An Efficient Quantum Message Authentication Scheme

by

Nicolas Perez

A thesis proposal submitted to the thesis proposal committee in partial fulfillment of the requirements for the degree of

Master

in

Computer Science

Carleton University

Ottawa, Ontario

Abstract

Quantum networking is the study of transmitting data that uses quantum mechanics principles (quantum messages). Quantum networking aims to connect two or more quantum computers over a network to cooperatively increase their capabilities, much like how the internet connects computers today. This thesis describes a new quantum message authentication (QMA) scheme, allowing to detect when an adversary maliciously modifies a quantum message (an integrity attack). The new scheme is analyzed in terms of its ability to detect an integrity attack and its efficiency in terms of classical computational complexity and the number of quantum gates used. It is rigorously compared to the Clifford code QMA scheme, as it is the most similar. It offers a trade-off compared to the Clifford code QMA scheme in terms of integrity detection versus efficiency: reduced detection rates and improved efficiency. The reduced detection rates may make the new scheme more appealing in communication applications with a small amount of passive noise, as the detection rate is positively correlated with the number of noisy qubits in the packet. The increased efficiency may benefit constrained devices or communications with very large data packets or data throughput. As a consequence of developing this new QMA scheme, a novel method for unranking permutations of multisets is also presented.

Acknowledgements

I am grateful for the support from my supervisors Evangelos Kranakis and Michel Barbeau, and the additional support from Joaquin Garcia-Alfaro. Their inquiry and feedback on my research throughout my degree have been very helpful. My supervisors were also eager to teach me quantum computing at the start of my degree, and ensured that I had a good foundation of knowledge for starting my research.

My girlfriend provided me with an immense amount of support in more ways than I can say. She, along with my cats Steve and Bucky, were an appreciated source of comfort and humour and kept me good company during the isolating times of the COVID pandemic.

I used to ask my dad repeatedly while I was a small child: ”What doing dad, what doing?”, whilst he was tinkering around. He would let me watch and explain to me what he was doing. I used to ask my mom repeatedly: ”Why? Why? Why?”, when asking her about how the world worked. She would try her best to answer my questions! Both of them helped me become curious and want to know more about the world.

I am thankful for the support of all my family and friends.

Contents

List of Tables

List of Figures

Chapter 1

Overview

The new quantum message authentication scheme presented in this thesis will be referred to herein as the 'block Clifford code'. This chapter will outline what is presented in all the other chapters of this thesis.

Chapter 2 explains the motivation behind, and practical relevance of, the contents of this thesis. Chapter 3 provides a table listing relevant symbols and their corresponding definitions. Chapter 4 summarizes the results of this thesis, by comparing the Clifford code and block Clifford code as well as briefly describing the efficiency of the novel multiset permutation unranking method. Chapter 5 goes over the necessary prerequisite knowledge for understanding the contents of this thesis. Chapter 6 reviews related work, with a focus on the Clifford code. Chapter 7 describes the block Clifford code. The novel multiset permutation unranking method is described in Subsection 7.2.4, and is used in the key mapping process of the block Clifford code. The last chapter, Chapter 8, discusses the results of this thesis and potential future work.

Chapter 2

Motivation

Quantum networking is like quantum computing's 'little brother', still very much new to the world and in its 'infancy'. The article, *The Internet Goes Quantum* [1], explains the motivations behind the emerging study of quantum networking and gives an interesting overview of current progress and main topics. In short, quantum computers have certain advantages over classical computers and enabling an internet for communicating quantum data will yield even more advantages. When mentioning the word 'classical' in this thesis, I am either using it to describe computation, a computer, or data concerning bits (what your typical modern computer uses). Authentication methods used on classical data must be fundamentally different than the ones applied to quantum data, due to the unique properties of quantum data [2]. The emerging quantum internet thus puts pressure on the further development of QMA schemes.

Quantum data inherently has security properties, due to the 'no cloning theorem' stating that quantum data can not be copied. Additionally, quantum data can only be read once by a computer, before its state is disturbed and irreversibly lost, and

typically the information gained from reading the quantum data is only a small portion of its entirety [3]. As a consequence, stolen quantum data has no perceivable use for the one who stole it, unless the characteristics of the data are known ahead of time. One could then assume that the most significant security threat to quantum data is modifying its contents. The method developed in this thesis is designed to detect an adversary modifying the contents of a quantum message.

The main goal of this thesis is to further advance the security capabilities of the quantum networks of the future. Some of the ideas on detecting integrity attacks may also one day apply to securing information in quantum computers, not just in quantum communications. This thesis describes a new solution for securing quantum data that shows improved efficiency, in terms of gates used and time complexity, compared to a similar QMA method. Efficiency may be paramount in applications with constrained computational resources, low latency requirements, large data packets or data throughput.

Chapter 3

Symbols

Table 3.1: A list of symbols and their definitions.

Symbol	**Definition**
C	An arbitrary Clifford matrix.
$C(n)$	The Clifford group acting on n qubits.
d	The number of signature qubits within a quantum message.
$divmod(a, b)$	A pseudocode method for Euclidean division, [4], where a is divided by b. The method returns a tuple, q, r, where q is the quotient and r is the remainder.
i	Usually used to denote the qubit position of a data qubit within a quantum message. Otherwise, it typically denotes a position of an arbitrary qubit.

Symbol	Definition
i'	The qubit position of a data qubit within a quantum message after it has been moved by a $CIRC_{i,i'}$ gate.
j	Usually used to denote the qubit position of a signature qubit within a quantum message. Otherwise, it typically denotes a position of an arbitrary qubit.
j'	The qubit position of a signature qubit within a quantum message after it has been moved by a $CIRC_{j,j'}$ gate.
k	Denotes either the security key k or the qubit position of an arbitrary qubit within a quantum message.
k_a	The subkey k_a such that $k = k_a, k_b$
k_b	The subkey k_b such that $k = k_a, k_b$
m	The number of data qubits within a quantum message.
n	The total number of qubits a quantum message is composed of. The message is often referred to as a 'packet'. $$n = m + d$$
$O\left(f(n_1, n_2, \cdots, n_k)\right)$	Reads as 'big O of $f(n_1, n_2, \cdots, n_k)$' or 'order $f(n_1, n_2, \cdots, n_k)$'. The big O notation indicates an asymptotic limit of a given function $f(n_1, n_2, \cdots, n_k)$. See Subsection 1.3.3 of (the open access) reference [5] for more info.

Symbol	Definition
$P(n)$	The set of matrices given by all possible n-qubit tensor products of Pauli matrices (including the identity matrix).
w	The number of non-identity single-qubit Pauli unitaries an adversarial attack is composed of.
$\left\{ {w \atop x} \right\}$	The Stirling number of the second kind. It counts the number of ways to partition a set of w distinct elements into x non-empty sets. $$\left\{ {w \atop x} \right\} = \frac{1}{x!} \sum_{i=0}^{x} (-1)^i \binom{x}{i} (x-i)^w \tag{3.1}$$
†	Indicates the conjugate transpose of a matrix. Also known as the Hermitian transpose.
$\|\psi\rangle$	An arbitrary quantum state.
ρ	A quantum state in density matrix form. Usually used to denote a quantum data packet. May also denote an arbitrary quantum state.
σ	A quantum state in density matrix form. Usually used to denote the message portion (the data qubits) of a quantum data packet. Also referred to as the 'quantum message'.

Symbol	Definition
()	Parentheses may be used to denote lists, whilst making use of set-builder notation [6]. The parentheses are used to indicate an order of elements. This is particularly useful for expressing for-loops in pseudocode. For example: $$x \in (0, 1, 2, 3...n-1)$$ denotes x taking on the values of 0 through to $n-1$ in order. Parantheses may also be used to denote accessing elements in a particular list. For example, denote the list B such that $B = (3, 8, 10)$. It then follows that $B(0) = 3, B(1) = 8$ and $B(2) = 10$.
$\Longleftarrow$	Assigns the value right of the arrow to the value left of the arrow. For example: $$x \Longleftarrow x + 1$$ denotes increasing the value of x by one.
$\Rightarrow$	Indicates an equation is being modified. See Section 5.3 for more details.

Symbol	Definition
$\cdot$	May be used to indicate multiplication instead of relying on brackets. For example: $$3 \cdot 5 = (3)(5) = 15$$

Chapter 4

Summary of Results

Table 4.1: A table comparing the Clifford code to the newly proposed block Clifford code.

Code	Number of gates	Time complexity	Integrity detection probability
Clifford code	$O\left(\frac{n^2}{\log_2(n)}\right)$	$O\left(n^3\right)$	$1-\frac{2^{2m+d}-1}{2^{2n}-1} \approx 1-\left(\frac{1}{2}\right)^d$
Block Clifford code	$\min\left(O\left(nd\right), O\left(\frac{n^2}{\log_2(n)}\right)\right)$	$O\left(n^2\log_2^2(n)\right)$	$\approx \sum_{x=1}^{\min(d,w)}\left(1-\left(\frac{1}{2}\right)^x\right)\binom{d}{x}\frac{\left\{ {w \atop x} \right\} x!}{d^w}$

See Table 4.1 for a comparison between the block Clifford code and the Clifford code. Note that the 'Integrity Detection Probability' column is only concerned with Pauli attacks (see the attack model described in Section 5.12). The probability of detection for the block Clifford code approaches the probability of detection for the Clifford code as w increases. Remark Table 3.1 for a description of symbols.

The novel multiset permutation unranking method is described in Algorithm 8 in Subsection 7.2.4. It runs in $O\left(n^2\log_2^2(n)\right)$ time.

Chapter 5

Required Background Knowledge

5.1 Introduction and External Resources

Quantum information science relies heavily on linear algebra and complex numbers. This thesis will not explain the basics of those topics. The required linear algebra and complex number concepts for understanding this thesis can be found on Khan Academy, YouTube, in textbooks on the subject(s), etc. The textbook *Quantum Computing for Computer Scientists* also provides a great explanation of complex numbers and many of the linear algebra concepts required for understanding quantum information science [7]. This thesis also makes use of concepts from probability, combinatorics and set theory as well as various proof techniques which are all explained in one of Michiel Smid's free textbooks [8]. Understating time complexity analysis and basic data structures will also be required, and are outlined in Pat Morin's free *Open Data Structures* textbook [5]. Finally, a basic understanding of pseudocode is also required.

There are many excellent resources on quantum information science. *Learn*

Quantum Computation Using Qiskit is an excellent open source free textbook managed by IBM, which provides coding examples in its quantum computing language 'Qiskit' [9]. *Quantiki* is also another free online source of quantum information science, which explains information in an open source manner similar to IBM's textbook, but without the coding examples [10]. The *Quantum Computing Stack Exchange* forum has many specific answered questions one may have about quantum computing and quantum networking [11]. *Wikipedia* also has some articles on quantum computing, but many of them are densely written and do not explain concepts from first principles. John Watrous also has a free textbook, *The Theory of Quantum Information* which is a good reference for the more advanced reader, although the notation used in that textbook is not always consistent with the notation used in this thesis [12]. *Quantum Computation and Quantum Information Theory* is generally regarded as a standard textbook on quantum computing [3]. *Quantum Computing for Computer Scientists* explains concepts in a similar way to *Quantum Computation and Quantum Information Theory*, but slightly more towards the language computer scientists are familiar with [7]. Finally, *Quantum Networking* is a helpful textbook specifically on quantum communications [13].

Most of the background knowledge described in this thesis is foundational knowledge used extensively in the field of quantum computing. Some more focused subtopics in quantum computing are elaborated on as well. The following sections in this chapter will describe the notation and foundational concepts that are required to understand this thesis.

5.2 A Comment on Conventions and Notation Used in this Thesis

Quantum computing and communications are fields welcome to physicists, mathematicians, network scientists, and computer scientists. These different branches of science all prefer a slightly different language (set of notations, and common conventions) to describe the ideas in quantum information science. This thesis, for instance, enumerates qubits from right to left, starting at 0, as seen in *Learn Quantum Computation Using Qiskit* [9], which may be unfamiliar to some readers. The notation and conventions in this textbook almost entirely follow the ones used in the standard textbook, *Quantum Computation and Quantum Information Theory* [3]. Any deviation from these conventions will be stated explicitly either in this chapter's various sections, or when needed.

5.3 What Does the Arrow Instead of the Equals Sign Mean?

In some equations in this thesis, the right arrow, $\Rightarrow$, is used in place of an equals sign, $=$, to emphasize when an equation is being modified.

Example 5.3.1. *Here is an example further clarifying. See the equations:*

$$10 + x + 3x + 4 = 4x + 14 \tag{5.1}$$
$$= 2(2x + 7) \tag{5.2}$$
$$\Rightarrow 10(2(2x + 7)) \tag{5.3}$$
$$= 20(2x + 7) \tag{5.4}$$
$$= 40x + 140 \tag{5.5}$$

In Equation (5.3) the right arrow is used, instead of the equals sign because the equation is being modified (in this case, being multiplied by 10). It may be any other sort of modification, like addition, adding an exponent, etc. Using an equals sign would have been incorrect, as $10(2(2x + 7)) \neq 2(2x + 7)$.

5.4 Quantum States

5.4.1 Definition

A quantum state is said to be in a 'superposition' of different states, meaning it has probabilities corresponding to each of the different states it can 'collapse' to. Once observed, the superposition collapses to a particular state with its associated probability. If it is not observed, it behaves as if it is in all of those states *simultaneously*. Quantum states can be described using state vectors or bra-ket notation.

Denote an arbitrary quantum state $|\psi\rangle$. Denote n as the number of qubits (also known as quantum bits) in the quantum state. A qubit is the quantum version of a

bit (also known as a classical bit). A state vector of n qubits is represented as:

$$|\psi\rangle = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{2^n-1} \end{bmatrix} \tag{5.6}$$

where each c_i corresponds to the complex amplitude for the state $|x_i\rangle$. The complex amplitudes are complex numbers whose squared magnitudes represent the probability the quantum state will collapse to the state $|x_i\rangle$ when measured. The complex amplitudes thus must be defined such that:

$$1 = \sum_{i=0}^{2^n-1} |c_i|^2 \tag{5.7}$$

In other words, the state vector must be a unit vector (which is also a unitary matrix with only one column).

The material in this thesis is concerned only with the quantum states of qubits (and not the more generalizable 'qudits', see Section 6.2). In the standard computational basis (see Subsection 5.4.3), each state $|x_i\rangle$ corresponds to the binary string whose decimal representation is equal to i. The notation for representing a state in bra-ket notation is:

$$|\psi\rangle = c_0 |x_0\rangle + c_1 |x_1\rangle + \cdots + c_{2^{n-1}} |x_{2^{n-1}}\rangle \tag{5.8}$$

Example 5.4.1. *If $n = 3$, then $|x_2\rangle = |010\rangle$.*

Example 5.4.2. *If $n = 4$, then $|x_5\rangle = |0101\rangle$.*

The number of qubits n is equal to the number of bits in each state $|x_i\rangle$. Each bit in $|x_i\rangle$ thus indicates a qubit in the state 1 or 0. The relative phases of each c_i represent the 'phase' of qubits, an additional property that quantum computers make use of.

5.4.2 Conventions

Some of the conventions used in this thesis when denoting quantum states are not widely used and may be unfamiliar to the reader. The conventions were adopted to make certain analyses more concise. The conventions are the following: This thesis

1. indexes qubits from right to left starting at 0 when representing states in bra-ket notation,

2. may sometimes omit normalizing factors from quantum states for convenience (i.e. Equation (5.7) will not always be satisfied),

3. refers to the 'terms' of a quantum state as the individual terms of the state in bra-ket notation. More formally written, given some arbitrary quantum state $|\psi\rangle$ expressed as $c_0|x_0\rangle + c_1|x_1\rangle + \cdots + c_{2^n-1}|x_{2^n-1}\rangle$ in bra-ket notation, one of the $c_i|x_i\rangle$ terms is simply referred to as a 'term' of the quantum state $|\psi\rangle$.

5.4.3 Basis States

The most common basis to represent quantum states is via the standard computational basis. For a single-qubit state, it is:

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

which may also be referred to as the 'z-basis'.

The x-basis is:

$$|+\rangle = \frac{|0\rangle + 1\rangle}{\sqrt{2}}, |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

and the y-basis is:

$$|i\rangle = \frac{|0\rangle + i\,|1\rangle}{\sqrt{2}}, |-i\rangle = \frac{|0\rangle - i\,|1\rangle}{\sqrt{2}}$$

Some sources denote the y-basis differently, but I prefer the one listed above [14].

Section 5.5 explains tensor products and how they are used to represent multi-qubit states. To generate the basis states for an n-qubit system of a given basis: Take the two corresponding single-qubit basis states and generate all possible 2^n number of n-tensor products with n of the single-qubit basis states. For the standard computational basis, this gives each of the $|x_i\rangle$ listed in Equation (5.8).

5.5 Tensor Products

The tensor product sign is denoted using the symbol: $\otimes$. It is also sometimes referred to as the Kronecker product. Oddly enough, despite tensor products being used extensively in quantum information science, a rigorous definition of tensor products of matrices is omitted from some textbooks on the subject. A tensor product of two matrices, denoted $A \in \mathbb{C}^{m \times n}$ and $B \in \mathbb{C}^{m' \times n'}$, is precisely defined as:

$$(A \otimes B)[i, j] = A[\lfloor i/m' \rfloor, \lfloor j/n' \rfloor] \cdot B[i \bmod m', j \bmod n']$$

The indices i and j indicate the value of the element at row i and column j in the corresponding matrices. Tensor products are also associative.

An example of a tensor product will help further explain how they are computed.

Essentially, a tensor product $A \otimes B$ creates a new matrix where each element of A is multiplied by the entirety of B.

Example 5.5.1.

$$A \otimes B = \begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \otimes \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} \tag{5.9}$$

$$= \begin{bmatrix} a_{0,0}B & a_{0,1}B \\ a_{1,0}B & a_{1,1}B \end{bmatrix} \tag{5.10}$$

$$= \begin{bmatrix} a_{0,0} \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} & a_{0,1} \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} \\ a_{1,0} \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} & a_{1,1} \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} \end{bmatrix} \tag{5.11}$$

$$= \begin{bmatrix} a_{0,0}b_{0,0} & a_{0,0}b_{0,1} & a_{0,1}b_{0,0} & a_{0,1}b_{0,1} \\ a_{0,0}b_{1,0} & a_{0,0}b_{1,1} & a_{0,1}b_{1,0} & a_{0,1}b_{1,1} \\ a_{1,0}b_{0,0} & a_{1,0}b_{0,1} & a_{1,1}b_{0,0} & a_{1,1}b_{0,1} \\ a_{1,0}b_{1,0} & a_{1,0}b_{1,1} & a_{1,1}b_{1,0} & a_{1,1}b_{1,1} \end{bmatrix} \tag{5.12}$$

To take n number of tensor products of the same matrix, call it A, the following notation may be used:

$$A^{\otimes n} = \underbrace{A \otimes A \otimes \cdots \otimes A}_{n \text{ times}}$$

Most textbooks go about simply stating the properties of tensor products of matrices, keeping the concept of tensor products very abstract. I derived a proof for a property of tensor products used throughout quantum information science and

provide it here, to provide further clarification.

Theorem 5.5.1. $(A \otimes B) \cdot (A' \otimes B') = (A \cdot A') \otimes (B \cdot B')$ *where* $A \in \mathbb{C}^{m \times n}$, $A' \in \mathbb{C}^{n \times p}$, $B \in \mathbb{C}^{m' \times n'}$ *and* $B' \in \mathbb{C}^{n' \times p'}$

Proof. To prove this claim, look at an arbitrary pair of elements indexed at the same position of the resulting matrices on the left and right side of the equality, indexed at row j, column k. If the elements are equal for all values of j and k, then the matrices are also equal. First, simplify the left side.

$$((A \otimes B) \cdot (A' \otimes B'))[j,k] \tag{5.13}$$

$$= \sum_{i=0}^{nn'-1} (A \otimes B)[j,i] \cdot (A' \otimes B')[i,k] \tag{5.14}$$

$$\begin{aligned} &= \sum_{i=0}^{nn'-1} A[\lfloor j/m' \rfloor, \lfloor i/n' \rfloor] \cdot B[j \bmod m', i \bmod n'] \\ &\cdot A'[\lfloor i/n' \rfloor, \lfloor k/p' \rfloor] \cdot B'[i \bmod n', k \bmod p'] \end{aligned} \tag{5.15}$$

$$\begin{aligned} &= \sum_{i=0}^{nn'-1} A[\lfloor j/m' \rfloor, \lfloor i/n' \rfloor] \cdot A'[\lfloor i/n' \rfloor, \lfloor k/p' \rfloor] \\ &\cdot B[j \bmod m', i \bmod n'] \cdot B'[i \bmod n', k \bmod p'] \end{aligned} \tag{5.16}$$

Then, simplify the right side.

$$((A \cdot A') \otimes (B \cdot B'))[j,k] \tag{5.17}$$

$$= (A \cdot A')[\lfloor j/m' \rfloor, \lfloor k/p' \rfloor] \cdot (B \cdot B')[j \bmod m', k \bmod p'] \tag{5.18}$$

$$= \sum_{w=0}^{n-1} A[\lfloor j/m' \rfloor, \lfloor w \rfloor] \cdot A'[\lfloor w \rfloor, \lfloor k/p' \rfloor]$$
$$\cdot \sum_{u=0}^{n'-1} B[j \bmod m', u] \cdot B'[u, k \bmod p'] \tag{5.19}$$

Equations (5.16) and (5.19) are equal for any $i = wn' + u$, and enumerating all possible values of w and u in Equation (5.19) gives the corresponding equivalence for each of the possible values of i for Equation (5.16). Thus, both summations are equal, proving the initial claim. □

Tensor products are primarily used in quantum information science for representing multi-qubit states and multi-qubit operations. The tensor product is used to combine qubits to form multi-qubit states. When using bra-ket notation, the tensor product symbol is usually implied and omitted to make quantum states easier to read and write.

Example 5.5.2. *For instance:*

$$|01\rangle = |0\rangle \otimes |1\rangle$$

Example 5.5.3. *The tensor product symbol may also be omitted like this:*

$$|0\rangle\,|1\rangle = |0\rangle \otimes |1\rangle$$

The tensor product is also used to combine qubit operations. This allows one to take the tensor product of single-qubit operations to construct multi-qubit operations. A multi-qubit operation is an operation that acts on more than one qubit at the same time.

Example 5.5.4. *The tensor product of qubit operations $A \in \mathbb{C}^{2\times 2}$ and $B \in \mathbb{C}^{2\times 2}$ gives the multi-qubit operation:*

$$A \otimes B$$

which can act on an arbitrary two-qubit state, such as $|01\rangle$, like so:

$$(A \otimes B)\,|01\rangle = A\,|0\rangle \otimes B\,|1\rangle$$

Another useful property is the distributive nature of the trace operation with respect to tensor products.

Theorem 5.5.2. $\mathrm{Tr}(A \otimes B) = \mathrm{Tr}(A)\,\mathrm{Tr}(B)$, *where $A \in \mathbb{C}^{n\times n}$ and $B \in \mathbb{C}^{n'\times n'}$.*

Proof. Let the entries of A be denoted by $a_{i,j}$ and the entries of B be denoted by $b_{i,j}$. First, the left side will be simplified.

$$\mathrm{Tr}(A \otimes B) \tag{5.20}$$

$$= \mathrm{Tr}\left(\begin{bmatrix} a_{0,0}b_{0,0} & a_{0,0}b_{1,0} & \dots & \dots & a_{0,n}b_{0,n'} \\ a_{0,0}b_{1,0} & a_{0,0}b_{1,1} & \dots & \dots & \dots \\ \vdots & \ddots & \dots & \dots & \dots \\ \vdots & \ddots & \dots & a_{n-1,n-1}b_{n'-1,n'-1} & \dots \\ a_{n,0}b_{n',0} & \dots & \dots & \dots & a_{n,n}b_{n',n'} \end{bmatrix}\right) \tag{5.21}$$

$$= a_{0,0}b_{0,0} + a_{0,0}b_{1,1} \cdots a_{1,1}b_{0,0} + a_{1,1}b_{1,1} \cdots a_{n,n}b_{n'-1,n'-1} + a_{n,n}b_{n',n'} \tag{5.22}$$

Then the right side will be simplified and shown to be equal to the left side.

$$\operatorname{Tr}(A)\operatorname{Tr}(B) \tag{5.23}$$

$$= \operatorname{Tr}\left(\begin{bmatrix} a_{0,0} & a_{0,1} & \dots & \dots \\ a_{1,0} & a_{1,1} & \dots & \dots \\ \vdots & \ddots & \dots & \dots \\ \vdots & \ddots & \dots & a_{n,n} \end{bmatrix}\right) \operatorname{Tr}\left(\begin{bmatrix} b_{0,0} & b_{1,0} & \dots & \dots \\ b_{1,0} & b_{1,1} & \dots & \dots \\ \vdots & \ddots & \dots & \dots \\ \vdots & \ddots & \dots & b_{n',n'} \end{bmatrix}\right) \tag{5.24}$$

$$= (a_{0,0} + a_{1,1} \cdots a_{n-1,n-1} + a_{n,n})(b_{0,0} + b_{1,1} \cdots b_{n'-1,n'-1} + b_{n',n'}) \tag{5.25}$$

$$= a_{0,0}b_{0,0} + a_{0,0}b_{1,1} \cdots a_{1,1}b_{0,0} + a_{1,1}b_{1,1} \cdots a_{n,n}b_{n'-1,n'-1} + a_{n,n}b_{n',n'} \tag{5.26}$$

□

5.6 Density Matrix

A density matrix may additionally be used to represent a quantum state. A density matrix ρ is simply defined as:

$$\rho = |\psi\rangle \langle\psi| \tag{5.27}$$

where $\langle\psi| = |\psi\rangle^\dagger$ and the 'dagger' symbol, $\dagger$, denotes the conjugate transpose of a matrix. Density matrices are useful for representing mixed states, which are defined as probabilistic mixtures of different quantum states. State vectors and bra-ket notation can only represent pure states, i.e. as a single quantum state vector or as listing out the terms in bra-ket notation. For simplicity, this thesis is only concerned with pure states. The theorems and concepts herein can all be extended to consider

mixed-states.

