R+n$ logical qubits, plus an additional $\max\left(\lfloor \frac{w_{t_i}}{d_{t_i}} \rfloor\right)n + \max\left(\lceil \frac{w_{t_i}}{d_{t_i}} \rceil\right)(n-1)$ ancillae.

Definition 3.2.4. (Split & Cyclic Shift) A unitary operator, $SCS_{n,r}$, where for all $l \in 1,...,r$, where $r<n$:

$$SCS_{n,r}\left|0\right\rangle^{\otimes r+1} = \left|0\right\rangle^{\otimes r+1}$$
$$SCS_{n,r}\left|0\right\rangle^{\otimes r+1-l}\left|1\right\rangle^{\otimes l} = \sqrt{\frac{l}{n}}\left|0\right\rangle^{\otimes r+1-l}\left|1\right\rangle^{\otimes l} + \sqrt{\frac{n-l}{n}}\left|0\right\rangle^{\otimes r-l}\left|1\right\rangle^{\otimes l}\left|0\right\rangle$$
$$SCS_{n,r}\left|1\right\rangle^{\otimes r+1} = \left|1\right\rangle^{\otimes r+1}$$

Using an inductive approach (lemma 2 from the original paper), the authors were then able to derive the following equation to construct an arbitrary $D_{n,r}$ using only SCS gates:

$$D_{n,r} := \prod_{l=2}^{r}(SCS_{l,l-1}\otimes \mathbb{1}^{\otimes n-l}) \cdot \prod_{l=r+1}^{n}(\mathbb{1}^{\otimes l-r-1}\otimes SCS_{l,r}\otimes \mathbb{1}^{\otimes n-l})$$

As an example, if you wanted to create the state $\left|D_3^5\right\rangle$ you would start with the state $\left|0\right\rangle^{\otimes 2}\left|1\right\rangle^{\otimes 3}$ and apply the operations $(\mathbb{1}\otimes SCS_{5,3})$, $(SCS_{4,3}\otimes\mathbb{1})$, $(SCS_{3,2}\otimes\mathbb{1}^{\otimes 2})$, $(SCS_{2,1}\otimes\mathbb{1}^{\otimes 3})$, we demonstrate this example in figure 3.9.

What remains now to show is how we can construct $SCS_{n,r}$, using Clifford+T gates. The authors demonstrate a method using r-many controlled Y-rotation gates, where if the control qubit is in the $\left|1\right\rangle$ state then $R_y(\theta)$ is applied to the state where,

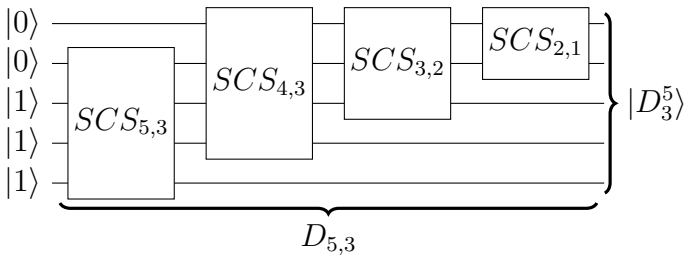


Figure 3.9: Quantum circuit to implement $D_{5,3}$ using recursive SCS gates.

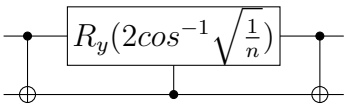
$$
\begin{aligned}
R_y(\theta) &= \begin{bmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix} \\
R_y(\theta)\,|0\rangle &= \cos(\frac{\theta}{2})\,|0\rangle + \sin(\frac{\theta}{2})\,|1\rangle \\
R_y(\theta)\,|1\rangle &= -\sin(\frac{\theta}{2})\,|0\rangle + \cos(\frac{\theta}{2})\,|1\rangle
\end{aligned}
\tag{3.2}
$$

Each of the r-many gates implements one of the mappings for $l \in 1...r$ given by definition 3.2.4. The authors define two more operators named (i) and $(ii)_l$, where (i) is a two qubit gate to implement $l = 1$ and $(ii)_l$ is a three qubit gate to implement the other $r - 1$ choices for $2 \leq l \leq r$.

(i) $\quad |00\rangle \to |00\rangle$

$\quad |11\rangle \to |11\rangle$

$\quad |01\rangle \to \sqrt{\frac{1}{n}}\,|01\rangle + \sqrt{\frac{n-1}{n}}\,|10\rangle$



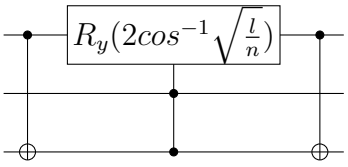
$(ii)_l$ $\quad |000\rangle \to |000\rangle$

$\quad |001\rangle \to |001\rangle$

$\quad |010\rangle \to |001\rangle$

$\quad |111\rangle \to |111\rangle$

$\quad |011\rangle \to \sqrt{\frac{l}{n}}\,|01\rangle + \sqrt{\frac{n-l}{n}}\,|10\rangle$



To continue on from our example before, if we wanted to perform an $SCS_{5,3}$ gate, we would construct a circuit using one (i) gate followed by $(ii)_2$ and $(ii)_3$. Figure 3.10 shows a full implementation of $D_{5,3}$ using (i) and $(ii)_l$ gates. In total this construction requires $(n-1)$ controlled rotation, CR_y gates and $(n-r)(r-1) + \sum_{i=3}^{r}(i-2)$ two-controlled rotation, CCR_y gates. The total depth cost of this construction is also linear in n, this is due to $(ii)_r$ of $SCS_{n,r}$ being capable of running in parallel with up to $r^* := \lfloor \frac{r+1}{3} \rfloor - 1$ many other gates, namely $(ii)_{r-3}, (ii)_{r-6}, ..., (ii)_{r-3r^*}$ of $SCS_{n-1,r}, SCS_{n-2,r}, ..., SCS_{n-r^*,r}$.

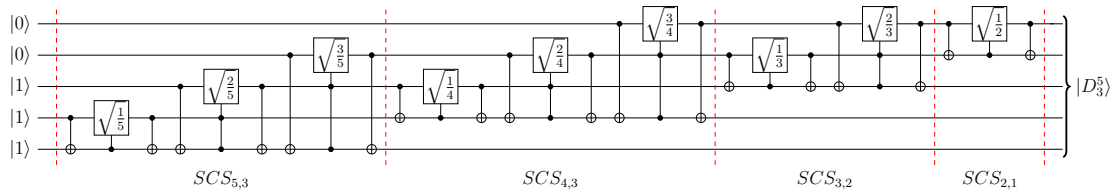


Figure 3.10: Quantum circuit from [26], to perform $D_{5,3}$ using SCS gates. $\sqrt{\frac{l}{n}}$-gates are shorthand for $R_y(2cos^{-1}\sqrt{\frac{l}{n}})$, mapping $|0\rangle \rightarrow \sqrt{\frac{l}{n}}\,|0\rangle + \sqrt{\frac{n-l}{n}}\,|1\rangle$.

In order to perform our controlled rotations using Clifford+T, we use a $CNOT$ gate to conditionally apply an X-gate, this X will then cause any Y-rotations to be inverted due the axis being orthogonal i.e, $XR_y(\theta)X = R_y(-\theta)$. Looking at figure 3.11, we can see that when the control qubit is in the state $|0\rangle$, we rotate the target qubit by $R_y(\theta/2)$ immediately followed by $R_y(-\theta/2)$ returning the qubit to its original state. However, when the control is in the state $|1\rangle$ the target will be rotated by $R_y(\theta/2)$ twice, causing a total rotation of $R_y(\theta)$. We use the same principle for our two-control CCR_y gate, however, this time we now split the angle up into four parts where the gates will cancel each other out in the event that either none, or one, of the controls is in the state $|0\rangle$ and rotate the state by $R_y(\theta)$ if both controls are in the $|1\rangle$ state.

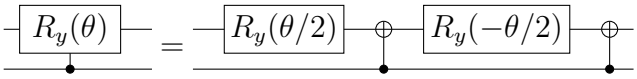


Figure 3.11: Implementation of a controlled Y-rotation utilizing $CNOT$ gates to invert the second rotation gate if the control is in the state $|1\rangle$.

