Information Theoretically Secure Hypothesis Test for Temporally Unstructured Quantum Computation

Daniel Mills

Master of Science by Research

Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh

2016

Abstract

Quantum computer provide great promise in terms of the computational speed-up that they could provide. Less than universal quantum computers too show promise and it is believed that some of these models of computation will prove classically hard to simulate. As such, it would be of great benefit, if only as a proof of principle demonstration of our ability to perform quantum computations, to construct such a device.

It is necessary then that a computer we believed to be more than classical can prove it is such. To do so it should pass a hypothesis test. In this work we move towards developing a hypothesis test to convince a sceptic of the capability of a device to perform temporally unstructured quantum computations. In particular we consider the instantaneous quantum polytime (IQP) machine developed by Shepherd and Bremner [1] which constructs computations from a commuting gate set.

We adapt the hypothesis test developed in [1] by re-framing IQP machine programs in the measurement based model for quantum computation. In doing so we are able to exploit tools from blind quantum computing and strengthen some of the claims made in [1] about their hypothesis test.

By developing a blind graph state creation resource we are able to conceal many of the details of the hypothesis test that a server must pass, weakening their ability to cheat. In doing so we develop a stronger hypothesis test based on fewer computational complexity assumption and instead on information theoretic assumptions.

Acknowledgements

Many thanks to my supervisors, Elham Kashefi and Anna Pappa, for their support, guidance and for introducing me to many new and exciting subjects. Thanks to Thodoris Kapourniotis for his insight and teachings relating to this project and other fields. Thanks to Petros Wallden for his insightful course on quantum computing and his assistance and guidance throughout the year. Thanks to Andru Gheorghiu for intriguing conversations and motivation. Thanks to Martyna Panasiuk for proofreading and correcting this work and for her support through the year.

This work was supported in part by the EPSRC Centre for Doctoral Training in Pervasive Parallelism, funded by the UK Engineering and Physical Sciences Research Council (grant EP/L01503X/1) and the University of Edinburgh.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Daniel Mills*)

Table of Contents

Chapter 1

Introduction and Background

1.1 Introduction

Initially the quantum computer was suggested as a way of simulating quantum systems [2, 3] but it has since been shown to have many other applications [4]. Protocols demonstrating the power of quantum computers include Shor's algorithm for prime factorisation [5], Grover's algorithm for unstructured search [6], and the BB84 protocol for public key exchange [7].

That said, it may be some time before a large scale universal quantum computer capable of demonstrating the computational power of these protocols is built. In the meantime several intermediate, non universal models of quantum computation, which may be easier to implement, have been developed. Examples include the one clean qubit model [8, 9] and the boson sampling model [10].

The instantaneous quantum poly-time (IQP) machine [1] is another, less than universal machine, with practical advantages [11]. This model uses a commuting gate set and although a universal gate set would be non-commuting this simple model is still thought not to be classically simulatable [12, 13]. This fact demonstrates why it would constitute exciting progress for such a machine to be built as it would prove our capacity to build quantum computers.

Simply claiming to have build such a machine would not, on its own, be sufficient and we must be able to prove this achievement. Indeed in [1] the authors develop a hypothesis test designed to be passable only by devices capable of efficiently simulating an IQP machine. Their protocol employs binary matroid theory and arguments about the computational complexity of finding hidden sub-matroids to achieve this goal.

In our work we build on the scheme of [1]. By framing the IQP machine in the

setting of measurement based quantum computing (MBQC) [14, 15] we are able to use tools taken from blind quantum computing [17, 18] to develop a hypothesis test strengthening that of [1] by removing some of the conjectures they make. These conjectures relate to the computational complexity of the hidden sub-matroid problem which we will introduce and discuss.

The structure of this work will be first to provide the necessary background in quantum computing and quantum mechanics. We will then formally introduce the IQP machine and the hypothesis test developed by [1] before moving to develop a protocol to implement an IQP computation distributively. Importantly we will do so without revealing the details of the computation to the device performing it. This property, in particular, will be of use in our hypothesis test, a description of which will then follow.

We proceed by introducing some of the key tools that will be used throughout. In section 1.2 we introduce some basics of quantum computing before coming, in section 1.3, to the aforementioned measurement based quantum computing. Our final introductory section is on the vital proof tool of abstract cryptography. This is found in section 1.4.

1.2 Quantum Mechanics and Computing

In this section, we will not cover all the necessary linear algebra but such background may be found in [4, 19]. There, a more thorough introduction to quantum computing generally may also be found and much of the work of this section relies on it.

Many texts provide good introductions to quantum physics but, in particular, [20] is of quality. Throughout, and when it becomes appropriate, we will also introduce the postulates of quantum mechanics.

1.2.1 The Qubit

The primary tool used in quantum computing is the *qubit*. While a classical bit exists in the state 0 or 1 the qubit may exist in a *superposition* of these states. We write a general qubit in the form of (1.1) where $\alpha, \beta \in \mathbb{C}$ and where $|a|^2 + |b|^2 = 1$. We can loosely understand this superposition as being 'α much of the $|0\rangle$ state and β much of the $|1\rangle$ state'.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1.1}$$

Notice that with a basis $\{|0\rangle, |1\rangle\}$, which we refer to a the *computational basis*,

we have defined a two dimensional vector space over the field $\mathbb{C}$. With the addition of an inner product, $\langle\cdot|\cdot\rangle$, with $\langle 0|1\rangle = 0$ and $\langle 1|1\rangle = \langle 0|0\rangle = 1$ we have defined a two dimensional Hilbert space

We would typically expect to write $\langle|0\rangle||1\rangle\rangle$ when talking about the inner product of vectors. We will, however, often resort to the above, more convenient, *Bra-Ket* notation. Using this notation, the state $\langle\psi|$ is the complex conjugate of the state $|\psi\rangle$.

The reader will realise that any qubit may be written in the form of expression (1.2). Since we will later see that the factor $e^{i\gamma}$ in that expression can be ignored, we can handily represent a single qubit as a point on surface of the *Bloch sphere* of Figure 1.1. The positive and negative Z axis of the Bloch sphere are the $|0\rangle$ and $|1\rangle$ states respectively.

$$|\psi\rangle = e^{i\gamma}\left(\cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle\right) \tag{1.2}$$

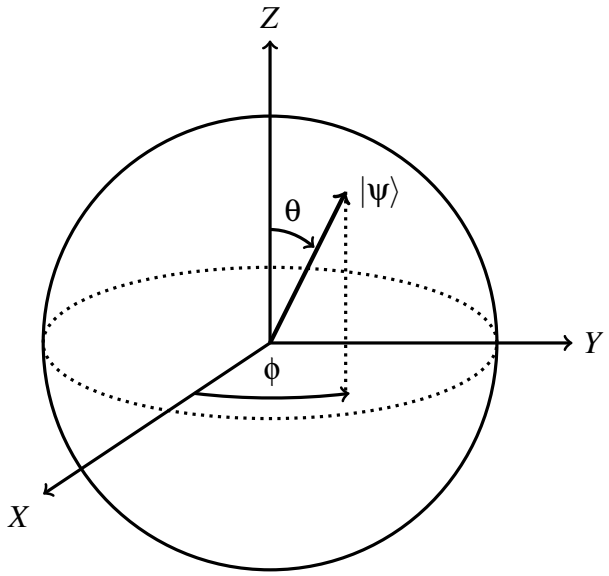

Figure 1.1: The Bloch sphere provides an intuitive way of representing any single qubit $|\psi\rangle$. Referring to expression (1.2) helps to understand the notation used. The state of a single qubit can be represented by a point on the surface of the sphere.

We note now that, as well as the computational basis mentioned, the Hadamard basis of expression (1.3) is also commonly used as a basis for this same space. These states are found on the X axis of the Bloch sphere.

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \quad \text{and} \quad |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{1.3}$$

1.2.2 Entanglement, Non-locality and the Tensor Product

Previously, we have considered single qubit systems but we can use the tensor product to construct larger, multi-qubits systems.

Definition 1.2.1. *The elements on the* tensor product, $V \otimes W$, *of two vector spaces* V *and* W, *are linear combinations of vectors* $|v\rangle \otimes |w\rangle$ *where* $|v\rangle \in V$ *and* $|w\rangle \in W$.

If $|i\rangle$ *and* $|j\rangle$ *form an orthonormal basis of* V *and* W *respectively, then vectors of the form* $|i\rangle \otimes |j\rangle$ *forms an orthonormal basis of* $V \otimes W$.

If systems 1 *to n are prepared in the states* $|\psi_1\rangle, ..., |\psi_n\rangle$ *then the* joint state *of the system is* $|\psi_1\rangle \otimes ... \otimes |\psi_n\rangle$.

More complicated systems can result from building states which cannot be described as the joint state of single qubit systems. It is these states that bring the power of quantum computing and which introduce entanglement into quantum mechanics. Entanglement is what introduces *non-locality*, or 'spooky action at a distance'; the phenomenon found so unpalatable by Einstein [21].

Definition 1.2.2. *A state is called* entangled *if the state of the composite system cannot be written as the tensor product of its components.*

The typical example of an entangled state is as in expression (1.4) which the reader can verify meets the conditions of Definition 1.2.2.

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{1.4}$$

We now have sufficient terminology to introduce the first postulate of quantum mechanics.

> **Postulate 1**: Associated to any physical system is a complex vector space with inner product known as the *state space* of the system. The system is completely described by its *state vector*, which is a unit vector in the system's state space.

1.2.3 Operators on Quantum States

Qubits can be acted upon and altered by *operators*. These operators can be represented by $m \times n$ matrices acting on the state space of the system. This representation may be arrived at by considering expression (1.5) where vectors, of the form $|k\rangle$, form an orthonormal basis.

$$A = \sum_{ij} |i\rangle \langle j| A_{ij} \quad \text{where} \quad A_{ij} = \langle i|A|j\rangle \tag{1.5}$$

Operators can also be formed from vectors by considering the *outer product* as outlined in expression (1.6). A description as a matrix can be found as in equation (1.7).

$$(|w\rangle \langle v|)|\widehat{v}\rangle = \langle v|\widehat{v}\rangle |w\rangle \tag{1.6}$$

$$|w\rangle \langle v| = \sum_{i,j} w_i v_j^* |i\rangle \langle j| \tag{1.7}$$

Some useful definitions regarding operators are found in Definition 1.2.3. These definitions will leave us sufficiently prepared to introduce the second postulate of quantum mechanics.

Definition 1.2.3. *The* adjoint, $A^\dagger$, *of an operator, A, is that satisfying expression* (1.8).

An operator is said to be Hermitian *if it is self adjoint and* unitary *if its adjoint is its inverse.*

$$\langle |v\rangle |A|w\rangle \rangle = \left\langle A^\dagger |v\rangle \middle| |w\rangle \right\rangle \tag{1.8}$$

Postulate 2: The state of the system at time t_1 is related, as seen in expression (1.9), to the state of the system at time t_2 by the unitary operator U which depends only on the times t_1 and t_2.

$$|\psi_{t_2}\rangle = U|\psi_{t_1}\rangle \tag{1.9}$$

A second and similar postulate addressing the matter of continuous evolution of states in time is that of postulate 2'.