5.7 Quantum Gates

5.7.1 Definition

A quantum gate (in its physical form) allows one to manipulate and change the state of a quantum state deliberately and predictably. This section will describe the gates used in this thesis and the notational conventions for gates that it uses. Quantum gates are mathematically represented via unitary matrices.

5.7.2 Unitary Transformations

Quantum computing in its standard form involves performing operations on qubits within closed systems via quantum gates. Unitary transformations are transformations that do not violate the closed system requirement. Unitary matrices are used to represent these transformations. A matrix U is unitary if and only if it satisfies the following:

$$UU^\dagger = I = U^\dagger U \tag{5.28}$$

When operating on n qubits, the unitary $U \in \mathbb{C}^{2^n \times 2^n}$.

5.7.3 Single-Qubit Gates

A 'single-qubit gate' is a gate that acts only on a single qubit. As a result, a single-qubit gate always corresponds to a 2 by 2 unitary matrix, $U \in \mathbb{C}^{2\times 2}$.

5.7.4 Subscripts on Quantum Gates

For a single-qubit quantum gate, U, the subscript i in U_i may be used to indicate which qubit position that gate is applied to.

Example 5.7.1. *For example, Z_2 acting on a system of four qubits would be explicitly represented as $I \otimes Z \otimes I \otimes I$.*

Example 5.7.2. *For example, Y_4 acting on a system of five qubits would be explicitly represented as $Y \otimes I \otimes I \otimes I \otimes I$.*

Theorems 5.7.2, 5.7.3, 5.7.6, 5.7.7 and 5.7.9 in Subsections 5.7.5 and 5.7.6 use subscripts on gates in a different way. This is because, in those theorems, C_i and P_j are denoted as arbitrary Clifford gates and Pauli gates respectively. The following theorem is another example of using subscripts in this way.

Theorem 5.7.1. *U_i commutes with V_k for any two values of i and k where $i \neq k$.*

Proof. In essence, this is because they act on different qubits.

For simplicity, let I_x denote a 2 by 2 identity matrix throughout this proof. The subscripted index is added to convey the number and position of each of the 2 by 2 I matrices each of U_i and V_k are composed of. Without loss of generality, assume $i < k$. Without loss of generality, $U_i = I_{n-1} \otimes I_{n-2} \otimes \cdots \otimes U \otimes \cdots \otimes I_0$ commutes with $V_k = I_{n-1} \otimes I_{n-2} \otimes \cdots \otimes V \otimes \cdots \otimes I_0$, as seen:

$$U_i V_k = I_{n-1}I_{n-1} \otimes I_{n-2}I_{n-2} \otimes \cdots \otimes UI_i \otimes I_{i-1}I_{i-1} \otimes \cdots \otimes I_k V \otimes \cdots \otimes I_0 I_0 \quad (5.29)$$

$$U_i V_k = I_{n-1}I_{n-1} \otimes I_{n-2}I_{n-2} \otimes \cdots \otimes I_i U \otimes I_{i-1}I_{i-1} \otimes \cdots \otimes V I_k \otimes \cdots \otimes I_0 I_0 \quad (5.30)$$

$$V_k U_i = I_{n-1}I_{n-1} \otimes I_{n-2}I_{n-2} \otimes \cdots \otimes I_i U \otimes I_{i-1}I_{i-1} \otimes \cdots \otimes V I_k \otimes \cdots \otimes I_0 I_0 \quad (5.31)$$

□

Asides from Theorem 5.7.1 and the theorems previously listed, all other instances of subscripts being used on gates follow the convention as shown in Examples 5.7.1 and 5.7.2. The context of the subscript should make the convention clear.

5.7.5 Pauli Matrices

The Pauli matrices (Pauli is pronounced as pow-lee) are matrices that are both unitary and Hermitian, and are a very important tool in quantum information science. A Hermitian matrix is a matrix that is equal to its conjugate transpose. The Pauli matrices are defined as:

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

Sometimes the identity matrix $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ is included in this set. The matrices X_i,Y_i and Z_i acting on an n-qubit system each denote a tensor product of a single-qubit Pauli and $n-1$ number of I matrices (remark Examples 5.7.1 and 5.7.2). The index i indicates the Pauli is acting on the qubit at position i.

The Pauli matrices X, Y and Z correspond to 180° rotations around the x, y and z axes of the Bloch sphere respectively. See Qiskit's textbook for an explanation of the Bloch sphere [9].

Let $P(n)$ be the set of all n-qubit tensor products of Pauli matrices, including the identity matrix, with a scalar of $1, -1, i$ or $-i$. This can be more formally written as:

$$P(n) = \left\{ e^{i\theta\pi/2} P_0 \otimes \cdots \otimes P_{n-1} \mid \theta = 0, 1, 2, 3, P_i = X, Y, Z, I \right\} \tag{5.32}$$

Additionally, let $P(n)/U(1)$ be $P(n)$ but without the multiplicative constants -1, i or $-i$.

Theorem 5.7.2. *A matrix $A \in P(n+m)$ where $n+m \geq 2$ if and only if it can be represented as a tensor product of an element $B \in P(n)$ and an element $C \in P(m)$.*

Proof. This is possible due to tensor products being associative. Let $B = P_0 \otimes P_1 \otimes \cdots \otimes P_{n-1}$ and let $C = P_0 \otimes P_1 \otimes \cdots \otimes P_{m-1}$.

$$A = B \otimes C \tag{5.33}$$

$$= (P_0 \otimes P_1 \cdots P_{n-1}) \otimes (P_n \cdots P_{n+m-2} \otimes P_{n+m-1}) \tag{5.34}$$

$$= P_0 \otimes P_1 \otimes \cdots \otimes P_{n-1} \otimes P_n \otimes \cdots \otimes P_{n+m-2} \otimes P_{n+m-1} \tag{5.35}$$

Both directions of the 'if and only if' are proven by following the above equations in forward, and then in reverse. □

The inner product of two matrices, denoted by $A \in \mathbb{C}^{m \times n}$ and $B \in \mathbb{C}^{m \times n}$, is defined as $\mathrm{Tr}(A^\dagger B)$.

Theorem 5.7.3. *The inner product of two Pauli matrices $P \in P(n)/U(1)$ and $P' \in P(n)/U(1)$, $\mathrm{Tr}(P^\dagger P') = \mathrm{Tr}(PP')$, is equal to 2^n if $P = P'$, and equal to 0 if $P \neq P'$.*

Proof. Note that this is first shown for the 2×2 Pauli matrices.

$$\mathrm{Tr}(II) = \mathrm{Tr}(XX) = \mathrm{Tr}(YY) = \mathrm{Tr}(ZZ) = \mathrm{Tr}(I) = 2$$

$$\mathrm{Tr}(XY) = \mathrm{Tr}(-YX) = -\mathrm{Tr}(YX) = Tr(iZ) = iTr(Z) = 0$$

$$\mathrm{Tr}(YZ) = \mathrm{Tr}(-ZY) = -\mathrm{Tr}(ZY) = Tr(iX) = iTr(X) = 0$$

$$\mathrm{Tr}(XZ) = \mathrm{Tr}(-ZX) = -\mathrm{Tr}(ZX) = Tr(-iY) = -iTr(Y) = 0$$

It is then derived for the n-fold tensor product Pauli matrices. This is given by the property $\mathrm{Tr}(A \otimes B) = \mathrm{Tr}(A)\,\mathrm{Tr}(B)$, proven in Theorem 5.5.2. Taking the inner product of two arbitrary Pauli matrices, denoted by $P = (P_0 \otimes P_1 \otimes \cdots \otimes P_n)$ and $P' = (P'_0 \otimes P'_1 \otimes \cdots \otimes P'_n)$, gives:

$$\mathrm{Tr}((P_0 \otimes P_1 \cdots P_n)(P'_0 \otimes P'_1 \cdots P'_n)) \tag{5.36}$$

$$= \mathrm{Tr}(P_0P'_0 \otimes P_1P'_1 \cdots P_nP'_n) \tag{5.37}$$

$$= \mathrm{Tr}(P_0P'_0)\,\mathrm{Tr}(P_1P'_1) \cdots \mathrm{Tr}(P_nP'_n) \tag{5.38}$$

If $P \neq P'$, then there exists at least one index i such that $P_i \neq P'_i$. Therefore, Equation (5.38) shows that if $P \neq P'$, then $\mathrm{Tr}(PP') = 0$, and otherwise if $P = P'$, then $\mathrm{Tr}(PP') = 2^n$. □

Corollary 5.7.1. *$P(n)/U(1)$ forms an orthogonal basis for $\mathbb{C}^{2^n \times 2^n}$.*

Proof. The inner product of any pair of Pauli matrices is 0 if the Pauli matrices are different, and 2^n if they are the same. This means they are all linearly independent. Any basis for $\mathbb{C}^{2^n \times 2^n}$ must contain 4^n elements, because each matrix has 4^n entries. Any subset of a vector space V with dimension N that consists of N linearly independent elements is a basis for V. As a result, all of the 4^n linearly independent Pauli matrices in $P(n)/U(1)$ are a basis for $\mathbb{C}^{2^n \times 2^n}$. □

Theorem 5.7.4. *The generators of $P(n)$ are X_i and Z_j, where $0 \leq i, j \leq n-1$, along with*

the coefficients, $1, -1, i, -i$.

Proof. Y_i can be generated from X_i and Z_i, along with the coefficient $-i$:

$$-iZ_iX_i = Y_i$$

I_i can be generated via:

$$X_iX_i = I_i$$

It then follows that any element from the formal definition of $P(n)$, in Equation (5.32), can be generated via multiplication (remark Theorem 5.7.2). □

Theorem 5.7.5. *Any element of* $P(n)$ *is also a part of* $C(n)$*, where* $C(n)$ *is defined in Subsection 5.7.6.*

Proof. This is due to the inherent property of the group $P(n)$ being closed under multiplication. That is, any two Pauli matrices multiplied by each other equals a Pauli matrix. □

5.7.6 Clifford Gates

A quantum gate is a Clifford gate if it can be represented by a unitary matrix belonging to the Clifford group, denoted by $C(n)$. (see [15])

$$C(n) = \{C \in U(2^n) \mid P \in P(n) \Rightarrow CPC^\dagger \in P(n)\}/U(1). \tag{5.39}$$

The term $CPC^\dagger$ is called the conjugation of P by C. Equation (5.39) states that any element of $P(n)$ must give another element of $P(n)$ when conjugated by any

element of $C(n)$[1]. The size of the Clifford group $C(n)$ is known to be

$$|C(n)| = \prod_{j=1}^{n} 2(4^j - 1)4^j = 2^{n^2+2n} \cdot \prod_{j=1}^{n} \left(4^j - 1\right) \tag{5.40}$$

The generators of the Clifford group are the S, H, and $CNOT$ gates via multiplication [16]. In this thesis, I will use the Clifford generators as a way to get a good estimate of how many basis gates are needed to implement a given unitary operation on a quantum computer. Different quantum computing hardware architectures will use different sets of basis gates for expressing arbitrary unitary operations.

Theorem 5.7.6. *An arbitrary Clifford C_0 times another arbitrary Clifford C_1 equals an arbitrary Clifford.*

Proof. This directly corresponds to the definition of Cliffords in Equation (5.39):

$$C_1 P C_1^\dagger \in P(n) \tag{5.41}$$

which means that

$$C_0 C_1 P C_1^\dagger C_0^\dagger \in P(n) \tag{5.42}$$

□

Theorem 5.7.7. *The tensor product of two arbitrary Clifford gates C_0 and C_1 is also a Clifford gate.*

Proof. Let C_0 and C_1 both be arbitrary Clifford gates belonging to the $C(m)$ and $C(n)$ groups respectively. Let P_2 be an arbitrary matrix belonging to $P(m+n)$. Let

[1] In group theoretic terminology, $C(n)$ is the normalizer of the Pauli group in the group $U(2^n)$, modulo the subgroup $U(1)$.

P_0 and P_1 be matrices belonging to the $P(m)$ and $P(n)$ groups respectively such that $P_0 \otimes P_1 = P_2$ (possible due to Theorem 5.7.2). We can see that $C_0 \otimes C_1$ belongs to the group $C(m+n)$ by:

$$(C_0 \otimes C_1) \cdot P_2 \cdot (C_0^\dagger \otimes C_1^\dagger) = (C_0 \otimes C_1) \cdot (P_0 \otimes P_1) \cdot (C_0^\dagger \otimes C_1^\dagger) \tag{5.43}$$

$$= ((C_0 \otimes C_1) \cdot (P_0 \otimes P_1)) \cdot (C_0^\dagger \otimes C_1^\dagger) \tag{5.44}$$

$$= ((C_0 \cdot P_0) \otimes (C_1 \cdot P_1)) \cdot (C_0^\dagger \otimes C_1^\dagger) \tag{5.45}$$

$$= ((C_0 \cdot P_0 \cdot C_0^\dagger) \otimes (C_1 \cdot P_1 \cdot C_1^\dagger)) \tag{5.46}$$

$$= P_0' \otimes P_1' \tag{5.47}$$

$$= P_2' \tag{5.48}$$

where P_0', P_1' and P_2' belong to $P(m)$, $P(n)$ and $P(m+n)$ respectively and the simplifications from line 5.44 to 5.46 are given by the property $(A \otimes B) \cdot (A' \otimes B') = (A \cdot A') \otimes (B \cdot B')$, see Theorem 5.5.1, of matrix multiplication and tensor products. We see that any arbitrary P_2 from $P(m+n)$ is transformed to another arbitrary matrix P_2' belonging to $P(m+n)$ when conjugated by $(C_0 \otimes C_1)$, therefore $(C_0 \otimes C_1)$ is a Clifford gate. □

Theorem 5.7.8. *$C(n)$ preserves the structure of $P(n)$.*

Proof. Let P and P' be two arbitrary Pauli matrices.

$$C\ P\ C^\dagger C\ P'\ C^\dagger = C\ P\ I\ P'\ C^\dagger = C\ P\ P'\ C^\dagger \tag{5.49}$$

This means that if P and P' commute, then $C\ P\ C^\dagger$ and $C\ P'\ C^\dagger$ must also commute. Also, it means if P and P' anti-commute, then $C\ P\ C^\dagger$ and $C\ P'\ C^\dagger$ must also anti-commute. □

Theorem 5.7.9. *A matrix C can be proven to be in $C(n)$ by showing that each of the $2n$ generators X_i and Z_j, where $0 \leq i, j \leq n-1$ (given in Theorem 5.7.4), give elements that are a part of $P(n)$ when conjugated by C.*

Proof. Assume $C \in C(n)$. Let P'' be an arbitrary element generated by the generators described previously; they are the generators of $P(n)$ excluding the multiplicative constants (see Theorem 5.7.4). These generators can be used to show $C\ P'\ C^\dagger \in P(n)$ where P' is an arbitrary element of $P(n)$. Let $P' = \alpha P''$, where $\alpha \in \{1, -1, i, -i\}$. Also let P'' be generated by w generators (as described previously), each denoted by P_k, where $0 \leq k \leq w-1$. Examine the following:

$$C\ P_0\ C^\dagger C\ P_1\ C^\dagger \cdots C\ P_{w-1}\ C^\dagger = C\ P_0\ P_1 \cdots\ P_{w-1}\ C^\dagger = C\ P''\ C^\dagger \tag{5.50}$$

Since $C \in C(n)$, it follows that $C\ P''\ C^\dagger \in C(n)$. This is because any two elements of $P(n)$ when multiplied together give another element of $P(n)$. It then follows that when any two elements $C\ P_i\ C^\dagger$ and $C\ P_j\ C^\dagger$ that also exist in $P(n)$ are multiplied together, where P_i and P_j are generators of $P(n)$, give an element in $P(n)$. Multiplying Equation (5.50) by α also allows for an arbitrary P' such that $C\ P'\ C^\dagger \in P(n)$:

$$\alpha C\ P_0\ C^\dagger C\ P_1\ C^\dagger \cdots C\ P_{w-1}\ C^\dagger = \alpha C\ P''\ C^\dagger = C\ \alpha P''\ C^\dagger = C\ P'\ C^\dagger \tag{5.51}$$

because any element from $P(n)$ multiplied by α is also in $P(n)$. Thus, checking the equality in Equation (5.50) automatically satisfies the equality in Equation (5.51) for any arbitrary $P' \in P(n)$ and ensures $C\ P'\ C^\dagger \in P(n)$. Therefore, if C can not satisfy these equations, then it must not be a part of the Clifford group. □

Additionally, it can be shown that a given mapping from the $2n$ generators

determines a unique Clifford operator [16].

5.7.7 Hadamard Gate

The Hadamard gate is simply just:

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

The nice thing about the Hadamard gate is that it is equal to its inverse, as seen by

$$H^\dagger H = HH^\dagger = HH = I.$$

Additionally, the Hadamard gate can be used to transform a qubit from the z-basis (the standard computational basis, remark Subsection 5.4.3) to the x-basis, or from the x-basis to the z-basis. The following illustrates the previous statement: Given some arbitrary single qubit $|\psi\rangle = c_0 |0\rangle + c_1 |1\rangle$, applying a H to it,

$$H |\psi\rangle = H(c_0 |0\rangle + c_1 |1\rangle) = c_0 H |0\rangle + c_1 H |1\rangle = c_0 |+\rangle + c_1 |-\rangle$$

shows a basis transformation, where $|+\rangle$ and $|-\rangle$ are the basis vectors for the x-basis and $|0\rangle$ and $|1\rangle$ are the basis vectors for the z-basis. Applying the Hadamard gate again transforms the quantum state from the x-basis back to its original representation in the z-basis.

When viewing a qubit using the Bloch sphere representation, the H gate corresponds to a rotation of 90° around the y-axis followed by a 180° around the x-axis. Equivalently, it can be seen as a rotation of 180° around the vector on the Bloch

sphere which is halfway between the x-axis and z-axis.

Theorem 5.7.10. *The H gate is a Clifford gate.*

Proof. Making use of Theorem 5.7.9, the proof can be done by showing each $H\ P\ H^\dagger$ is an element of $P(n)$, where P takes on the values of two generators of $P(1)$:

Table 5.1: A table showing H is a Clifford gate.

P	$H\ P\ H^\dagger$
X	Z
Z	X

□

5.7.8 S Gate

The S gate, also known as the 'phase gate' is simply just:

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$$

Applying the S gates three times is equal to its inverse, as seen by

$$SSSS = I = SS^\dagger = S^\dagger S.$$

When viewing a qubit using the Bloch sphere representation, the S gate corresponds to a rotation of 90° around the z-axis.

Theorem 5.7.11. *The S gate is a Clifford gate.*

Proof. Making use of Theorem 5.7.9, the proof can be done by showing each $S\ P\ S^\dagger$ is an element of $P(n)$, where P takes on the values of two generators of $P(1)$:

Table 5.2: A table showing S is a Clifford gate.

P	$S\ P\ S^\dagger$
X	Y
Z	Z

□

5.7.9 *CNOT* Gate

The *CNOT* gate (also known as the *CX* gate), short for controlled-NOT, is a common gate used in quantum computing that acts on two adjacent qubits. For each term of a quantum state, the *CNOT* gate takes the control qubit (the left qubit), and applies the *NOT* gate (also known as the X gate) to the target qubit (the right qubit) if the value of the control qubit is 1. The matrix representation for the *CNOT* gate is:

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

The *CNOT* gate also has the nice property $CNOT = CNOT^\dagger$.

Theorem 5.7.12. *The CNOT gate is a Clifford gate.*

Proof. Making use of Theorem 5.7.9, the proof can be done by showing each *CNOT P CNOT* is an element of $P(n)$, where P takes on the value of four generators of $P(2)$:

Table 5.3: A table showing *CNOT* is a Clifford gate.

P	*CNOT P CNOT*
$X \otimes I$	$X \otimes X$
$I \otimes X$	$I \otimes X$
$Z \otimes I$	$Z \otimes I$
$I \otimes Z$	$Z \otimes Z$

□

Example 5.7.3. *The following example shows a CNOT gate being applied to the state* $|00\rangle + |01\rangle - |10\rangle + i|11\rangle$.

$$CNOT(|00\rangle + |01\rangle - |10\rangle + i|11\rangle) \tag{5.52}$$

$$= CNOT|00\rangle + CNOT|01\rangle - CNOT|10\rangle + iCNOT|11\rangle \tag{5.53}$$

$$= I|00\rangle + I|01\rangle - (I \otimes X)|10\rangle + i(I \otimes X)|11\rangle \tag{5.54}$$

$$= |00\rangle + |01\rangle - |11\rangle + i|10\rangle \tag{5.55}$$

5.7.10 *SWAP* Gate

The *SWAP* gate is defined as:

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

The *SWAP* gate 'swaps' two qubits.

Example 5.7.4. *The following example shows how the SWAP operation affects an arbitrary 2-qubit state. Denote some arbitrary state* $|\psi\rangle = c_0 |00\rangle + c_1 |01\rangle + c_2 |10\rangle + c_3 |11\rangle$. *The SWAP operation works as follows:*

$$\begin{aligned} SWAP\,|\psi\rangle &= SWAP(c_0 |00\rangle + c_1 |01\rangle + c_2 |10\rangle + c_3 |11\rangle) && (5.56)\\ &= c_0 SWAP\,|00\rangle + c_1 SWAP\,|01\rangle + c_2 SWAP\,|10\rangle + c_3 SWAP\,|11\rangle && (5.57)\\ &= c_0 |00\rangle + c_1 |10\rangle + c_2 |01\rangle + c_3 |11\rangle && (5.58)\\ &= c_0 |00\rangle + c_2 |01\rangle + c_1 |10\rangle + c_3 |11\rangle && (5.59) \end{aligned}$$

The *SWAP* gate also has the nice property $SWAP = SWAP^\dagger$.

Theorem 5.7.13. *The SWAP gate is a Clifford gate.*

Proof. Making use of Theorem 5.7.9, the proof can be done by showing each *SWAP P SWAP* is an element of $P(n)$, where P takes on the value of four generators of $P(2)$:

Table 5.4: A table showing *SWAP* is a Clifford gate.

P	$SWAP\ P\ SWAP$
$X \otimes I$	$I \otimes X$
$I \otimes X$	$X \otimes I$
$Z \otimes I$	$I \otimes Z$
$I \otimes Z$	$Z \otimes I$

Another way to prove the *SWAP* gate is a Clifford gate is by making use of Theorems 5.7.6 and 5.7.7 and constructing the *SWAP* gate from the generators of the Clifford group:

$$SWAP = CNOT(H \otimes H)CNOT(H \otimes H)CNOT \tag{5.60}$$

□

5.7.11 $CIRC_{i,j}$ Gate

As far as I am aware, no one else has given a name to the gate I am about to describe. I have decided to call it a *CIRC* gate, which stands for circular shift [17]. A circular shift involves 'rotating' elements in an array such that the first element becomes the last element, and all the other elements are shifted over once. That is, given an array with n elements, the element at position 0 is moved to position $n-1$, and all elements originally at positions i where $1 \leq i \leq n-1$ move to position i' where $i' = i - 1$.

The quantum equivalent of a circular shift operation requires applying multiple

SWAP gates. To make the gate more useful, the subscripts i and j will denote performing a circular shift on qubits $i, i+1, \cdots, j-1, j$. The $CIRC_{i,j}$ moves a qubit at position i to position j, and all qubits originally at positions k where $i < k \leq j$ move to position k' where $k' = k - 1$. The qubit at position i may be referred to as the source qubit and the qubit at position j may be referred to as the destination qubit. Denote $SWAP_{k+1,k}$ as a *SWAP* gate swapping qubits at position $k+1$ and k. The $CIRC_{i,j}$ gate can be implemented using Algorithm 1.

Algorithm 1 $CIRC_{i,j}$ Gate(i,j)

```
1: U ⟸ I
2: if i > j then
3:     for each k ∈ (i − 1, i − 2, i − 3, ··· , j + 1, j) do
4:         U ⟸ SWAP_{k+1,k} U
5:     end for
6: else
7:     for each k ∈ (i, i + 1, i + 2, ··· , j − 2, j − 1) do
8:         U ⟸ SWAP_{k+1,k} U
9:     end for
10: end if
11: return U
```

Note that $CIRC_{i,j}^{\dagger} = CIRC_{j,i}$. This can be seen by carefully examining Algorithm 1. In essence, $CIRC_{i,j}$ applies all the same *SWAP* gates as $CIRC_{j,i}$ but in reverse order.