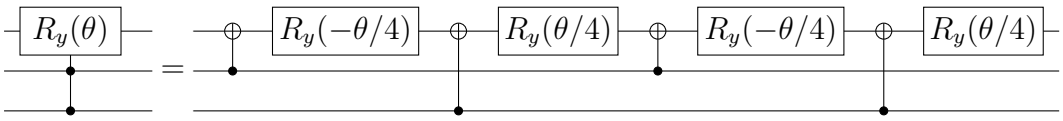


Figure 3.12: Implementation of a two-controlled Y-rotation utilizing $CNOT$ gates to invert the Y-rotation gates.

Breaking up our CR_y and CCR_y gates allows us to calculate the total number of single qubit rotations, $\#R_y$, we require:

$$\begin{aligned} \#R_y = 2 \cdot \#CR_y + 4 \cdot \#CCR_y &= 4rn - 2r^2 - 2r - 2n + 2 \\ &= 2(n-1)^2 - 2(k^2 - k) \end{aligned} \tag{3.3}$$

Now all that remains is a method to construct arbitrary Y-rotation gates using the Clifford+T gate set. This topic is one of great importance and as a result has been thoroughly explored with many proposed algorithms to synthesize an arbitrary rotation gate within some margin of error ϵ. One of the major reasons behind this research is that by synthesizing arbitrary rotation gates, we can synthesize any single

qubit unitary [57, 78]. The major result we will be using in our analysis is from Ross and Selinger [77], who demonstrated that with access to a factoring oracle i.e., a quantum computer, it is possible to optimally synthesize a Z-rotation, within a margin of error ϵ, using $O(log(1/\epsilon))$ T-gates, which can easily be extended to apply for both Y and X rotations, as they only differ by Clifford operators [78]. Their results also show that even without such an oracle, their algorithm still finds a solution using $3log_2(1/\epsilon) + O(log(log(1/\epsilon)))$ in the typical case. Due to current estimates [44] demonstrating that Shor's algorithm is *relatively* inexpensive compared to our analysis, it seems fair to assume that we are able to synthesize single qubit Y-rotations using $3log_2(1/\epsilon)$ many T-gates. The value of ϵ is chosen such that the total sum of error for each rotation gate is negligible e.g., 0.01%, giving $\epsilon < \frac{0.0001}{\#R_y}$. From this we can derive the total cost of the Bärtschi and Eidenbenz method:

$$T^c_{BE} = 3log_2(\frac{\#R_y}{0.0001}) \cdot \#R_y$$

$$T^d_{BE} = 3log_2(\frac{\#R_y}{0.0001}) \cdot (2 \cdot \#CR_y + \frac{4 \cdot \#CCR_y}{r^*})$$

Comparison of Methods

The previous sections introduced two possible methods for the state preparation step of our circuit. But the question remains as to which method is optimal. We report on table 3.7 the cost to implement both methods for the parameter sets outlined in table 3.1.

Parameters			CNS		B&E	
n	k	t	T^c	T^d	T^c	T^d
3488	2720	64	1.20×10^{33}	7.64×10^{29}	1.40×10^{31}	6.50×10^{28}
4608	3360	96	8.50×10^{39}	3.43×10^{36}	7.66×10^{37}	2.20×10^{35}
6688	5024	128	1.08×10^{52}	3.15×10^{48}	8.95×10^{49}	1.92×10^{47}
6960	5413	119	1.56×10^{52}	4.65×10^{48}	1.28×10^{50}	2.96×10^{47}
8192	6528	128	7.57×10^{57}	2.03×10^{54}	6.09×10^{55}	1.30×10^{53}

Table 3.7: Comparison of costs for two methods of preparing a Dicke state for various proposed parameter sets of Classic McEliece [17].

It is clear to see that the Bärtschi & Eidenbenz (B&E) method surpasses CNS in both count and depth of T-gates, further more CNS requires additional space as it needs to hold both the integer N, the n-length Dicke state, and requires additional ancillae for performing the qRAM look up, whereas the B&E method constructs the Dicke state in place and requires no additional space. However, CNS does have one key advantage over B&E, where B&E prepares the entire Dicke state, CNS is

able to prepare only parts of the state dependent on the values of N provided. This allows for our construction to be easily run distributively, across multiple quantum computers if desired. Furthermore, CNS as presented in this work is suboptimal, as we have ignored a number of potential optimization tricks for our qRAM [25, 66, 81], or alternative implementation approaches, that require a deeper knowledge of the internal structure. Ultimately, for our analysis, we will still use the costs of the B&E as we won't be analysing distributed costs, but due to the importance of Dicke states, we felt it worthwhile to present CNS as a potential alternative in future analysis.

3.2.4 Permuting the Parity-Check Matrix

With a method now set out for achieving a Dicke state, the next stage in the algorithm is to be able to apply the combinations to the parity-check matrix H to achieve H_π. The goal of this process is to use a combination to select its respective r-many chosen columns (represented by 1s) of H so that they can be confirmed, via row reduction, if they form an information set. One particular facet that will feature in this section is that Prange's algorithm only requires us to perform operations on the selected columns, a trait that is not shared by the other ISD variants. This allows us to save on space as we only need to store the required columns of H.

With that in mind we move on to the construction of a circuit that loads only the columns selected by a given combination state. First, as H is public and classically stored we are able to load its respective columns using a network of X gates. However, we need to find a method of only loading the columns specified by the combination. The method we propose in this section is to create a matrix of qubits where each row will have *at most* one non-zero entry and each column will have *exactly* one non-zero entry. We will then use each row of this matrix to determine which column of H is loaded and where. We begin by defining two operators used in our construction:

Definition 3.2.5. (Prepend 1s) Unitary operator, PRE_n, acting on $n+1$ qubits with a control qubit and n targets, where for all l, with $0 \leq l < n$:

$$PRE_n \left|1\right\rangle \left|1\right\rangle^l \left|0\right\rangle^{n-l} = \left|1\right\rangle \left|1\right\rangle^{l+1} \left|0\right\rangle^{n-l-1}$$
$$PRE_n \left|0\right\rangle \left|1\right\rangle^l \left|0\right\rangle^{n-l} = \left|0\right\rangle \left|1\right\rangle^{l} \left|0\right\rangle^{n-l}$$

Definition 3.2.6. (Eliminate 1s) Unitary operator $ELIM$, acting on n qubits, where for all l, with $1 \leq l \leq n$:

$$ELIM \left|0\right\rangle^{n-l} \left|1\right\rangle^l = \left|0\right\rangle^{n-l} \left|1\right\rangle \left|0\right\rangle^{l-1}$$

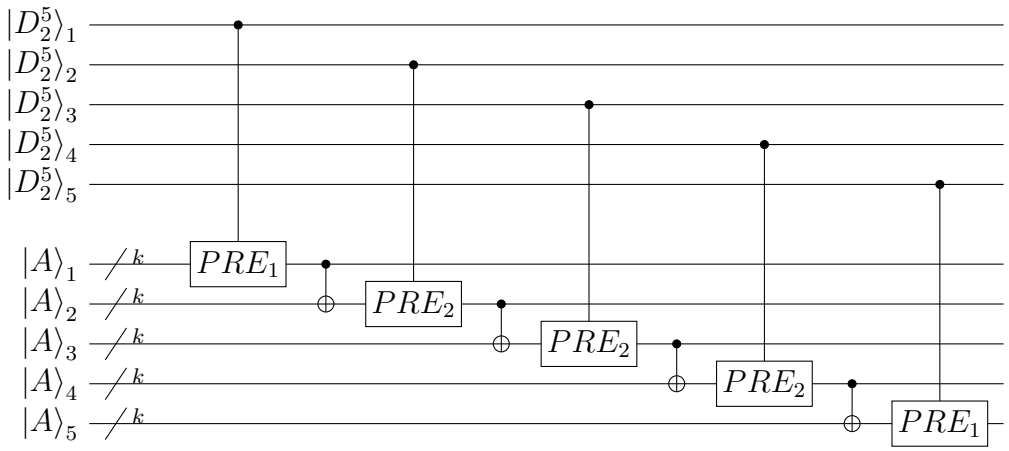


Figure 3.13: Quantum circuit implementation to generate a matrix to be used for permuting the parity-check matrix for the Dicke state $|D_2^5\rangle$. Where $|D_2^5\rangle_i$ and $|A\rangle_i$ indicates the i-th qubit of the Dicke state and i-th row of the matrix respectively.