Postulate 2': In the absence of any external influence the state of the system changes smoothly in time according to the Scrödinger equation seen in line (1.10). Here H is the hermitian operator, called the *Hamiltonian*, corresponding to the energy of the system.

$$i\hbar \frac{d|\psi_t\rangle}{dt} = H|\psi_t\rangle \tag{1.10}$$

1.2.4 Operators as Part of Quantum Circuits

In quantum computing all gates are unitary operators and we now consider some common examples.

The quantum equivalent of the classical NOT gate is the Pauli-X gate which, when acting on the computational basis states, behaves according to expression (1.11). It may also be summarised in matrix notation as in expression (1.12).

$$|0\rangle \to |1\rangle \quad , \quad |1\rangle \to |0\rangle \tag{1.11}$$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{1.12}$$

Expression (1.13) shows, in matrix notation, the Pauli-Y gate, the Pauli-Z gate, the Hadamard gate, H, and the phase gate, R_θ. These too are commonly used gates. One may note, in particular, that the Hadamard gate can be used to change between the computational and Hadamard basis of (1.3) and that $R_\pi = Z$.

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad , \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad , \quad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad , \quad R_\theta = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix} \tag{1.13}$$

On a notational note, if there is ambiguity as to which qubit the gates are being applied, we will use a subscript, such as X_i, to indicate that it is applied to the i^{th} qubit.

The above are only a few examples of a single qubit gates and any of the uncountably many unitary operators may perform this role.

Some simple examples of two qubit gates are the controlled gates. This simply means applying an operation to a *target* qubit depending on the value of a *control* qubit. Two examples of these are the controlled-X (also known as the CNOT gate) and the controlled-Z gates which are seen in expression (1.14). These matrices are written using the basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$.

$$\text{controlled-}X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad , \quad \text{controlled-}Z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \tag{1.14}$$

One may write the controlled-X and controlled-Y gates as $X_{i,j}$ and $Z_{i,j}$ respectively when they are applied to the control qubit, i, and the target qubit, j. In the case of the

controlled-Z gate, which of the two qubits is the control and which is the target makes no difference while in the case of the controlled-X gate this choice is important.

Later, we will draw these gates as part of quantum circuits and will use the representation as in Figure 1.2. The flow of time is from left to right and the horizontal wires carry qubits.

controlled-X controlled-Z

Figure 1.2: The controlled-X and controlled-Z gates in circuit notation.

As mentioned, the set of all possible unitary gates is uncountable. However all gates, including multi-qubit gates, may be approximated by some finite set of gates called a *finite universal gate set*. One such set is that of expression (1.15).

$$\left\{H, R_{\frac{\pi}{4}}, CNOT\right\} \tag{1.15}$$

If we allow for infinitely many gates to be in our *universal gate set* then we can exactly recreate any gate[1]. Two such examples are the sets of expression (1.16) where R_θ represent phase gates for all θ and U represents all single qubit gates.

$$\{X, R_\theta, CNOT\} \quad and \quad \{CNOT, U\} \tag{1.16}$$

1.2.5 Measurement

As well as the smooth evolution under a unitary operator, a system may undergo a change due to measurement. This fact is addressed in the third postulate of quantum mechanics.

> **Postulate 3**: A measurement may have m different outcomes. Each is represented by an operator acting on the Hilbert space of the system being measured. Together these operators form the collection $\{M_1, ..., M_m\}$ and must satisfy the *completeness equation* of expression (1.17).

$$\sum_m M_m^\dagger M = 1 \tag{1.17}$$

[1] I.e. not just approximately like in the finite gate set case.

The probability of each outcome depends on the initial state[2] $|\psi\rangle$ of the system according to the relationship of expression (1.18).

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle \tag{1.18}$$

The state of the system after a measurement outcome of m is as in expression (1.19).

$$|\psi\rangle \to \frac{1}{\sqrt{p(m)}} M_m |\psi\rangle \tag{1.19}$$

The completeness equation ensures the total probability of all measurement outcomes is 1 while the reader may check that the outcome of the measurement, as seen in expression (1.19), is a unit vector.

One will also note that the leading factor of expression (1.2) does not contribute to the value of the probability as calculated in expression (1.18). Hence, as mentioned previously, it is often ignored.

Measurement, in circuit notation, will appear later as in Figure 1.3 where a double wire carries classical information.

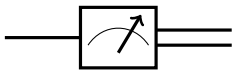

Figure 1.3: A representation of a measurement in circuit notation.

A particularly important class of measurements are the *projective measurements*. These occur when all operators M_i are projective operators, which we will now define.

Definition 1.2.4. *A Hermitian operator, P, is called a* projection operator *if* $P^2 = P$

These operators restrict a vector to a particular subspace of the total Hilbert space and, in particular, $P = |\psi\rangle\langle\psi|$ gives a projection onto the subspace defined by $|\psi\rangle$. Often we will refer to things like an Z basis measurement which will involve, when considering only a single qubit measurements, the projective operators $|0\rangle\langle 0|$ and $|1\rangle\langle 1|$ defined by the eigenvectors of the Z operator.

The *spectral decomposition* of expression (1.20) allows us to write any hermitian matrix, A, in terms of projection operators, P_a, projecting along the substance defined by eigenvectors, $|a\rangle$, of A and the corresponding eigenvalues, a. This fact, along with proposition 3, allows us to write the expected value, $\langle A\rangle_\psi$, of measuring the state $|\psi\rangle$ with the observable A as that of expression (1.21).

[2]By initial we here mean immediately prior to the measurement.

$$A = \sum_a a P_a \tag{1.20}$$

$$\langle A \rangle_\psi = \sum_a a \langle \psi | P_a | \psi \rangle = \langle \psi | \sum_a a P_a | \psi \rangle = \langle \psi | \sum_a A | \psi \rangle \tag{1.21}$$

Finally, although we do not go through the details of the proof, instead referring the reader to [4] for this, it is true that no measurement can ever distinguish between non-orthogonal pure states with certainty. For completeness we state this now.

Theorem 1.2.1. *Non-orthogonal quantum states cannot be distinguished with certainty.*

1.2.6 Density Matrices And Mixed States

We may represent states, not just by vectors $|\psi\rangle$, but also as operators $\rho_\psi = |\psi\rangle \langle\psi|$. This description of a state is known as a *density matrix* and using this notation allows us to represent our own uncertainty about the system, as well as the uncertainty which arises from the nature of quantum mechanics, as described by postulate 3.

Suppose we have an *ensemble*, $\{p_i, |\psi_i\rangle\}$, of states, where the state we have in our possession is the state $|\psi_i\rangle$ with probability p_i. Then we can describe this state as in expression (1.22).

$$\rho = \sum_i p_i |\psi_i\rangle \langle\psi_i| \tag{1.22}$$

Considering states as linear compilations of basis states lead us to introduce the, rather counter intuitive, entangled states. Now, considering states as linear combinations of density matrices leads us to define mixed states which expand our repertoire of states yet further.

Definition 1.2.5. *If the state* ρ *cannot be expressed in terms of a single pure state then we call it a* mixed state.

Of particular use to us in this work is the observation that different ensembles give rise to the same density matrices. One will note that this is true for the particular case of expression (1.23) were $\mathbb{I}$ is the identity operator.

$$\frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| = \mathbb{I} = \frac{1}{2}|+\rangle\langle +| + \frac{1}{2}|-\rangle\langle -| \tag{1.23}$$

In fact, states having density matrices equal to the identity $\mathbb{I}$, as in expression (1.23), are referred to as *maximally mixed*.

This section requires us to re-express the unitary evolution, expectation value and measurement operations mentioned before.

- Unitary evolution: $\rho \rightarrow U\rho U^{\dagger}$.
- Expectation value: $\langle A \rangle_{\rho} = Tr(\rho A)$
- Measurement: $p(m) = Tr\left(M_m \rho M_m^{\dagger}\right)$ and the state become $\frac{1}{p(m)} M_m \rho M_m^{\dagger}$.

1.2.7 No Cloning Theorem

We bring this introduction to quantum computing to a close by mentioning one final theorem which we will use implicitly throughout. We refer the reader to [4] for the proof.

Theorem 1.2.2. *It is impossible to clone (i.e. to make two identical copies of) unknown quantum states.*

1.3 Single Party MBQC

We now introduce measurement based quantum computing (MBQC) which was mentioned in the introduction. The tools we discuss here are sufficient for the construction of a universal quantum computer. The reader should, however, keep in mind that these tools will be altered somewhat in later sections to add the blindness mentioned and to adapt to the IQP machine.

Measurement based quantum computing (MBQC) [14, 15, 16] is a method of performing quantum computations which uses measurement to drive the process. This is as opposed to the circuit model discussed in Section 1.2 and the adiabatic quantum computing model [22], for example. Recalling the randomness of measurements, as noted in postulate 3 of quantum mechanics, it seems surprising that it can be used as a tool for computation. However, MBQC and, in particular, blind quantum computing [17] which relies on it, are tools key to the development of our protocol. We introduce some background to MBQC now.

There are three steps to implementing a computation in MBQC.

1. A collection of qubits are set up in an entangled state.
2. Measurements are made on individual qubits. The results of measurements on some qubits may be used to determine measurements basis of others.

3. Depending on the outcome of these measurements, unitary operators, called corrections, may be applied to some qubits.

The entangled state required for a measurement based quantum computation has a description in the form of an *open graph*.

Definition 1.3.1. *A* graph $G=(V,E)$ *is a pair consisting of the vertex set, V, containing vertices, v_i, and edge set E described by an adjacency matrix. The* adjacency matrix*, E has a* 1 *in entry* (i,j) *if there is an edge from vertex v_i to the vertex v_j. This entry should be* 0 *if there is no edge. The adjacency matrix will be symmetric in the case that the graph is undirected.*

Definition 1.3.2. *An* open graph *is a triplet (G,I,O), where $G=(V,E)$ is an undirected graph, and $I,O\subseteq V$ are respectively called input and output vertices.*

A *uniformity condition* ensures that unreasonable computational power is not hidden in the description of the graph. This condition limits the graphs we consider to those that, for a given input, can be classically efficiently described. Hence each computation is associated with a *uniform* family of open graphs $\{(G_n,I_n,O_n)\}_n$ describing the initial entangled state. Often it is the case that the open graph depended only on the input size rather than the exact input.

An *MBQC graph state*, or just *graph state*, $|G\rangle$, is constructed as follows. Given an input state, corresponding to the input vertices of the graph, $|V|-|I|$ qubits in the state $|+\rangle$ are prepared and assigned a correspondence to non-input vertices I^c. A controlled-Z operation is then applied between two qubits if the corresponding vertices in the graph G are connected by the edge set E.

In our protocols we require a different process of building graphs state which we will introduce later but which only requires some alterations be made to the initial states of the non-input qubits. When the time comes, we must then also ensure that unreasonable computational power is used to determine the initial state of the qubits.