Theorem 5.7.14. *The $CIRC_{i,j}$ gate is a Clifford gate.*

Proof. First, it will be shown for the case when $i < j$. Like before, let k be any value such that $i < k \leq j$. Additionally, let h be any value such that $0 \leq h < i$ or $j < h < n$. Making use of Theorem 5.7.9, the proof can be done by showing each $CIRC_{i,j}\ P\ CIRC_{j,i}$ is an element of $P(n)$, where P takes on the value of $2n$ generators of $P(n)$:

Table 5.5: A table showing $CIRC_{i,j}$ is a Clifford gate for when $i < j$.

P	$CIRC_{i,j}\ P\ CIRC_{j,i}$
X_k	X_{k-1}
X_i	X_j
X_h	X_h
Z_k	Z_{k-1}
Z_i	Z_j
Z_h	Z_h

Now the transformations for when $i > j$ will be shown. Let k be any value such that $j < k \leq i$ and let h be any value such that $0 \leq h < j$ or $i < h < n$. The following table shows it is still a Clifford:

Table 5.6: A table showing $CIRC_{i,j}$ is a Clifford gate for when $i > j$.

P	$CIRC_{i,j}\ P\ CIRC_{j,i}$
X_k	X_{k+1}
X_i	X_j
X_h	X_h
Z_k	Z_{k+1}
Z_i	Z_j
Z_h	Z_h

Another way to prove the $CIRC_{i,j}$ gate is a Clifford gate is by making use of Theorems 5.7.6 and 5.7.7 and constructing the $CIRC_{i,j}$ gate from the generators

of the Clifford group. Algorithm 1 combined with the results in Theorem 5.7.13 showing that the *SWAP* gate can be constructed by the generators of the Clifford group proves this. □

Corollary 5.7.2. *$CIRC_{i,j}$ can be implemented using $O(|i-j|)$ gates from the Clifford generators.*

Proof. As seen in Theorem 5.7.13, the *SWAP* gate can be implemented using five of the Clifford generators. By the construction in Algorithm 1, the $CIRC_{i,j}$ gate can be implemented using $|i-j|$ number of *SWAP* gates. That means the $CIRC_{i,j}$ gate can be implemented using $5(|i-j|) = O(|i-j|)$ gates from the Clifford generators. □

5.7.12 $CNOT_{i,j}$ Gate

The $CNOT_{i,j}$ gate (also known as the $CX_{i,j}$ gate) is an extension of the normal *CNOT* gate, by being able to choose what is the control qubit, and what is the target qubit. The control qubit is the qubit at position i and the target qubit is the qubit at position j. When $i = j + 1$ it is equivalent to the *CNOT* gate.

Example 5.7.5. *The following example shows a $CNOT_{i,j}$ gate, where $i = 1$ and $j = 3$ being applied to the state $|0001\rangle - |0110\rangle + |1001\rangle + |1110\rangle$.*

$$CNOT_{1,3}(|0001\rangle - |0110\rangle + |1001\rangle + |1110\rangle) \tag{5.61}$$

$$= CNOT_{1,3}|0001\rangle - CNOT_{1,3}|0110\rangle + CNOT_{1,3}|1001\rangle + CNOT_{1,3}|1110\rangle \tag{5.62}$$

$$= I|0001\rangle + (X \otimes I \otimes I \otimes I)|0110\rangle - I|1001\rangle + (X \otimes I \otimes I \otimes I)|1110\rangle \tag{5.63}$$

$$= |0001\rangle + |1110\rangle - |1001\rangle + |0110\rangle \tag{5.64}$$

Just like the $CNOT$ gate, the $CNOT_{i,j}$ gate also has the nice property $CNOT_{i,j} = CNOT_{i,j}^{\dagger}$. Note that $i \neq j$.

Theorem 5.7.15. *The $CNOT_{i,j}$ gate is a Clifford gate.*

Proof. Making use of Theorem 5.7.9, the proof can be done by showing each $CNOT_{i,j}\ P\ CNOT_{i,j}$ is an element of $P(n)$, where P takes on the value of $2n$ generators of $P(n)$. First, it will be shown for only four of the generators:

Table 5.7: A table showing $CNOT_{i,j}$ is a Clifford gate.

P	$CNOT_{i,j}\ P\ CNOT_{i,j}$
X_iI_j	X_iX_j
I_iX_j	I_iX_j
Z_iI_j	Z_iI_j
I_iZ_j	Z_iZ_j

For the other $2n-4$ generators, $P = CNOT_{i,j}\ P\ CNOT_{i,j}^{\dagger}$ because the $CNOT_{i,j}$ gate only acts on qubits at positions i and j. □

The $CNOT_{i,i+1}$ gate can be constructed from the Clifford generators like so:

$$H_iH_{i+1}CNOT_{i+1,i}H_iH_{i+1} \tag{5.65}$$

5.7.13 $CY_{i,j}$ Gate

The $CY_{i,j}$ gate is like the $CNOT_{i,j}$ gate, except a Y gate acts on the target qubit when the control qubit is set to 1, instead of an X gate. If $i = j+1$ then it can just be called a CY gate. The $CY_{i,j}$ gate also has the nice property $CY_{i,j} = CY_{i,j}^{\dagger}$. Note that $i \neq j$.

Theorem 5.7.16. *The $CY_{i,j}$ gate is a Clifford gate.*

Proof. Making use of Theorem 5.7.9, the proof can be done by showing each $CY_{i,j}\ P\ CY_{i,j}$ is an element of $P(n)$, where P takes on the value of $2n$ generators of $P(n)$. First, it will be shown for only four of the generators:

Table 5.8: A table showing $CY_{i,j}$ is a Clifford gate.

P	$CY_{i,j}\ P\ CY_{i,j}$
X_iI_j	X_iY_j
I_iX_j	Z_iX_j
Z_iI_j	Z_iI_j
I_iZ_j	Z_iZ_j

For the other $2n-4$ generators, $P = CY_{i,j}\ P\ CY_{i,j}^{\dagger}$ because the $CY_{i,j}$ gate only acts on qubits at positions i and j. □

The $CY_{i,i-1}$ gate can be constructed from the Clifford generators like so:

$$S_{i-1}CNOT_{i,i-1}S_{i-1}S_{i-1}S_{i-1} \tag{5.66}$$

The $CY_{i,i+1}$ gate can be constructed from the Clifford generators like so:

$$S_{i+1}H_iH_{i+1}CNOT_{i+1,i}H_iH_{i+1}S_{i+1}S_{i+1}S_{i+1} \tag{5.67}$$

5.7.14 $CZ_{i,j}$ Gate

The $CZ_{i,j}$ gate is like the $CNOT_{i,j}$ gate, except a Z gate acts on the target qubit when the control qubit is set to 1, instead of an X gate. If $i = j+1$ then it can just be called

a CZ gate. The $CZ_{i,j}$ gate also has the nice property $CZ_{i,j} = CZ_{i,j}^{\dagger}$. Note that $i \neq j$.

Theorem 5.7.17. *The $CZ_{i,j}$ gate is a Clifford gate.*

Proof. Making use of Theorem 5.7.9, the proof can be done by showing each $CZ_{i,j}\ P\ CZ_{i,j}$ is an element of $P(n)$, where P takes on the value of $2n$ generators of $P(n)$. First, it will be shown for only four of the generators:

Table 5.9: A table showing $CZ_{i,j}$ is a Clifford gate.

P	$CZ_{i,j}\ P\ CZ_{i,j}$
$X_i I_j$	$X_i Z_j$
$I_i X_j$	$Z_i X_j$
$Z_i I_j$	$Z_i I_j$
$I_i Z_j$	$I_i Z_j$

For the other $2n - 4$ generators, $P = CZ_{i,j}\ P\ CZ_{i,j}^{\dagger}$ because the $CZ_{i,j}$ gate only acts on qubits at positions i and j. □

The $CZ_{i,i-1}$ gate can be constructed from the Clifford generators like so:

$$H_{i-1} CNOT_{i,i-1} H_{i-1} \tag{5.68}$$

The $CZ_{i,i+1}$ gate can be constructed from the Clifford generators like so:

$$H_i CNOT_{i+1,i} H_i \tag{5.69}$$

5.7.15 $CU_{i,j}$ Gate

The $CX_{i,j}$, $CY_{i,j}$ and $CZ_{i,j}$ gates are all considered $CU_{i,j}$ gates (see Subsections 5.7.9, 5.7.13 and 5.7.14). A $CU_{i,j}$ gate is a controlled-unitary gate that applies a single-qubit unitary to the target qubit j if the control qubit i is set to 1. If $i = j + 1$ then it can just be called a CU gate.

Theorem 5.7.18. *Let $CU_{i,j}$ be an arbitrary $CU_{i,j} \in \{CX_{i,j}, CY_{i,j}, CZ_{i,j}\}$ and let $CU_{k,j}$ be set to the same type of controlled-U gate as $CU_{i,j}$. The operations $CU_{i,j}$ and $CU_{k,j}$ commute for any values of i, j and k where $i \neq j \neq k$.*

Proof. $CU_{i,j}$ commutes with $CU_{k,j}$ for any values of i, j and k where $i \neq j \neq k$. This is because the only qubit they both perform a transformation on is at position j, and the transformations that are applied are either both Xs, Y's or Z's which of course commute with themselves: $XX = XX$, $YY = YY$ and $ZZ = ZZ$. For some state $|\psi\rangle = c_0|x_0\rangle + c_1|x_1\rangle + c_2|x_2\rangle \cdots + c_{2^n-1}|x_{2^n-1}\rangle$ observe what may happen for an arbitrary $c_w |x_w\rangle$. Each of $CU_{i,j}$ and $CU_{k,j}$ may either apply a U, or an I to the qubit at position j for $c_w |x_w\rangle$. Denote $U(q_i)_j$ and $I(q_i)_j$ as a U_j and I_j respectively applied to $c_w |x_w\rangle$ due to $CU_{i,j}$. Observe that for an application of $CU_{i,j}CU_{k,j}$, the transformation on $c_w |x_w\rangle$ for any arbitrary w is either $U(q_i)_j U(q_k)_j = I_j$, $U(q_i)_j I(q_k)_j = U_j$ or $I(q_i)_j U(q_k)_j = U_j$. The transformation by $CU_{i,j}CU_{k,j}$ is the same as the one by $CU_{k,j}CU_{i,j}$, as shown by the following equations:

$$U(q_i)_j U(q_k)_j = I_j = U(q_k)_j U(q_i)_j \tag{5.70}$$

$$U(q_i)_j I(q_k)_j = U_j = I(q_k)_j U(q_i)_j \tag{5.71}$$

$$I(q_i)_j U(q_k)_j = U_j = U(q_k)_j I(q_i)_j \tag{5.72}$$

where the left side of each equation is the transformation done by

$$CU_{i,j}CU_{k,j}$$

and the right side of each equation is the transformation done by

$$CU_{k,j}CU_{i,j}.$$

□

Theorem 5.7.19. *Let $CU_{i,j}$ be an arbitrary $CU_{i,j} \in \{CX_{i,j}, CY_{i,j}, CZ_{i,j}\}$ and let $CU_{k,w}$ also be an arbitrary $CU_{k,w} \in \{CX_{k,w}, CY_{k,w}, CZ_{k,w}\}$. The operations $CU_{i,j}$ and $CU_{k,w}$ commute for any values of i, j, k and w where $i \neq j \neq k \neq w$.*

Proof. A density matrix can be rewritten as a linear combination of Pauli matrices, as they form a basis (remark Corollary 5.7.1). As a result, if one can show that two given matrices commute when conjugating any given Pauli, then they will also commute when conjugating any given density matrix. Obersving Tables 5.3, 5.8 and 5.9 shows that $CU_{i,j}$ and $CU_{k,w}$ commute (remark how single-qubit gates applied to different qubits commute, Theorem 5.7.1). □

Theorem 5.7.20. *Each of the $CY_{i,j}$, $CY_{i,j}$ and $CZ_{i,j}$ gates can be constructed using $O(|i-j|)$ number of gates from the Clifford generators. Making use of Theorems 5.7.6 and 5.7.7, this also proves that they are all Clifford gates.*

Proof. Using the results from Theorem 5.7.14 showing that the $CIRC_{i,j}$ gate can be constructed by the generators of the Clifford group allows the proof to be done by showing that $CX_{i,j}$, $CY_{i,j}$ and $CZ_{i,j}$ can be constructed from *CNOT* gates, *H* gates, *S*

gates and $CIRC_{i,j}$ gates. First for $CX_{i,j}$ gates when $i > j$:

$$CX_{i,j} = CIRC_{i-1,j}CNOT_{i,i-1}CIRC_{j,i-1} \tag{5.73}$$

and in the other case where $i < j$,

$$CX_{i,j} = CIRC_{i+1,j}CNOT_{i,i+1}CIRC_{j,i+1} \tag{5.74}$$

$$= CIRC_{i+1,j}H_iH_{i+1}CNOT_{i+1,i}H_iH_{i+1}CIRC_{j,i+1} \tag{5.75}$$

Next for $CY_{i,j}$ gates when $i > j$:

$$CY_{i,j} = CIRC_{i-1,j}CY_{i,i-1}CIRC_{j,i-1} \tag{5.76}$$

$$= CIRC_{i-1,j}S_{i-1}CNOT_{i,i-1}S_{i-1}S_{i-1}S_{i-1}CIRC_{j,i-1} \tag{5.77}$$

and in the other case where $i < j$,

$$CY_{i,j} = CIRC_{i+1,j}CY_{i,i+1}CIRC_{j,i+1} \tag{5.78}$$

$$= CIRC_{i+1,j}S_{i+1}H_iH_{i+1}CNOT_{i+1,i}H_iH_{i+1}S_{i+1}S_{i+1}S_{i+1}CIRC_{j,i+1} \tag{5.79}$$

Lastly, for $CZ_{i,j}$ gates when $i > j$:

$$CZ_{i,j} = CIRC_{i-1,j}CZ_{i,i-1}CIRC_{j,i-1} \tag{5.80}$$

$$= CIRC_{i-1,j}H_{i-1}CNOT_{i,i-1}H_{i-1}CIRC_{j,i-1} \tag{5.81}$$

and in the other case where $i < j$,

$$CZ_{i,j} = CIRC_{i+1,j} CZ_{i,i+1} CIRC_{j,i+1} \tag{5.82}$$

$$= CIRC_{i+1,j} H_i CNOT_{i+1,i} H_i CIRC_{j,i+1} \tag{5.83}$$

The constructions make use of the constructions for $CU_{i,i-1}$ and $CU_{i,i+1}$ in Subsections 5.7.12, 5.7.13 and 5.7.14. A given $CU_{i,j}$ is constructed by moving the target qubit next to the control qubit, applying a $CU_{i,i-1}$ or $CU_{i,i+1}$, followed by moving the (now modified) target qubit back to its original position. To additionally verify the correctness, combining the results from the tables in Subsections 5.7.11, 5.7.12, 5.7.13 and 5.7.14 shows that the $2n$ Pauli generators are each mapped onto the correct Pauli (as per Theorem 5.7.9). Each mapping of the $2n$ Pauli generators determines a unique Clifford [16].

Corollary 5.7.2 showed that a $CIRC_{i,j}$ gate can be implemented using $O(|i-j|)$ number of gates from the Clifford generators. The constructions shown for each of $CX_{i,j}$, $CY_{i,j}$, and $CZ_{i,j}$ use two $CIRC$ gates where the source and destination qubits take on values of $i+1$, $i-1$ or j. That means at most $2(|i-j|+1)$ $SWAP$ gates would be used in the construction, which equals $5(2(|i-j|+1) = O(|i-j|))$ gates from the Clifford generators (see Corollary 5.7.2). The remaining gates used in each of the constructions use at most nine gates from the Clifford generators. Thus, $5(2(|i-j|)) + 9 = O(|i-j|)$. □

Theorem 5.7.21. *For control qubits $\{i \mid 0 \leq i < n, i \neq j\}$ and a fixed target qubit j, $n-1$ or less number of consecutively applied $CU_{i,j}$ gates can be implemented using $O(n)$ number of gates from the Clifford generators.*

Proof. This theorem makes use of the constructions provided in Theorem 5.7.20 and

Algorithm 1. Each index i of a control qubit can be denoted as a part of a list $B(j)$, where each element in the list is accessible via $B(j,k)$ where $1 \leq k < n$. Additionally, assume the list is sorted in ascending order, such that $B(j,k) < B(j,k+1)$ for all k. By Theorem 5.7.18, the order that the $CU_{i,j}$ gates are applied does not matter. For the proof, the $CU_{i,j}$ gates will be applied in ascending order of the control qubits' indices (the same order in $B(j)$). The following construction shows the number of gates required:

Algorithm 2 $CU_{i,j}$ Gates(CU,$B(j)$,j)

1: $U \Longleftarrow I$
2: $i' \Longleftarrow j$
3: **for each** $i \in (i \mid i \in B(j), i < j)$ **do**
4: $\quad U \Longleftarrow CU_{i,i+1} CIRC_{i',i+1} U$
5: $\quad i' \Longleftarrow i+1$
6: **end for**
7: **for each** $i \in (i \mid i \in B(j), i > j)$ **do**
8: $\quad U \Longleftarrow CU_{i,i-1} CIRC_{i',i-1} U$
9: $\quad i' \Longleftarrow i-1$
10: **end for**
11: $U \Longleftarrow CIRC_{i',j} U$
12: **return** U

The idea behind this algorithm here is that the consecutive $CU_{i,j}$ gates have redundant *SWAP* gates omitted which cancel out each other to equal the identity. Typically a single $CU_{i,j}$ gate requires moving the target qubit next to the control qubit, applying a $CU_{i,i+1}$ or $CU_{i,i-1}$ gate, followed by moving the target qubit back to its original position. In Algorithm 2, moving the target qubit back to its original position is omitted for every application of each $CU_{i,j}$ and is only moved back to its original position at the very end, on Line 11. The algorithm performs less than $2n$ number of *SWAP* operations, and less than n number of $CU \in \{CU_{i,i-1}, CU_{i,i+1}\}$

operations. Making use of the constructions in Theorems 5.7.13, 5.7.14 and 5.7.20, this means less than $(2n)(5) + 9n$ gates from the Clifford generators are used, which is $O(n)$. □

5.8 Ranking and Unranking Functions

A ranking function is a bijective function which maps an element $s \in S$, where S is some defined set, to an integer $i \in \{0, 1, \cdots |S| - 1\}$. It thus gives a 'rank' to each element in the set. The inverse of a ranking function is called an unranking function: This is a bijective function which maps an integer $i \in \{0, 1, \cdots |S| - 1\}$ to a corresponding s from a defined set S [18]. Unranking and ranking functions are useful for certain methods of encryption and have been studied extensively for permutations of sets [19].

5.9 Quantum Message Authentication

Quantum Message Authentication (QMA) involves performing operations on a quantum message to ensure the message has not been tampered with by an adversary. Quantum states place the additional requirement that authentication can not be performed without encryption. Thus, the message must be modified by the authentication process [2]. The work in [20] showed that the Clifford code QMA scheme can also be extended to include replay detection.

5.9.1 Authenticity, Integrity and Replay Detection

In cryptography, authenticity, integrity and replay detection are all distinct but closely related terms. Authenticity detection ensures the message received by the receiver is sent from the sender. Integrity detection ensures that the message has not been tampered with by an adversary while in transit. Replay detection ensures that the message has not been resent or delayed by an adversary while in transit.

5.9.2 Keys and Key Mappings

The operations performed on the quantum message by a given sender and receiver are determined by some key k along with a function that maps the key to its corresponding operations. The receiver applies the inverse of the operations the sender applies. The key is assumed to be classical, and is stored as a bit string.

5.9.3 Data Qubits and Signature Qubits

Data qubits are the qubits that make up the desired message the sender wants to send to the receiver. There are m number data qubits in the quantum message ρ. Signature qubits are additional qubits added to the message for facilitating authentication. There are d number data qubits in the quantum message ρ. The quantum message ρ has $n = m + d$ qubits.

5.10 Measurement

The measurement result of a quantum state is a probabilistic outcome dependent on the complex amplitudes of the state. Measurement can be interpreted via any

of one of the three quantum state representations discussed in this thesis: bra-ket notation, state vector and density matrix.

Remark that $|\psi\rangle^\dagger = \langle\psi|$. In the state vector form, a measurement outcome m along with measurement operator M_m occurs with probability $p(m)$ such that

$$p(m) = \left\langle \psi \left| M_m^\dagger M_m \right| \psi \right\rangle \tag{5.84}$$

and the resulting state after being measured is

$$\frac{M_m|\psi\rangle}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}} \tag{5.85}$$

Equation (5.85) precisely characterizes how a quantum state is disturbed when measured. For this thesis, measurements will always be in the standard computational basis. One way to denote the measurement operators in the standard computational basis is as:

$$Z_+ = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \tag{5.86}$$

and

$$Z_- = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{5.87}$$

with corresponding measurement outcomes of $+1$ and -1 respectively. Note that these operators can only measure single-qubit states. Different notations may be given for the measurement outcomes; I chose $+1$ and -1 as they correspond to the

eigenvector/eigenvalue pairs of the Z matrix. That is, the resulting state after being measured,

$$\frac{Z_+|\psi\rangle}{\sqrt{\langle\psi\,|Z_+^\dagger Z_+|\,\psi\rangle}} \tag{5.88}$$

is equal to the eigenvector $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ with eigenvalue $+1$, and

$$\frac{Z_-|\psi\rangle}{\sqrt{\langle\psi\,|Z_-^\dagger Z_-|\,\psi\rangle}} \tag{5.89}$$

is equal to the eigenvector $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ with eigenvalue -1. For extending this single-qubit measurement to multi-qubit states, all that is needed is to take the tensor product of the measurement operators. In this thesis, the following measurement operators for measuring the rightmost d signature qubits are given by:

$$\underbrace{I \otimes I \otimes \cdots \otimes I}_{m \text{ times}} \otimes \underbrace{Z_\pm \otimes Z_\pm \otimes \cdots \otimes Z_\pm}_{d \text{ times}} \tag{5.90}$$

where each $Z_\pm \in \{Z_-, Z_+\}$. This results in 2^d measurement operators with 2^d corresponding measurement outcomes. This measurement operator can be extended to a more general case: for each qubit that one wants to measure, apply a $Z_\pm$, else apply a I.

Conveniently, this thesis makes use of relatively simple measurements using the operators in Equation (5.90). That is, the state that is being measured is almost

always in the form:

$$|\phi\rangle \otimes \underbrace{|b\rangle \otimes |b\rangle \otimes \cdots |b\rangle}_{d \text{ times}} \tag{5.91}$$

where $|\phi\rangle$ is an m-qubit state tensored with the d-qubit state $\underbrace{|b\rangle \otimes |b\rangle \otimes \cdots \otimes |b\rangle}_{d \text{ times}}$. Each $|b\rangle \in \{|0\rangle, |1\rangle\}$, as an example:

$$\underbrace{|b\rangle \otimes |b\rangle \otimes \cdots \otimes |b\rangle}_{d \text{ times}} = |01 \cdots 0\rangle \tag{5.92}$$

This kind of state results in a very straightforward measurement of the rightmost d qubits. Whatever state the rightmost d qubits are in, directly corresponds to the measurement results. The measurement also does not disturb the state. The following examples will help clarify.

Example 5.10.1. *If the state $|\phi\rangle \otimes |0\rangle$ is measured using the $2^d = 2^1 = 2$ measurement operators in Equation* (5.90)*, the measurement outcome will be:*

$$+1$$

with probability 1 *and the resulting state will be unaffected.*

Example 5.10.2. *If the state $|\phi\rangle \otimes |011010\rangle$ is measured using the $2^d = 2^6 = 64$ measurement operators in Equation* (5.90)*, the measurement outcome will be:*

$$+1, -1, -1, +1, -1, +1$$

with probability 1 *and the resulting state will be unaffected.*

Each qubit set to $|0\rangle$ has a measurement result of $+1$ and each qubit set to $|1\rangle$ has a measurement result of -1. Also note that in the vast majority of situations in quantum computing it is unusual for the measurement to not affect the state.

5.11 Noise

In quantum computing, there is always a chance that some of the qubits will be affected by noise. There are a few common ways to model noise in quantum computing. One of these ways is via a depolarizing channel. The single-qubit depolarizing channel is:

$$\mathcal{E}(\rho) = (1-p)\rho + \frac{p}{3}(X\rho X + Y\rho Y + Z\rho Z) \tag{5.93}$$

This can be interpreted as applying a random non-identity Pauli (see Subsection 5.7.5) with probability p to the state ρ and otherwise leaving the state undisturbed with probability $1-p$. Conveniently, extending this channel to the multi-qubit case yields a representation where a non-identity Pauli is randomly applied to the data with probability p [21]. Later, the block Clifford code and Clifford code's probability of detecting a given Pauli modifying the data packet will be described.