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

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

Figure 3.14: Matrix representation of Fig. 3.13, looking at one such r-weighted string, 01100, that is part of the Dicke state $|D_2^5\rangle$ and showing how each PRE gate and $CNOT$ effects the matrix state $|A\rangle$.

We now use an $r \times n$ matrix of ancillae, $|A\rangle$ with all entries set to $|0\rangle$, we then iterate through qubits $i \in 1, ..., r$ of the Dicke state using PRE_i on the i-th row of $|A\rangle$ controlled on $|D_r^n\rangle_i$ making sure to use $CNOT$ gates after each PRE gate to copy the completed row down to the next. Once we reach PRE_r we can abuse what we know of the structure of the Dicke state, and rather than increase to PRE_{r+1} we can continue to use PRE_r as we know there will be at most r 1s in the state. We continue to use PRE_r another $n - 2r$ times at which point we can then perform PRE_i for $i \in r, ..., 1$ as we again can determine which positions must have a 1 based on the fixed weight of the Dicke state. This gives us a total of n PRE gates. Figure 3.13 shows an example of how to use our PRE gates to prepare our ancilla matrix for the state $|D_2^5\rangle$, figure 3.14, shows what the matrix $|A\rangle$ would look like for one of the strings contained in $|D_2^5\rangle$.

After we have applied all of our n PRE gates, we then perform $ELIM$ on each column of $|A\rangle$, this will result in every column having only one non-zero value and each row having at most one non-zero value, an example of how to construct $ELIM$

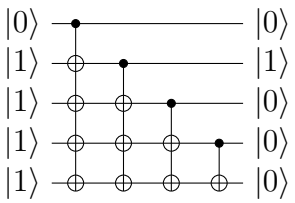


Figure 3.15: Implementation of $ELIM$ using cascading $CNOT$ gates.

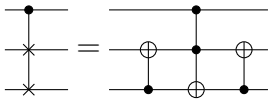


Figure 3.16: Construction of a Fredkin gate using 2 $CNOT$ gates and a Toffoli gate. When the top control qubit is set to $|1\rangle$, the target qubits will swap positions.

is given in figure 3.15. We will then iterate through each entry of $|A\rangle$, if the entry a_{ij} is equal to 1, then we will load the ith column of H into the jth column of H_π conversely if it is equal to 0 then we ignore it. The circuits for $ELIM$ and loading can both be performed using only $CNOT$ gates and as such we can assume they are free in our cost model. This means all that remains is to show a construction for our PRE gates. In order to construct PRE_r we will simply use $r-1$ cascading Fredkin gates (also known as controlled-$SWAP$ gates) which, if the control qubit is in the $|1\rangle$ state will transform the state $|a_{i,1}, a_{i,2}, ..., a_{i,r-1}, a_{i,r}\rangle$ to $|a_{i,r}, a_{i,1}, a_{i,2}..., a_{i,r-1}\rangle$, after which a $CNOT$ will transform the qubit into a 1. We show in figure 3.16 that a Fredkin gate can be constructed using only one Toffoli gate and 2 $CNOT$ gates [86]. In order to perform PRE_r we require $r-1$ Fredkin gates, figure 3.17 shows an example for the gate PRE_5. From this we can calculate the total cost for permuting as:

$$T^c_{Perm} = 7(nr - n - r^2 + r)$$

$$T^d_{Perm} = 3(nr - n - r^2 + r)$$

We also require an additional nr ancillae to store $|A\rangle$, which can be uncomputed and reused.

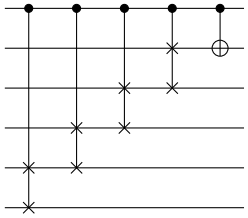


Figure 3.17: Implementation of PRE_5 using 4 Fredkin gates and a single $CNOT$. When the top control qubit is set to $|1\rangle$ a 1 is prepended to the binary string e.g., $PRE(|1\rangle \otimes |11000\rangle) = |1\rangle \otimes |11100\rangle$.

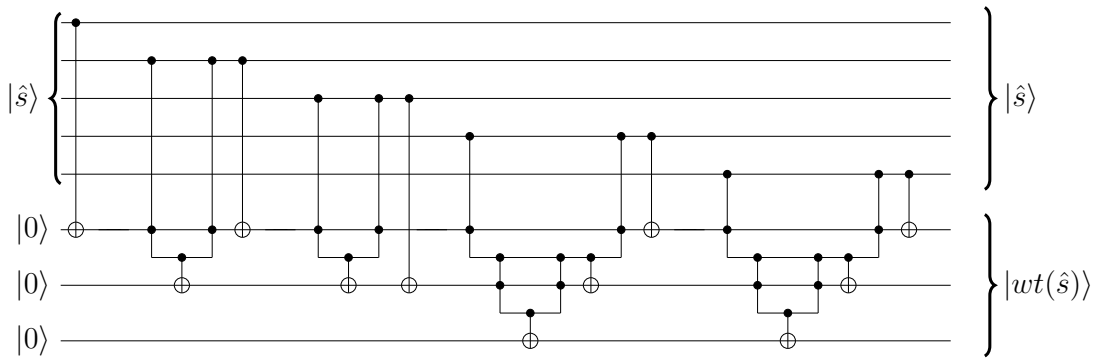


Figure 3.18: Circuit to calculate Hamming weight of a 5 length input constructed using quantum AND gates.

3.2.5 Hamming Weight & Phase Kickback

We can now move on to the final part of our oracle function, the goal of this section is to calculate the Hamming weight of our syndrome $\hat{s}$ and use that to perform phase kickback, conditioned on $\text{wt}(\hat{s} = t)$. Assuming we have a syndrome of length $r = n-k$ of unknown weight, we require a temporary register of length $R = \lfloor \log_2 r \rfloor + 1$ to store a binary integer up to r. We then will use each qubit of $\hat{s}$ as an incoming carry to a series of half adders, which will act as a controlled increment gate. As an example, if we wanted to compute the hamming weight of a syndrome of length 5, we require 3 qubits to store the sum, then let us assume we have a gate INC_i, which will increment an i qubit register by 1, we are able to use INC_i a maximum of 2^i times before having to scale up to INC_{i+1}. Therefore, for our example of length 5 we will use 1 INC_1, 2 INC_2, and 2 INC_3 gates. To implement our INC gates we will use the adder built from AND gates introduced in section 3.2.2, giving us the cost of $T^c_{INC_i} = 4(i-1)$, $T^d_{INC_i} = i-1$, and $i-1$ ancillae to store the output of our AND gates. Figure 3.18 shows our example of a syndrome of length 5 implemented using AND gates.

The largest increment gate required is R, which we need to use a total of $(r+1)-2^{R-1}$ times, with each copy costing $4R-4$. We can then sum the cost of the smaller increment gates using $\sum_{i=1}^{R-1} 2^{i-1}(4i-4) = 2^{R+1}(R-3)+8$.

Once we have calculated the total weight of $\hat{s}$ the last step is to apply phase kickback using our register initialized to $|-\rangle$, this is easily performed using just one R sized MPMCT gate with the controls set to our desired weight t. After we have applied our MPMCT gate the last step is to uncompute our weight function which is done by running the whole circuit in reverse such that it acts as a decrement gate. We display a summary of the costs of this section in table 3.8.