For each non-output qubit, j, there is a corresponding angle $\phi_j\in[0,2\pi)$ defining an initial measurement in the basis of (1.24).

$$\left\{\frac{1}{\sqrt{2}}\left(|0\rangle+e^{i\phi_j}|1\rangle\right),\frac{1}{\sqrt{2}}\left(|0\rangle-e^{i\phi_j}|1\rangle\right)\right\}. \tag{1.24}$$

These measurement basis may need to change as the computation proceeds and it is by building a structure of dependency into the measurements (i.e. measurement basis used to measure one qubit can depend on the outcome of measurements of another)

that MBQC overcomes the probabilistic nature of quantum computing to perform deterministic computations.

A measurement on a qubit i can be either X or Y dependent on s_j; the outcome of measuring qubit j. The actual measurement angle we use is then $(-1)^{s_j}\phi_i$ or $\phi_i + s_j\pi$ respectively. Formalising this dependency requires the introduction of the concept of flow.

Definition 1.3.3. *An open graph (G,I,O) has* flow *if there exists a map $f : O^c \to I^c$ from the measured qubits to the non input qubits and a partial order $\preceq$ over the qubits.*

Using the notation $N_G(i)$ to mean the vertices in G which neighbour the vertex i we have the following conditions.

1. $i \in N_G(f(i))$
2. $i \preceq f(i)$
3. $\forall j \in N_G(f(i))$ *we have* $i \preceq j$

The order of measurement should respect this partial order and one may choose to read $\preceq$ as 'in the past of'. Condition 3 ensures that there are no cyclic dependencies.

Each qubit k is X dependent on $f^{-1}(k)$ and Z dependent on all qubits l such that $k \in N_G(f(l))$. We now have an exact expression for the actual measurement angles that we should use.

$$\phi'_i = (-1)^{s_{f^{-1}(i)}}\phi_1 + \pi\left(\sum_{j:i\in N_G(f(j))} s_j\right) \tag{1.25}$$

This sequence of dependent measurements is referred to as an *MBQC measurement pattern*.

1.4 Introduction to Abstract Cryptography

The approach originally used to prove the security of early quantum key distribution protocols was to show the mutual information content[3], between the information gained by the eavesdropper and the key produced, is small. However, the BB84 [7] and E91 protocols [24], which satisfied this condition, where shown [25] to be secure only if the key is never used.

[3]Background in Information theory is not given her but can be found in [4] or [23]

It was later found [26] that proving the joint state of the final key and quantum information obtained by the eavesdropper was close to an ideal key, which is independent of the adversaries information, provided a better security criterion than the mutual information content. This idea was originally a classical technique [27, 28] but was adapted to quantum computing [29, 30] and has since been simplified by the introduction of abstract cryptography[31, 32].

We introduce some of the necessary methodology and definitions now. Much of this section relies on [32] and the reader will also find illustrative examples in that work. The work in that paper is re-framed here to change the setting to that of two party distributed computation and may be compared to [33, 34].

The intuition on which this technique is based is that an author, hoping to develop a secure protocol, will define first the *ideal functionality* of their resource which will complete perfectly, but without considering the details of the computation, the task in mind. The *real functionality*, which must take into consideration these details, can then be introduced and compared. By showing the two to be indistinguishable we prove they may be reasoned about interchangeably.

We will formalise this intuition and begin by defining a resource.

Definition 1.4.1. *An* I-resource *is an abstract system with interfaces specified by a set* I*. Each interface* $i \in I$ *is accessible to a user* i *and provides them with the ability to present inputs and read outputs.*

Resources are equipped with a parallel composition operator, $||$*, that maps two resources to another.*

The real functionality will be more complicated than the ideal one and could involve many rounds of communication between the *honest player* and *adversary*. The communication and computations are defined by the protocols π_A and π_B of each player respectively. Resources implementing the real and ideal functionality, called the real and ideal resource respectively, can be compared by observing Figure 1.4.

The approach of abstract cryptography is to abandon the notion of the eavesdropper's information in favour of the *distinguisher's* state. The distinguisher may pick the inputs of the honest players and collect their outputs while, all the time, playing the role of the adversary. In that way the distinguisher acts as everything outside of the honest player.

The distinguisher is given access to the either the real or ideal system and must decide which of the two it has in it's possession. If they are unable to do so then the

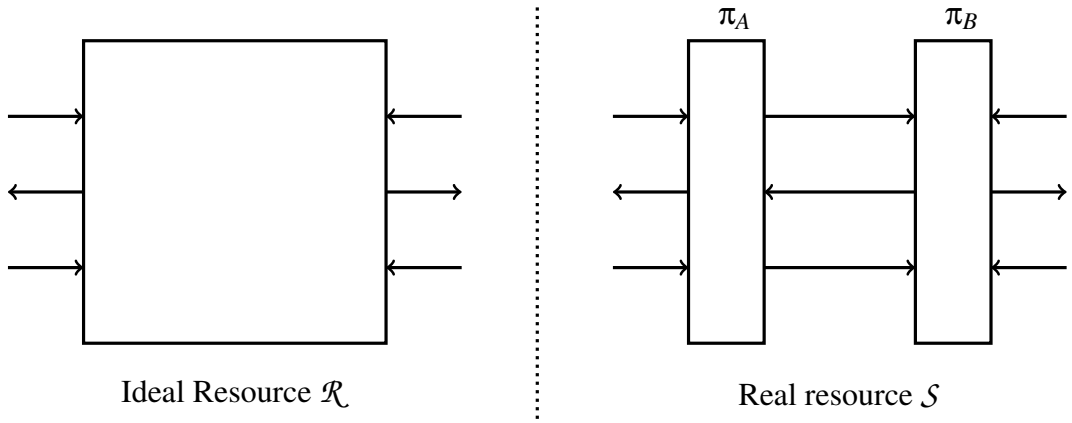

Figure 1.4: A comparison of the real, $\mathcal{S}$, and ideal, $\mathcal{R}$, resources. $\mathcal{R}$ does not have any consideration for the details of the computation while $\mathcal{S}$ must. The protocols π_A and π_B are used, in $\mathcal{S}$, to achieve the security of $\mathcal{R}$. The honest party is on the left while the adversary is on the right.

resources are indistinguishable. This can be visualised in Figure 1.5.

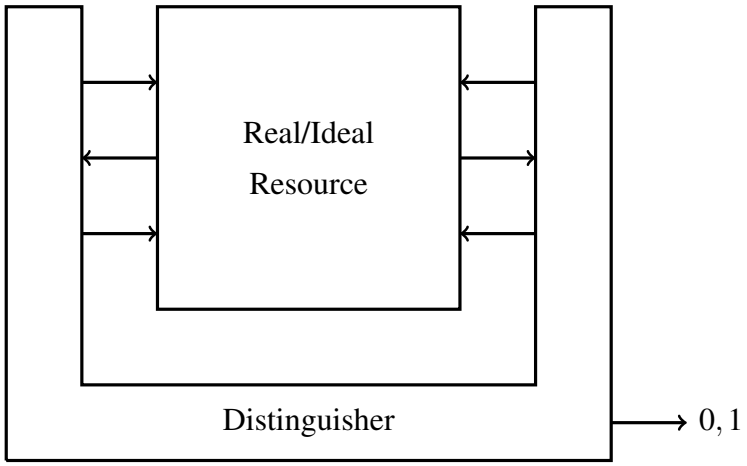

Figure 1.5: The distinguisher interacts with one of either the real and ideal resources with the aim of guessing which of the two the are interacting

Figure 1.6 illustrates that, as discussed, the distinguisher may interact with the communication during the protocol as well as the inputs and outputs of the adversary. As such the outputs of the real resource are more complicated than those of the ideal resource and we summarise this in Figure 1.7.

The distinguisher may than realise that they are interacting with a real system simply because the interface is different. To address this we introduce a simulator which acts as an interface between the ideal resource and the adversary and which produces outputs indistinguishable from the communication of the real resource.

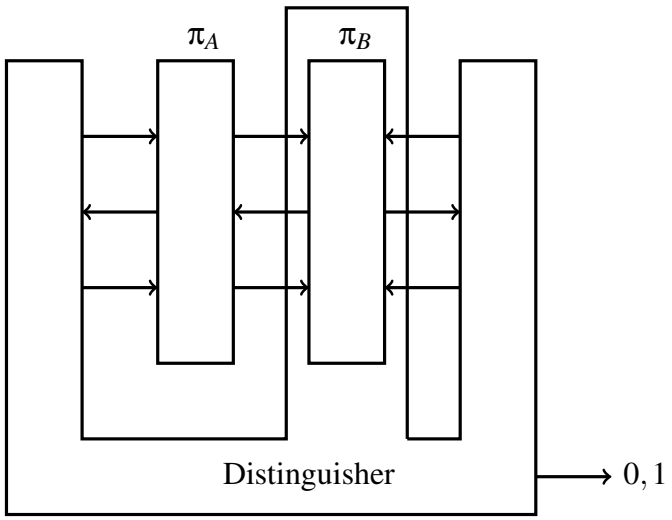

Figure 1.6: The distinguisher interacts with the real resource. The distinguisher has control over both parties inputs and outputs and has the responsibility of implementing the adversaries protocol, π_B. This fact is captured by the envelopment of this protocol by the box representing the distinguisher.

The simulator only learns that which is outputted by the ideal resource and so only weaken the adversary as they could reproduce the simulators behaviour on their own. Then, if the ideal resource along with the simulator is indistinguishable from the real resource, the real resource is considered secure as it does not reveal anything more than the ideal resource. We now formalise what is meant by a simulator in Definition 1.4.3 but first introduce a converter. The simulator may be visualised in Figure 1.8.

Definition 1.4.2. *A* Converter *transforms one resource into another. These are abstract systems with two interfaces, an inside interface and an outside interface. The inside interface connects to an interface of a resource and the outside becomes the new interface of the constructed resource.*

We write $\alpha_i\mathcal{R}$ to denote the new resource with the converter α connected to the interface i of the resource $\mathcal{R}$ and $\alpha\mathcal{R}$ for a set of converters $\alpha = \{\alpha_i\}_i$, for which it is clear to which interface they are connected.