5.12 Attack Model

Unless specified otherwise, the adversary in this thesis can only modify the data packet, denoted by ρ, while in transit by applying Pauli operations to it (see Subsection 5.7.5). That is, the adversary is restricted to some operation $A \in P(n)$. This attack model will be applied to the Clifford code and the block Clifford code. I may

in certain instances also specify the number of non-identity single-qubits the Pauli operator is composed of, as shown in Chapter 7.

Denote C and $C^\dagger$ as Clifford operations for encrypting and decrypting a data packet respectively, as described by either the Clifford code or the block Clifford code. The data packet ρ will be transformed in such a way if the adversary attacks using $A \in P(n)$:

$$C^\dagger A C \rho C^\dagger A^\dagger C = A' \rho A'^\dagger \tag{5.94}$$

where $A' \in P(n)$ by definition of the Clifford group (see Equation (5.39)).

Chapter 6

Related Work

6.1 Integrity Detection Using Quantum Tomography

A technique for detecting integrity attacks using quantum tomography is clearly outlined in [22]. The specific protocol for the tomography method is explained in more detail in [23]. Quantum tomography is the process of repeatedly measuring copies of the same quantum state to determine what the state is.

The core idea of this method of integrity detection is to use (what one may call) 'indicator qubits' which solely serve the purpose of monitoring the health of the channel the quantum message travels along. The health of the channel is monitored using quantum tomography. The indicator qubits are inserted at random positions within the entire transmitted message (one might also call the indicator qubits signature qubits). Both the sender and receiver must agree on which qubit positions in the message are for holding the data to transmit (the data qubits), and which ones are used for indicator qubits. The sender and receiver must keep the positions of the indicator qubits secret, to ensure the protocol works properly. If

an adversary tampers with the message, it may tamper with some of the indicator qubits. If enough of the indicator qubits are modified then an attack is indicated. This method is particularly interesting because it can detect at exactly which node within a network the attack is happening, by making use of the 'entanglement swapping' process.

6.2 The Signed Polynomial Code

The signed polynomial code, [24, 25], was designed with qudits in mind and offers a level of integrity detection similar to the Clifford code. Qudits are the higher-dimension equivalent of qubits. A qubit is considered a two-state system, while a qudit is a d-state system where $d > 2$. The integrity detection of the method is affected by the dimension of the qudits.

6.3 The Trap Code

The trap code makes use of quantum error correction to detect adversaries. The security of the trap code is dependent on the number of errors the chosen error correction scheme can detect and thus also dependent on how many physical qubits are encoding each logical qubit in the data packet. See [3] for an in-depth explanation of quantum error correction.

The key mapping for the trap code is similar to the block Clifford code, as the key determines Pauli operations applied to the qubits in the packet and a permutation of qubits [26]. As far as I am aware, the key mapping process for the trap code has not been explained in rigorous detail and is abstracted away whenever it

appears in the literature. The multiset permutation unranking algorithm presented in this thesis may be used to facilitate the permuting of qubits in the trap code (see Subsection 7.2.4).

6.4 The Clifford Code

6.4.1 Introduction

In reference [20], the Clifford code was described in its entirety. It describes how a Clifford operation (see Subsection 5.7.6) and its inverse Clifford operation can be used for QMA. The scheme makes use of the unranking function (see Section 5.8) for elements from the Clifford group [27], so that a given classical encryption key has a unique corresponding Clifford operation for operating on the quantum message (along with a unique inverse operation). The scheme first appends a 'signature' portion to the data before encryption, consisting of d signature qubits that are measured after the decryption process. If there is an integrity attack on the encrypted data, the scheme detects the attack with high probability. If any of the signature qubits get a measurement result of -1, an integrity attack is indicated.

6.4.2 Algorithm

Given some encryption key k, a corresponding Clifford operation, C, for encrypting the data is determined via a mapping. The mapping runs in $O\left(n^3\right)$ time and makes use of the symplectic group [27]. The mapping of k to its corresponding C will be denoted by $k \rightarrow C$. The decryption operator corresponding to k is determined via $C^\dagger$. C acts on a quantum state ρ such that $\rho = \sigma \otimes (|0\rangle\langle 0|)^{\otimes d}$ where σ is the

desired quantum state to encrypt and $(|0\rangle\langle 0|)^{\otimes d}$ is the signature portion of the message. When an adversary tampers with the encrypted data, the decryption operator may propagate some of the modifications made by the adversary onto the signature portion of the message, much like how checksums in classical data integrity detection work [28]. If the signature portion of the message is modified from its original value after the decryption process due to an integrity attack, some (or all) of the measurement results on the signature qubits may be -1 instead of all $+1$. Measuring a -1 on one of the signature qubits after decrypting the message indicates an integrity attack.

The asymptotically optimal number of gates from the Clifford generators to construct a given Clifford operation requires $O\left(\frac{n^2}{\log_2(n)}\right)$ gates [29, 30]. Thus, the Clifford code uses $O\left(\frac{n^2}{\log_2(n)}\right)$ gates from the Clifford generators. The largest part of the construction comes from applying *CNOT* gates.

The Clifford code detects an adversary which applies attacks from $P(n)$ with a probability of:

$$1 - \frac{2^{2m+d} - 1}{2^{2n} - 1} \approx 1 - \left(\frac{1}{2}\right)^d \tag{6.1}$$

Example 6.4.1 shows a simple example of encryption and decryption without an adversary present. To concretely illustrate how this encryption scheme can detect integrity attacks, Examples 6.4.2 and 6.4.3 are provided. Both examples have the same data being encrypted, and the same operation done to the data by the adversary, but differ only by the encryption and decryption operation chosen. They illustrate how which key is chosen to encrypt and decrypt the data can affect the ability to detect integrity attacks.

Example 6.4.1. *The following details an example of encrypting and decrypting a message with no adversary present. Let the encryption operator* $C = CNOT(X \otimes X)$*, and the data to encrypt* $\sigma = |1\rangle\langle 1|$ *and the length of the signature* $d = 1$*. This means that C will be acting on the state* $\rho = |1\rangle\langle 1| \otimes |0\rangle\langle 0| = |10\rangle\langle 10|$*. First, the data is encrypted:*

$$\rho' = C\rho C^\dagger = CNOT(X \otimes X)|10\rangle = |01\rangle \tag{6.2}$$

Then the data is decrypted back to its original contents:

$$\Rightarrow C^\dagger|01\rangle = (X \otimes X)CNOT|01\rangle = (X \otimes X)|01\rangle = |10\rangle \tag{6.3}$$

The original data $\sigma = |1\rangle\langle 1|$ *may still be read or used, and measuring the signature of the decrypted data does not indicate an integrity attack occurred. Example 6.4.2 uses the same data to encrypt and the same encryption/decryption operators, but outlines what happens if there is an adversary present.*

Example 6.4.2. *The following details an example of an adversary's integrity attack being detected with probability 1. The same as in Example 6.4.1, let* $C = CNOT(X \otimes X)$*,* $\sigma = |1\rangle\langle 1|$ *and* $d = 1$*. This means that C will be acting on the state* $\rho = |1\rangle\langle 1| \otimes |0\rangle\langle 0| = |10\rangle\langle 10|$*. First, the data is encrypted:*

$$\rho' = C\rho C^\dagger = CNOT(X \otimes X)|10\rangle = |01\rangle \tag{6.4}$$

Then the adversary modifies the encrypted data by applying an X gate to it:

$$\Rightarrow (X \otimes I)\rho'(X \otimes I) = (X \otimes I)|01\rangle = |11\rangle \tag{6.5}$$

Decrypting the modified message results in:

$$\Rightarrow C^\dagger|11\rangle = (X \otimes X)CNOT|11\rangle = (X \otimes X)|10\rangle = |01\rangle \tag{6.6}$$

The decrypter of the data then measures -1 on the signature qubit, indicating an integrity attack.

Example 6.4.3. *The following details an example of an adversary's integrity attack not being detected. Let $C = X \otimes X$, $\sigma = |1\rangle\langle 1|$ and $d = 1$. As a result, $\rho = |1\rangle\langle 1| \otimes |0\rangle\langle 0| = |10\rangle\langle 10|$, just like in Example 6.4.2. If an adversary modifies the encrypted data by applying an X gate to it, then the integrity attack will be not detected when measuring the signature qubit. First, the data is encrypted:*

$$\rho' = C\rho C^\dagger = (X \otimes X)|10\rangle = |01\rangle \tag{6.7}$$

Then the adversary modifies the encrypted data by applying an X gate to it:

$$\Rightarrow (X \otimes I)\rho'(X \otimes I) = (X \otimes I)|01\rangle = |11\rangle \tag{6.8}$$

Decrypting the modified message results in:

$$\Rightarrow C^\dagger|11\rangle = (X \otimes X)|11\rangle = |00\rangle \tag{6.9}$$

The decrypter of the data then measures $+1$ on the signature qubit, meaning the integrity attack is not detected, despite the adversary successfully tampering with the data.

Chapter 7

The Block Clifford Code

7.1 Introduction

Chapter 6.4 showed how to use Clifford operations to detect integrity attacks. This chapter describes a new type of code, the block Clifford code, which uses a subset of the Clifford operations. The block Clifford code uses asymptotically fewer gates (under reasonable assumptions) and has an asymptotically faster key mapping compared to the Clifford code. The one downside of this new method is that the probability of detecting integrity attacks is dependent on the number of qubits that are modified by the adversary. As a result of this, it gives lower levels of detection than the Clifford code when the adversary attacks a small number of qubits (this will be precisely quantified in the analysis). Under certain assumptions, this can be advantageous. Particularly, in situations where the receiver still wants to accept data packets that have a very small number of noisy qubits. Qubits may acquire noise when using an imperfect channel to travel from the sender to the receiver.

The method described in this chapter works by first appending signature qubits

to the message, just like how the Clifford code does. The block Clifford code assigns a $CX_{i,j}$, $CY_{i,j}$ or $CZ_{i,j}$ gate (see Subsection 5.7.15) to each data qubit at position i so that operations the adversary may perform on a given data qubit of the packet can subsequently affect a corresponding signature qubit at position j. Additionally, single-qubit operations are performed on each data qubit to ensure that the data is encrypted and that the adversary can not modify the packet with a certain operation that the code can never detect. The previously described operations are applied during the 'encryption step'. To decrypt the data, the inverse of the encryption step is applied. Just like with the Clifford code, if the signature qubits are measured to all be $+1$ (which is guaranteed to happen if they are all in the state $|0\rangle$ upon measurement) then the code does not indicate an adversarial attack. Otherwise, an attack is indicated.

Example 6.4.2 showed how a $CX_{i,j}$ gate which spans the data portion and signature portions of the packet enables integrity attack detection. Example 6.4.3 showed that without a $CX_{i,j}$, $CY_{i,j}$ or $CZ_{i,j}$ gate affecting the data and signature portion of the message, integrity attacks are harder to detect. The strength of the Clifford code arises when it performs many operations that span both the data qubits and signature qubits (via the *CNOT* gates). The block Clifford code aims to preserve this property as much as possible.

7.2 Algorithm

7.2.1 Quantum Operations

Just like with the Clifford Code, the block Clifford code first appends d signature qubits to the message σ so that $\rho = \sigma \otimes (|0\rangle \langle 0|)^{\otimes d}$. Let m be the number of data

qubits the message σ is made up of and let $n = m + d$. For simplicity, assume m is evenly divisible by d (the reason for this will be evident soon). Next, each signature qubit has a $U_j \in \{I_j, H_j, S_j^\dagger H_j\}$ applied to it based on the key (the key mapping is described in Subsection 7.2.4) followed by m number of $CU_{i,j}$ gates, where $CU_{i,j} \in \{CX_{i,j}, CY_{i,j}, CZ_{i,j}\}$, being applied to ρ after each application of a U_j gate, where $d \leq i < n$ and $0 \leq j < d$. Each data qubit is acted on by exactly one $CU_{i,j}$. Each signature qubit is acted on by exactly $\frac{m}{d}$ number of $CU_{i,j}$'s, where each value of i is determined by the key. The I_j, H_j and $S_j^\dagger H_j$ gates convert the signature qubits to $|0\rangle$, $|+\rangle$ and $|-i\rangle$ qubits respectively (remark Subsection 5.4.3). For each $CU_{i,j}$, $CX_{i,j}$ is used if j is the position of a $|0\rangle$ signature qubit, $CY_{i,j}$ is used if j is the position of a $|+\rangle$ signature qubit and $CZ_{i,j}$ is used j is the position of a $|-i\rangle$ signature qubit. After that, each data qubit at position i is acted on by a $U_i \in \{I_i, H_i, S_i^\dagger H_i\}$, determined by the key. Finally, a permutation of the signature qubits is performed. That is, each signature qubit is moved to a new position j' within the packet (again, determined by the key), where $0 \leq j \leq j' < n$, by applying a $CIRC_{j,j'}$ gate (see Subsection 5.7.11). As a result, each data qubit at position i is moved to a new position i' where $0 \leq i' \leq i < n$ to make room for the shifting signature qubits.

To decrypt, the previously described operations are performed in reverse order, and with their conjugate transpose taken. Finally, the signature qubits are measured. All the signature qubits are measured along the z basis (see Section 5.10) and if they all have a measurement result of $+1$ then no attack is indicated, otherwise an attack is indicated.

To further illustrate how this algorithm works, examples are provided next. The examples showcased are simplified by always setting $CU_{i,j}$ to $CX_{i,j}$ and omitting the permutation of signature qubits step (i.e. always using the identity permutation).

Afterward, the pseudocode for the algorithm will be presented followed by a mathematical analysis of its effectiveness and efficiency and then documented simulation results.

Example 7.2.1. *The following details an example of encrypting and decrypting a message with no adversary present. Let the encryption operator $U' = Z \otimes H$, let the data to encrypt be $\sigma = (|00\rangle + |10\rangle)(\langle 00| + \langle 10|)$ and let $\rho = \sigma \otimes |0\rangle\langle 0|$. Let $U = U' \otimes I$. Here are the detailed steps for encrypting the data:*

$$\Rightarrow CNOT_{1,0}\rho CNOT_{1,0}^{\dagger} \tag{7.1}$$

$$= CNOT_{1,0}(|000\rangle + |100\rangle) = |000\rangle + |100\rangle \tag{7.2}$$

$$\Rightarrow CNOT_{2,0}(|000\rangle + |100\rangle) = |000\rangle + |101\rangle \tag{7.3}$$

$$\Rightarrow U(|000\rangle + |101\rangle) = (Z \otimes H \otimes I)(|000\rangle + |101\rangle) \tag{7.4}$$

$$= |0\rangle(|0\rangle + |1\rangle)|0\rangle - |1\rangle(|0\rangle + |1\rangle)|1\rangle \tag{7.5}$$

$$= |000\rangle + |010\rangle - |101\rangle - |111\rangle \tag{7.6}$$

The process for decrypting is:

$$\Rightarrow U^{\dagger}(|000\rangle + |010\rangle - |101\rangle - |111\rangle) = |000\rangle + |101\rangle \tag{7.7}$$

$$\Rightarrow CNOT_{2,0}|000\rangle + |101\rangle = |000\rangle + |100\rangle \tag{7.8}$$

$$\Rightarrow CNOT_{1,0}|000\rangle + |100\rangle = |000\rangle + |100\rangle \tag{7.9}$$

The receiver of the message measures the signature bit, resulting in no integrity attack detection. Now, the same example will be shown, but with an adversary attacking the data after it has been encrypted.

Example 7.2.2. *Let U and ρ have the same values as in Example 7.2.1. This time, however, an adversary will perform an integrity attack by applying a Hadamard gate to the rightmost data qubit of the encrypted quantum state. This example illustrates the integrity attack detection capabilities of the proposed method, by detecting the attack with $\frac{1}{2}$ probability. The data is encrypted the same way as before:*

$$\Rightarrow U(CNOT_{2,0})CNOT_{1,0}\rho CNOT_{1,0}^{\dagger}CNOT_{2,0}^{\dagger}U^{\dagger} \tag{7.10}$$

$$= |000\rangle + |010\rangle - |101\rangle - |111\rangle \tag{7.11}$$

Then, the adversary attacks by applying a Hadamard gate to the rightmost data qubit.

$$\Rightarrow (I \otimes H \otimes I)(|000\rangle + |010\rangle - |101\rangle - |111\rangle) \tag{7.12}$$

$$= (I \otimes H \otimes I)(|0\rangle(|0\rangle + |1\rangle)|0\rangle - |1\rangle(|0\rangle + |1\rangle)|1\rangle) \tag{7.13}$$

$$= |000\rangle - |101\rangle \tag{7.14}$$

The process for decrypting the altered data is:

$$\Rightarrow U^{\dagger}(|000\rangle - |101\rangle) \tag{7.15}$$

$$= |0\rangle(|0\rangle + |1\rangle)|0\rangle + |1\rangle(|0\rangle + |1\rangle)|1\rangle \tag{7.16}$$

$$= |000\rangle + |010\rangle + |101\rangle + |111\rangle \tag{7.17}$$

$$\Rightarrow CNOT_{2,0}(|000\rangle + |010\rangle + |101\rangle + |111\rangle) \tag{7.18}$$

$$= |000\rangle + |010\rangle + |100\rangle + |110\rangle \tag{7.19}$$

$$\Rightarrow CNOT_{1,0}(|000\rangle + |010\rangle + |100\rangle + |110\rangle) \tag{7.20}$$

$$= |000\rangle + |011\rangle + |100\rangle + |111\rangle \tag{7.21}$$

The receiver of the message measures the rightmost qubit, and detects an integrity attack with probability $\frac{1}{2}$

Example 7.2.2 shows the probability of integrity attack detection given the adversary applies an H gate to the rightmost data qubit. Regardless of which qubit the adversary applies the H gate to, the proposed method gives the same probability of integrity attack detection. Examples 7.2.3 and 7.2.4 demonstrate this.

Example 7.2.3. *Let U and ρ have the same values as in Example 7.2.1. This time, however, an adversary will perform an integrity attack by applying a Hadamard gate to the leftmost data qubit of the encrypted quantum state. This example illustrates the integrity attack detection capabilities of the proposed method, by detecting the attack with $\frac{1}{2}$ probability. The data is encrypted the same way as before:*

$$\Rightarrow U(CNOT_{2,0})CNOT_{1,0}\rho CNOT_{1,0}^{\dagger}CNOT_{2,0}^{\dagger}U^{\dagger} \tag{7.22}$$

$$= |000\rangle + |010\rangle - |101\rangle - |111\rangle \tag{7.23}$$

Then, the adversary attacks by applying a Hadamard gate to the leftmost data qubit.

$$\Rightarrow (H \otimes I \otimes I)(|000\rangle + |010\rangle - |101\rangle - |111\rangle) \tag{7.24}$$

$$= (H \otimes I \otimes I)(|0\rangle(|0\rangle + |1\rangle)|0\rangle - |1\rangle(|0\rangle + |1\rangle)|1\rangle) \tag{7.25}$$

$$= (|0\rangle + |1\rangle)(|0\rangle + |1\rangle)|0\rangle - (|0\rangle - |1\rangle)(|0\rangle + |1\rangle)|1\rangle) \tag{7.26}$$

Decrypting the altered data gives:

$$\Rightarrow U^{\dagger}(|0\rangle + |1\rangle)(|0\rangle + |1\rangle)|0\rangle - (|0\rangle - |1\rangle)(|0\rangle + |1\rangle)|1\rangle) \tag{7.27}$$

$$= (|0\rangle - |1\rangle)|00\rangle - (|0\rangle + |1\rangle)|01\rangle \tag{7.28}$$

$$= |000\rangle - |100\rangle - |001\rangle + |101\rangle \tag{7.29}$$

$$\Rightarrow CNOT_{2,0}(|000\rangle - |100\rangle - |001\rangle + |101\rangle) \tag{7.30}$$

$$= |000\rangle + |100\rangle - |001\rangle - |101\rangle \tag{7.31}$$

$$\Rightarrow CNOT_{1,0}(|000\rangle + |100\rangle - |001\rangle - |101\rangle) \tag{7.32}$$

$$= |000\rangle + |100\rangle - |001\rangle - |101\rangle \tag{7.33}$$

The receiver of the message measures the rightmost qubit, and detects an integrity attack with probability $\frac{1}{2}$

Example 7.2.4. *Let U and ρ have the same values as in Example 7.2.1. This time, however, an adversary will perform an integrity attack by applying a Hadamard gate to the signature qubit of the encrypted quantum state. This example illustrates the integrity attack detection capabilities of the proposed method, by detecting the attack with* $\frac{1}{2}$ *probability. The data is encrypted the same way as before:*

$$\Rightarrow U(CNOT_{2,0})CNOT_{1,0}\rho CNOT_{1,0}^{\dagger}CNOT_{2,0}^{\dagger}U^{\dagger} \tag{7.34}$$

$$= |000\rangle + |010\rangle - |101\rangle - |111\rangle \tag{7.35}$$

Then, the adversary attacks by applying a Hadamard gate to the signature qubit.

$$\Rightarrow (I \otimes I \otimes H)(|000\rangle + |010\rangle - |101\rangle - |111\rangle) \tag{7.36}$$

$$= |00\rangle(|0\rangle + |1\rangle) + |01\rangle(|0\rangle + |1\rangle) - |10\rangle(|0\rangle - |1\rangle) - |11\rangle(|0\rangle - |1\rangle) \tag{7.37}$$

$$= (|00\rangle + |01\rangle)(|0\rangle + |1\rangle) - (|10\rangle + |11\rangle)(|0\rangle - |1\rangle) \tag{7.38}$$

$$= |000\rangle + |001\rangle + |010\rangle + |011\rangle - |100\rangle + |101\rangle - |110\rangle + |111\rangle \tag{7.39}$$

$$= (|0\rangle + 1\rangle)(|0\rangle + 1\rangle)|1\rangle + (|0\rangle - 1\rangle)(|0\rangle + 1\rangle)|0\rangle \tag{7.40}$$

Decrypting the altered data gives:

$$\Rightarrow U^{\dagger}((|0\rangle + 1\rangle)(|0\rangle + 1\rangle)|1\rangle + (|0\rangle - 1\rangle)(|0\rangle + 1\rangle)|0\rangle) \tag{7.41}$$

$$= (|0\rangle - 1\rangle)|01\rangle + (|0\rangle + 1\rangle)|00\rangle \tag{7.42}$$

$$= |001\rangle - |101\rangle + |000\rangle + |100\rangle \tag{7.43}$$

$$\Rightarrow CNOT_{2,0}(|001\rangle - |101\rangle + |000\rangle + |100\rangle) \tag{7.44}$$

$$= |001\rangle - |100\rangle + |000\rangle + |101\rangle \tag{7.45}$$

$$\Rightarrow CNOT_{1,0}(|001\rangle - |100\rangle + |000\rangle + |101\rangle) \tag{7.46}$$

$$= |001\rangle - |100\rangle + |000\rangle + |101\rangle \tag{7.47}$$

The decrypter of the message measures the rightmost qubit, and detects an integrity attack with probability $\frac{1}{2}$

A formal algorithm describing the encryption and decryption processes is presented using pseudocode in Algorithms 3 and 4 respectively. The long left arrow, $\Longleftarrow$, indicates the variable to the left of the arrow being assigned the value to the right of the arrow. Unless otherwise specified, U_i refers to the single-qubit

unitaries applied at lines 5, 7 and 9 of Algorithm 3 and $CU_{i,j}$ refers to the gates applied at lines 17, 19 and 21 of Algorithm 3. The corresponding inverse operations via conjugate transpose are labelled in Algorithm 4 (remark that $CU_{i,j}^{\dagger} = CU_{i,j}$). Additionally, denote i' and j' as the positions the qubits at the original positions i and j respectively are moved to due to the *CIRC* operations performed by the permutation of signature qubits. Also, denote the parenthesis $()$ in Algorithms 3 and 4 as a list of ordered elements. The standard set-builder notation, [6], is used to build these lists. As in, they are equivalent to sets in set-builder notation, but they additionally have an ordering of elements.

The key k is made up of two subkeys such that $k = k_a, k_b$. Algorithms 5 and 8 (see Subsection 7.2.4) transform each of the subkeys k_a and k_b into k'_a and k'_b respectively. Algorithms 3 and 4 expect the processed subkeys k'_a and k'_b as input parameters, which determines which gates a part of the block Clifford code to use. Subkey k_a specifies the single-qubit unitary operations applied to each data qubit and signature qubit. Algorithm 5 takes as input a subkey k_a and maps that subkey to a ternary string, k'_a, of length n. Each value stored in k'_a at a given index i corresponds to one of the three single-qubit unitaries to apply to the qubit at position i (here, i denotes the position of either a data qubit or signature qubit). The other subkey, k_b, specifies the permutation of signature qubits and which signature qubit j is the target for each data qubit at position i for each $CU_{i,j}$. Algorithm 8 takes as input a subkey k_b and maps that subkey to a permutation, k'_b, of the multiset:

$$\{\underbrace{0,0,\cdots 0}_{\frac{m}{d} \text{ times}}\underbrace{1,1,\cdots 1}_{\frac{m}{d} \text{ times}}\cdots\underbrace{d-1,d-1,\cdots d-1}_{\frac{m}{d} \text{ times}}\underbrace{d,d,\cdots d}_{d \text{ times}}\} \tag{7.48}$$

Each index j' of an element whose value is equal to d in k'_b corresponds to a signa-

ture qubit being moved to position j' within the data packet (the last step of the encryption process). Each index i' of an element whose value is j where $j < d$ in k'_b corresponds to a data qubit which is $CU_{i,j}$'d with the signature qubit at position j. Each Subsection 7.2.4 will describe in more detail the generation of k'_a and k'_b.