Process	T-count	T-depth	Ancilla
$\text{INC}_{1\leq i<R}$	$2^{R+1}(R-3)+8$	$2^{R+1}(R-3)+8$	$R-1$
INC_R	$(4R-4)(2^R-2r-2)$	$(R-1)(r-2^{R-1}+1)$	R
MPMCT	$12R-20$	$4R-8$	$R-1$
Total	$8r(R-1)-2^{R+3}+20R-12$	$2r(R-1)-(11\cdot 2^R)+3(2^R+2)R+6$	R

Table 3.8: Total cost to perform phase kickback based on a Hamming weight function constructed using AND gates where $r=n-k$ and $R=\lfloor \log_2 r \rfloor + 1$. Total ancillae is reduced by reusing the ancillae, total cost also includes the extra step of uncomputing the weight function.

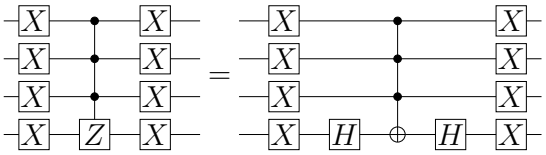


Figure 3.19: Construction of U_0, using single qubit gate operations and one MPMCT gate.

3.2.6 Grover's Diffusion Operator

The final piece required for our construction is Grover's diffusion operator. In section 2.2 we detailed that to perform the diffusion operator we implement the gates $VU_0V^\dagger$, over our input qubits where V is the operation used to prepare our search space, and $V^\dagger$ is its respective inverse. In our case V is dependent on whether we're using the CNS or B&E method of state preparation, in the case of CNS V, and $V^\dagger$ are the Hadamard gate as we are searching over integers, whereas B&E is prepared directly and so V would be the same algorithm as outlined in that section. Ultimately, these costs are the same as with the CNS method we are still required to uncompute our state preparation prior to diffusion. What remains to be analysed is how to construct the gate U_0. Using equation 2.24, we are able to construct this operator by first applying X gates to all of our input qubits followed by using a multiply-controlled Z gate, and then finally returning our qubits to their original state by applying X gates again. Figure3.19 demonstrates how we can instead construct U_0 by replacing the controlled Z gate with two Hadamard gates and an MPMCT gate. Overall the cost of U_0 is equivalent to a single MPMCT gate, with a number of controls equal to the number of input qubits we have, in the case of CNS unranking the cost is $T^c = 12N-20$, $T^d = 4N-8$, and uses $N-1$ ancillae, where $N = \lfloor \log_2 \binom{n}{r} \rfloor + 2$, and for B&E the cost is $T^c = 12n-20$, $T^d = 4n-8$, and uses $n-1$ ancillae.

3.2.7 Putting it all Together

With our construction outlined, we can now derive the full cost for our circuit. Table 3.9 shows a breakdown of the costs for each part of our circuit for a single parameter set, $n=3488$, $k=2720$, $t=64$, from this we can see that the cost of state

preparation is the major bottleneck of quantum Prange, contributing over 99% of the overall T-count. Table 3.10 shows the overall costs for the suggested parameters for Classic McEliece [17].

Process	T-count	T-depth
State preparation	1.40×10^{31}	6.50×10^{28}
Permute	1.31×10^{29}	5.65×10^{28}
Hamming weight	2.14×10^{26}	1.51×10^{26}
Diffusion	1.88×10^{26}	6.30×10^{25}
Total	1.41×10^{31}	1.22×10^{29}

Table 3.9: Full costs for running our construction of quantum Prange, for parameters $n = 3488$, $k = 2720$, $t = 64$. The algorithm also requires 3.28×10^6 logical qubits, including ancillae.

Parameters			T-count	T-depth	Logical Qubits
n	k	t			
3488	2720	64	1.41×10^{31}	1.22×10^{29}	3.28×10^6
4608	3360	96	7.69×10^{37}	3.44×10^{35}	7.32×10^6
6688	5024	128	8.98×10^{49}	3.05×10^{47}	1.40×10^7
6960	5413	119	1.28×10^{50}	4.61×10^{47}	1.32×10^7
8192	6528	128	6.11×10^{55}	2.01×10^{53}	1.64×10^8

Table 3.10: Costs of running quantum Prange, given in terms of T-gates and qubits (including ancillae) for parameter sets suggested in [17].

3.3 Fault Tolerance

The next stage of our methodology is to take our circuit constructions and embed them within a suitable error correction scheme such that they can be considered fault-tolerant. In section 3.2.1 we made the assumption that we intend to use the surface code scheme, this is due to current research indicating that surface codes are the best-known method for fault-tolerant quantum computing. There are a number of papers available discussing different methods of estimating algorithms using surface code error correction schemes [4, 25, 36, 38, 63]. For our estimates we will be using the 'defect' approach introduced in [38], while this approach may report larger estimates than the more recent 'lattice surgery' methods [38, 63], unfortunately the lattice surgery methods do not report methodologies for handling algorithms as large-scale as the one we present here.

Algorithm 6 Estimating the required number of rounds of magic state distillation and the corresponding distances of the concatenated codes

Input:

p_g, p_{out}

Output:

$d = [d_1, \ldots, d_i]$

1: $d \leftarrow$ empty list [ ]
2: $p \leftarrow p_{out}$
3: **while** $p < p_{in}$ **do**
4: Find minimum odd integer d_i such that $192 d_i (100 p_g)^{\frac{d_i+1}{2}} < \frac{p}{2}$
5: $p \leftarrow \sqrt[3]{p/70}$
6: d.prepend(d_i)
7: **end while**
8: **return** d

As discussed in section 2.3.3, in order to perform T-gates on the surface code we require magic states, $|A_L\rangle$. We prepare these states faultily with error probability p_{in} using state injection, and then use dedicated circuits known as magic state distilleries in order to purify them to our desired fidelity. As errors are additive, our required error rate is $p_{out} = 1/T^c$. For our analysis we will be using the Reed-Muller 15-to-1 distillation scheme, which takes 15 magic states of error rate p_{in} in order to produce one magic state with an error rate $p_{out} = 35 p_{in}^3$[20]. Using the results from [62], we assume that our injected states have an error rate equal to the per gate error rate of the surface code p_g. We will follow the most commonly reported optimistic value for the per gate error rate of $p_g = 10^{-4}$. This means using our Reed-Muller distillation schemes we are only capable of producing magic states with error rate $p_{out} = 3.5 \times 10^{-11}$, which is insufficient for the number of T-gates in our constructions. However, it is possible for us to concatenate multiple layers of distilleries in order to further purify our states, if we produce 15 states of error rate 3.5×10^{-11}, and then use them as input to another distillery we are able to achieve error rates of $35(3.5 \times 10^{-11})^3 = 1.50 \times 10^{-30}$. This same process can be repeated until we have states of sufficiently high fidelity. Algorithm 6, first presented in [4], provides a method for estimating the number of layers of distillation to produce magic states of our desired fidelity, as well as the surface code distances required to keep our magic state distilleries fault-tolerant.

We will now detail the process of resource estimation of a surface code based fault-tolerant quantum computation for the parameter set $n = 3488$, $k = 2720$, $t = 64$. From table 3.10, we have $T^c = 1.41 \times 10^{31}$, therefore our desired error rate for our magic states is $p_{out} = 1/T^c = 7.09 \times 10^{-32}$. Using algorithm 6 assuming $p_g = 10^{-4}$, we find that we require 3 layers of distillation with code distances, $d_1 = 7$, $d_2 = 15$, $d_3 = 35$.

Our first layer of distillation requires the most resources, in order to supply enough

magic states to successive layers we require $N_{d_1} = 16 \times 15 \times 15 = 3600$ logical qubits encoded on a distance $d_1 = 7$ surface code[2]. Using results from [39], we find that we require $\lceil 2.5 \times 1.25 \times d_1^2 \rceil = 154$ physical qubits per logical qubit, giving us a total footprint of $N_{d_1} \times 154 = 5.54 \times 10^5$ physical qubits, and completes a round of distillation in $10d_1 = 70$ surface code cycles.

The second layer of distillation requires $N_{d_2} = 16 \times 15 = 240$ logical qubits encoded on a distance $d_2 = 15$ surface code, which requires $\lceil 2.5 \times 1.25 \times d_2^2 = 704 \rceil$ physical qubits per logical qubit, for a total of $N_{d_2} \times 704 = 1.69 \times 10^5$ physical qubits and completes a round of distillation in $10d_2 = 150$ surface code cycles.