Serial and parallel composition of converters is defined as in (1.27) *and* (1.26) *respectively.*

$$(\alpha\beta)_i\,\mathcal{R} := \alpha_i\,(\beta_i\mathcal{R}) \tag{1.26}$$

$$(\alpha||\beta)_i\,(\mathcal{R}||\mathcal{S}) := (\alpha_i\mathcal{R})\,||\,(\beta_i\mathcal{S}) \tag{1.27}$$

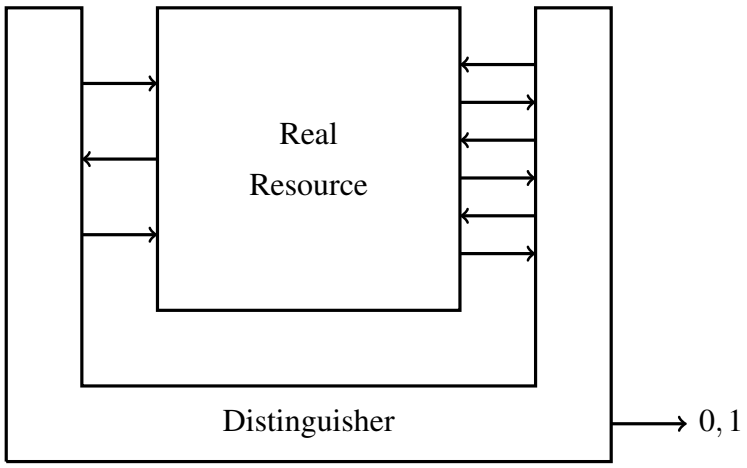

Figure 1.7: It was demonstrated in Figure 1.6 that the interactions between the distinguished and the real resource are more complicated than implied by Figure 1.5. This is summarised here with some of the redundant arrows implying communication with the distinguisher and itself having been removed.

Definition 1.4.3. *A* simulator, σ, *is a converter connected to the adversaries interface to the ideal system. It is defined by a set of operations* $(\sigma_1, ..., \sigma_t)$, *one for each step of the protocol. The simulator may call the ideal resource at any time.*

A final converter to be introduced before we can move to the formal definition of cryptographic security is the filter. The adversary having control of one half of the protocol means they may choose to deviate from it. We prevent this with a filter. We also use the filter to prevent the adversary from interacting with the communications during the computation.

Definition 1.4.4. *A* filter *is a converter which, when placed over the adversaries interface, prevents access to controls necessary to act maliciously and to anything other than the standard inputs and outputs.*

The reader will notice that the filter makes the simulator irrelevant as it is no longer necessary that the communications of the resource be reproduced. If the filtered ideal and filtered real resource are indistinguishable then we know that, in the case the adversary behaves honestly, the real resource is correct.

Definition 1.4.4 can, and will, be understood as defining a filter to prevent an adversary inputting deviation instructions. In this way we will always assuming that one of the adversaries inputs is instructions on how to deviate from honest behaviour. Blocking this interface then simulates honest behaviour.

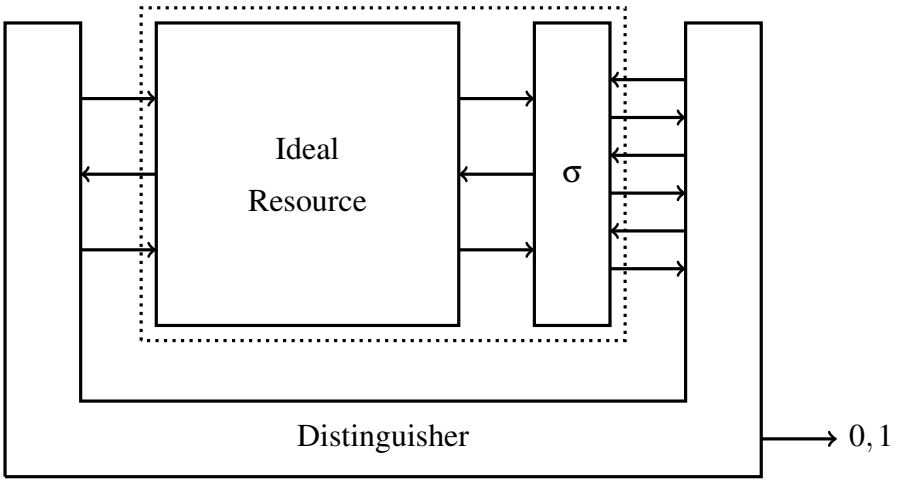

Figure 1.8: As noted in Figure 1.7 the interface between the real system is different from that of the ideal system alone. This is addressed with the definition of the simulator σ. Intuitively, the dotted region of this figure is now interfaced with in an equivalent way to the real resource of Figure 1.7.

Definition 1.4.5. *A pair,* $(\mathcal{R},\#)$*, of a resource* $\mathcal{R}$ *and a filter* # *together specify define a* filtered resource *which may be written* $\mathcal{R}_{\#}$.

Throughout this section we have spoken about the distinguisher ability to guess which of either the real or the ideal resource they are interacting with. Intuitively guessing correctly half of the time means the resources are indistinguishable and the real resource may be considered secure. To define this notion more formally we must define a metric which enumerates the degree to which resources are distinguishable. Actually we use a pseudo-metric, defined in Definition 1.4.6.

Definition 1.4.6. *A* pseudo-metric $d(\cdot,\cdot)$ *on the space of resources has the following properties. Consider the three resources* $\mathcal{R}$*,* $\mathcal{S}$*,* $\mathcal{T}$.

- $d(\mathcal{R},\mathcal{R}) = 0$
- $d(\mathcal{R},\mathcal{S}) = d(\mathcal{S},\mathcal{R})$
- $d(\mathcal{R},\mathcal{S}) \leq d(\mathcal{R},\mathcal{T}) + d(\mathcal{T},\mathcal{S})$

The pseudo metric we require is further restricted by demanding it is also non-increasing under composition with resources and converters. Intuitively, this is because a converter should not be able to make it easier to distinguish between two resources. This would otherwise mean the converter has added some information, separate from

the outputs of the resources, that helps with distinguishing them. So, for any converter α and resources $\mathcal{R}, \mathcal{S}, \mathcal{T}$, we require the relations of expression (1.28).

$$d(\alpha\mathcal{R}, \alpha\mathcal{S}) \leq d(\mathcal{R}, \mathcal{S}) \quad \text{and} \quad d(\mathcal{R}||\mathcal{T}, \mathcal{S}||\mathcal{S}) \leq d(\mathcal{R}, \mathcal{S}). \tag{1.28}$$

Finally we can formalise what it means to achieve cryptographic security.

Definition 1.4.7. *Let $\pi_{AB} = (\pi_A, \pi_B)$ be a protocol and $\mathcal{R}_{\#} = (\mathcal{R}, \#)$ and $\mathcal{S}_{\Diamond} = (\mathcal{S}, \Diamond)$ denote two filtered resources. we say that π_{AB} constructs $\mathcal{S}_{\Diamond}$ from $\mathcal{R}_{\#}$ within ε if the two following conditions hold.*

1. **Correctness condition:** *We have*

$$d(\pi_{AB}\mathcal{R}\#_E, \mathcal{S}\Diamond_E) \leq \varepsilon \tag{1.29}$$

2. **Security condition:** *There exists a converter σ_E - which we call simulator - such that*

$$d(\pi_{AB}\mathcal{R}, \mathcal{S}\sigma_E) \leq \varepsilon \tag{1.30}$$

The two conditions in definition 1.4.7 capture, respectively, the notions of correctness and security. Condition 1 tells us that, when the adversary behaves honestly, the resources are indistinguishable. I.e. the outcome of the protocol is correct if it is not deviated from. Condition 2 tells us that the resources are indistinguishable even if the adversary behaves maliciously. I.e. try as they might, the adversary cannot extract more information from the real resource than the ideal one.

We can now be more specific and define the pseudo-metric we will use.

Definition 1.4.8. *The* distinguishing advantage *that a computationally unbounded distinguisher, which can guess with probability $p_{distinguish}$ whether it is interacting with the resource $\mathcal{R}$ or $\mathcal{S}$, as is given by* (1.31).

$$d(\mathcal{R}, \mathcal{S}) := 2p_{distinguish}(\mathcal{R}, \mathcal{S}) - 1 \tag{1.31}$$

One may choose to define a weaker distinguisher than that of Definition 1.4.8 and, as a result, build a different metric. Here we are concerned with only information theoretic security and so we do not do this.

This definition adheres to our intuition that when the resources can be correctly identified with probability $p_{distinguish} = \frac{1}{2}$ the distinguisher has no knowledge of the resource they are interacting with. In that case the distinguishing advantage is $d(\mathcal{R}, \mathcal{S}) = 0$.

Chapter 2

The Instantaneous Quantum Polytime Machine

2.1 X-Programs and the IQP Oracle

In this section we formally introduce the IQP oracle first defined in [1]. This machine will be defined by its capacity to implement a so called *X-program* which we will now introduce.

Definition 2.1.1. *An X-*program *is characterised by a polysize*[1] *list of pairs* $(\theta_{\mathbf{p}}, \mathbf{p}) \in [0, 2\pi] \times \mathbb{F}_2^n$. *Each pair is interpreted as the action of a Hamiltonian given by the product of Pauli X operators. The product is over all qubits i for which* $\mathbf{p}_i = 1$ *and the Hamiltonian is applied for action* $\theta_{\mathbf{p}}$.

The program input is the computational basis state $|\mathbf{0}^n\rangle$. *The output is a classical vector* $\mathbf{x} \in \mathbb{F}_2^n$ *corresponding to the outcome of a computational basis measurement after all Hamiltonians have been applied.*

Throughout this work we will refer to the pairs $(\theta_{\mathbf{p}}, \mathbf{p}) \in [0, 2\pi] \times \mathbb{F}_2^n$ as *program elements*.

Using the random variable X to represent the distribution of output samples we are able to use definition 2.1.1 to derive equation (2.1) as the probability distribution for the outcomes.

$$\mathbb{P}(X = \mathbf{x}) = |\langle \mathbf{x}| \exp\left(\sum_{\mathbf{p}} i\theta_{\mathbf{p}} \bigotimes_{j:\mathbf{p}_j=1} X_j\right) |\mathbf{0}^n\rangle|^2 \tag{2.1}$$

[1] I.e. can be described by a classical Turing machine in polynomial time

As discussed, an IQP oracle is defined by its capability to implement an X-program. We now understand this as the ability to draw a sample from the distribution of (2.1). We formalise this now.

Definition 2.1.2. *Given some X-program an* IQP oracle *is any computational method capable of efficiently returning a sample* $\mathbf{x} \in \mathbb{F}_2^{\mathbf{n}}$ *from the probability distribution given in* (2.1).

In fact, in all applications of interest in this work, the action, $\theta_{\mathbf{p}}$, for each program element will be the same (i.e. $\theta_{\mathbf{p}} = \theta$ for all $\mathbf{p}$). As such we will talk about the action as applied to the the X-program as a whole while the reader should now understand the term *program element* to refer only to the individual vectors $\mathbf{p}$.

Say that there are m program elements, $\mathbf{p}$, each being of length n. An X-program can now be referred to simply by the pair $(\mathbf{P}, \theta) \in \mathbb{F}^{m \times n} \times [0, 2\pi]$ where $\mathbf{P}_i$, the i^{th} rows of the matrix $\mathbf{P}$, is the i^{th} program element.

We mention, although do not explore the reasoning, that it is widely believed [12] that a machine of this description is classically hard to simulate, amusing a non-collapse of the polynomial hierarchy to it's third level. This result is further strengthened by [13].

Similar results can be shown for the slightly different IQP* machine [35] which differs in the uniformity condition used. Where, as the reader will notice from Definition 2.1.1, the IQP machine of [1] allows the circuit to depend only on the input, in the case of the IQP* machine it depends on the input size.