Algorithm 3 returns the encrypted quantum data packet. Algorithm 4 returns a tuple. If there is no integrity attack detected, the decrypted quantum data packet is returned along with the 'accept' flag. If an integrity attack is detected, the decrypted (and maliciously modified) quantum data packet is returned along with the 'reject' flag.

Algorithm 3 Block Clifford Code Encryption(σ,n,d,k'_a,k'_b)

$\rho \Longleftarrow \sigma \otimes |0\rangle \langle 0|^{\otimes d}$
$k''_b \Longleftarrow (x \mid x \in k'_b, x \neq d)$
procedure APPLY U_i GATE(ρ,k'_a,i)
    if $k'_a(i) = 0$ **then**
        $\rho \Longleftarrow I_i \rho I_i$
    else if $k'_a(i) = 1$ **then**
        $\rho \Longleftarrow S_i^{\dagger} H_i \rho H_i S_i$
    else if $k'_a(i) = 2$ **then**
        $\rho \Longleftarrow H_i \rho H_i$
    end if
end procedure
for each $j \in (0, 1 \cdots d-1)$ **do**
    APPLY U_i GATE(ρ,k'_a,j)
    for each $i \in (d, d+1 \cdots n-1)$ **do**
        if $k''_b(i-d) = j$ **then**
            if $k'_a(j) = 0$ **then**
                $\rho \Longleftarrow CX_{i,j} \rho CX_{i,j}$
            else if $k'_a(j) = 1$ **then**
                $\rho \Longleftarrow CZ_{i,j} \rho CZ_{i,j}$
            else if $k'_a(j) = 2$ **then**
                $\rho \Longleftarrow CY_{i,j} \rho CY_{i,j}$
            end if
        end if
    end for
end for
for each $i \in (d, d+1 \cdots n-1)$ **do**
    APPLY U_i GATE(ρ,k'_a,i)
end for
$j \Longleftarrow d-1$
for each $i \in (n-1, n-2 \cdots 0)$ **do**
    if $k'_b(i) = d$ **then**
        $\rho \Longleftarrow CIRC_{j,i} \rho CIRC_{i,j}$
        $j \Longleftarrow j-1$
    end if
end for
return ρ

Algorithm 4 Block Clifford Code Decryption(ρ,n,d,k'_a,k'_b)

$k''_b \Longleftarrow (x \mid x \in k'_b, x \neq d)$
procedure APPLY $U_i^\dagger$ GATE(ρ,k'_a,i)
    if $k'_a(i) = 0$ **then**
        $\rho \Longleftarrow I_i \rho I_i$
    else if $k'_a(i) = 1$ **then**
        $\rho \Longleftarrow H_i S_i \rho S_i^\dagger H$
    else if $k'_a(i) = 2$ **then**
        $\rho \Longleftarrow H_i \rho H_i$
    end if
end procedure
$j \Longleftarrow 0$
for each $i \in (0, 1 \cdots n-1)$ **do**
    if $k'_b(i) = d$ **then**
        $\rho \Longleftarrow CIRC_{i,j} \rho CIRC_{j,i}$
        $j \Longleftarrow j + 1$
    end if
end for
for each $i \in (d, d+1 \cdots n-1)$ **do**
    APPLY $U_i^\dagger$ GATE(ρ,k'_a,i)
end for
for each $j \in (0, 1 \cdots d-1)$ **do**
    for each $i \in (d, d+1 \cdots n-1)$ **do**
        if $k''_b(i-d) = j$ **then**
            if $k'_a(j) = 0$ **then**
                $\rho \Longleftarrow CX_{i,j} \rho CX_{i,j}$
            else if $k'_a(j) = 1$ **then**
                $\rho \Longleftarrow CZ_{i,j} \rho CZ_{i,j}$
            else if $k'_a(j) = 2$ **then**
                $\rho \Longleftarrow CY_{i,j} \rho CY_{i,j}$
            end if
        end if
    end for
    APPLY $U_i^\dagger$ GATE(ρ,k'_a,j)
end for
measure the d signature qubits
if at least one signature qubit measurement returns -1 **then**
    return ρ,**reject**
else
    return ρ,**accept**
end if

Theorem 7.2.1. *Algorithms 3 and 4 run in $O(nd)$ time.*

Proof. Algorithms 3 has three loops. The nested loop at line 12 loops through d values, with its inner loop at line 14 looping through m values, equating to $O(md)$. The loop at line 26 loops through $m = O(m)$ values, and the loop at line 30 loops through $n = O(n)$ values. Additionally, the filtering at line 2 takes $O(n)$ time. This means Algorithms 3 runs in $O(md) + O(m) + O(n) + O(n) = O(md) = O(nd)$ time.

Algorithms 4 has three loops. The nested loop at line 21 loops through d values, with its inner loop at line 22 looping through m values, equating to $O(md)$. The loop at line 18 loops through $m = O(m)$ values, and the loop at line 12 loops through $n = O(n)$ values. Additionally, the filtering at line 1 takes $O(n)$ time. This means Algorithms 4 runs in $O(md) + O(m) + O(n) + O(n) = O(md) = O(nd)$ time.

It makes most sense to consider m as being asymptotically the same as n, because m must be $\frac{n}{2} \leq m < n$ because of the way Algorithms 3 and 4 are constructed. If $m < \frac{n}{2}$, then that would mean at least one data qubit would not be acted on by a $CU_{i,j}$ with a signature qubit. □

7.2.2 Construction from the Clifford Generators

As mentioned in Section 6.4, a Clifford gate can be constructed using $O\left(\frac{n^2}{\log_2(n)}\right)$ number of gates from the Clifford generators. A construction for the block Clifford code from the Clifford generators is provided in this subsection which uses $O(nd)$ number of gates. More efficient constructions may be given, but this at least shows that for a fixed d the block Clifford code uses asymptotically fewer gates from the Clifford generators than the Clifford code.

Theorem 7.2.2. *There exists an implementation of Algorithm 3 which applies $O(nd)$ gates from the Clifford generators.*

Proof. Each data qubit at position i is acted on by a single $U_i \in \{I_i, H_i, S_i^\dagger H_i\}$ gate. Each signature qubit at position j is acted on by a single $U_j \in \{I_j, H_j, S_j^\dagger H_j\}$ gate. These two previous sets of gates require $O(n)$ number of gates from the Clifford generators. That is, H requires one gate, $S^\dagger H$ requires four gates and I requires no gates, thus $4n = O(n)$.

Making use of Theorem 5.7.21, each of the d executions of the loop at line 14 can be implemented using $O(n)$ number of gates from the Clifford generators. That means the loop at line 12 applies $O(nd)$ number of gates.

Making use of Corollary 5.7.2, each of the d executions of line 32 can be implemented using $O(n)$ number of gates from the Clifford generators. That means the loop at line 30 applies $O(nd)$ number of gates from the Clifford generators.

This accounts for all the gates applied by the block Clifford code. Thus, $O(n) + O(nd) + O(nd) = O(nd)$ gates. □

Corollary 7.2.1. *There exists an implementation of Algorithm 4 which applies $O(nd)$ gates from the Clifford generators.*

Proof. Algorithm 4 applies the same operations as Algorithm 3, except in reverse order and with their conjugate transposes taken. □

7.2.3 Security Analysis

Next, a mathematical analysis of the security of the proposed method is provided. Note that the transformations applied to ρ in Algorithms 3 and 4 are all from the Clifford group. That means, the entire operation acting on ρ can be expressed using

a single element from the Clifford group, call it C (see Theorems 5.7.6 and 5.7.7). The following proofs will be concerned with an adversary that can only perform Paulis on the encrypted data. This simplifies the proofs, as the resulting transformation performed on the data packet will also be a Pauli. As in, with C denoting the Clifford operation described by Algorithm 3, the data packet ρ will be transformed in such a way if the adversary attacks using $A \in P(n)$:

$$C^\dagger A C \rho C^\dagger A^\dagger C = A' \rho A'^\dagger \tag{7.49}$$

where $A' \in P(n)$ by definition of the Clifford group (see Equation (5.39)). The adversary will be detected if and only if A' has at least one X_j or one Y_j gate that is applied to one of the d signature qubits at positions $j \in 0, 1, 2 \cdots d-1$. This can be seen by the following equalities:

$$Z|0\rangle = |0\rangle \tag{7.50}$$

$$X|0\rangle = |1\rangle \tag{7.51}$$

$$Y|0\rangle = i|1\rangle \tag{7.52}$$

and remarking measurement of quantum states in Section 5.10 Also, since the transformation on the data packet is a Pauli, measuring the signature qubits will not disturb the state.

Also note that the proof for the security of a QMA scheme when attacked by an adversary from $P(n)$ can be used to prove the security against any type of attack. See Section 3.1 of reference [25]. This relies on the fact that $P(n)$ forms a basis (see Corollary 5.7.1) which means that any given adversarial attack can be expressed as

a probabilistic mixture of Paulis.

The proofs in this subsection will regularly make use of A_i denoting a single-qubit Pauli an adversary applies to qubit position i. Next, establishing some Lemmas will aid in the security analysis of the proposed method.

Lemma 7.2.1. *Let U_i and V_k be arbitrary single-qubit unitaries. Also let $CU_{i,j}$ be an arbitrary $CU_{i,j} \in \{CX_{i,j}, CY_{i,j}, CZ_{i,j}\}$ and let $CU_{k,j}$ be set to the same type of controlled-U gate as $CU_{i,j}$. The operations $CU_{i,j}$ and U_i commute with any of $CU_{k,j}$ and V_k, for any values of i, j and k where $i \neq j \neq k$.*

Lemma 7.2.2. *Let $A_{i'}$ be a single-qubit adversarial attack applied to a qubit at position i', and let A_i be the same operation but applied to a data qubit position i. When using the block Clifford code, the operation applied to the state ρ as a result of the operations for encrypting, followed by an adversary attacking a single data qubit using $A_{i'}$ and then decrypting, can be simplified in such a way:*

$$U_j CU_{i,j} U_i^\dagger A_i U_i CU_{i,j} U_j \tag{7.53}$$

Next, the probability of detecting a single-qubit adversarial attack is shown. That is, the adversarial attack can take on a value of X_i, Y_i or Z_i for i where $0 \leq i < n$. Afterward, the proofs for the single-qubit attack will be extended to the general case where $A \in P(n)$.

Theorem 7.2.3. *Let the data qubit at position i be encrypted by a unitary U_i. Assume the adversary applies a single-qubit Pauli operation, $A_{i'} \in \{X_{i'}, Y_{i'}, Z_{i'}\}$. Also assume that after $A_{i'}$ is conjugated by all the SWAP operations from the CIRC gates during the encryption and decryption process, it becomes A_i where $d \leq i < n$. If U_i is equal to one of I_i, H_i or $S_i^\dagger H_i$ with equal probability, the adversary will be detected with probability $\frac{2}{3}$.*

Proof. This can be seen by making use of the tables listed in Theorems 5.7.14, 5.7.15, 5.7.16 and 5.7.17. Due to Lemma 7.2.2,

$$U_j CU_{i,j} U_i^\dagger A_i U_i CU_{i,j} U_j$$

is the transformation on the data packet ρ as a result of the adversarial attack. A given $CU_{i,j} \in \{CX_{i,j}, CY_{i,j}, CZ_{i,j}\}$ 'flips' the signature qubit at position j if $U_i^\dagger A_i U_i$ is equal to X_i or Y_i. The transformation is fully described using the following tables:

Table 7.1: A table showing how a single-qubit Pauli attack applied to a data qubit affects the data packet when considering the $CX_{i,j}$ gate.

A_i	$CX_{i,j}A_iCX_{i,j}$	$CX_{i,j}H_iS_iA_iS_i^\dagger H_iCX_{i,j}$	$CX_{i,j}H_iA_iH_iCX_{i,j}$
X_i	X_iX_j	$-Y_iX_j$	Z_i
Y_i	Y_iX_j	$-Z_i$	$-Y_iX_j$
Z_i	Z_i	X_iX_j	X_iX_j

Table 7.2: A table showing how a single-qubit Pauli attack applied to a data qubit affects the data packet when considering the $CY_{i,j}$ gate.

A_i	$H_jCY_{i,j}A_iCY_{i,j}H_j$	$H_jCY_{i,j}H_iS_iA_iS_i^\dagger H_iCY_{i,j}H_j$	$H_jCY_{i,j}H_iA_iH_iCY_{i,j}H_j$
X_i	$-X_iY_j$	Y_iY_j	Z_i
Y_i	$-Y_iY_j$	$-Z_i$	Y_iY_j
Z_i	Z_i	$-X_iY_j$	$-X_iY_j$

Table 7.3: A table showing how a single-qubit Pauli attack applied to a data qubit affects the data packet when considering the $CZ_{i,j}$ gate.

A_i	$H_jS_jCZ_{i,j}A_iCZ_{i,j}S_j^\dagger H_j$	$H_jS_jCZ_{i,j}H_iS_iA_iS_i^\dagger H_iCZ_{i,j}S_j^\dagger H_j$	$H_jS_jCZ_{i,j}H_iA_iH_iCZ_{i,j}S_j^\dagger H_j$
X_i	X_iX_j	$-Y_iX_j$	Z_i
Y_i	Y_iX_j	$-Z_i$	$-Y_iX_j$
Z_i	Z_i	X_iX_j	X_iX_j

In the above tables, each row headers enumerate the possible values for A_i, and all the column headers enumerate the possible encryption operations. In each row of each table, $\frac{2}{3}$ of the resulting transformations on the data packet have an X_j or Y_j, thus indicating an attack with probability $\frac{2}{3}$. □

For some of the following analyses, the 2-dimensional list B will be used to improve conciseness. Denote $B(j,k)$ as the k^{th} qubit position i of a data qubit used as the control qubit for a $CU_{i,j}$ with the target qubit as the signature qubit at position j. In other words, each $B(j,k)$ for all $k \in \{0,1,2\cdots\frac{m}{d}-1\}$ and a fixed j corresponds to an element in k_b'' equal to j (remark, lines 15 and 23 in Algorithms 3 and 4 respectively). For additional convenience, define $B(j)$ as a 'block': The positions of all the data qubits that get $CU_{i,j}$'d with the signature qubit at position j.

Lemma 7.2.3. *When using the block Clifford code, the operation applied to the state ρ due to encrypting, followed by an adversary attacking a single signature qubit using $A_{j'}$ and then decrypting, can be simplified in such a way:*

$$U_j^\dagger CU_{B(j,0),j}\cdots CU_{B(j,\frac{m}{d}-1),j}A_jCU_{B(j,\frac{m}{d}-1),j}\cdots CU_{B(j,0),j}U_j \tag{7.54}$$

Theorem 7.2.4. *Let the signature qubit at position j be encrypted by a unitary U_j. Assume the adversary applies a single-qubit Pauli operation, $A_{j'} \in \{X_{j'}, Y_{j'}, Z_{j'}\}$. Also assume that after $A_{j'}$ is conjugated by all the SWAP operations from the CIRC gates during the encryption and decryption process, it becomes A_j where $0 \leq j < d$. If U_j is equal to one of I_j, H_j or $S_j^\dagger H_j$ with equal probability, the adversary will be detected with probability $\frac{2}{3}$.*

Proof. This can be seen by making use of the tables listed in Theorems 5.7.14, 5.7.15, 5.7.16 and 5.7.17. Additionally, denote A'_k as the single-qubit Pauli applied at position k a part of the resulting transformation $A' = C^\dagger A_j C$. Due to Lemma 7.2.3, the resulting transformation on the data packet can be evaluated as:

$$U_j^\dagger CU_{B(j,0),j} \cdots CU_{B(j,\frac{m}{d}-1),j} A_j CU_{B(j,\frac{m}{d}-1),j} \cdots CU_{B(j,0),j} U_j \tag{7.55}$$

$$= U_j^\dagger CU_{B(j,0),j} \cdots CU_{B(j,\frac{m}{d}-2),j} A'_{B(j,\frac{m}{d}-1)} A_j CU_{B(j,\frac{m}{d}-2),j} \cdots CU_{B(j,0),j} U_j \tag{7.56}$$

$$= A'_{B(j,\frac{m}{d}-1)} U_j^\dagger CU_{B(j,0),j} \cdots CU_{B(j,\frac{m}{d}-2),j} A_j CU_{B(j,\frac{m}{d}-2),j} \cdots CU_{B(j,0),j} U_j \tag{7.57}$$

$$\vdots$$

$$= A'_{B(j,\frac{m}{d}-1)} A'_{B(j,\frac{m}{d}-2)} \cdots A'_{B(j,0)} A_j \tag{7.58}$$

From Equation (7.55) to Equation (7.56) the following simplification is made:

$$CU_{B(j,\frac{m}{d}-1),j} A_j CU_{B(j,\frac{m}{d}-1),j} = A'_{B(j,\frac{m}{d}-1)}$$

and by the previous definition at the start of the proof,

$$A'_k = A'_{B(j,\frac{m}{d}-1)} = CU_{B(j,\frac{m}{d}-1),j} A_j CU_{B(j,\frac{m}{d}-1),j} = C^\dagger A_j C$$

Evaluating Equation (7.58) results in the following transformations on the data

packet due to the adversary's attack:

Table 7.4: A table showing how a single-qubit Pauli attack applied to a signature qubit affects the data packet.

A_j	$CX_{i,j}A_jCX_{i,j}$	$H_jCY_{i,j}A_jCY_{i,j}^\dagger H_j$	$H_jS_jCZ_{i,j}A_jCZ_{i,j}^\dagger S_j^\dagger H_j$
X_j	X_j	$Z_{B(j,\frac{m}{d}-1)}\cdots Z_{B(j,0)}Z_j$	$-Z_{B(j,\frac{m}{d}-1)}\cdots Z_{B(j,0)}Y_j$
Y_j	$Z_{B(j,\frac{m}{d}-1)}\cdots Z_{B(j,0)}Y_j$	$-Y_j$	$-Z_{B(j,\frac{m}{d}-1)}\cdots Z_{B(j,0)}Z_j$
Z_j	$Z_{B(j,\frac{m}{d}-1)}\cdots Z_{B(j,0)}Z_j$	$Z_{B(j,\frac{m}{d}-1)}\cdots Z_{B(j,0)}X_j$	X_j

In the above table, the row headers enumerate the possible values for A_i, and the column headers enumerate the possible encryption operations. In each row of the table, $\frac{2}{3}$ of the resulting transformations on the data packet have an X_j or Y_j, thus indicating an attack with probability $\frac{2}{3}$. □

Corollary 7.2.2. *Let a given signature qubit at position j be acted on by U_j which is equal to I_j, H_j or $S_j^\dagger H_j$ with equal probability. The adversary described in Theorem 7.2.4 has a $\frac{2}{3}$ probability that the resulting transformation on the data packet ρ will contain $\frac{m}{d}$ number of Z_i gates for $i \neq j$.*

Theorem 7.2.5. *There is an injection between the preprocessed subkeys k'_a, k'_b given as input to Algorithms 3 and 4 and the resulting unitary transformations, call them C and $C^\dagger$, given by those algorithms. In other words, each k'_a, k'_b gives a unique C (for encryption) and $C^\dagger$ (for decryption) that are not given by any other possible value of k'_a, k'_b.*

Proof. The injection can be proven by showing that there is an injection between subkeys k'_a, k'_b and the mappings of the Pauli operators to themself when conjugated by the subkeys' corresponding C. This is because if two different Cs give two

different mappings from the Pauli operators to themselves then they must be different unitaries. The proof can be derived in part by the tables in Theorems 7.2.3 and 7.2.4. A proof by case will be used. The first case will show that fixing k'_b to an arbitrary value produces a different C for each different k'_a. The second case will show that two different k'_bs will give two different Cs (regardless of whether or not k'_a is fixed to an arbitrary value).

Case 1: First, fix k'_b to an arbitrary value. It will be shown that each k'_a produces a different C. Each column of each of Tables 7.1, 7.2, 7.3 and 7.4 corresponds to one of the three U_i operators as determined by the i^{th} element in k'_a where $0 \leq i < n$. Let $X_{i'}$ have the same definition as in Theorem 7.2.3, but with the corresponding X_i such that $0 \leq i < n$ (thus $X_{i'}$ can be applied to any qubit position). By only considering the Pauli $X_{i'}$, one can see that each value k'_a can take on produces a different mapping of $X_{i'}$ to $C^\dagger X_{i'} C$. Remark that these tables do not describe the effect that $CIRC_{i,j}$ gates have on the Pauli mapping. This is not an issue however, because k'_b is fixed to an arbitrary value. The conjugation of a single-qubit Pauli by one or more $CIRC_{i,j}$ gates determined by k'_b only changes the position of the Pauli (remark Theorem 5.7.14).

Case 2: Next, it will be shown that two different k'_bs will give two different Cs (regardless of whether or not k'_a is fixed to an arbitrary value). Only the transformation described by Table 7.4 will be considered for this case. This case has two subcases. The first subcase will show that if elements with value d are placed differently in two k'_bs, then that will give two different Cs. The second subcase will show that if elements with some value j such that $0 \leq j < d$ are placed differently in two k'_bs, then that will give two different Cs.

Subcase 1: Each possible combination of placing the elements with value d in k'_b

determines a different permutation of the Paulis $A_{j'} \in \{X_{j'}, Y_{j'}, Z_{j'}\}$ ($A_{j'}$ is defined in Theorem 7.2.4) when conjugated by the $CIRC_{i,j}$ gates a part of the construction of C. Remark that each $A_{j'}$ for a given k'_b is conjugated by a $CIRC_{i,j}$ to a qubit position j where $0 \leq j < d$. Note that each of the single-qubit Paulis A_j in the range of the mapping given by C are each mapped onto by one $A_{j'}$ in the domain. This is because each column in Table 7.4 has one resulting Pauli that does not apply additional Z gates to the data packet. As a result, two given k'_bs with two different placements of elements with value d will give two mappings via each C which have different sets of elements in the domain that map to $A_j \in \{X_j, Y_j, Z_j\}$. Thus, two given k'_bs with different placements of elements with value d will give a different C. Note that the previously mentioned Paulis may also be transformed to a different single-qubit Pauli, due to the U_i gates determined by k'_a, in addition to being moved. However, this is not necessary to take into account: Only the positions that single-qubit Paulis are moved to is required to describe this subcase.

Subcase 2: For the next subcase, remark that the placement of each value j, where $0 \leq j < d$, in k'_b determines which data qubits are used as the control qubit for the signature qubit at position j. Two thirds of the outcomes in Table 7.4 show the placements of each value j in k'_b determine the positions of the $\frac{m}{d}$ number of Z gates applied to the corresponding control qubits. Each row of the table has exactly one resulting Pauli with $\frac{m}{d}$ number of Z gates. Like in subcase 1, considering how the U_i gates change the resulting transformation is not necessary. As in, for two mappings given by keys where one or more of the elements with a value of j are placed in different positions in both of the k'_bs, there exists at least one single-qubit Pauli in the domain that results in a different combination of $\frac{m}{d}$ number of Z gates for each of the two mappings.

Let $P_i \in \{X_i, Y_i, Z_i\}$ where $0 \leq i < n$. Highlighting the two previously described subcases shows there are two different mappings from P_i to $C^\dagger P_i C$ for two pairs of subkeys where k'_b is different in each pair. □

Theorem 7.2.6. *The transformations described by the tables in Theorems 7.2.3 and 7.2.4, which show the resulting transformation applied to the data packet from a single-qubit Pauli attack, can be extended to attacks from $P(n)$.*

Proof. Repeated application of Theorem 5.7.8 proves this. Denote

$$A = A_{n-1} A_{n-2} \cdots A_0 \tag{7.59}$$

where each A_k is the single-qubit Pauli applied at position k a part of the adversarial attack A. Evaluating each $C^\dagger A_k C$ separately via Theorems 7.2.3 and 7.2.4, and then multiplying them all together gives the resulting transformation $C^\dagger A C$. Note that any of A_k may also be equal to I_k, in such a case $C^\dagger I_k C = I$. □

Theorem 7.2.7. *Let each qubit in the data packet be acted on by a U_i from the set $\{I_i, H_i, S_i^\dagger H_i\}$ with equal probabilities. An adversary that attacks using a Pauli composed of w non-identity single-qubit Paulis that are each placed at an arbitrary signature qubit position j or positions $i \in B(j)$ will be detected with probability:*

$$f(w) = \frac{1}{3} f(w-1) + \frac{2}{3}(1 - f(w-1)) \tag{7.60}$$

$$= \frac{1 - \left(\frac{-1}{3}\right)^w}{2} \tag{7.61}$$

with a base case $f(0) = 0$. Additionally, $f(w) \approx \frac{1}{2}$ and $\frac{4}{9} \leq f(w) \leq \frac{2}{3}$.