Finally, the third layer of magic state distillation uses a distance $d_3 = 35$ surface code, which uses $\lceil 2.5 \times 1.25 \times d_3^2 \rceil = 3829$ physical qubits per logical. This gives us a total footprint of $16 \times 3829 = 6.13 \times 10^4$ physical qubits, and completes a round of distillation in $10d_3 = 350$ surface codes cycles.

We are able to reuse the qubits required for the first layer of distillation in the second and third layers, for this reason the total number of physical qubits for our concatenated distillation scheme is 5.54×10^5, and can produce a sufficiently error-prone magic state in $70 + 150 + 350 = 570$ surface code cycles. Furthermore, as the second layer of distillation has a smaller footprint than the first, we can simultaneously produce $\lfloor (5.54 \times 10^5)/(1.69 \times 10^5) \rfloor = 3$ magic states per round of distillation. Therefore, to produce our 1.41×10^{31} T-gates, a single distillery would require $t_{dist} = 570 \times (1.41 \times 10^{31}/3) = 2.68 \times 10^{33}$ surface code cycles.

We now have to calculate the surface code distance required to keep our logical qubits in our circuit error-free. Fowler and Gidney [36] show that the error rate per logical qubit per code cycle can be approximated as:

$$P_L(p_g, d) = 0.1(100p_g)^{(d+1)/2} \tag{3.4}$$

As we require a total error probability for our whole algorithm to be below 1%, we can solve the following inequality to find a sufficient code distance, d:

$$\text{logical qubits} \times \text{surface code cycles} \times P_L(p_g, d) < 0.01 \tag{3.5}$$

From table 3.10 we find that the total number of logical qubits for our parameter set is 3.28×10^6, and we previously found that the total number of surface code cycles required to produce our T-gates is 2.68×10^{33}, giving us a surface code distance of $d = 41$. Using a $d = 41$ surface code requires $\lceil 2.5 \times 1.25 \times 41^2 \rceil = 5254$ physical qubits per logical qubit, giving us a total of $5254 \times 3.28 \times 10^6 = 1.72 \times 10^{10}$ physical qubits.

[2]The Reed-Muller scheme requires 16 logical qubits, 15 to hold our magic states that are consumed and an additional ancilla that can be reused.

Using only one distillery we are able to produce 3 magic states per 570 code cycles, therefore the time taken to produce all $T^c = 1.41 \times 10^{31}$ would require $(T^c/3) \times 570 = 2.70 \times 10^{33}$ surface code cycles. However, by adding more distilleries we can reduce this time significantly at the cost of some additional qubits. Let $T^w = T^c/T^d$ denote the T-width of an algorithm, the width tells us the average number of T-gates per layer of T-depth. If we construct enough distilleries such that we can distil T^w magic states per round of distillation, then we are able to execute our circuit in a time proportional to our T-depth, rather than T-count. Continuing on with our example values, we have $T^w = 1.41 \times 10^{31}/1.22 \times 10^{29} = 115$, as each distillery produces 3 magic states per distillation round we would therefore require $\lceil T^w/3 \rceil = 39$ distilleries in total. The number of physical qubits required for 39 distilleries is $39 \times 5.54 \times 10^5 = 2.16 \times 10^7$.

We can now derive the full cost of our surface code implementation of quantum Prange. We require $1.72 \times 10^{10} + 2.16 \times 10^7 = 1.73 \times 10^{10}$ physical qubits and $570 \times 1.22 \times 10^{29} = 6.95 \times 10^{31}$ surface code cycles. Assuming the time taken to perform a single surface code cycle $t_{sc} = 1\mu s$ [36, 63], our computation will take 6.95×10^{25} seconds. We detail the results of our resource estimation for all proposed parameter sets in table 3.11.

		Proposed Classic McEliece Parameter Set — $[n, k, t]$				
		[3488, 2720, 64]	[4608, 3360, 96]	[6688, 5024, 128]	[6960, 5413, 119]	[8192, 6528, 128]
Prange	T-count	1.41×10^{31}	7.69×10^{37}	8.98×10^{49}	1.28×10^{50}	6.11×10^{55}
	T-depth	1.22×10^{29}	3.44×10^{35}	3.05×10^{35}	4.61×10^{47}	2.01×10^{53}
	Logical qubits	3.28×10^{6}	7.32×10^{6}	1.40×10^{7}	1.32×10^{7}	1.64×10^{8}
	Code distance	41	47	61	61	67
	Physical qubits	1.71×10^{10}	5.05×10^{10}	1.63×10^{11}	1.54×10^{11}	2.30×10^{12}
Distilleries	Logical qubits per distillery	3600	3600	3600	3600	3600
	No. of distilleries	39	56	147	139	101
	Code distances	$[7, 15, 35]$	$[9, 17, 43]$	$[9, 21, 55]$	$[9, 21, 55]$	$[11, 23, 61]$
	Physical qubits	2.16×10^{7}	5.12×10^{7}	1.34×10^{8}	1.27×10^{8}	1.38×10^{8}
Totals	Physical qubits	1.72×10^{10}	5.06×10^{10}	1.63×10^{11}	1.54×10^{11}	2.30×10^{12}
	Seconds	6.95×10^{25}	2.37×10^{32}	2.59×10^{44}	3.92×10^{44}	1.91×10^{50}
	Cost	119.85	143.11	184.78	185.28	208.10

Table 3.11: Estimation of the resources required to run quantum Prange against proposed parameter sets of Classic McEliece [17]. We define cost as $\log_2$ (# of physical qubits $\times$ # of seconds).

Parameters			Classical	Quantum	Security
n	k	t	Security	Security	Reduction (%)
3488	2720	64	195	119.85	38.54
4608	3360	96	240	143.11	40.37
6688	5024	128	320	184.78	42.26
6960	5413	119	321	185.28	39.48
8192	6528	128	358	208.10	41.87

Table 3.12: Comparison of cost estimates to perform Prange's algorithm using a classical computer, versus a quantum computer against proposed parameter sets of Classic McEliece [17]. Classic estimates obtained from [32].

3.4 Analysis

We have presented an estimate for performing Prange's algorithm fault-tolerantly on a surface code based quantum computer, using our construction we have derived estimates for attacks against the proposed parameter sets of Classic McEliece [17], one of the leading candidates of the NIST post-quantum cryptography standardization process [1]. In table 3.12, we compare the cost of our quantum attack with current estimates for classical variants of Prange's algorithm against the same Classic McEliece parameter sets [32]. Naïvely, we would expect for the cost of our quantum algorithm to cut our classical bit security in half as Grover's algorithm allows us to search a space of size 2^n in $O(2^{n/2})$ versus classical exhaustive search which takes $O(2^n)$. However, table 3.12 shows that on average the reduction is only around 40%. Note, that despite our construction not factoring in the row reduction process of Prange's algorithm (the process commonly used to benchmark Prange's algorithm [15]) we still find a 10% overhead due to excessive costs incurred by preparation of our Dicke state, if we were to assume that row reduction is more expensive than our state preparation, this overhead would only increase further.

Finally, we will analyse the post-quantum security of the proposed Classic McEliece parameters. NIST's call for proposals set out various levels of post-quantum security defined by the computational effort required for a quantum adversary to perform Grover's search against different symmetric key cryptosystems. The proposed parameters for Classic McEliece fall under levels 1, 3, and 5 in order for these parameter sets to meet the criteria of these security levels they must be as computationally hard to break as AES 128, 192, and 256 respectively [68]. In order to compare our costs with that of AES we have performed the same fault-tolerant resource analysis discussed in section 3.3 for the constructions of AES from [54], we summarize these results in table 3.13, and provide a comparison of these costs with those of our construction of quantum Prange in table 3.14. From these results we can see that not only do the parameter sets meet the required benchmark for their suggested security level, but

		AES 128	AES 192	AES 256
Grover	T-count	1.60×10^{24}	7.63×10^{33}	6.07×10^{43}
	T-depth	1.75×10^{21}	7.45×10^{30}	3.38×10^{40}
	Logical qubits	3329	3969	6913
	Code distance	31	41	51
	Physical qubits	1.00×10^{7}	2.09×10^{7}	5.62×10^{7}
Distilleries	Logical qubits per distillery	240	3600	3600
	No. of distilleries	913	205	599
	Code distances	$[13, 29]$	$[9, 15, 39]$	$[9, 19, 49]$
	Physical qubits	1.26×10^{8}	1.87×10^{8}	5.48×10^{8}
Totals	Physical qubits	1.36×10^{8}	2.08×10^{8}	6.04×10^{8}
	Seconds	7.33×10^{17}	4.70×10^{27}	2.60×10^{37}
	Cost	86.37	119.56	153.46

Table 3.13: Fault-tolerant resource analysis of the AES constructions from [54], we report cost as $\log_2$ (# of physical qubits $\times$ # of seconds).