2.2 The Hypothesis Test

In [1] the authors develop a *hypothesis test* designed as a way for a server to convince a classical client that they are in possession of an IQP oracle.

The idea used is that the client should conceal some structure in the program elements, $\mathbf{p}$, used to build an X-program defining an IQP-hard problem. This would then result in some known (to the client) structure in the distribution of the outputs from implementing said X-program. The server should, however, remain oblivious to the form of this structure.

A server possessing an IQP oracle can reproduce this structure by implementing the X-program while a server not in possession of an IQP oracle, being unable to return

samples from the correct distribution, could not generate outputs obeying the same rules. As such, they would expose themselves as a fraud.

We may summarise this discussion by three conditions that a hypothesis test using this method must meet.

1.1 The client asks the server to perform an IQP-hard computation.

By testing that the server solves an IQP-hard problem we ensure that they can solve all problems in the IQP class.

1.2 The client can check the solution to this computation because they know some secret structure that makes this checking processes efficient.

The intuition behind the contribution of this second point is that the client can efficiently check the output because they 'know where to look' for its important features. This is the point eluded to by the above discussion of hidden structure.

1.3 The server must be unable to uncover this structure in polynomial time.

Demanding that the server cannot access the secret structure prevents them from using it, rather than an IQP machine, to solve the problem. It should also be the case that there is no other piece of knowledge that the sever knows which helps them, thus making the IQP-hard computation unavoidable

During the remainder of this work we discuss hypothesis tests which use these conditions. One method, which can be found in chapter 4, is our own but is motivated by that of [1] which we recall this now.

The particular 'known structure' of the output which is used in [1] to satisfy condition 1.2 is its *bias* in some direction. We define this object in Definition 2.2.1.

Definition 2.2.1. *If X is a random variable taking values in $\mathbb{F}_2^n$ and $\mathbf{s} \in \mathbb{F}_2^n$ then the bias of X in the direction $\mathbf{s}$ is $\mathbb{P}(X \cdot \mathbf{s} = 0)$ where $\cdot$ is the usual dot product in $\mathbb{F}_2$.*

We may say then that the bias of a distribution in the direction $\mathbf{s}$ is the probability of a sample from the distribution being orthogonal to $\mathbf{s}$.

In [1] the authors are able to derive an expression for the value of the bias for the distribution of outputs of an X-program. This expression can be used to make predictions about the output from a true IQP oracle.

Before stating this quantity we must introduce some other mathematical objects. We refer the reader to [23] for an extensive study of the information theoretic objects discussed now.

Definition 2.2.2. *A* linear binary code*, $\mathcal{C}$, of length k is a linear subspace of the vector space $\mathbb{F}_2^k$. Elements $\mathbf{c} \in \mathcal{C}$ are called* codewords *and each has a* Hamming weight*, $wt(\mathbf{c}) \in [0,...,k]$, defined by the number of* 1*s it has as entries.*

Linear binary codes are frequently represented using a generator matrix*, $\mathbf{G}$, whose columns form a basis for the code. Then $\mathcal{C} = \{\mathbf{G}\cdot\mathbf{d} : \mathbf{d} \in \mathbb{F}_2^{\mathbf{k}}\}$.*

We can form a linear code $\mathcal{C}_\mathbf{s}$ from the combination of an X-program, $(\theta, \mathbf{P}) \in [0, 2\pi] \times \mathbb{F}_2^{m\times n}$, and a vector, $\mathbf{s} \in \mathbb{F}_2^n$, by selecting all rows, $\mathbf{P}_i$, of $\mathbf{P}$ such that $\mathbf{P}_i \cdot \mathbf{s}^\mathbf{T} = \mathbf{1}$ and forming the rows of the generator matrix, $\mathbf{P}_\mathbf{s}$, of $\mathcal{C}_\mathbf{s}$, from these $\mathbf{P}_i$. Defining $n_\mathbf{s}$ to be the number of such $\mathbf{P}_i$ (i.e. the number of rows of $\mathbf{P}_\mathbf{s}$) allows us to understand the following expression for the bias of the distribution of the outputs from an X-program. The derivation of this can be found in [1].

$$\mathbb{P}\left(X\cdot\mathbf{s}^T = 0\right) = \mathbb{E}_{\mathbf{c}\sim\mathcal{C}_\mathbf{s}}\left[\cos^2\left(\theta\left(n_s - 2\cdot wt\left(\mathbf{c}\right)\right)\right)\right] \tag{2.2}$$

Interestingly, we find that the probability of an output from an X-program being orthogonal to the vector $\mathbf{s}$ depends only on θ and the linear code defined by a generator matrix $\mathbf{P}_\mathbf{s}$.

One can imagine a hypothesis test which could be derived from these facts. Although the X-program, $(\theta, \mathbf{P})$, to be implement needs to be made public, the value of $\mathbf{s}$, and so which of those program elements are non-orthogonal to $\mathbf{s}$ (the ones that are important in calculating the bias value of expression (2.2)), need not be. This gives a client, with the computational power to calculate the quantity of expression (2.2), knowledge of the bias, but, assuming $\mathbf{s}$ cannot be derived from the X-program, does not afford the server the same privilege. In that case we hope the only way the servers could produce an output with the correct bias is to reproduce the distribution of outcomes in expression (2.1), i.e. to use an IQP oracle. If the server could, however, uncover $\mathbf{s}$ then they could calculate the value of the expression in (2.2) and return vectors to the client which are orthogonal to $\mathbf{s}$ with the correct probability.

Intuitively, we can now see the way in which $\mathbf{s}$ produces the required hidden structure and also tells us 'where to look' when checking the output. Without knowing $\mathbf{s}$ the server does not know where the client will look and so must be faithful everywhere.

We can now specialise the conditions we mentioned at the beginning of this section to this particular method.

2.1 The X-Program send to a server must represent an IQP-hard computation.

2.2 It must be possible for a client, having knowledge of $\mathbf{s}$ and the X-program, to calculate the quantity of expression (2.2).

2.3 Knowing the X-program to be implemented is not enough to learn the value of $\mathbf{s}$ in polynomial time forcing the server to perform the original IQP-hard computation in order to create the correct bias value.

Beginning with condition 2.2, let us now work through the solutions [1] provide to address these conditions.

Knowing the value of $\mathbf{s}$ is enough to produce, from the X-program $(\theta, \mathbf{P})$, the code $\mathcal{C}_{\mathbf{s}}$ used to calculate (2.2). In general, however, the calculation of (2.2) is not efficient as the expectation value is calculated over exponentially many terms. Ensuring that this calculation can be performed efficiently requires adding some structure to the X-program but the reader may notice that it is not given that this is achievable without revealing $\mathbf{s}$.

In [1] the authors develop a protocol, which we include in Algorithm 2.1, for building an X-program and vector $\mathbf{s}$ to be used in this type of hypothesis test. In that case the code $\mathcal{C}_{\mathbf{s}}$ in expression 2.2 is a quadratic residue code. They demonstrate that condition 2.2 is satisfied by calculating, for their choice of X-program and $\mathbf{s}$, that the bias value is $\cos^2\left(\frac{\pi}{8}\right)$ and, in doing so, prove that it can be calculated in polynomial time.

Condition 2.1 is, only in part, satisfied by their X-program by introducing a distribution which is a classically optimal simulation of the IQP distribution and showing that the server cannot recreate the same output properties as an IQP oracle. They calculate that a bias value of a distribution produced by such a simulation would be 0.75, differing from the $\cos^2\left(\frac{\pi}{8}\right)$ of an IQP oracle.

This dies not meet condition 2.1 as t only shows the problem to be outside of classical, and not necessarily IQP-hard. This is, however, progress as it demonstrates, at least, that the machine is more powerful than classical.

For the construction of the distribution we direct the reader to [1] but we refer to it by the random variable $\mathbf{Y}$ and include below the conjuncture which motivated its derivation.

Conjecture 2.2.1. *The random variable* $\mathbf{Y}$ *is asymptotically classically optimal (when compared with the worst-case behaviour and restricting to polynomial time) for the simulation of IQP distributions arising from constant-action* $\theta = \frac{\pi}{8}$ *X-programs.*

The contribution of our work is a method for addressing condition 2.3 in a way

which differers from that developed by [1]. We will first introduce their argument as many of the tools they develop we will employ in our work.

Consider expression (2.3) which should be compared to the right hand side of equation (2.2).

$$\mathbb{E}_{\mathbf{c}\sim\mathcal{C}}\left[\cos^2\left(\theta\left(k-2\cdot wt\left(\mathbf{c}\right)\right)\right)\right] \tag{2.3}$$

Suppose the linear binary code $\mathcal{C}$ has the generator matrix $\mathbf{G}\in\mathbb{F}_2^{k\times n}$ and notice the following two things.

- The expression depends only on the code $\mathcal{C}$. Importantly it is independent of $\mathbf{G}$ and one would even obtain the same result using a generator matrix with more columns than the rank of $\mathcal{C}$.
- For each code word $\mathbf{c}$, the only property used in calculating (2.3) is the number of zeros it contains. This would not change if, for example, we permuted the rows of $\mathbf{G}$.

Together theses two facts mean that the value of the quantity in (2.3) is actually equal for all generator matrices in a much larger collection of matrices. We define this collection as an equivalence class called a *matroid*.

Definition 2.2.3. *A k-point binary matroid is an equivalence class of matrices defined over $\mathbb{F}_2$ where each matrix in the equivalence class has k rows. Two matrices, M_1 and M_2, are said to be equivalent if, for some permutation matrix Q, the column echelon reduced form of M_1 is the same as the column echelon reduced form of $Q\cdot M_2$.*

We define equivalence in the case where the column dimensions do not match by deleting columns containing only 0*s after column echelon reduction and comparing the result.*

Throughout we will use $\mathbf{M}$ *to refer to the matrix itself and to the matroid to which it belongs.*

Hence, given a matrix $\mathbf{P}$, there are four operations we can perform on this matrix while remaining in the same matroid. Namely, we can permute its rows with each other, we can permute its columns with each other, we can add multiples of columns to one and other and we can add additional columns to the matrix if they are picked from the code generated by $\mathbf{P}$. As discussed, none of these operations change the value of expression (2.3) and so we call it *matroid invariant*.

In equation (2.2) we constructed our generator matrix, $\mathbf{P_s}$, from the rows of $\mathbf{P}$ which are non-orthogonal to $\mathbf{s}$. We now understand that we would not alter the value of the

quantity in expression (2.3) if we used a different matrix but remained in the same matroid as $\mathbf{P_s}$. If we performed a sequence of operations on the matrix $\mathbf{P_s}$ which did not cause us to leave its matroid it would not, however, be true that the rows of the new matrix would be orthogonal to $\mathbf{s}$. To address this we conduct the following discussion.