Proof. First, remark that Theorem 7.2.4 showed that single-qubit Pauli attacks on

a signature qubit result in a X_j, Y_j or Z_j applied to the signature qubit with equal probability. We can see that the base case $f(1) = \frac{2}{3}$, agrees with the results from Theorem 7.2.4. Similarly, Theorem 7.2.3 showed that single-qubit Pauli attacks on a data qubit result in a X_j, Y_j or Z_j applied to the corresponding signature qubit at position j with equal probability. Again, we can see that the base case $f(1) = \frac{2}{3}$, agrees with the results from Theorem 7.2.3. The base case $f(0) = 0$ when there is no adversarial attack also makes sense.

Denote B' as equal to B, excluding the indices of qubits that have not been attacked by a single-qubit Pauli. When considering $w > 1$, use of Theorem 7.2.6 gives Equation (7.60) by evaluating each

$$C^\dagger A_{B'(j,w-1)} C, C^\dagger A_{B'(j,w-2)} C, \cdots, C^\dagger A_{B'(j,0)} C$$

separately to acquire $C^\dagger AC$. Since the adversary is detected if and only if there is an X_j or Y_j in the resulting transformation $C^\dagger AC$ on the data packet, $f(w)$ is the probability the resulting transformation $C^\dagger AC$ has an X_j or Y_j. The first term

$$\frac{1}{3} f(w-1)$$

is the $\frac{1}{3}$ probability that a Z is applied by the w^{th} single-qubit Pauli

$$C^\dagger A_{B'(j,w-1)} C$$

multiplied by the probability that an X or Y is applied by

$$C^\dagger A_{B'(j,w-2)} C C^\dagger A_{B'(j,w-3)} C \cdots C^\dagger A_{B'(j,0)} C.$$

The second term

$$\frac{2}{3}(1 - f(w-1))$$

is the $\frac{2}{3}$ probability that an X or Y is applied by

$$C^\dagger A_{B'(j,w-1)} C$$

multiplied by the probability that a Z is applied by

$$C^\dagger A_{B'(j,w-2)} C C^\dagger A_{B'(j,w-3)} C \cdots C^\dagger A_{B'(j,0)} C.$$

The negation in the second term corresponds to the fact that each X or Y applied to the signature qubit 'flips' it, changing the integrity detection results, while a Z does not alter the integrity detection results. Another way of seeing the recursion is as the probability that all the $C^\dagger A_{B'(j,w-1)} C$, $C^\dagger A_{B'(j,w-2)} C, \cdots, C^\dagger A_{B'(j,0)} C$ have an odd number of X_j and Y_j. This is more precisely seen by repeated application of the following equations:

$$XX = ZZ = YY = I \tag{7.62}$$

$$XY = YX = Z \tag{7.63}$$

$$ZX = XZ = Y \tag{7.64}$$

$$ZY = YZ = X \tag{7.65}$$

where the constants $\{e^{i\theta\pi/2} \mid \theta = 0, 1, 2, 3\}$ are omitted (as they do not have any affect on quantum states).

To show that $f(w) = \frac{1-(\frac{-1}{3})^w}{2}$ an expansion of the recursion reveals its summa-

tion form, which is then simplified using the geometric series summation formula:

$$
\begin{aligned}
f(w) &= \frac{1}{3}f(w-1) + \frac{2}{3}(1 - f(w-1)) && (7.66)\\
&= \frac{-1}{3}f(w-1) + \frac{2}{3} && (7.67)\\
&= \frac{-1}{3}(\frac{-1}{3}f(w-2) + \frac{2}{3}) + \frac{2}{3} && (7.68)\\
&= \frac{1}{3^2}f(w-2) - \frac{2}{3^2} + \frac{2}{3} && (7.69)\\
&= \frac{-1}{3^3}f(w-3) + \frac{2}{3^3} - \frac{2}{3^2} + \frac{2}{3} && (7.70)\\
&\quad \vdots && (7.71)\\
&= \frac{-1}{3^w}f(w-w) + \frac{2}{3^w} - \frac{2}{3^{w-1}} + \frac{2}{3^{w-2}} \cdots + \frac{2}{3} && (7.72)\\
&= \sum_{i=1}^{w}(-1)^{i+1}\frac{2}{3^i} && (7.73)\\
&= \sum_{i=1}^{w} -2\frac{(-1)^i}{3^i} && (7.74)\\
&= \sum_{i=1}^{w} -2\left(\frac{-1}{3}\right)^i && (7.75)\\
&= \left(\sum_{i=0}^{w} -2\left(\frac{-1}{3}\right)^i\right) + 2 && (7.76)\\
&= -2\left(\frac{1 - (\frac{-1}{3})^{w+1}}{1 - (\frac{-1}{3})}\right) + 2 && (7.77)\\
&= \frac{1 - (\frac{-1}{3})^w}{2} && (7.78)
\end{aligned}
$$

In Equation (7.78) the term $(\frac{-1}{3})^w$ becomes vanishingly small as w increases. This is expressed more precisely by $|\frac{1-(\frac{-1}{3})^{w+1}}{2} - \frac{1}{2}| < |\frac{1-(\frac{-1}{3})^w}{2} - \frac{1}{2}|$. As a result, $f(w) \approx \frac{1}{2}$ and $\frac{4}{9} \leq f(w) \leq \frac{2}{3}$ for $w \geq 1$. □

To provide a formula for the security of the block Clifford code, one must

consider how many non-identity single-qubit Paulis the adversary's attack A is composed of. Keeping this in mind, the following analyses will lead to an approximation for the probability of detection given the adversary can attack using an arbitrary Pauli from $P(n)$.

Corollary 7.2.3. *Given an adversary applies at least one non-identity single-qubit Pauli to each of x blocks, the probability of detection, p_d, is:*

$$1 - \left(1 - \frac{4}{9}\right)^x \leq p_d \approx 1 - \left(\frac{1}{2}\right)^x \leq 1 - \left(1 - \frac{6}{9}\right)^x \tag{7.79}$$

Proof. Theorem 7.2.7 showed that if a Pauli A has all w of its non-identity single-qubit Paulis applied to a single block of qubits, the probability of the block's corresponding signature qubit being flipped and thus indicating a detection is $\frac{4}{9} \leq f(w) \approx \frac{1}{2} \leq \frac{6}{9}$. That means there is a probability less than $(1 - \frac{4}{9})^x$, approximately $(\frac{1}{2})^x$ and greater than $(1 - \frac{6}{9})^x$ that none of the x signature qubits are flipped. The negation of this gives the probability that at least one of them is flipped, thus detecting an attack. □

The 'balls into bins' and 'urn problem' are both closely related to the analysis of the block Clifford code's security for the more general case $A \in P(n)$ [31, 32]. Let the adversary's attack $A \in P(n)$ be composed of $0 < w \leq n$ non-identity single-qubit Paulis. Framing the analysis in the terminology of the analogous 'urn problem', each block $B(j)$ is an urn that contains $\frac{n}{d}$ balls with the colour j. Each non-identity single-qubit Pauli corresponds to a single removal of one of the balls from the urn. The number of distinctly coloured balls removed from the jar after sequentially removing w balls corresponds to the number of blocks the adversary applies a non-identity single-qubit Pauli to. The urn problem with multiple coloured balls

models the multivariate hypergeometric distribution (sampling with replacement).

To make the analysis simpler and more compact, an approximation is given for the probability of detection making use of the 'balls into bins' analogy (sampling without replacement).

Theorem 7.2.8. *The probability of detecting an attack $A \in P(n)$ composed of $0 < w \leq n$ non-identity single-qubit Paulis, when using d signature qubits is approximately:*

$$g(w,d) = \sum_{x=1}^{\min(d,w)} \left(1 - \left(\frac{1}{2}\right)^x\right) h(x,w,d) \tag{7.80}$$

where

$$h(x,w,d) = \binom{d}{x} \left(\sum_{i=0}^{x} (-1)^i \binom{x}{i} \left(\frac{x-i}{d}\right)^w\right) \tag{7.81}$$

Proof. The theorem makes use of the Stirling number of the second kind, $\{ {w \atop x} \}$, which calculates the number of ways to partition w distinct elements (the single-qubit Pauli attacks) into x non-empty sets (the blocks that are attacked) [33]. Thus, the problem is modeled using a multinomial distribution (balls into bins) instead of the hypergeometric multivariate distribution (urn problem). This entails an approximation, as the two different distributions converge to equal each other [34]. Note that this approximation is more accurate for small values of w. For larger values of w, it is sufficient to say that this approximation will prompt the probability of detection to be lower than the exact value because sampling without replacement would encourage a given single-qubit Pauli to be applied to a block that hasn't been attacked by another single-qubit Pauli (this will become more apparent by the end of the proof). First Equation (7.81) will be explained followed by Equation (7.80).

Equation (7.82) simplifies to equal Equation (7.81). The numerator of the fraction in Equation (7.82) consists of the Stirling number of the second kind multiplied by $x!$. Multiplying by $x!$ ensures the number of permutations of each of the $\left\{ {w \atop x} \right\}$ partitions are counted. The denominator is d^w, which counts the total number of ways to apply w single-qubit Paulis (the 'balls') to qubits in d blocks (the 'bins'). The numerator and denominator together is the probability of a particular selection of x blocks each being attacked by at least one of the w single-qubit attacks. Finally, this fraction is multiplied by $\binom{d}{x}$ to count the number of combinations of x of the d blocks there are. Thus, simplifying this out:

$$h(x,w,d) = \binom{d}{x} \frac{\left\{ {w \atop x} \right\} x!}{d^w} \tag{7.82}$$

$$= \binom{d}{x} \frac{\left(\frac{1}{x!} \sum_{i=0}^{x} (-1)^i \binom{x}{i} (x-i)^w \right) x!}{d^w} \tag{7.83}$$

$$= \binom{d}{x} \frac{\sum_{i=0}^{x} (-1)^i \binom{x}{i} (x-i)^w}{d^w} \tag{7.84}$$

$$= \binom{d}{x} \left(\sum_{i=0}^{x} (-1)^i \binom{x}{i} \left(\frac{x-i}{d} \right)^w \right) \tag{7.85}$$

In Equation (7.80), $1 - \left(\frac{1}{2}\right)^x$ is the approximation for the probability of detecting an adversary given they have applied at least one single-qubit Pauli to each of x blocks (shown in Corollary 7.2.3). Equation (7.81) is the probability of an adversary attacking x blocks, as a function of the w number of single-qubit Pauli attacks the adversary's attack is composed of. Thus, multiplying these two terms together and summing over all possible values of $x = 1, 2, \cdots \min(d, w)$ gives the approximation for the probability of detection. □

Next, the approximate probabilities of detection for the block Clifford code

(Equation (7.80)) and the Clifford code (Equation (6.1)) will be compared.

Theorem 7.2.9. *The approximate probability of detection for the block Clifford code is less than or equal to the approximate probability of detection for the Clifford code.*

Proof. Taking Equations (7.80) and (6.1) for the block Clifford code and the Clifford code respectively gives:

$$\sum_{x=1}^{\min(d,w)} \left(1 - \left(\frac{1}{2}\right)^x\right) \binom{d}{x} \frac{\left\{ {w \atop x} \right\} x!}{d^w} \leq 1 - \left(\frac{1}{2}\right)^d \tag{7.86}$$

As described in Theorem 7.2.8, the function given by Equation (7.81) gives the probability of exactly x number of blocks each having at least one non-identity single-qubit applied to them. Thus, evaluating Equation (7.81) for all possible values of $x \in \{1, 2, \cdots, \min(d, w)\}$ should be equal to 1, because the probabilities of all possible outcomes must sum to 1. As a result, Equation (7.80) can be seen as a weighted sum of $1 - \left(\frac{1}{2}\right)^x$ for all possible values of x, where the weights sum to equal 1. Remark that x is at most equal to d. That means that:

$$1 - \left(\frac{1}{2}\right)^x \leq 1 - \left(\frac{1}{2}\right)^d$$

for $x \leq d$. Therefore Equation (7.86) holds true. □

Theorem 7.2.10. *The approximate probability of detection for the block Clifford code approaches the approximate probability of detection for the Clifford code as w increases.*

Proof. This can be seen by referring back to the 'balls into bins' analogy. Remark that the non-identity single-qubit Paulis are the 'balls' and the blocks are the 'bins'. As the number of balls, w, thrown into the bins increases, the probability of only

a small number of bins having at least 1 ball in them becomes smaller. In other terms, the probability distribution described by Equation (7.81) evaluated for all possible values of x becomes more heavily weighted towards high values of x, as w increases.

This can be explained more formally by making use of the asymptotic approximation [35]:

$$\left\{ {w \atop x} \right\} \approx \frac{x^w}{x!} \tag{7.87}$$

for a fixed x and increasing w. Substituting this into Equation (7.80) gives:

$$\sum_{x=1}^{\min(d,w)} \left(1 - \left(\frac{1}{2}\right)^x\right) \binom{d}{x} \frac{\frac{x^w}{x!} x!}{d^w} = \sum_{x=1}^{\min(d,w)} \left(1 - \left(\frac{1}{2}\right)^x\right) \binom{d}{x} \left(\frac{x}{d}\right)^w \tag{7.88}$$

and the term $\left(\frac{x}{d}\right)^w$ approaches 0, for all $x \neq d$, as w increases. Therefore Equation (7.88) approaches $1 - \left(\frac{1}{2}\right)^d$ as w increases. □

7.2.4 Key Mapping

Now, the key mapping for the block Clifford code will be explained in detail along with a time complexity analysis. Typically time complexity analyses consider arithmetic operations as $O(1)$. This is a good-enough approximation in most instances, but the key mapping for the block Clifford code requires multiplication and division with very large numbers, thus $O(1)$ is not appropriate for these operations. The analyses will make this clear.

The key mapping takes as a input a binary string, k and splits it into two 'subkeys', call them k_a and k_b, meaning that $k = k_a, k_b$. Remark that subkey k_a

specifies the single-qubit unitary operations applied to both the data qubits and the signature qubits. The other subkey, k_b, specifies the permutation of signature qubits and which signature qubit j is the target for each data qubit at position i for each $CU_{i,j}$. When talking about binary, ternary and decimal numbers in this thesis, digits will be indexed from right to left starting at 0 (just like how qubits are indexed in this thesis).

The subkey k_a mapping, described using pseudocode in Algorithm 5, determines which single-qubit unitary gates to apply by converting the binary representation of the subkey to a ternary number of length n, call it k'_a. The conversion from k_a to k'_a is done via a base conversion [36]. There is an additional step at line 10 in Algorithm 5 which removes the n^{th} digit from the ternary number representation if it is length $n+1$ (contains $n+1$ digits). This means that if line 10 were commented out (meaning it would no longer execute), then $k_a = k'_a$ when interpreted in their respective bases. Since the subkey k_a specifies 3^n sets of operations, it must consist of $\lceil \log_2(3^n) \rceil = \lceil n \log_2(3) \rceil \approx \lceil 1.585n \rceil$ bits to express all possible 3^n gate combinations. This also means that, depending on the value of n, some bit strings of length $\lceil n \log_2(3) \rceil$ may cause Algorithm 5 to execute line 10. In particular, any bit string k_a representing a decimal number greater than $3^n - 1$ will result in a k'_a with $n+1$ digits prior to executing line 10.

Example 7.2.5. *If $n = 2$, k_a will be specified using $\lceil \log_2(3^2) \rceil = 4$ bits. The bit string* 1001 *when converted to ternary is* 100*, which is length $n+1$. The bit string* 1000 *when converted to ternary is* 22*, which is length n.*

As a result of using a base conversion method, the subkey space will have some 'collisions', meaning that for a given k_a, there will be at most one other subkey that specifies the same operations to apply via Algorithm 5. Again, this isn't an issue as

there is a very large number of values k_a can take on. In the case that there are no collisions, the resulting set of k_a's will give a perfectly uniform selection of each of the U_i gates, thus satisfying the security requirements laid out in Theorems 7.2.3 and 7.2.4. There is a negligible effect on the uniformity of the selected gates if there are collisions. If someone implementing the block Clifford code wants to be extra-diligent, they may opt to exclude each k_a greater than $3^n - 1$ from the underlying security protocol to remove all collisions.

Theorem 7.2.11. *For a given k_a, there will be at most one other subkey that specifies the same operations to apply via Algorithm 5.*

Proof. In the worst case, $2(3^n) - 2 = 2^{\lceil \log_2(3^n) \rceil}$. This occurs when the n-digit ternary number with n number of 2s is equal to a $\lceil \log_2(3^n) \rceil$-digit binary number with a single 1 followed by $\lceil \log_2(3^n) \rceil - 1$ number of 0s. This means the first $2^{\lceil \log_2(3^n) \rceil - 1} + 1$ binary numbers that k_a can take on specify all 3^n ternary numbers, and the next $2^{\lceil n \log_2(3) \rceil - 1} - 1$ binary numbers specify each of the first $3^n - 2$ ternary numbers again. See the following:

$$\underbrace{2,2,2,\cdots,2,2,2}_{n \text{ times}} = 1,\underbrace{0,0,0,\cdots,0,0,0}_{\lceil \log_2(3^n) \rceil - 1 \text{ times}} \tag{7.89}$$

$$1,\underbrace{0,0,0,\cdots,0,0,0}_{n \text{ times}} = 1,\underbrace{0,0,0,\cdots,0,0,0}_{\lceil \log_2(3^n) \rceil - 2 \text{ times}},1 \tag{7.90}$$

$$\vdots$$

$$1,\underbrace{2,2,2,\cdots,2,2,2}_{n-2 \text{ times}},1,2 = \underbrace{1,1,1,\cdots,1,1,1}_{\lceil \log_2(3^n) - 1 \rceil \text{ times}}0 \tag{7.91}$$

$$1,\underbrace{2,2,2,\cdots,2,2,2}_{n-1 \text{ times}},0 = \underbrace{1,1,1,\cdots,1,1,1}_{\lceil \log_2(3^n) \rceil \text{ times}} \tag{7.92}$$

□

Although there may be faster methods for converting a number from binary to ternary [37], Algorithm 5 can be used for converting k_a to ternary. Algorithm 5 is a standard way to perform base conversion, which involves successive divisions with remainder [36]. On line 3 of Algorithm 5, *divmod* denotes division with remainder, also known as Euclidean division [4]. Here the subkey k_a is the dividend and 3 is the divisor, which returns q as the quotient and r as the remainder. The same *divmod* method will be used for the other subkey mapping. Line 5 denotes prepending a single ternary digit onto the ternary string k'_a, which determines which single-qubit gates are applied to which qubit (remark Algorithms 3 and 4). In Algorithm 5, k'_a is referred to as a 'ternary string' instead of a 'ternary number' as the prepend operation is typically associated with strings in most computer languages. In practical terms, the results of the algorithm are the same (whether or not it returns a string or a number). Line 8 ensures that the resulting ternary string is at least length n.

Algorithm 5 Subkey k_a Mapping(k_a,n)

1: initialize k'_a as empty ternary string
2: **while** $k_a \neq 0$ **do**
3: $\quad q, r \Longleftarrow divmod(k_a, 3)$
4: $\quad k_a \Longleftarrow q$
5: $\quad k'_a.prepend(r)$
6: **end while**
7: **if** $length\_of(k'_a) < n$ **then**
8: $\quad k'_a.prepend(\underbrace{0,0,0,\cdots 0,0}_{n-length\_of(k'_a)})$
9: **else if** $length\_of(k'_a) = n+1$ **then**
10: $\quad k'_a.pop(0)$
11: **end if**
12: **return** k'_a

The main while loop of Algorithm 5 will iterate at most $\log_3(3^n) + 1 = n + 1$ times, where n denotes the number of qubits in the data packet ρ. The division in the while loop at line 3 will execute in $O\left(D(\lceil n \log_2(3) \rceil)\right)$ time, where $D(n)$ specifies the time complexity to divide an at most n-bit number by an at most n-bit number. Asymptotically, the fastest method for dividing two numbers is to take the reciprocal of the divisor, and then perform multiplication. This method takes $O(M(n))$ time, where $M(n)$ specifies the time complexity to multiply two n-bit numbers [38, 39]. The asymptotically fastest multiplication algorithm takes $M(n) = O(n \log_2(n))$ time [40]. Utilizing the asymptotically fastest method of division, each iteration of the while loop takes:

$$O\left(D(\lceil n \log_2(3) \rceil)\right) = O\left(\lceil n \log_2(3) \rceil \log_2 \lceil n \log_2(3) \rceil\right) \tag{7.93}$$

$$= O(n \log_2(n)) \tag{7.94}$$

time, and with the loop iterating at most $n + 1$ times, Algorithm 5 takes:

$$(n+1)(O(n \log_2(n))) = O\left(n^2 \log_2(n)\right) \tag{7.95}$$

time.

The mapping for subkey k_b makes use of the unranking algorithm for the combinatorial number system [41]. First, the unranking algorithm will be explained, followed by how it is used for the subkey k_b mapping. Remark that an unranking function is a bijective function which maps a given integer $i \in \{0, 1, \cdots |S| - 1\}$ to a corresponding element $s \in S$, where S is some defined defined set [18]. For the combinatorial number system unranking function, each s is a k-combination from the set S of all $|S| = \binom{n}{k}$ combinations. The combinatorial number system is used to

map an index N where $0 \leq N < \binom{n}{k}$ to a unique k-combination. The combinatorial number system relies on the following unique representation for each N [42]:

$$N = \binom{c_k}{k} + \cdots + \binom{c_2}{2} + \binom{c_1}{1} \tag{7.96}$$

where each $c_{i+1} > c_i$. The element c_i denotes the index of a chosen element in the k-combination. A greedy algorithm can be used to efficiently map an integer N to its corresponding combination. It essentially works by finding the largest c_k such that $\binom{c_k}{k} \leq N$, then recursively repeats the process to find the largest c_{k-1} such that $\binom{c_{k-1}}{k-1} \leq N - \binom{c_k}{k}$, largest c_{k-2} such that $\binom{c_{k-2}}{k-2} \leq N - \binom{c_k}{k} - \binom{c_{k-1}}{k-1}$, etc. until each c_i has been chosen. The bulk of the computation in this algorithm is from the calculation of each $\binom{m}{j}$ where $0 \leq m \leq n$ and $0 < j \leq k$. Algorithm 7 describes the greedy algorithm for implementing the combinatorial number system unranking function, which requires calculating at most $2n$ number of $\binom{m}{j}$s.

Theorem 7.2.12. *When making the same assumptions for multiplication and division operations as in Algorithm 5, an efficient algorithm for computing factorials, [43], allows calculating* $2n$ *number of* $\binom{m}{j}$*s in* $O\left(n^2 \log_2^2(n) \log_2(\log_2(n))\right)$ *time.*

Proof. The algorithm presented in [43] shows that calculating $n!$ can be computed in $O\left(\log_2(\log_2(n))M(n\log_2(n))\right)$ time. Subsequently, that means that $\binom{n}{k}$ can be calculated in the same amount of time, as it requires calculating three factorials all at most $n!$ in size, followed by two division operations, each costing $O\left(M(n\log_2(n))\right)$.

Calculating the $2n$ number of $\binom{m}{j}$s would then result in a time complexity of:

$$O\left(n\log_2(\log_2(n))M(n\log_2(n))\right) \tag{7.97}$$

$$=O\left(n\log_2(\log_2(n))n\log_2(n)\log_2(n\log_2(n))\right) \tag{7.98}$$

$$=O\left(n\log_2(\log_2(n))n\log_2(n)(\log_2(n)+\log_2(\log_2(n)))\right) \tag{7.99}$$

$$=O\left(n\log_2(\log_2(n))(n\log_2^2(n)+n\log_2(n)\log_2(\log_2(n)))\right) \tag{7.100}$$

$$=O\left(n^2\log_2^2(n)\log_2(\log_2(n))+n^2\log_2(n)(\log_2(\log_2(n)))^2\right) \tag{7.101}$$

$$=O\left(n^2\log_2^2(n)\log_2(\log_2(n))\right) \tag{7.102}$$

□

A more efficient algorithm allows incrementally calculating each $\binom{m}{j}$ by reusing the previously calculated $\binom{m}{j}$, bringing the cost of calculating the $2n$ number of $\binom{m}{j}$s to:

$$O\left(nM(k\log_2(n))\right) = O\left(nk\log_2(n)(\log_2(k\log_2(n)))\right) \tag{7.103}$$

$$= O\left(nk\log_2(n)(\log_2(k)+\log_2(\log_2(n)))\right) \tag{7.104}$$

Each time a new $\binom{m}{j}$ is calculated, it only differs from the previously calculated $\binom{m}{j}$ by one multiplication operation and one division operation (see Algorithm 7). Algorithm 6 shows how to calculate each successive $\binom{m}{j}$ in $O\left(M(k\log_2 n)\right)$ time. Let n' and k' denote the previously calculated $\binom{m}{j}$'s coefficients such that $m = n'$ and $j = k'$. Additionally, define $n\_choose\_k' = \binom{n'}{k'}$. Algorithm 7 uses Algorithm 6 as a subroutine on lines 12, 24 and 27.