Security Level	AES Cost	McEliece Parameters			Prange Cost	Security Margin
		n	k	t		
1 — AES 128	86.37	3488	2720	64	119.85	33.48
3 — AES 192	119.56	4608	3360	96	143.11	23.55
5 — AES 256	153.46	6688	5024	128	184.78	31.32
		6960	5413	119	185.28	31.82
		8192	6528	128	208.10	54.64

Table 3.14: Proposed Classical McEliece parameter sets [17], and their suggested security level, compared against the cost of breaking the respective AES schemes. We define cost as $\log_2$ (# of physical qubits $\times$ # of seconds).

in fact exceed it by a large margin. One of the most notable parameter sets is [3488, 2720, 64] which was suggested by the authors of Classic McEliece to achieve level 1 security (comparable to AES 128) but actually meets the requirements for level 3 security (comparable to AES 192). We observe that, with respect to our estimates of quantum Prange, these results are indicative of proposed parameter sets being rather conservative in their security assumptions, and as a result have space to be lowered while still ensuring the correct level of post-quantum security.

Chapter 4

Comparison with Recent Work

During the writing up of our research a new publication by Perriello, Barenghi, and Pelosi [71] was released, this research sought to answer the same question as our work, i.e., calculating overhead costs of implementing a quantum circuit for information set decoding. In this chapter we explore their work and compare their findings with our own.

4.1 Summary of Our Work

First let us quickly summarize our contributions presented in this work. We presented a quantum circuit implementing the majority of Quantum Prange (first shown in [15]), the method of state preparation using CNS is, to the best of our knowledge, the first attempt to use such a method in the preparation of Dicke States. We then continued by providing a resource estimate to perform QISD using our circuits fault-tolerantly, deriving estimates for the costs of attacking the five proposed parameter sets of Classic McEliece [17]. In order to gauge the security provided by these parameter sets, we performed the same fault-tolerant resource estimate for the construction of AES presented in [54]. This allowed us to come to the conclusion that the proposed parameter sets provide a significantly larger level of security than is necessary.

4.2 Summary of Results

We will now detail the work of [71]. In the paper the authors present a complete quantum circuit to perform Prange's algorithm, they use their construction in order to derive gate counts for attacking a variety of parameter sets for two proposed post-quantum code based cryptosystems; BIKE [5] and Classic McEliece [17]. The authors compare the gate counts of their construction with the asymptotic costs reported in [15]. They report a total gate cost smaller than the asymptotic cost by approximately a factor of 2^4. Furthermore, they translate their constructions into the Clifford+T gate model in order to provide their costs in terms of T-count,

T-depth, and number of logical qubits. Using these values they compare their costs versus the AES implementation of [93] in order to determine the security level of proposed code-based parameter choices for BIKE and Classic McEliece. Their results conclude that both investigated cryptosystems suggest parameter sizes that require considerably more effort to be broken than the symmetric ciphers they are compared against to determine their security level.

4.3 Comparison with our Work

To compare our work with that of [71], we will look at three facets; the approach, constructions, and conclusions of both bodies of work.

First let's discuss the approach taken across the two bodies of work. The authors of [71] only discuss the construction of Prange's variant of ISD, similar to our work, the parameters investigated in [71] look at a parameter set for each of the NIST post-quantum security levels for two different cryptosystems, whereas our analysis only features the five parameter sets of Classic McEliece. One of major differences between the two pieces of work, is that the authors of [71] only evaluate the cost of their circuit in terms of logical gate counts, and make no attempt to produce fault-tolerant estimates, in contrast our work analyses the overheads incurred by performing our quantum circuits fault-tolerantly, following the methodology of [4].

The second part of comparison is the constructions presented. The methods used for construction differ slightly, however overall report similar costs. One of the major differences in construction is that in our work we omit a construction of the row reduction function of Prange however, we show in our analysis that even without such a construction, the overheads incurred by state preparation lead to some meaningful analysis. In contrast, the authors of [71] provide details of such a construction. We provide on table4.1 a breakdown of the T-count for each process involved in quantum Prange for the Classic McEliece parameter set [3488, 2720, 64]. We note at this stage that there appears to be some issues with the reported T-count figures of [71], in the work they state that when synthesizing Y-rotation gates that $\epsilon = 10^{-15}$ is sufficient, however their constructions feature significantly more than $1/\epsilon$ Y-rotation gates. Furthermore, they state X gates require a T-count of 4, and Fredkin (referred to as CSWAP) gates require a T-depth of 4. We present in section 3.2.2 and section 3.2.4 respectively different figures for these values, however we should note that these don't wildly affect the overall resource estimates.

Finally, we note that the conclusions drawn by the authors of [71] align with the same conclusions drawn in section 3.4 of our work. Specifically, that the current proposed parameter suggestions for code based cryptosystems suggest parameter

Process	T-count	
	[71]	[this work]
Dicke state	1.16×10^{31}	1.40×10^{31}
Permute	2.15×10^{31}	1.31×10^{29}
Row reduction	1.76×10^{32}	—
Hamming weight	3.05×10^{26}	2.14×10^{26}
Diffusion	3.12×10^{26}	1.88×10^{26}
Total	2.09×10^{32}	1.41×10^{31}

Table 4.1: Comparison of T-count estimates from [71] and our work, for each process required to perform quantum Prange against proposed Classic McEliece parameters [3488, 2720, 64].

sets that exceed the security requirements of their respective post-quantum security levels, with regard to attacks utilizing quantum Prange.

Chapter 5

Conclusions and Future Direction

We have investigated the construction of a quantum information set decoding algorithm. Using our construction we were able to estimate the overhead incurred due to the expensive process of error-correction required to keep our algorithm fault-tolerant. We discussed the impact that this estimate has on current proposed parameters of the Classic McEliece cryptosystem, notably that the suggested parameter sizes were conservative in their security estimates and as such present opportunity to be decreased whilst still maintaining the same level of post-quantum security. Our results corroborate with those from the recent publication by Perriello, Barenghi, and Pelosi [71].

During our analysis a number of key areas of for further research can be identified. First is the investigation of other variants of information set decoding, in our work we only investigated Prange's algorithm, however, research has presented quantum versions of more advanced ISD variants [55, 56].

Another key piece of research is the lattice surgery techniques [36], which claims to reduce the overall physical qubit requirement by roughly a factor of 5, leading to reducing the number of bits of security by approximately 2 bits. While this number does not invalidate our results, it would be an interesting area of research to further investigate the impact of lattice surgery on large scale quantum computations.
Furthermore, our constructions are by no means final, or optimal. We specifically chose to follow a methodology that allows for our estimates to be updated such that any improvements to our circuits can be easily adopted. We note, that one potential avenue for improvement is the usage of optimization algorithms such as T-par [2].
Finally, as the area of research continues to expand there will likely be numerous new results that impact our estimates. Ultimately, this was our motivation behind our choice of methodology as it allows for easier adoption of new research. The most likely areas to change are improvements to quantum error-correcting codes and distillation that could boast significant reductions during the fault-tolerant layer. One particularly interesting area of research is the possibility of new attacks

impacting the very first layer of the stack, by being able to reformulate or find new algorithms that apply to code-based cryptography such as those found in [21].