Suppose we had performed a sequence of operations on $\mathbf{P}$ in order to generate a new matrix which belongs to the same matroid as $\mathbf{P}$. We would like still to be able to locate a sub-matrix of this new $\mathbf{P}$ which is in the same matroid as $\mathbf{P_s}$. This is made possible by making the following observation. If we acted the invertible matrix $\mathbf{A}$ on $\mathbf{P}$ to move to a new matrix within the matroid then notice how $\mathbf{p}\cdot\mathbf{s}^T = \mathbf{p}\cdot\mathbf{A}\mathbf{A}^{-1}\cdot\mathbf{s}^T = (\mathbf{p}\cdot\mathbf{A})\left(\mathbf{s}\cdot\mathbf{A}^{-T}\right)^T$. We can see that rows which were originally non-orthogonal to $\mathbf{s}$ are now non-orthogonal to $\mathbf{s}\cdot\mathbf{A}^{-T}$ while the orthogonal rows are similarly orthogonal to $\mathbf{s}\cdot\mathbf{A}^{-T}$. Hence we can locate the matroid $\mathbf{P_s}$ in $\mathbf{P}$ but we need to use the vector $\mathbf{s}\cdot\mathbf{A}^{-T}$ and not $\mathbf{s}$.

One can now imagine that a way of ensuring that the server could not know $\mathbf{s}$ would be to randomise it with some set of operations $\mathbf{A}$. We now understand what we would have to do to the X-program we are considering in order to not change the value of the bias. This is the approach used by [1] and is the way in which they address condition 2.3.

We now have sufficient background to understand the approach of Algorithm 2.1 as introduced in [1]. We introduce it now before formally introducing the details of how it addresses condition 2.3.

Algorithm 2.1. Generation of X-program used in the hypothesis test of [1]

This protocol describes how a sceptical client should design an X-program that can be used to test a server's ability to perform IQP computations. This protocol concerns a vector s and the bias of the distribution produced by the X-program built with respect to $\mathbf{s}$ is known.

Input:

1. q (a prime) chosen so that $q+1$ is a multiple of 8.

Output:

1. X-program, $(\theta, \mathbf{P}) \in [0, 2\pi] \times \mathbb{F}_2^{2q\times\frac{q+1}{2}}$.
2. A vector, $\mathbf{s} \in \{0,1\}^{\frac{q+1}{2}}$.

Algorithm:

1. Generate the generator matrix, $\mathbf{G} \in \mathbb{F}_2^{q \times \frac{q+1}{2}}$, for a quadratic residue code, over $\mathbb{F}_2$, of length q.
2. To this matrix, the client appends an additional column consisting entirely of ones. Call this new matrix $\widehat{\mathbf{P}}_s \in \mathbb{F}_2^{q \times \left(\frac{q+1}{2}+1\right)}$.
3. Append q additional rows to the matrix which have random entries with the exception of the last entry (the final column) which must contain 0. Call this new matrix $\widehat{\mathbf{P}} \in \mathbb{F}_2^{2q \times \left(\frac{q+1}{2}+1\right)}$.
4. Reorder the rows of $\widehat{\mathbf{P}}$ randomly.
5. Column reduce this new matrix to obtain $\mathbf{P}$.
6. Output the X-program $\left(\frac{\pi}{8}, \mathbf{P}\right)$.
7. Suppose this reordering of rows and column reduction can be achieved by applying the matrix $\mathbf{A} \in \mathbb{F}_2^{\left(\frac{q+1}{2}+1\right) \times \left(\frac{q+1}{2}+1\right)}$. Return the vector $\mathbf{s} = (0,...,0,1)\,\mathbf{A}^{-T} \in \{0,1\}^{\frac{q+1}{2}}$.

We now justify that, in the setting of Algorithm 2.1, condition 2.3 is met and that the server could not know the value of $\mathbf{s}$. This requires us to be able to say that we could not find the matrix $\mathbf{P_s}$ in the matrix $\mathbf{P}$ given only; the matrix $\mathbf{P}$, an understanding of the randomisation process and that the matrix $\mathbf{P_s}$ is in the same matroid as a generator matrix for the quadratic residue code. More formally we would like to be able to say that it is NP complete to decide whether or not $\mathbf{P}$ contains a $\mathbf{P_s}$ of the appropriate form. The authors of [1] make the following conjecture.

Conjecture 2.2.2. *The language of matroids P that contain a quadratic residue code submatroid Q by point deletion, where the size of Q is at least half the size of P, is NP complete under polytime reduction.*

Their reasoning for making this conjecture is that it is similar to following theorem which is a classical result and may be found in [37].

Theorem 2.2.1. *The language of graphs G that contain a complete graph K by vertex deletion, where the size of K is at least half the size of G, is NP complete under polytime reduction.*

In this work we move to developing a protocol based on information theoretic assumptions rather than assumptions based on the computational complexity of the hidden matroid problem. We proceed towards this goal in the following sections.

However, for completeness and future reference we summarise the solutions found by [1] for conditions 1.1, 1.2 and 1.3.

3.1 It was shown, for quadratic residue codes, that the bias of the output of an IQP machine is significantly different to that produced by the best known classical sampling technique. This point relies on Conjecture 2.2.1.

3.2 It was shown that the value of the bias, in the case of the X-program developed in Algorithm 2.1, can be calculated explicitly to be $\cos\left(\frac{\pi}{8}\right)$.

3.3 The randomisation process of Algorithm 2.1 conceals which parts of the X-program are important. This point relies on Conjecture 2.2.2.

It is now simply a matter for the server to implement the X-program and for the client to check the outputs are orthogonal to $\mathbf{s}$ with the correct probability. To achieve this the X-program generation and implementation must be completed many times.

The complete hypothesis test would look roughly like indicated by Figure 2.1.

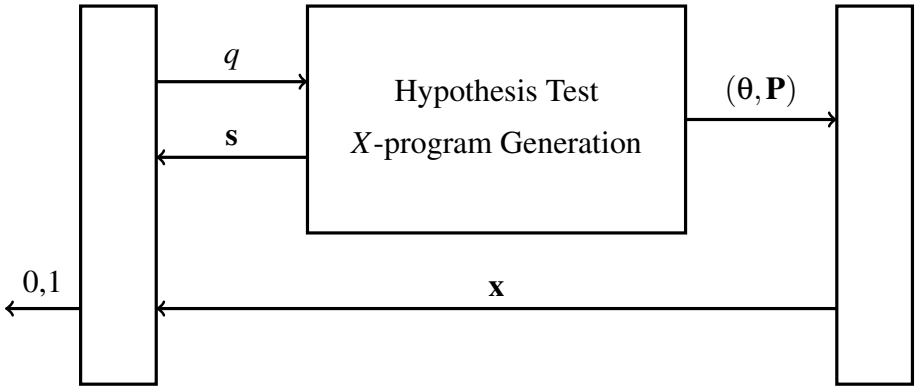

Figure 2.1: One round of the hypothesis test algorithm of [1]. The Hypothesis Test X-program Generation resource is as described in Algorithm 2.1 while the protocol of the server should be to implement said X-program and return the result, $\mathbf{x}$. The client then checks if the result is orthogonal to their vector $\mathbf{s}$.

2.3 IQP In MBQC

In this section we work to understand the implementation of an X-program using measurement based quantum computing.

Recall, definition 2.1.2 taught us the aim of an IQP oracle is to be able to recreate the distribution of outputs seen in equation (2.1). An equivalent distribution can be derived from using the relationship in equation (2.4) which uses H_n to represent the n-qubit Hadamard operator.

$$\exp\left(i\theta \otimes_{j:\mathbf{p}_j=1} X_j\right) = H_n \exp\left(i\theta \otimes_{j:\mathbf{p}_j=1} Z_j\right) H_n \tag{2.4}$$

We now come to realise the equality of equation (2.5).

$$\exp\left(\sum_{\mathbf{p}} i\theta_{\mathbf{p}} \otimes_{j:\mathbf{p_j}=\mathbf{1}} X_j\right) = H_n \left(\prod_{\mathbf{p}\in\mathbf{P}} \exp\left(i\theta \otimes_{j:\mathbf{p}_j=1} Z_j\right)\right) H_n \tag{2.5}$$

This, in tern, shows us that the distribution in equation (2.1) can be equivalently represented by equation (2.7) where we denote by $\tilde{\mathbf{x}}$ the vector $H_n\mathbf{x}$ for $\mathbf{x}$ a computational basis state (i.e. $\tilde{\mathbf{x}}$ is a n qubit Hadamard basis state).

$$\mathbb{P}(X = \mathbf{x}) = \left|\langle \mathbf{x}| H_n \left(\prod_{\mathbf{p}\in\mathbf{P}} \exp\left(i\theta \otimes_{j:\mathbf{p}_j=1} Z_j\right)\right) H_n |\mathbf{0}^n\rangle\right|^2 \tag{2.6}$$

$$= \left|\langle \tilde{\mathbf{x}}| \left(\prod_{\mathbf{p}\in\mathbf{P}} \exp\left(i\theta \otimes_{j:\mathbf{p}_j=1} Z_j\right)\right) |+^n\rangle\right|^2 \tag{2.7}$$

We proceed by developing a circuit which produces distribution of (2.7). As a first step towards doing so we will derive an implementation the unitary of (2.8).

$$\exp\left(i\theta \otimes_{j:\mathbf{p}_j=1} Z_j\right) \tag{2.8}$$

Lemma 2.3.1. *The circuit seen in figure 2.2 implements the an example unitary of the form seen in expression* (2.8).

In the proof of this lemma we will use the measurement basis and notation of expression (2.9).

$$\left\{\frac{1}{\sqrt{2}}\left(exp\left(-i\theta\right)|+\rangle + exp\left(i\theta\right)|-\rangle\right), \frac{1}{\sqrt{2}}\left(exp\left(-i\theta\right)|+\rangle - exp\left(i\theta\right)|-\rangle\right)\right\} = \left\{|0_\theta\rangle, |1_\theta\rangle\right\}. \tag{2.9}$$

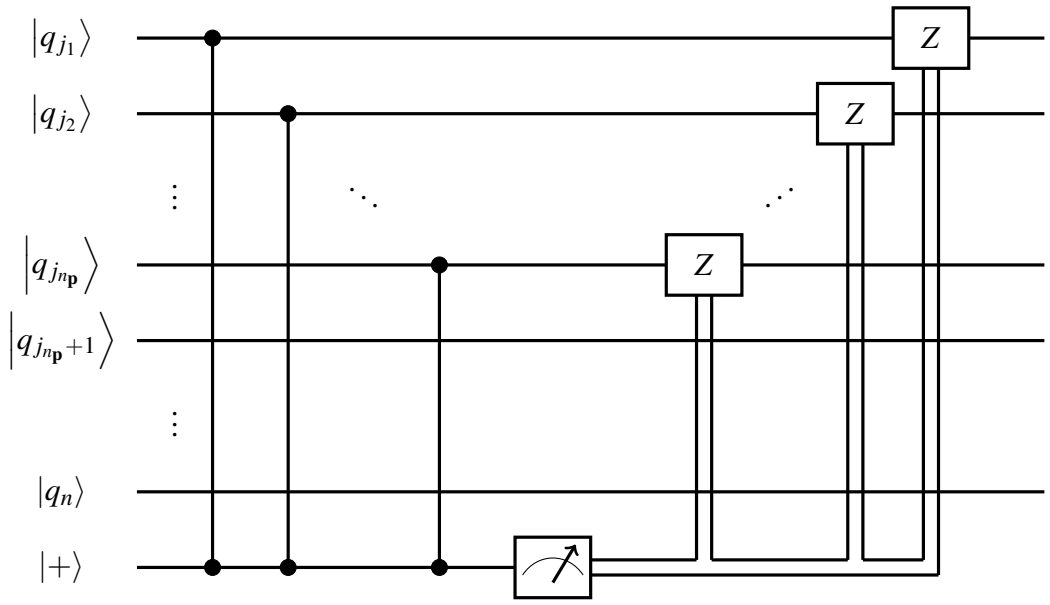

Figure 2.2: Circuit to implement (2.8). $\mathbf{p}_{j_i} = 1$ for $i \in \{1,...,n_\mathbf{p}\}$, $\mathbf{p}_{j_i} = 0$ for $i \in \{n_\mathbf{p}+1,...,n\}$ and $n_\mathbf{p}$ is the number of 1s in $\mathbf{p}$. $\{|q_1\rangle,...,|q_n\rangle\}$ is the set of all input qubits and the the measurement is in the basis $\{|0_\theta\rangle,|1_\theta\rangle\}$. We refer to the $|+\rangle$ state on the bottom wire in figure 2.2 as an ancilla qubit.