Algorithm 6 Quick N Choose K(n,k,n',k',n_choose_k')

1: $n\_choose\_k \Longleftarrow 0$
2: **if** $n\_choose\_k' = 0$ **then**
3: **if** $n = k$ **then**
4: $n\_choose\_k \Longleftarrow 1$
5: **else**
6: $n\_choose\_k \Longleftarrow 0$
7: **end if**
8: **else if** $n - n' = 1$ **and** $k' = k$ **then**
9: $n\_choose\_k \Longleftarrow \frac{(n)(n\_choose\_k')}{n'-k'+1}$
10: **else if** $n' - n = 1$ **and** $k' = k$ **then**
11: $n\_choose\_k \Longleftarrow \frac{(n'-k')(n\_choose\_k')}{n'}$
12: **else if** $n' - n = 1$ **and** $k' - k = 1$ **then**
13: $n\_choose\_k \Longleftarrow \frac{(k')(n\_choose\_k')}{n'}$
14: **end if**
15: **return** n_choose_k

Algorithm 6 relies on the following equalities:

$$\binom{n}{k} = \frac{n\binom{n-1}{k}}{n-k} \tag{7.105}$$

$$= \frac{(n-k+1)\binom{n+1}{k}}{n+1} \tag{7.106}$$

$$= \frac{(k+1)\binom{n+1}{k+1}}{n+1} \tag{7.107}$$

where Equation (7.105) corresponds to line 9, Equation (7.106) to line 11 and Equation (7.107) to line 13 of Algorithm 6.

Theorem 7.2.13. *Each execution of Algorithm 6 takes*

$$O\left(M(k\log_2(n))\right) = O\left(k\log_2(n)(\log_2(k) + \log_2(\log_2(n)))\right)$$

time.

Proof. Each execution has at most one division and one multiplication operation, each involving numbers with at most $\log_2(\binom{n}{k})$ number of bits.

$$\log_2\left(\binom{n}{k}\right) < \log_2(n^k) \tag{7.108}$$

$$= k\log_2(n) \tag{7.109}$$

$$\tag{7.110}$$

Assuming multiplication and division cost $n\log_2 n$ where n is the number of bits required to represent the largest of the 2 numbers, a single execution of Algorithm 6 costs:

$$2(M(k\log_2(n)) = 2(k\log_2(n)\log_2(k\log_2(n))) \tag{7.111}$$

$$= 2(k\log_2(n)(\log_2(k) + \log_2(\log_2(n)))) \tag{7.112}$$

$$= O\left(k\log_2(n)(\log_2(k) + \log_2(\log_2(n)))\right) \tag{7.113}$$

□

Algorithm 7 Combinatorial Number System Unranking(n,k,N)

1: $combination \Longleftarrow ()$
2: $c_i \Longleftarrow k - 1$
3: $i \Longleftarrow k$
4: $c_i\_choose\_i \Longleftarrow 0$
5: $c_i\_choose\_i' \Longleftarrow c_i\_choose\_i$
6: **while** $c_i\_choose\_i \leq N$ **do**
7: $\quad c_i\_choose\_i' \Longleftarrow c_i\_choose\_i$
8: $\quad$ **if** $c_i\_choose\_i = N$ **then**
9: $\quad\quad c_i \Longleftarrow c_i + 1$
10: $\quad\quad$ **break**
11: $\quad$ **end if**
12: $\quad c_i\_choose\_i \Longleftarrow$ Quick N Choose K(n,k,n',k',n_choose_k')
13: $\quad c_i \Longleftarrow c_i + 1$
14: **end while**
15: $c_i\_choose\_i \Longleftarrow c_i\_choose\_i'$
16: $N \Longleftarrow N - c_i\_choose\_i$
17: $c_i \Longleftarrow c_i - 1$
18: $combination.append(c_i)$
19: **for each** $i \in (k-1, k-2 \cdots 1)$ **do**
20: $\quad$ **if** N = 0 **then**
21: $\quad\quad combination.append(i-1)$
22: $\quad$ **else**
23: $\quad\quad c_i \Longleftarrow c_i - 1$
24: $\quad\quad c_i\_choose\_i \Longleftarrow$ Quick N Choose K(n,k,n',k',n_choose_k')
25: $\quad\quad$ **while** $c_i\_choose\_i > N$ **do**
26: $\quad\quad\quad c_i \Longleftarrow c_i - 1$
27: $\quad\quad\quad c_i\_choose\_i \Longleftarrow$ Quick N Choose K(n,k,n',k',n_choose_k')
28: $\quad\quad$ **end while**
29: $\quad\quad N \Longleftarrow N - c_i\_choose\_i$
30: $\quad\quad combination.append(c_i)$
31: $\quad$ **end if**
32: **end for**
33: **return** *combination*

Theorem 7.2.14. *Algorithm 7 takes* $O\left(nk \log_2(n)(\log_2(k) + \log_2(\log_2(n)))\right)$ *time.*

Proof. The most expensive lines in Algorithm 7 are calls to Algorithm 6 on lines 12,

24 and 27. Theorem 7.2.13 showed that Algorithm 6 runs in

$$O\left(k\log_2(n)(\log_2(k)+\log_2(\log_2(n)))\right)$$

time. Careful inspection of Algorithm 7 reveals that Algorithm 6 is called at most $n-k+2$ times in the first loop and at most $n-1$ times in the second loop. This can be seen by how in the worst case c_i sequentially takes on values from $(k, k+1, \cdots, n)$ followed by sequentially taking on values from $(n-2, n-3, \cdots, 1)$ when being used for a call to Algorithm 6. Since $1 \leq k \leq n$, Algoritm 6 is called at most $(n-k+1)+(n-2) = 2n-k-1 \leq 2(n-1)$ times. Thus, the run time of Algorithm 7 is:

$$2(n-1)(O\left(k\log_2(n)(\log_2(k)+\log_2(\log_2(n)))\right)) \tag{7.114}$$

$$=O\left(nk\log_2(n)(\log_2(k)+\log_2(\log_2(n)))\right) \tag{7.115}$$

□

Algorithm 7 is used in Algorithm 8. Next, the pseudocode for Algorithm 8 is presented followed by an explanation and analysis.

Algorithm 8 Subkey k_b Mapping(n,d,k_b)

1: $k'_b \Longleftarrow (\underbrace{d, d, \cdots d}_{n \text{ times}})$
2: $unpicked\_elements = (0, 1, 2, \cdots, n-1)$
3: $m = n - d$
4: **for each** $i \in (0, 1, 2, \cdots, d-1)$ **do**
5: $\quad q, r \Longleftarrow divmod(k_b, \binom{n-i\frac{m}{d}}{\frac{m}{d}})$
6: $\quad k_b \Longleftarrow q$
7: $\quad combination =$ Combinatorial Number System Unranking($n - i\frac{m}{d}$,$\frac{m}{d}$,r)
8: $\quad$ **for each** $element \in combination$ **do**
9: $\quad\quad k'_b(unpicked\_elements.pop(element)) \Longleftarrow i$
10: $\quad$ **end for**
11: **end for**
12: **return** k'_b

Remark that Algorithm 8 takes as input a subkey k_b and maps that subkey to a permutation, k'_b, of the multiset:

$$\{\underbrace{0, 0, \cdots 0}_{\frac{m}{d} \text{ times}} \underbrace{1, 1, \cdots 1}_{\frac{m}{d} \text{ times}} \cdots \underbrace{d-1, d-1, \cdots d-1}_{\frac{m}{d} \text{ times}} \underbrace{d, d, \cdots d}_{d \text{ times}}\} \tag{7.116}$$

It generates a given permutation by applying the combinatorial number system unranking d times, once for each block of qubits (see how line 7 executes Algorithm 7). The output of Algorithm 8 is a decimal string where each value i, where $0 \leq i < d$, determines which block each data qubit is associated with, and each value of $i = d$ denotes the position a signature qubit is moved to (recall Algorithms 3 and 4 for further details). The input, k_b, is in the range

$$0 \leq k_b < \binom{n}{\frac{m}{d}} \binom{n - \frac{m}{d}}{\frac{m}{d}} \binom{n - 2\frac{m}{d}}{\frac{m}{d}} \cdots \binom{\frac{m}{d} + d}{\frac{m}{d}}$$

The input k_b is converted to a d-digit number with variable-sized base where the

value of each digit at position i is at most $\binom{n-i\frac{m}{d}}{\frac{m}{d}} - 1$. The method of base conversion uses the same technique as in Algorithm 5 by successively applying Euclidean division to 'extract' the value at each digit (see line 5 of Algorithm 8). The value and index of each digit in the resulting variable-sized base number correspond to a set of inputs for an execution of the combinatorial number system algorithm (recall line 7).

Each output of the combinatorial number system algorithm determines the locations of all the elements of some value a part of the multiset. For instance, the first combinatorial number system algorithm call determines the positions of all $\frac{m}{d}$ of the 0s, followed by the next call determining the positions of the $\frac{m}{d}$ number of 1s out of the remaining $n - \frac{m}{d}$ positions, etc. Line 9 shows the values from an output of Algorithm 7 being used to assign values to the output of Algorithm 8. The $pop(x)$ method held by the *unpicked_elements* list denotes removing and returning the element at position x in the list. An efficient tree-based data structure may be used to represent the list, enabling $pop(x)$ to run in $O(\log_2 n)$ time. To avoid having to 'reinvent the wheel', the Sorted Containers Python library is used to represent such a data structure in my implementation of Algorithm 8 [44]. The *unpicked_elements* list ensures that values in the output k'_b are only ever assigned at most once. Note that k'_b is initialized with its values all set to d for convenience because line 9 executes m times, making the remaining unassigned elements in k'_b default to a value of d.

Theorem 7.2.15. *Algorithm 8 runs in* $O\left(n^2 \log_2^2(n)\right)$ *time.*

Proof. Theorem 7.2.14 showed that Algorithm 7 runs in

$$O\left(nk \log_2(n)(\log_2(k) + \log_2(\log_2(n)))\right)$$

time. Algorithm 8 calls Algorithm 7 as a subroutine d times, each time with values at most n and $\frac{m}{d}$ for input parameters n and k respectively. This results in a run time of:

$$d\left(O\left(n\frac{m}{d}\log_2(n)\left(\log_2(\frac{m}{d})+\log_2(\log_2(n))\right)\right)\right)=O\left(n^2\log_2^2(n)\right) \tag{7.117}$$

The other main computation in Algorithm 8 is the base conversion on line 5. This results in d division operations, each with numbers involving at most $\log_2(n!)$ number of bits:

$$\log_2(k_b)=\log_2\left(\binom{n}{\frac{m}{d}}\binom{n-\frac{m}{d}}{\frac{m}{d}}\binom{n-2\frac{m}{d}}{\frac{m}{d}}\cdots\binom{\frac{m}{d}+d}{\frac{m}{d}}\right) \tag{7.118}$$

$$=\log_2\left(\left(\frac{n(n-1)\cdots n-\frac{m}{d}+1}{\frac{m}{d}!}\right)\left(\frac{(n-\frac{m}{d})(n-\frac{m}{d}-1)\cdots}{\frac{m}{d}!}\cdots\right)\right) \tag{7.119}$$

$$<\log_2(n!) \tag{7.120}$$

This results in a run time of:

$$O\left(d(M(\log_2(n!)))\right)<O\left(d(M(n\log_2(n)))\right) \tag{7.121}$$

$$=O\left(d(M(n\log_2(n)))\right) \tag{7.122}$$

$$=O\left(dn\log_2(n)\log_2(n\log_2(n))\right) \tag{7.123}$$

$$=O\left(dn\log_2(n)(\log_2(n)+\log_2(\log_2(n)))\right) \tag{7.124}$$

$$=O\left(dn\log_2^2(n)\right) \tag{7.125}$$

Adding together the two time complexities in Equations (7.117) and (7.125) concludes the proof. □

This idea of 'variable-base numbers' seems to be uncommon within the broader scope of computer science and mathematics. The only other known usage of variable-base numbers I know of is with the Lehmer code and its similar variants [19]. As a result, I made sure that the proof of correctness for Algorithm 8 is elaborate.

Theorem 7.2.16. *Algorithm 8 is correct.*

Proof. To prove its correctness, it must be shown that every possible permutation is mapped onto by a value from $0 \leq k_b < \binom{n}{\frac{m}{d}}\binom{n-\frac{m}{d}}{\frac{m}{d}}\binom{n-2\frac{m}{d}}{\frac{m}{d}}\cdots\binom{\frac{m}{d}+d}{\frac{m}{d}}$. Remark that for determining the positions of the elements with the first value, all the 0s, there are $\binom{n}{\frac{m}{d}}$ possible ways to place them. For determining the positions of the elements with a value of i, there are $\binom{n-i\frac{m}{d}}{\frac{m}{d}}$ possible ways to place them. Each of $\binom{n-i\frac{m}{d}}{\frac{m}{d}}$ corresponds to the number of different values a digit at position i in the variable-base number can take on.

I will define a few additional variables to prove the correctness of the algorithm. Denote q_i as an integer such that:

$$q_i \in \{0, 1, \cdots, \left(\binom{n-i\frac{m}{d}}{\frac{m}{d}}\binom{n-(i+1)\frac{m}{d}}{\frac{m}{d}}\cdots\binom{\frac{m}{d}+d}{\frac{m}{d}}\right)-1\} \tag{7.126}$$

Also let r_i be the remainder and q_i be the dividend when performing Euclidean division on line 5 of Algorithm 8. A given execution of line 5 rewritten using this new notation,

$$q_{i+1}, r_i \Longleftarrow divmod\left(q_i, \binom{n-i\frac{m}{d}}{\frac{m}{d}}\right) \tag{7.127}$$

gives the following:

$$q_{i+1} \in \{0, 1, \cdots, \left(\binom{n-(i+1)\frac{m}{d}}{\frac{m}{d}} \binom{n-(i+2)\frac{m}{d}}{\frac{m}{d}} \cdots \binom{\frac{m}{d}+d}{\frac{m}{d}} \right) - 1\} \quad (7.128)$$

and

$$r_i \in \{0, 1, \cdots, \binom{n-i\frac{m}{d}}{\frac{m}{d}} - 1\} \quad (7.129)$$

Additionally, let $R_i = \{0, 1, \cdots, \binom{n-i\frac{m}{d}}{\frac{m}{d}} - 1\}$, meaning $r_i \in R_i$. Remark that using Euclidean division means that

$$q_i = q_{i+1} \binom{n-i\frac{m}{d}}{\frac{m}{d}} + r_i \quad (7.130)$$

Rewriting Equation (7.130) by substituting q_{i+1} and r_i for their maximum values, as previously defined, gives:

$$q_i = \left(\left(\binom{n-(i+1)\frac{m}{d}}{\frac{m}{d}} \binom{n-(i+2)\frac{m}{d}}{\frac{m}{d}} \cdots \right) - 1 \right) \binom{n-i\frac{m}{d}}{\frac{m}{d}} + \binom{n-i\frac{m}{d}}{\frac{m}{d}} - 1 \quad (7.131)$$

$$= \left(\binom{n-i\frac{m}{d}}{\frac{m}{d}} \binom{n-(i+1)\frac{m}{d}}{\frac{m}{d}} \cdots \binom{\frac{m}{d}+d}{\frac{m}{d}} \right) - 1 \quad (7.132)$$

which is the maximum value q_i can take on (as previously defined). As a result, any combination of r_i and q_{i+1} will satisfy q_i to be within its valid range of values. Moreover, due to Euclid's division lemma, each value of q_i gives a unique q_{i+1} and r_i. The inverse case also holds true of course, a given q_{i+1} and r_i gives a unique q_i (remark that in both cases the divisor $\binom{n-i\frac{m}{d}}{\frac{m}{d}}$ is fixed). This means there is a bijection

between q_i and (q_{i+1}, r_i). That is, there is a function

$$f(q_i) = (q_{i+1}, r_i)$$

which has an inverse

$$g(q_{i+1}, r_i) = q_i$$

given by Equation (7.130) for a fixed $\binom{n-i\frac{m}{d}}{\frac{m}{d}}$. Any function that has an inverse is a bijection.

Remark that i takes on values $0, 1, 2 \cdots d-1$. An inductive proof will show that each possible combination of $r_0, r_1 \cdots r_{d-1}$ can be given by each possible value of k_b. The base case for $i = 0$ is as follows:

$$q_1, r_0 \Longleftarrow divmod\left(q_0, \binom{n-0\frac{m}{d}}{\frac{m}{d}}\right) \tag{7.133}$$

resulting in

$$q_0 = q_1\binom{n-0\frac{m}{d}}{\frac{m}{d}} + r_0 \tag{7.134}$$

$$= q_1\binom{n}{\frac{m}{d}} + r_0 \tag{7.135}$$

As shown previously, each possible combination of q_1 and r_0 is given by each possible value of $q_0 = k_b$ (due to the bijection). Given q_{i+1} and r_i can take on all possible combinations, the proof for the inductive step follows as a result of the bijection between q_i and (q_{i+1}, r_i), and between q_{i+1} and (q_{i+2}, r_{i+1}). As in, each possible combination of q_{i+2} and r_{i+1} is given by each possible value of q_{i+1} while also allowing no restrictions on r_i. This results in any given r_i and r_{i+1} being

able to take on any given combination, which then allows for any combination of $r_0, r_1 \cdots r_{d-1}$.

Now I will finalize the proof. At execution i of the main loop on line 4, there are $n - i\frac{m}{d}$ elements left to assign in *permutation*. Each possible value of r_i corresponds to the $|R_i| = \binom{n-i\frac{m}{d}}{\frac{m}{d}}$ ways to place the next $\frac{m}{d}$ elements with value i in *permutation*. Thus, all possible ways to permute the multiset:

$$\binom{n}{\frac{m}{d}}\binom{n-\frac{m}{d}}{\frac{m}{d}}\binom{n-2\frac{m}{d}}{\frac{m}{d}}\cdots\binom{\frac{m}{d}+d}{\frac{m}{d}} = (|R_0|)(|R_1|)(|R_2|)\cdots(|R_{d-1}|) \quad (7.136)$$

are mapped onto by a value k_b. □

Remark that $|k'_b| = \binom{n}{\frac{m}{d}}\binom{n-\frac{m}{d}}{\frac{m}{d}}\binom{n-2\frac{m}{d}}{\frac{m}{d}}\cdots\binom{\frac{m}{d}+d}{\frac{m}{d}}$. Just like with Algorithm 5, the number of bits used to express k_b for Algorithm 8 is equal to $\lceil \log_2 |k'_b| \rceil$. Also similar to Algorithm 5, a given subkey k_b may 'collide' with another subkey. In other words, Algorithm 8 can return the same k'_b for at most two different subkey inputs.

Theorem 7.2.17. *For a given k_b, there will be at most one other subkey that specifies the same operations to apply via Algorithm 8.*

Proof. In the worst case, $2|k'_b| - 2 = 2^{\lceil \log_2 |k'_b| \rceil}$. This occurs when the largest possible n-digit variable-base number (as defined earlier) is equal to a $\lceil \log_2 |k'_b| \rceil$-digit binary number with a single 1 followed by $\lceil \log_2 |k'_b| \rceil - 1$ number of 0s. This means the first $2^{\lceil \log_2 |k'_b| \rceil - 1} + 1$ binary numbers that k_b can take on specify all $|k'_b|$ of the variable-base numbers, and the next $2^{\lceil \log_2 |k'_b| \rceil - 1} - 1$ binary numbers specify each of the first

$|k'_b| - 2$ variable base numbers again. See the following:

$$\binom{\frac{m}{d}+d}{\frac{m}{d}} - 1, \cdots, \binom{n-2\frac{m}{d}}{\frac{m}{d}} - 1, \binom{n-\frac{m}{d}}{\frac{m}{d}} - 1, \binom{n}{\frac{m}{d}} - 1 = 1, \underbrace{0, 0, 0, \cdots, 0, 0, 0}_{\lceil \log_2 |k'_b| \rceil - 1 \text{ times}} \tag{7.137}$$

$$1, \underbrace{0, 0, 0, \cdots, 0, 0, 0}_{d \text{ times}} = 1, \underbrace{0, 0, 0, \cdots, 0, 0, 0}_{\lceil \log_2 |k'_b| \rceil - 2 \text{ times}}, 1 \tag{7.138}$$

$$\vdots$$

$$1, \binom{\frac{m}{d}+d}{\frac{m}{d}} - 1, \cdots, \binom{n-2\frac{m}{d}}{\frac{m}{d}} - 1, \binom{n-\frac{m}{d}}{\frac{m}{d}} - 1, \binom{n}{\frac{m}{d}} - 4 = \underbrace{1, 1, 1, \cdots, 1, 1, 1}_{\lceil \log_2 |k'_b| \rceil - 1 \text{ times}} 0 \tag{7.139}$$

$$1, \binom{\frac{m}{d}+d}{\frac{m}{d}} - 1, \cdots, \binom{n-2\frac{m}{d}}{\frac{m}{d}} - 1, \binom{n-\frac{m}{d}}{\frac{m}{d}} - 1, \binom{n}{\frac{m}{d}} - 3 = \underbrace{1, 1, 1, \cdots, 1, 1, 1}_{\lceil \log_2 |k'_b| \rceil \text{ times}} \tag{7.140}$$

□

Note that although Theorems 7.2.11 and 7.2.17 showed a given subkey may have up to two collisions, the effect this has on the security of the protocol is negligible, given how large the key size is in comparison:

$$|k_a| \cdot |k_b| = 2^{\lceil \log_2 (\binom{n}{\frac{m}{d}} \binom{n-\frac{m}{d}}{\frac{m}{d}} \binom{n-2\frac{m}{d}}{\frac{m}{d}} \cdots \binom{\frac{m}{d}+d}{\frac{m}{d}}) \rceil \lceil \log_2 3^n \rceil} \tag{7.141}$$

Example 7.2.6. *The number of different keys the block Clifford code can have when* $n = 32$

and $d = 16$ *is:*

$$42,535,295,865,117,307,932,921,825,928,971,026,432$$

which is about 42.5 *undecillion, or* 42.5 *billion billion billion billion.*

The gates performed due to subkey k'_b did not affect the probability of detection in the security analysis in Theorems 7.2.3 and 7.2.4. However, subkey k'_b improves the block Clifford code in three ways. First, it increases the key size. Second, it protects against the vulnerability outlined in Table 7.4 where an adversary can apply a single-qubit Pauli to the packet and end up modifying $\frac{m}{d} + 1 = \frac{n}{d}$ qubits in the packet. This is a vulnerability in the sense that: The attack is detected with probability $\frac{2}{3}$ but will affect a non-negligible number of qubits. The number of qubits that are unaltered by the adversary may be of interest in certain applications (see Section 8.1 for a brief discussion on the usage of noisy qubits in quantum computing). By randomly permuting the signature qubits in the message, an adversary can not consistently perform the attack outlined in Table 7.4. A rigorous analysis of this should be left to potential future work. Lastly, it ensures that qubits belonging to the same block are randomly placed within the encrypted message. This is similar to the previous point: It ensures the adversary can't focus a disruptive attack on only a single block. As in, if the blocks were not designed in this way, the adversary could more consistently apply many non-identity single-qubit Paulis to a single block's qubits, and only be detected with $\approx \frac{1}{2}$ probability while being able to corrupt many qubits (remark Theorem 7.2.7). Future work may more rigorously analyze this.

Algorithm 8 can readily be extended to unrank permutations of any given

multiset. As far as I am aware, this is the most efficient algorithm of its kind. There is some literature on generating successive permutations of multisets [45] as well as unranking permutations of sets [46]. I could not find any published academic literature (papers, textbooks, etc.) on unranking permutations of multisets. I did find an online forum post describing an algorithm for unranking permutations of a multiset [47]. The author of the algorithm claims it runs in $O\left(n^2\right)$ time, however, it is $O\left(n^3 \log_2^2(n)\right)$ when considering the cost of multiplication operations within its main loop. Further work, outside the scope of this thesis, can fully analyze and expand upon this.

7.3 Simulation

A simulation of the block Clifford code implemented with Qiskit is available to the public[1]. It has been tested using Python 3.8.8. Unit tests are included in the repository which additionally verify

1. the correctness of both subkey mappings,
2. the constructions from the Clifford generators of the gates used in the block Clifford code and
3. the tables listed throughout this thesis which describe how a given Clifford operator transforms $2n$ Pauli generators.