Bibliography

[1] Gorjan Alagic et al. "Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process". In: *US Department of Commerce, NIST* (2020).

[2] Matthew Amy, Dmitri Maslov, and Michele Mosca. "Polynomial-Time T-Depth Optimization of Clifford+T Circuits Via Matroid Partitioning". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33.10 (2014), pp. 1476–1489. DOI: 10.1109/TCAD.2014.2341953.

[3] Matthew Amy et al. "A Meet-in-the-Middle Algorithm for Fast Synthesis of Depth-Optimal Quantum Circuits". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32.6 (2013), pp. 818–830.

[4] Matthew Amy et al. "Estimating the Cost of generic Quantum Pre-Image Attacks on SHA-2 and SHA-3". In: *International Conference on Selected Areas in Cryptography*. Springer. 2016, pp. 317–337.

[5] Nicolas Aragon et al. "BIKE: Bit Flipping Key Encapsulation". In: (2017).

[6] Srinivasan Arunachalam et al. "On the Robustness of Bucket Brigade Quantum RAM". In: *New Journal of Physics* 17.12 (2015), p. 123010.

[7] Marco Baldi et al. "A Finite Regime Analysis of Information Set Decoding Algorithms". In: *Algorithms* 12.10 (2019), p. 209.

[8] Adriano Barenco et al. "Elementary Gates for Quantum Computation". In: *Physical review A* 52.5 (1995), p. 3457.

[9] Andreas Bärtschi and Stephan Eidenbenz. "Deterministic Preparation of Dicke States". In: *International Symposium on Fundamentals of Computation Theory*. Springer. 2019, pp. 126–139.

[10] Anja Becker et al. "Decoding Random Binary Linear Codes in 2 n/20: How 1+ 1= 0 Improves Information Set Decoding". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2012, pp. 520–536.

[11] Paul Benioff. "The Computer as a Physical System: A Microscopic Quantum Mechanical Hamiltonian Model of Computers as Represented by Turing Machines". In: *Journal of Statistical Physics* 22.5 (1980), pp. 563–591.

[12] Charles H Bennett and Gilles Brassard. "Quantum Cryptography: Public key distribution and Coin Tossing". In: *International Conference on Computers, Systems & Signal Processing, Bangalore, India* (1984), pp. 175–179.

[13] Charles H Bennett et al. "Teleporting an Unknown Quantum State via Dual Classical and Einstein-Podolsky-Rosen Channels". In: *Physical Review Letters* 70.13 (1993), p. 1895.

[14] Elwyn Berlekamp, Robert McEliece, and Henk Van Tilborg. "On the Inherent Intractability of Certain Coding Problems (corresp.)" In: *IEEE Transactions on Information Theory* 24.3 (1978), pp. 384–386.

[15] Daniel J Bernstein. "Grover vs. McEliece". In: *International Workshop on Post-Quantum Cryptography*. Springer. 2010, pp. 73–80.

[16] Daniel J Bernstein, Tanja Lange, and Christiane Peters. "Attacking and Defending the McEliece Cryptosystem". In: *International Workshop on Post-Quantum Cryptography*. Springer. 2008, pp. 31–46.

[17] Daniel J Bernstein et al. "Classic McEliece: Conservative Code-Based Cryptography". In: *NIST Submissions* (2017).

[18] Gilles Brassard and Peter Hoyer. "An Exact Quantum Polynomial-Time Algorithm for Simon's Problem". In: *Proceedings of the Fifth Israeli Symposium on Theory of Computing and Systems*. IEEE. 1997, pp. 12–23.

[19] Gilles Brassard, Peter Høyer, and Alain Tapp. "Quantum Counting". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 1998, pp. 820–831.

[20] Sergey Bravyi and Alexei Kitaev. "Universal Quantum Computation with Ideal Clifford Gates and Noisy Ancillas". In: *Physical Review A* 71.2 (2005), p. 022316.

[21] Yu-Ao Chen and Xiao-Shan Gao. "Quantum Algorithm for Boolean Equation Solving and Quantum Algebraic Attack on Cryptosystems". In: *Journal of Systems Science and Complexity* (2021), pp. 1–40.

[22] Lily Chen et al. *Report on Post-Quantum Cryptography*. Vol. 12. US Department of Commerce, National Institute of Standards and Technology, 2016.

[23] Steven A Cuccaro et al. "A New Quantum Ripple-Carry Addition Circuit". In: *arXiv preprint quant-ph/0410184* (2004).

[24] David Deutsch and Richard Jozsa. "Rapid Solution of Problems by Quantum Computation". In: *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 439.1907 (1992), pp. 553–558.

[25] Olivia Di Matteo, Vlad Gheorghiu, and Michele Mosca. "Fault-Tolerant Resource Estimation of Quantum Random-Access Memories". In: *IEEE Transactions on Quantum Engineering* 1 (2020), pp. 1–13.

[26] Robert H Dicke. “Coherence in Spontaneous Radiation Processes”. In: *Physical review* 93.1 (1954), p. 99.

[27] Thomas G Draper. “Addition on a Quantum Computer”. In: *arXiv preprint quant-ph/0008033* (2000).

[28] Thomas G Draper et al. “A Logarithmic-Depth Quantum Carry-Lookahead Adder”. In: *arXiv preprint quant-ph/0406142* (2004).

[29] Ilya Dumer. “On Minimum Distance Decoding of Linear Codes”. In: *Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory*. 1991, pp. 50–52.

[30] Taher ElGamal. “A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms”. In: *IEEE Transactions on Information Theory* 31.4 (1985), pp. 469–472.

[31] Daniela Engelbert, Raphael Overbeck, and Arthur Schmidt. “A Summary of McEli-ece Type Cryptosystems and their Security”. In: *Journal of Mathematical Cryptology* 1.2 (2007), pp. 151–199.

[32] Andre Esser and Emanuele Bellini. *Syndrome Decoding Estimator*. Cryptology ePrint Archive, Report 2021/1243. `https://ia.cr/2021/1243`. 2021.

[33] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. “A Quantum Approximate Optimization Algorithm”. In: *arXiv preprint arXiv:1411.4028* (2014).

[34] RP Feynman. “Simulating Physics with Computers”. In: *International Journal of Theoretical Physics* 21.6 (1982).

[35] Matthieu Finiasz and Nicolas Sendrier. “Security Bounds for the Design of Code-Based Cryptosystems”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2009, pp. 88–105.

[36] Austin G Fowler and Craig Gidney. “Low Overhead Quantum Computation using Lattice Surgery”. In: *arXiv preprint arXiv:1808.06709* (2018).

[37] Austin G Fowler, Ashley M Stephens, and Peter Groszkowski. “High-Threshold Universal Quantum Computation on the Surface Code”. In: *Physical Review A* 80.5 (2009), p. 052312.

[38] Austin G Fowler et al. “Surface codes: Towards Practical Large-Scale Quantum Computation”. In: *Physical Review A* 86.3 (2012), p. 032324.

[39] Austin G Fowler et al. “Surface codes: Towards Practical Large-Scale Quantum Computation”. In: *Physical Review A* 86.3 (2012), p. 032324.

[40] GeeksforGeeks. *Space and Time Efficient Binomial Coefficient*. 2012. URL: `https://www.geeksforgeeks.org/space-and-time-efficient-binomial-coefficient/` (visited on 12/09/2019).

[41] V Gheorghiu and M Mosca. “Quantum Cryptanalysis of Symmetric, Public-Key and Hash-Based Cryptographic Schemes”. In: *arXiv preprint arXiv:1902.02332* (2019).

[42] Vlad Gheorghiu and Michele Mosca. “A Resource Estimation Framework For Quantum Attacks Against Cryptographic Functions: Recent Developments”. In: (2021).

[43] Craig Gidney. “Halving the Cost of Quantum Addition”. In: *Quantum* 2 (2018), p. 74.

[44] Craig Gidney and Martin Ekerå. “How to Factor 2048-Bit RSA Integers in 8 Hours Using 20 Million Noisy Qubits”. In: *arXiv preprint arXiv:1905.09749* (2019).