Proof. We develop a complete understanding of the behaviour of the unitary in expression (2.8) by understanding its effect on basis states. As such we will consider only the case of computational basis inputs in the following.

Notice that, representing the n qubit identity operator by $\mathbb{I}_n$, we can rewrite expression (2.8) in the following way.

$$\mathbb{I}_n \cos\theta + i \otimes_{j:\mathbf{p}_j=1} Z_j \sin\theta \tag{2.10}$$

If we act on a computational basis state $|\phi\rangle$ with the operator of expression (2.10) then one of two things will happen.

1. If the state $|\phi\rangle$ has an even number of 1s in the locations $j_1,...,j_{n_\mathbf{p}}$ then there will be a phase change of $\cos\theta + i\sin\theta$ as the $\otimes_{j:\mathbf{p}_j=1} Z_j$ operator will extract an even number of negatives.

2. If the number of 1s is odd then the phase change will be $\cos\theta - i\sin\theta$.

As such, the effect is to produce one of the two states in expression (2.11) depending on the parity of entries $j_1,...,j_{n_\mathbf{p}}$ of the input.

$$(\cos\theta \pm i\sin\theta)|\phi\rangle = \exp(\pm i\theta)|\phi\rangle \tag{2.11}$$

We now need to understand if the effect of the circuit in Figure 2.2 is the same. The action of the controlled-Z gates is to check the parity of the input as each appearance of a 1 will flip the ancilla qubit between the states $|+\rangle$ and $|-\rangle$. Hence, after the action of all controlled-Z operators we have the state $|\phi\rangle|+\rangle$ if there is an even number of 1s in the locations $j_1, ..., j_{n_\mathbf{p}}$ and $|\phi\rangle|-\rangle$ if this number is odd.

Making a measurement of the ancilla qubit in the described basis leaves us with one of the two states $\pm\exp(-i\theta)|\phi\rangle$ in the odd parity case and the state $\exp(i\theta)|\phi\rangle$ in the even parity case. The negative sign preceding the exponential term in the odd parity case comes from measuring the state $|1_\theta\rangle$ (a measurement outcome of 1) and the positive sign comes from measuring $|0_\theta\rangle$. By then applying Z operators to all unmeasured qubits in the case of a measurement outcome of 1 we ensure that the resulting states are as in expression (2.11) and with the same dependency of the sign on the parity of $|\phi\rangle$ as before. □

Producing the distribution in equation (2.7) (and so equivalently, that of equation (2.1)) can be achieved by inputting the state $|+^n\rangle$ into a circuit made from composing circuits of the form seen in Figure 2.2 (one for each term in the product of (2.7)) and measuring the result in the Hadamard basis.

Lemma 2.3.2. *A graph and measurement pattern can always be designed to simulate a X program efficiently.*

Proof. Algorithm 2.2 gives a description of a process generating the necessary graph and measurement pattern. This follows, in part, from the above discussion and, in particular, lemma 2.3.1.

The creation of a three step entanglement, measurement and correction process is possible as the Z corrections commute with the controlled-Z operations so can be moved to the end of the new, larger circuit (i.e. that created by composing multiple circuits of the form seen in Figure 2.2).

Notice also that without dependency of the measurements on each other they can be performed in any order (and so simultaneously). The Z corrections, conditional on the measurement outcomes, are then implemented via bit flips, after the measurements. □

Algorithm 2.2. MBQC implementation of X-program.

Here we develop an MBQC graph and measurement pattern to implement the X-program, $(\theta, \mathbf{P})$. This method of implementation differs from the discussion of Section 1.3 in the measurement basis used. As such we will return the measurement basis itself and not simple an angle as before. This time there is no input and so we consider the open graph (G, O).

- **Input:**
  1. The X-program $(\theta, \mathbf{P})$.
- **Output:**
  1. The graph $G = (V, E)$.
  2. The output set O.
  3. The measurement basis for each qubit.

- **Algorithm:**
  1. The vertex set V contains $p_1, ..., p_{n_P}, a_1, ..., a_{n_A}$
  2. The entry (i, j) of $\mathbf{P}$ indicates if qubit p_j is connected to a_i. This may easily be re-interpreted as an adjacency matrix E.
  3. The measurement basis for qubits $p_1, ..., p_{n_P}$ is the Hadamard basis.
  4. Using the notation of expression (2.9), the measurement basis for the qubits $a_1, ..., a_{n_A}$ is $\{|0_\theta\rangle, |1_\theta\rangle\}$.
  5. The output set is $O = \{p_1, ..., p_{n_P}\}$.

We have the following useful definition which describes the type of graph built in Algorithm 2.2. This form of graph may be visualised in Figure 2.3.

Definition 2.3.1. *A particular type of graph is the* bipartite graph *which has a partition of its vertices. We have that* $V = \{P, A\}$ *where* $P = \{p_1, ..., p_{n_P}\}$ *and* $A = \{a_1, ..., a_{n_A}\}$. *The adjacency matrix E describes connections between these sets of vertices but connections within them are not permitted.*

An undirected bipartite graph *may be described by the matrix* $\mathbf{P} \in \mathbb{F}_2^{n_A \times n_P}$ *where entry* (i, j) *is* 1 *if there is a connection between vertices* a_i *and* p_j.

From here on we will use the term IQP graph states or bipartite graph states in the same way we spoke about MBQC graph states.

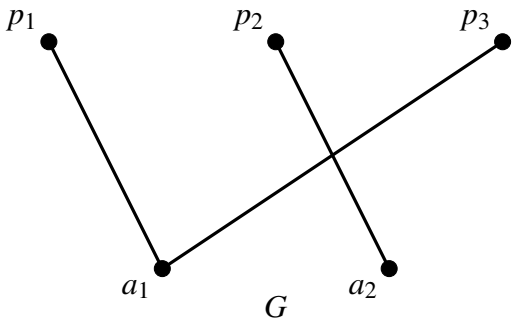

Figure 2.3: An example of the bipartite graph introduced in Definition 2.3.1. Here $n_p = 3$ and $n_a = 2$ while the partition used is $P = [p_1, p_2, p_3]$ and $A = [a_1, a_2]$.

Chapter 3

Distributed Blind IQP Graph State Generation

In Section 2.3 we developed a method for one party to implement an IQP X-program. That method used an MBQC graph state described by a bipartite graph and one round of measurement as outlined in Algorithm 2.2.

In this Chapter we discuss a different method, based on that of Section 2.3, but with two major differences. Firstly, the computation will be performed in a distributed setting with a client of limited quantum power requesting a computation from a more powerful server. Secondly, the method will be such that it allows the implementation of an X-program even when it's full description is kept secret from the server.

The X-program is kept secret by hiding the graph state used. The intuition exploited by the technique we develop to perform this hiding is that the client should produce a quite general graph state and move, without the server's knowledge, from this to the one they require for the computation. If the movement is done secretly then the server only has knowledge of the general starting state from which any number of other graph state may have been built.

The complication is that the creation of the general graph state and the movement to the new state should be done on the server's side. Hence, there are two key problems.

1. Moving from a general graph state to another more specialised one.

2. How to do so secretly when in a distributed setting.

These two problems are addressed by Section 3.1 and Section 3.2 respectively. In neither of these sections will we specialise to IQP graph states, instead focusing on all

forms of graphs. In section 3.3 we do, with some minor alterations to the techniques developed in the sections preceding it, restrict to the generation of IQP graph states.

3.1 Bridge and Break Operations

The bridge and break operations [18] are exactly those necessary to solve problem 1 from this chapter's introduction. We begin by introducing these operations on graphs and later define actions on the corresponding graph states which replicate their effect.

Definition 3.1.1. *The* break *operator acts on a vertex $v \in V$ of degree 2 in a graph G. It takes G to the graph G_{break} with a vertex and edge set equivalent to that of G with the exception that v has been removed from the vertex set and edges connected to v have been removed from the edge set.*

The bridge *operator has the same effect on G with the exception that the graph created, G_{bridge}, also includes a new edge between the neighbours of v.*

Some intuition about the behaviour of these operations can be gained from studying Figure 3.1.

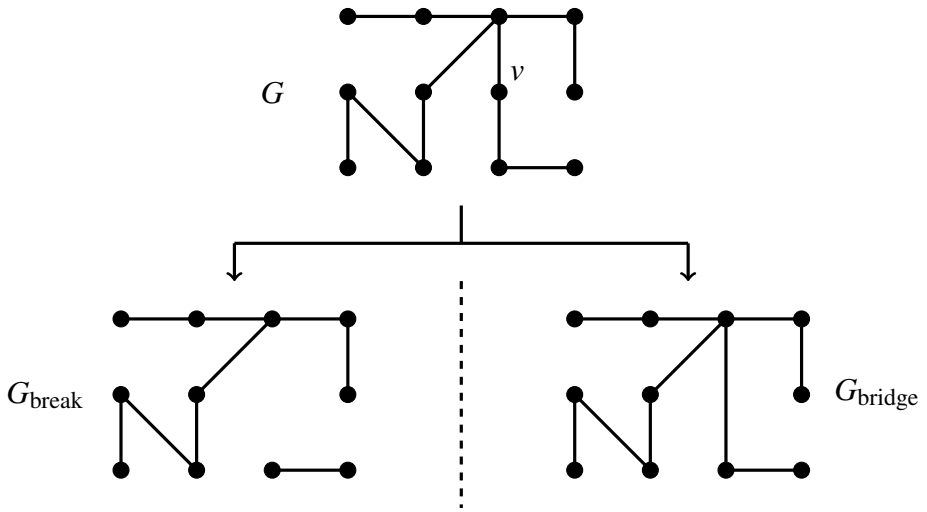

Figure 3.1: The effect of the bridge and break operations defined in Definition 3.1.1. The bridge or break operation is applied to the vertex v of the graph G and results in the graph G_{bridge} or G_{break} respectively.