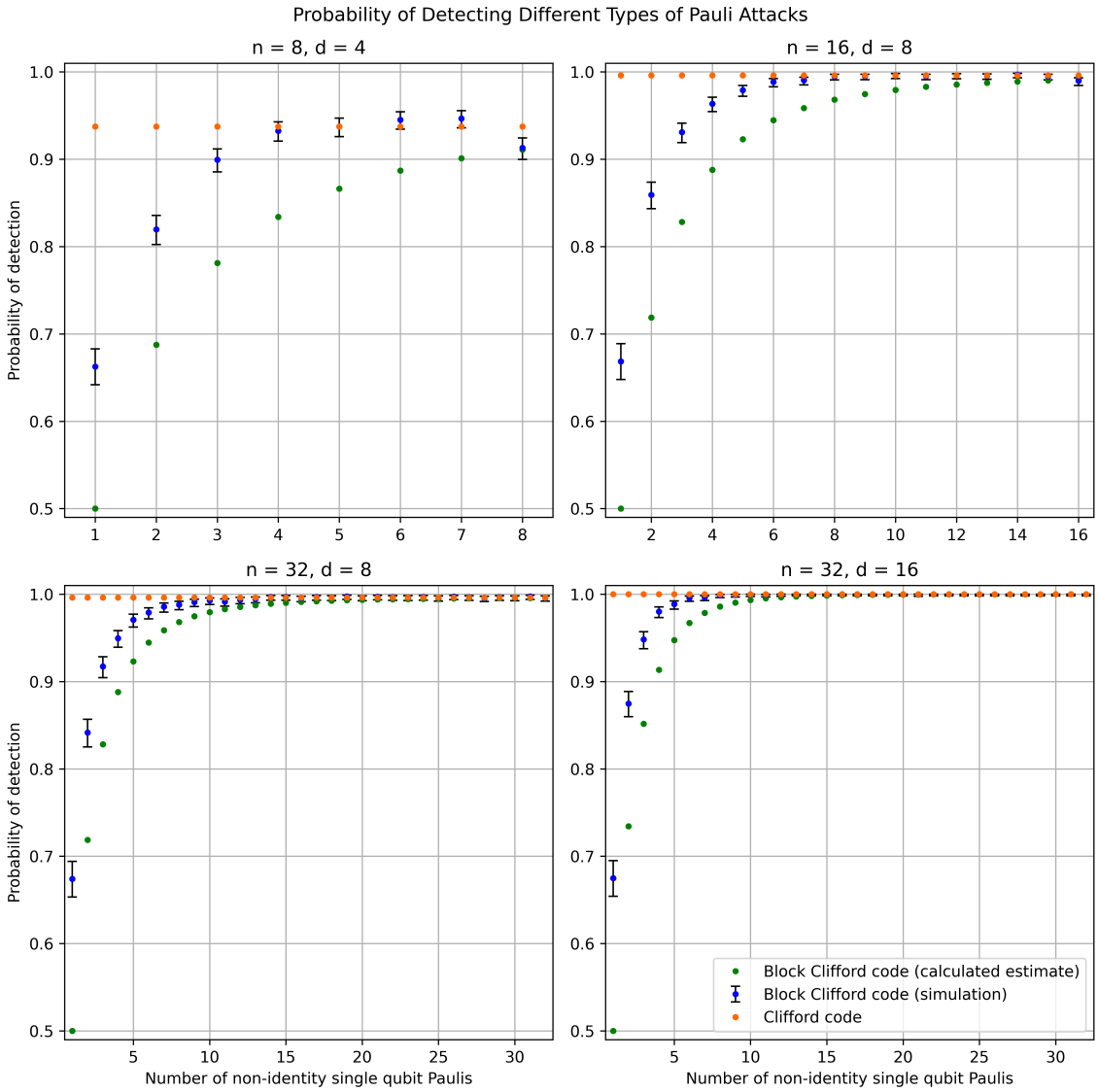

Figure 7.1: Figures showing the probability of detection for the Clifford code and the block Clifford code. Each x value on the x-axis denotes an adversary attacking only using random Paulis from $P(n)$ containing x number of non-identity single-qubit Paulis.

7.4 Results

Figure 7.1 compares the probability of detection for the Clifford code and the block Clifford code. Each subgraph corresponds to an experiment. The title for each subgraph depicts the values of n and d for each experiment. The horizontal sequence of dots (orange) is the probability of detection for the Clifford code given by Equation (6.1). The curved sequence of dots (blue) that has error bars on it depicts a confidence interval of 99.9% based on the Wilson score interval [48]; it shows the results from a simulation of the block Clifford code with 5000 trials for each x value on the graph. The Wilson score interval is a confidence interval used for distributions arising from Bernoulli trials (trials with an outcome of 1 indicating success and 0 indicating failure). In this case, 1 denotes the adversary was detected in a trial and 0 denotes the adversary was not detected in a trial. The curved sequence of dots (green) in Figure 7.1 without error bars is the approximated probability of detection for the block Clifford code given by Equation (7.80).

For all experiments, the block Clifford code quickly approaches a probability of detection comparable to the Clifford code as the number of non-identity single-qubit Paulis, x, in the attack increases. The experiment with $n = 32$ and $d = 16$ had the block Clifford code detect all 5000 attacks for 19 out of the 32 values of x.

[1] `https://github.com/catproof/block-Clifford-code`

Chapter 8

Conclusion

8.1 Discussion

The Clifford code's key mapping runs in $O\left(n^3\right)$ time, and the block Clifford code's key mapping runs in $O\left(n^2 \log_2^2(n)\right)$ time. It is not yet known how large data packets in a quantum internet might be. However, in classical networks data packets can contain thousands of bits [49], which may indicate the size of packets in quantum networks of the future. Assuming quantum data packets will have thousands of qubits, one would want to invest in efficient key mappings. For example, if a data packet had 1000 qubits, an $O\left(n^3\right)$ key mapping would result in a computation on the order of a billion operations. Although modern computers can perform these operations in a fraction of a second, it is still a relatively expensive operation for a computer to be performing frequently, especially in situations with stringent low latency requirements or limited computing power. Figure 8.1 shows how much faster n^3 grows in comparison to $n^2 \log_2^2(n)$.

Lemma 8.1.1. *$O\left(n^3\right)$ is asymptotically larger than $O\left(n^2 \log_2^2(n)\right)$.*

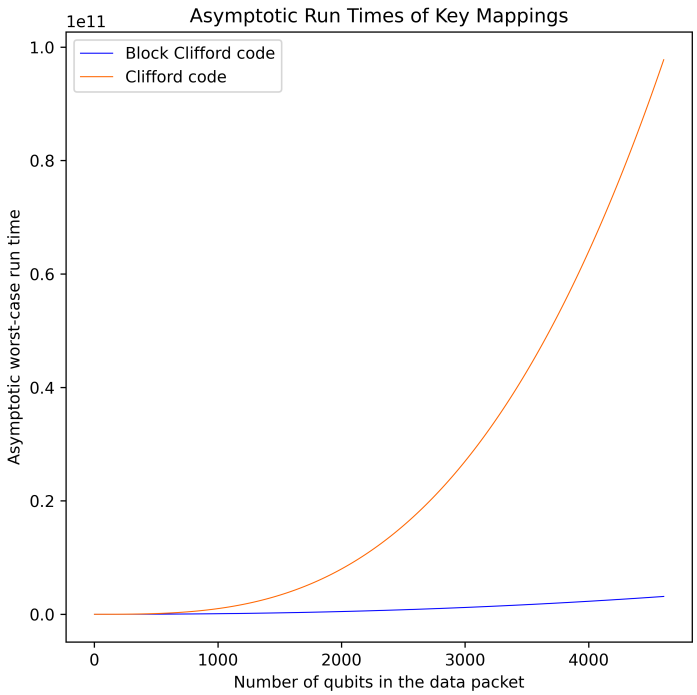

Figure 8.1: A figure comparing n^3 (Clifford code) to $n^2 \log_2^2(n)$ (block Clifford code).

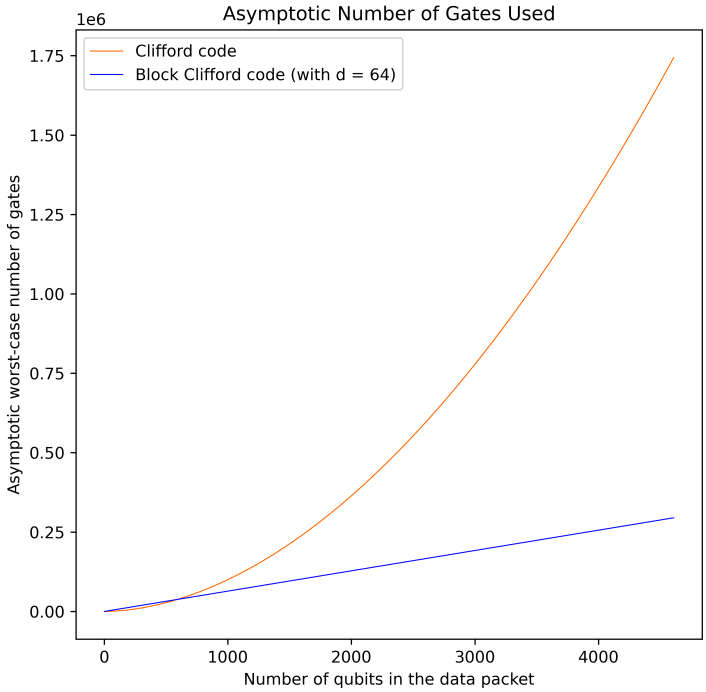

Figure 8.2: A figure comparing $\frac{n^2}{\log_2(n)}$ (Clifford code) to nd (block Clifford code).

In Section 5.11 a type of noise called depolarizing noise was characterized. There is reason to believe that many applications in quantum computing will work well with noisy qubits [50, 51, 52, 53, 54]. Given this assumption, a protocol such as the block Clifford code which can accept data packets with one or two noisy qubits with non-negligible probability (remark Figure 7.1) may be advantageous.

The block Clifford code also has a construction which uses $O(nd)$ gates from the Clifford generators. This construction may be optimal for certain values of n and d; more investigation is needed. For a fixed d, this is certainly asymptotically less than the worst-case of $O\left(\frac{n^2}{\log_2^2(n)}\right)$ for the Clifford code. Figure 8.2 shows how much faster $\frac{n^2}{\log_2(n)}$ grows in comparison to nd with $d = 64$.

8.2 Future Work

More precisely characterizing how the block Clifford code handles noisy data may be the focus of future work. Finding an exact (as opposed to an approximate) characterization of the block Clifford code's security would be beneficial.

The time complexity of the key mapping method for the block Clifford code could potentially be reduced. One way to tackle this problem would be to increase the memory it uses, in hopes that it will increase computational efficiency. The subkey k_a mapping could be sped up by assigning groupings of two adjacent bits in the binary key to each single-qubit unitary. The mapping could read the groupings of two bits from the key sequentially, where 00, 01 and 10 could correspond to each of I, H, and $S^\dagger H$. A grouping with 11 could correspond to the single-qubit unitary determined by the previously read grouping of two bits. If 11 were the first read grouping of bits, it could be assigned I. This would cause more key collisions, but could still determine an essentially uniform distribution of unitaries to apply to the packet. The subkey k_b mapping could be sped up by computing all the $\binom{m}{j}$s ahead of time and storing them in memory so that they don't have to be calculated every time the subkey mapping is computed.

The multiset permutation unranking algorithm may also be extended to describe a multiset permutation ranking algorithm. The unranking algorithm may be used to fully describe the key mapping for the trap code as well.

Bibliography

[1] Gabriel Popkin. The internet goes quantum, 2021.

[2] Howard Barnum, Claude Crépeau, Daniel Gottesman, Adam Smith, and Alain Tapp. Authentication of quantum messages. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 449–458. IEEE, 2002.

[3] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[4] Euclidean division. `https://en.wikipedia.org/wiki/Euclidean_division`. Accessed: 2022-06-08.

[5] Pat Morin. *Open Data Structures: An Introduction*, volume 9. Athabasca University Press, 2013.

[6] Set-builder notation. `https://en.wikipedia.org/wiki/Set-builder_notation`. Accessed: 2022-07-18.

[7] Noson S Yanofsky and Mirco A Mannucci. *Quantum computing for computer scientists*. Cambridge University Press, 2008.

[8] Discrete structures for computer science: Counting, recursion, and probability. https://cglab.ca/~michiel/DiscreteStructures/DiscreteStructures.pdf. Accessed: 2022-07-13.

[9] Abraham Asfaw, Antonio Corcoles, Luciano Bello, Yael Ben-Haim, Mehdi Bozzo-Rey, Sergey Bravyi, Nicholas Bronn, Lauren Capelluto, Almudena Carrera Vazquez, Jack Ceroni, Richard Chen, Albert Frisch, Jay Gambetta, Shelly Garion, Leron Gil, Salvador De La Puente Gonzalez, Francis Harkins, Takashi Imamichi, Hwajung Kang, Amir h. Karamlou, Robert Loredo, David McKay, Antonio Mezzacapo, Zlatko Minev, Ramis Movassagh, Giacomo Nannicini, Paul Nation, Anna Phan, Marco Pistoia, Arthur Rattew, Joachim Schaefer, Javad Shabani, John Smolin, John Stenger, Kristan Temme, Madeleine Tod, Stephen Wood, and James Wootton. Learn quantum computation using qiskit, 2020.

[10] Quantiki. https://quantiki.org/. Accessed: 2021-09-13.

[11] Quantum computing stack exchange. https://quantumcomputing.stackexchange.com/. Accessed: 2021-09-13.

[12] John Watrous. *The theory of quantum information*. Cambridge university press, 2018.

[13] Rodney Van Meter. *Quantum networking*. John Wiley & Sons, 2014.

[14] Quantum inspire: Qubit basis states. https://www.quantum-inspire.com/kbase/qubit-basis-states/. Accessed: 2022-08-05.

[15] Daniel Gottesman. *Stabilizer codes and quantum error correction*. California Institute of Technology, 1997.

[16] Maris Ozols. Clifford group. *Essays at University of Waterloo, Spring*, 2008.

[17] Circular shift. `https://en.wikipedia.org/wiki/Circular_shift`. Accessed: 2022-06-09.

[18] Toshihiro Shimizu, Takuro Fukunaga, and Hiroshi Nagamochi. Unranking of small combinations from large sets. *Journal of Discrete Algorithms*, 29:8–20, 2014.

[19] Jörg Arndt. *Matters Computational: ideas, algorithms, source code*. Springer Science & Business Media, 2010.

[20] Michel Barbeau, Evangelos Kranakis, and Nicolas Perez. Authenticity, integrity, and replay protection in quantum data communications and networking. *ACM Transactions on Quantum Computing*, 3(2):1–22, 2022.

[21] Qiskit depolarizing channel. `https://qiskit.org/documentation/stubs/qiskit.providers.aer.noise.depolarizing_error.html?highlight=depolarizing_error#qiskit.providers.aer.noise.depolarizing_error`. Accessed: 2022-07-24.

[22] Takahiko Satoh, Shota Nagayama, Takafumi Oka, and Rodney Van Meter. The network impact of hijacking a quantum repeater. *Quantum Science and Technology*, 3(3):034008, 2018.

[23] Takafumi Oka, Takahiko Satoh, and Rodney Van Meter. A classical network protocol to support distributed quantum state tomography. In *2016 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2016.

[24] Michael Ben-Or, Claude Crépeau, Daniel Gottesman, Avinatan Hassidim, and Adam Smith. Secure multiparty quantum computation with (only) a strict honest majority. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 249–260. IEEE, 2006.

[25] Dorit Aharonov, Michael Ben-Or, Elad Eban, and Urmila Mahadev. Interactive proofs for quantum computations. *arXiv preprint arXiv:1704.04487*, 2017.

[26] Anne Broadbent, Gus Gutoski, and Douglas Stebila. Quantum one-time programs. In *Annual Cryptology Conference*, pages 344–360. Springer, 2013.

[27] Robert Koenig and John A Smolin. How to efficiently select an arbitrary clifford group element. *Journal of Mathematical Physics*, 55(12):122202, 2014.

[28] Checksum. `https://en.wikipedia.org/wiki/Checksum`. Accessed: 2022-07-17.

[29] Sergey Bravyi and Dmitri Maslov. Hadamard-free circuits expose the structure of the clifford group. *IEEE Transactions on Information Theory*, 2021.

[30] Ketan N Patel, Igor L Markov, and John P Hayes. Optimal synthesis of linear reversible circuits. *Quantum Inf. Comput.*, 8(3):282–294, 2008.

[31] Balls into bins problem. `https://en.wikipedia.org/wiki/Balls_into_bins_problem`. Accessed: 2022-06-28.

[32] Urn problem. `https://en.wikipedia.org/wiki/Urn_problem`. Accessed: 2022-06-28.

[33] Stirling number of the second kind. `https://mathworld.wolfram.com/StirlingNumberoftheSecondKind.html`. Accessed: 2022-06-21.

[34] The multivariate hypergeometric distribution. https://stats.libretexts.org/Bookshelves/Probability_Theory/Probability_Mathematical_Statistics_and_Stochastic_Processes_(Siegrist)/12%3A_Finite_Sampling_Models/12.03%3A_The_Multivariate_Hypergeometric_Distribution. Accessed: 2022-07-19.

[35] Digital library of mathematical functions, set partitions: Stirling numbers. https://dlmf.nist.gov/26.8#vii/. Accessed: 2022-08-04.

[36] Positional notation, base conversion. https://en.wikipedia.org/wiki/Positional_notation#Base_conversion. Accessed: 2022-07-04.

[37] Hua Zhang, Xuemei Wei, Rui Wang, and Fanli Meng. An efficient base conversion using variable length segmentation and remainder transfer. *IEEE Signal Processing Letters*, 26(8):1227–1231, 2019.

[38] Paul Zimmermann. Asymptotically fast division for gmp. https://members.loria.fr/PZimmermann/papers/invert.pdf. Accessed: 2022-06-08.

[39] Torbjörn Granlund. Gnu multiple precision arithmetic library. *http://gmplib.org/*, 2010.

[40] David Harvey and Joris Van Der Hoeven. Integer multiplication in time o (n log n). *Annals of Mathematics*, 193(2):563–617, 2021.

[41] Generating elementary combinatorial objects. https://www.site.uottawa.ca/~lucia/courses/5165-09/GenCombObj.pdf. Accessed: 2022-06-21.

[42] Abu Bakar Siddique, Saadia Farid, and Muhammad Tahir. Proof of bijection for combinatorial number system. *arXiv preprint arXiv:1601.05794*, 2016.

[43] Peter B Borwein. On the complexity of calculating factorials. *Journal of Algorithms*, 6(3):376–380, 1985.

[44] Sorted containers. `https://grantjenks.com/docs/sortedcontainers/`. Accessed: 2022-07-20.

[45] Tadao Takaoka. Multi-level loop-less algorithm for multi-set permutations. *arXiv preprint arXiv:1502.06062*, 2015.

[46] Wendy Myrvold and Frank Ruskey. Ranking and unranking permutations in linear time. *Information Processing Letters*, 79(6):281–284, 2001.

[47] Algorithm for finding multiset permutation given lexicographic index. `https://stackoverflow.com/a/24508736/9480689`. Accessed: 2022-07-04.

[48] Binomial proportion confidence interval. `https://en.wikipedia.org/wiki/Binomial_proportion_confidence_interval#Wilson_score_interval`. Accessed: 2022-08-04.

[49] Rfc 879. `https://www.rfc-editor.org/rfc/rfc879#section-1`. Accessed: 2022-07-24.

[50] Ibm's latest trick: Turning noisy quantum bits into machine learning magic. `https://thenextweb.com/news/ibms-latest-trick-turning-noisy-quantum-bits-into-machine-learning-magic`. Accessed: 2022-07-24.

[51] Sarah Sheldon. Quantum computing with noisy qubits. In *Frontiers of Engineering: Reports on Leading-Edge Engineering from the 2018 Symposium*. National Academies Press, 2019.

[52] Dominic Rosch-Grace and Jeremy Straub. From quantum fuzzing to the multiverse: Possible effective uses of quantum noise. In *Future of Information and Communication Conference*, pages 399–410. Springer, 2022.

[53] Approximate quantum computing advantage applications. https://www.ibm.com/blogs/research/2017/12/approximate-quantum-computing-advantage-applications/. Accessed: 2022-07-24.

[54] Scientists work to turn noise on quantum computer to their advantage. https://scitechdaily.com/scientists-work-to-turn-noise-on-quantum-computers-to-their-advantage/. Accessed: 2022-07-24.

.1 Proofs for Section 7.2 (Algorithm)

Lemma 7.2.1. *Let U_i and V_k be arbitrary single-qubit unitaries. Also let $CU_{i,j}$ be an arbitrary $CU_{i,j} \in \{CX_{i,j}, CY_{i,j}, CZ_{i,j}\}$ and let $CU_{k,j}$ be set to the same type of controlled-U gate as $CU_{i,j}$. The operations $CU_{i,j}$ and U_i commute with any of $CU_{k,j}$ and V_k, for any values of i, j and k where $i \neq j \neq k$.*

Proof. The proofs for all three cases are as follows:

Case 1: U_i and V_k commute. This case is proven in Theorem 5.7.1.

Case 2: V_k commutes with $CU_{i,j}$ for any values of i, j and k where $i \neq j \neq k$ because, like in case 1, they act on different qubits. For some state $|\psi\rangle = c_0|x_0\rangle + c_1|x_1\rangle + c_2|x_2\rangle \cdots + c_{2^n-1}|x_{2^n-1}\rangle$ observe what may happen for an arbitrary $c_w |x_w\rangle$. $CU_{i,j}$ either applies an I, or a U_j to a given $c_w |x_w\rangle$. For the subcase when $CU_{i,j}$ applies a U_j to $c_w |x_w\rangle$, the transformation applied to $c_w |x_w\rangle$ is $U_j V_k = V_k U_j$. The proof for U_j and V_k commuting is the same as the one given for case 1. For the subcase when $CU_{i,j}$ applies a I, this of course commutes with V_k.

Case 3: $CU_{i,j}$ and $CU_{k,j}$ commute. This case is proven in Theorem 5.7.18. □

Lemma 7.2.2. *Let $A_{i'}$ be a single-qubit adversarial attack applied to a qubit at position i', and let A_i be the same operation but applied to a data qubit position i. When using the block Clifford code, the operation applied to the state ρ as a result of the operations for encrypting, followed by an adversary attacking a single data qubit using $A_{i'}$ and then decrypting, can be simplified in such a way:*

$$U_j CU_{i,j} U_i^\dagger A_i U_i CU_{i,j} U_j \tag{53}$$

Proof. The permuting and un-permuting of signature qubits result in the adversary's

attack $A_{i'}$ being 'moved' to become A_i. That is, the $A_{i'}$ unitary is conjugated by a series of *SWAP* gates to become A_i where $d \leq i < n$ and $i' \leq i$ (remark lines 32 and 14 or Algorithms 3 and 4). After simplifying all the *SWAP* gates for permuting and then un-permuting the signature qubits, the only gates left to simplify are the $CU_{k,w}$, $CU_{k,j}$, U_k and U_w gates for all $k \neq i$ and $w \neq j$. U_k denotes gates the block Clifford code applies to the data qubits and U_w denotes gates the block Clifford code applies to the signature qubits. Applying Lemma 7.2.1 and Theorem 5.7.19 multiple times allows to rearrange gates applied by the code into groupings of $CU_{k,w}U_k^\dagger U_k CU_{k,w}$ and $CU_{k,j}U_k^\dagger U_k CU_{k,j}$, which simplify to each equal I. Following that simplification, gates can be rearranged into groupings of $U_w^\dagger U_w$ which also simplify to equal I. This results in the only gates left to simplify or evaluate as the ones shown in Equation (7.53). □

Lemma 7.2.3. *When using the block Clifford code, the operation applied to the state ρ due to encrypting, followed by an adversary attacking a single signature qubit using $A_{j'}$ and then decrypting, can be simplified in such a way:*

$$U_j^\dagger CU_{B(j,0),j} \cdots CU_{B(j,\frac{m}{d}-1),j} A_j CU_{B(j,\frac{m}{d}-1),j} \cdots CU_{B(j,0),j} U_j \tag{54}$$

Proof. The permuting and un-permuting of signature qubits result in the adversary's attack $A_{j'}$ being 'moved' to become A_j. That is, the $A_{j'}$ unitary is conjugated by a series of *SWAP* gates to become A_j where $d \leq j \leq j' < n$ (remark lines 32 and 14 or Algorithms 3 and 4). After simplifying all the *SWAP* gates for permuting and then un-permuting the signature qubits, the only gates left to simplify are the $CU_{k,w}$, U_k and U_w gates for all $w \neq j$ and $k \notin B(j)$ and $k \neq j$. U_k denotes gates the block Clifford code applies to the data qubits and U_w denotes gates the block Clifford

code applies to the signature qubits. Applying Lemma 7.2.1 and Theorem 5.7.19 multiple times allows to rearrange gates applied by the code into groupings of $CU_{k,w}U_k^\dagger U_k CU_{k,w}$, which simplify to each equal I. Following that simplification, gates can be rearranged into groupings of $U_w^\dagger U_w$ which also simplify to equal I. This results in the only gates left to simplify or evaluate as the ones shown in Equation (7.54). □

.2 Proofs for Section 8.1 (Discussion)

Lemma 8.1.1. *$O\left(n^3\right)$ is asymptotically larger than $O\left(n^2 \log_2^2(n)\right)$.*

Proof. First, some simplifications are made:

$$n^2 \log_2^2(n) < n^3 \tag{1}$$

$$\log_2^2(n) < n \tag{2}$$

$$\log_2(n) < \sqrt{n} \tag{3}$$

The proof can be completed by using the racetrack principle. First, note that substituting $n = 16$ into Equation (3) shows that:

$$\log_2(16) = \sqrt{16} \tag{4}$$

Now, it must be shown that taking the derivatives of both sides of Equation (3)

satisfies the inequality for all $n \geq 16$. Taking the derivatives of both sides gives:

$$\frac{1}{n\ln(2)} < \frac{1}{2\sqrt{n}} \tag{5}$$

$$\frac{2\sqrt{n}}{n\ln(2)} < 1 \tag{6}$$

$$\frac{4n}{n^2(\ln^2(2))} < 1 \tag{7}$$

$$\frac{4}{n(\ln^2(2))} < 1 \tag{8}$$

$$\frac{4}{(\ln^2(2))} \approx 8.33 < n \tag{9}$$

Thus concluding the proof. □