[45] Daniel Gottesman. *Stabilizer Codes and Quantum Error Correction*. 1997. eprint: `arXiv:quant-ph/9705052`.

[46] Daniel Gottesman. “The Heisenberg Representation of Quantum Computers”. In: (1998). eprint: `arXiv:quant-ph/9807006`.

[47] Markus Grassl et al. “Applying Grover’s Algorithm to AES: Quantum Resource Estimates”. In: *Post-Quantum Cryptography*. Springer. 2016, pp. 29–43.

[48] Lov K Grover. “A Fast Quantum Mechanical Algorithm for Database Search”. In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. 1996, pp. 212–219.

[49] Stuart Hadfield et al. “From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz”. In: *Algorithms* 12.2 (2019), p. 34.

[50] Thomas Häner and Mathias Soeken. “Lowering the T-depth of Quantum Circuits by Reducing the Multiplicative Depth of Logic Networks”. In: *arXiv preprint arXiv:2006.03845* (2020).

[51] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. “Quantum Algorithm for Linear Systems of Equations”. In: *Physical review letters* 103.15 (2009), p. 150502.

[52] Raymond Hill. *A First Course in Coding Theory*. Oxford University Press, 1986.

[53] David Hobach. “Practical Analysis of Information Set Decoding Algorithms”. Ruhr-University Bochum, 2012. URL: `hackingthe.net/downloads/isd.pdf`.

[54] Samuel Jaques et al. *Implementing Grover oracles for quantum key search on AES and LowMC*. Cryptology ePrint Archive, Report 2019/1146. `https://ia.cr/2019/1146`. 2019.

[55] Ghazal Kachigar and Jean-Pierre Tillich. “Quantum Information Set Decoding Algorithms”. In: *International Workshop on Post-Quantum Cryptography*. Springer. 2017, pp. 69–89.

[56] Elena Kirshanova. “Improved Quantum Information Set Decoding”. In: *International Conference on Post-Quantum Cryptography*. Springer. 2018, pp. 507–527.

[57] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. “Practical Approximation of Single-qubit Unitaries by Single-qubit Quantum Clifford and T Circuits”. In: *IEEE Transactions on Computers* 65.1 (2015), pp. 161–172.

[58] Donald Ervin Knuth. *The Art of Computer Programming: Generating All Combinations and Partitions*. 2005.

[59] Brandon Langenberg, Hai Pham, and Rainer Steinwandt. “Reducing the Cost of Implementing the Advanced Encryption Standard as a Quantum Circuit”. In: *IEEE Transactions on Quantum Engineering* 1 (2020), pp. 1–12.

[60] Pil Joong Lee and Ernest F Brickell. “An Observation on the Security of McEliece’s Public-Key Cryptosystem”. In: *Workshop on the Theory and Application of Cryptographic Techniques*. Springer. 1988, pp. 275–280.

[61] DH Lehmer. *The Machine Tools of Combinatorics, Applied Combinatorial Mathematics (E. F. Beckenbach, ed.)* 1964.

[62] Ying Li. “A Magic State’s Fidelity can be Superior to the Operations that Created it”. In: *New Journal of Physics* 17.2 (2015), p. 023037.

[63] Daniel Litinski. “A Game of Surface Codes: Large-scale Quantum Computing with Lattice Surgery”. In: *Quantum* 3 (2019), p. 128.

[64] Alexander May, Alexander Meurer, and Enrico Thomae. “Decoding Random Linear Codes in $\tilde{\mathcal{O}}(2^{0.054n})$”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2011, pp. 107–124.

[65] Robert J McEliece. “A Public-Key Cryptosystem Based on Algebraic Coding Theory”. In: *Coding Thv* 4244 (1978), pp. 114–116.

[66] Alan Mishchenko and Marek Perkowski. “Fast Heuristic Minimization of Exclusive-Sums-of-Products”. In: (2001).

[67] Michael A. Nielsen and Isaac L. Chuang. “The Solovay-Kitaev Theorem”. In: *Quantum Computation and Quantum Information*. Cambridge University Press, 2010, pp. 617–624.

[68] NIST. *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*. 2016. URL: `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/evaluation-criteria/security-(evaluation-criteria)`.

[69] Joe O'Gorman and Earl T Campbell. "Quantum Computation with Realistic Magic-State Factories". In: *Physical Review A* 95.3 (2017), p. 032338.

[70] Sahin K Özdemir, Junichi Shimamura, and Nobuyuki Imoto. "A Necessary and Sufficient Condition to Play Games in Quantum Mechanical Settings". In: *New Journal of Physics* 9.2 (2007), p. 43.

[71] Simone Perriello, Alessandro Barenghi, and Gerardo Pelosi. "A Complete Quantum Circuit to Solve the Information Set Decoding Problem". In: *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE. 2021, pp. 366–377.

[72] Christiane Peters, Tanja Lange, and Daniel J Bernstein. "Curves, Codes, and Cryptography". In: (2011).

[73] Eugene Prange. "The Use of Information Sets in Decoding Cyclic Codes". In: *IRE Transactions on Information Theory* 8.5 (1962), pp. 5–9.

[74] Robert Prevedel et al. "Experimental Realization of Dicke States of up to Six Qubits for Multiparty Quantum Networking". In: *Physical Review Letters* 103.2 (2009), p. 020503.

[75] Ronald L Rivest, Adi Shamir, and Leonard Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[76] 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.

[77] Neil J Ross and Peter Selinger. "Optimal Ancilla-free Clifford+T Approximation of Z-rotations". In: *arXiv preprint arXiv:1403.2975* (2014).

[78] Peter Selinger. "Efficient Clifford+T Approximation of Single-Qubit operators". In: *arXiv preprint arXiv:1212.6253* (2012).

[79] Peter Selinger. "Quantum Circuits of T-Depth One". In: *Physical Review A* 87.4 (2013), p. 042302.

[80] Nicolas Sendrier. "Code-Based Cryptography: State of the Art and Perspectives". In: *IEEE Security & Privacy* 15.4 (2017), pp. 44–50.

[81] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. "Reversible Logic Synthesis of k-input, m-output Lookup Tables". In: *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2013, pp. 1235–1240.

[82] Claude Elwood Shannon. "A Mathematical Theory of Communication". In: *Bell system technical journal* 27.3 (1948), pp. 379–423.

[83] Peter W Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE. 1994, pp. 124–134.

[84] Peter W Shor. "Scheme for Reducing Decoherence in Quantum Computer Memory". In: *Physical Review A* 52.4 (1995), R2493.

[85] Daniel R Simon. "On the Power of Quantum Computation". In: *SIAM Journal on Computing* 26.5 (1997), pp. 1474–1483.

[86] John A. Smolin and David P. DiVincenzo. "Five Two-bit Quantum Gates are Sufficient to Implement the Quantum Fredkin Gate". In: *Phys. Rev. A* 53 (4 Apr. 1996), pp. 2855–2856. DOI: 10.1103/PhysRevA.53.2855.

[87] Andrew Steane. "Multiple-particle Interference and Quantum Error Correction". In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 452.1954 (1996), pp. 2551–2577.

[88] Jacques Stern. "A Method for Finding Codewords of Small Weight". In: *International Colloquium on Coding Theory and Applications*. Springer. 1988, pp. 106–113.

[89] Géza Tóth. "Multipartite Entanglement and High-Precision Metrology". In: *Physical Review A* 85.2 (2012), p. 022322.

[90] Vlatko Vedral, Adriano Barenco, and Artur Ekert. "Quantum Networks for Elementary Arithmetic Operations". In: *Physical Review A* 54.1 (1996), p. 147.

[91] Dominic Welsh. *Codes and Cryptography*. Oxford University Press, 1988.

[92] Paolo Zialcita. *Google Claims To Achieve Quantum Supremacy — IBM Pushes Back*. 2019. URL: https://www.npr.org/2019/10/23/772710977/google-claims-to-achieve-quantum-supremacy-ibm-pushes-back (visited on 01/09/2022).

[93] Jian Zou et al. "Quantum Circuit Implementations of AES with Fewer Qubits". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2020, pp. 697–726.