Once a new graph is formed, through a break or bridge operation on another, one may continue and apply bridge or break operations to this newly formed graph. Indeed, this may continue to many levels. In order to refer to the resulting graph we introduce the br-sub-graph and br-sup-graph.

Definition 3.1.2. *A* br-sub-graph*, G, of a graph, $\widetilde{G}$, is one which can be obtained by applying break or bridge operation to the graph $\widetilde{G}$ and the graphs which result from these operations. The vertices to which bridge and break operations are applied must not neighbour each other in the original graph $\widetilde{G}$.*

Similarly, a br-sup-graph*, $\widetilde{G}$, of a graph, G, is such that G is a br-sub-graph of $\widetilde{G}$.*

Figure 3.2 allows the reader to develop some intuition about definition 3.1.2 by presenting a possible chain of bridge and break operations. Notice then that any given graph may have many br-sub-graphs and may be a br-sub-graph of many graphs.

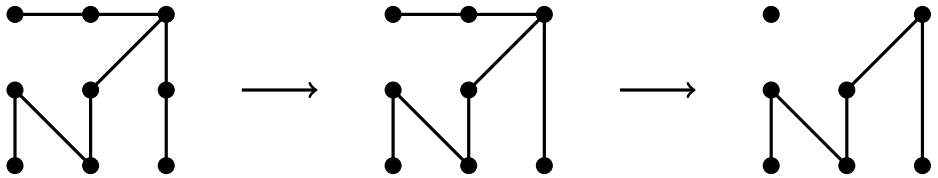

Figure 3.2: A possible chain of bridge and break operations. Each graph in the chain is a br-sub-graph of its predecessors while also being a br-sup-graph of its successors.

The use of the $\widetilde{G}$ notation to mean that of which G is a br-sub-graph will be used throughout. We will also index the vertices of the graph G with $v_1,...,v_n$ and those of $\widetilde{G}$ with $\widetilde{v}_1,...,\widetilde{v}_n,b_1,...,b_m$. The break and bridge operations are applied to the vertices $b_1,...,b_m$ and once the chain of operations is complete the resulting graphs are equivalent using the obvious mapping, $\widetilde{v}_i \leftrightarrow v_i$.

We will refer to the vertices b_i as bridge or break vertices and the v_i vertices as stationary vertices. From here on, we will consistently use n to mean the number of stationary vertices and m to mean the number of bridge or break vertices. When it comes to talking about the graph state $\left|\widetilde{G}\right\rangle$ we will use the terminology bridge and break qubits and stationary qubits.

Algorithm 3.1 describes operations on a graph state which reproduce the effects of performing a bridge or break operation on the corresponding graph. This algorithm is the one employed to prove lemma 3.1.1. The statement and proof of that lemma is similar to that of [18].

Lemma 3.1.1. *Take a graph state $|G\rangle$ with a graph structure given by G. Suppose that G is a br-sub-graph of the graph $\widetilde{G}$.*

There is a graph state $\left|\widetilde{G}\right\rangle$, with graph structure given by $\widetilde{G}$, which can be transformed into the graphs state $|G\rangle$ through a sequence of Pauli-Y measurements and local rotations about the Z axis through angles from the set $\left\{0,\frac{\pi}{2},\pi,\frac{3\pi}{2}\right\}$.

Proof. This proof is by construction. We will define a scheme for building the a graph state $\left|\widetilde{G}\right\rangle$ meeting the conditions of the lemma.

We begin by considering the two simpler cases but in both $m = 1$. The graph G can be built, in the first case, by applying a break operation to the vertex b_1 in the graph $\widetilde{G}$, and, in the second case, by applying a bridge operation to this same vertex. Hence, we are only consider cases where G can be arrived at by one of either a bridge or break operation on $\widetilde{G}$.

We also assume, without loss of generality, that the vertices $\widetilde{v}_1$ and $\widetilde{v}_2$ are the neighbours of b_1. The reader should also note that we make no assumptions about the state $|G\rangle$ other than the pattern of entanglement.

We now move to consider the two separate cases.

- **Break:**

  In this case the graph G can be arrived at by applying a break operation to the vertex b_1 in the graph $\widetilde{G}$.

  We claim that the state seen in expression (3.1), where $|b_1\rangle = |d\rangle$ with $d \in \{0,1\}$ (i.e a computational basis state), is the necessary state fulfilling the conditions of the lemma.

$$\left|\widetilde{G}\right\rangle = Z_{v_1,b_1} Z_{v_2,b_1} |G\rangle |b_1\rangle \tag{3.1}$$

  The reader will quickly notice that the graph $\widetilde{G}$ describing the structure of the state in (3.1) does indeed require only a break operation in the vertex b_1 to be transformed into the graph G.

  Applying the controlled-Z operations, Z_{v_1,b_1} and Z_{v_2,b_1}, as seen in expression (3.1), is equivalent to applying the operator Z^d to each of the qubits v_1 and v_2. We can conclude the identity in equation (3.2).

$$\left|\widetilde{G}\right\rangle = Z_{v_1}^d Z_{v_2}^d |G\rangle |b_1\rangle \tag{3.2}$$

  Measuring the qubit b_1 in the Pauli-Y basis causes a collapse to either of the Pauli-Y basis states with equal likelihood. It does not, however, have any other effect on the state as the qubit b_1 is disentangled.

  Discarding qubit b_1 then leaves us with $Z_{v_1}^d Z_{v_2}^d |G\rangle$ which differs from $|G\rangle$ only by local rotations about the Z axis.

- **Bridge:**

  In this case the graph G can be arrived at by applying a bridge operation to the vertex b_1 in the graph $\widetilde{G}$.

  We claim that the state seen in expression (3.3), where $|b_1\rangle = |d\rangle$ with $d \in \{+,-\}$ (i.e a Hadamard basis state), is the necessary state fulfilling the conditions of the lemma.

$$\left|\widetilde{G}\right\rangle = Z_{v_1,b_1} Z_{v_2,b_1} Z_{v_1,v_2} |G\rangle |b_1\rangle \tag{3.3}$$

  The reader will again notice that the graph $\widetilde{G}$ requires only a bridge operation in the vertex b_1 to be transformed into the graph G.

  Applying the controlled-Z operations, Z_{v_1,b_1} and Z_{v_2,b_1} in expression (3.3), to the state $Z_{v_1,v_2} |G\rangle |d\rangle$ is equivalent to applying the operator of expression (3.4) to the state $Z_{v_1,v_2} |G\rangle$.

$$\frac{1}{\sqrt{2}} |0\rangle \otimes \mathbb{I}_{v_1} \otimes \mathbb{I}_{v_2} + d \frac{1}{\sqrt{2}} |1\rangle \otimes Z_{v_1} \otimes Z_{v_2} \tag{3.4}$$

  Following this with measurement of qubit b_1 in the Pauli-Y basis, and a discarding of the resulting qubit, is equivalent to applying the operator of expression (3.5) to $Z_{v_1,v_2} |G\rangle$. In expression (3.5) we have used the notation that $y_1 = 0$ when the the positive Y basis state is measured and $y_1 = 1$ when the the negative Y basis state is measured. The expression results from post multiplication of expression 3.4 by $\frac{1}{\sqrt{2}} \langle 0| + (-1)^{1-y_1} i \frac{1}{\sqrt{2}} \langle 1|$, the conjugate of the Pauli-Y basis states, followed by the appropriate normalisation.

$$\frac{1}{\sqrt{2}} \mathbb{I}_{v_1} \otimes \mathbb{I}_{v_2} + (-1)^{1-y_1} di \frac{1}{\sqrt{2}} Z_{v_1} \otimes Z_{v_2} \tag{3.5}$$

  The reader will notice that y_1 takes either of the values 0 or 1 with equal probability. For completeness and future reference we note that the state $\left|\widetilde{G}\right\rangle$ has become, after measurement, the stare of equation 3.6.

$$\left(\frac{1}{\sqrt{2}} \mathbb{I}_{v_1} \otimes \mathbb{I}_{v_2} + (-1)^{1-y_1} di \frac{1}{\sqrt{2}} Z_{v_1} \otimes Z_{v_2} \right) Z_{v_1,v_2} |G\rangle \tag{3.6}$$

  We now break the flow of this calculation somewhat and notice that the controlled-Z operator can be written as is shown in equation (3.7). This may be verified by matrix

multiplication or otherwise.

$$Z_{1,2} = \frac{1}{2}\left(\mathbb{I}_1 \otimes \mathbb{I}_2 + Z_1 \otimes \mathbb{I}_2 + \mathbb{I}_1 \otimes Z_2 + Z_1 \otimes Z_2\right). \tag{3.7}$$

Using this fact allows us to see that both of the expressions in (3.8) and (3.9) are equal to the controlled-Z operation of expression (3.7). Again this can be verified by matrix multiplication.

$$Z_{1,2} = (S_1 \otimes S_2)\left(\frac{1}{\sqrt{2}}\mathbb{I}_1 \otimes \mathbb{I}_2 + i\frac{1}{\sqrt{2}}Z_1 \otimes Z_2\right) \tag{3.8}$$

$$Z_{1,2} = \left(S_1^{-1} \otimes S_2^{-1}\right)\left(\frac{1}{\sqrt{2}}\mathbb{I}_1 \otimes \mathbb{I}_2 - i\frac{1}{\sqrt{2}}Z_1 \otimes Z_2\right) \tag{3.9}$$

In particular, equation (3.10) holds.

$$Z_{v_1,v_2} = \left(S_{v_1}^{(-1)^{1-y_1}(-d)} \otimes S_{v_2}^{(-1)^{1-y_1}(-d)}\right)\left(\frac{1}{\sqrt{2}}\mathbb{I}_{v_1} \otimes \mathbb{I}_{v_2} - (-1)^{1-y_1}di\frac{1}{\sqrt{2}}Z_{v_1} \otimes Z_{v_2}\right) \tag{3.10}$$

Substituting this into (3.6), and with some rearranging, we realise the resulting state is actually that of equation (3.11).

$$\left(S_1^{(-1)^{y_1}d} \otimes S_2^{(-1)^{y_1}d}\right)|G\rangle \tag{3.11}$$

Once again this differs from the state $|G\rangle$ only by local rotations around the Z axis.

We now turn to the extension to more general G and $\widetilde{G}$. In this case the number of break and bridge operations needed to move from $\widetilde{G}$ to G is more than one.

The reader will notice that this problem could be solved by building the state $\left|\widetilde{G}\right\rangle$ a step at a time by repeating the steps above. This would, indeed build a graph state described by the graph $\widetilde{G}$.

The proof that the state resulting from measurements of the qubits $b_1, ..., b_m$ would result in the graph $|G\rangle$ follows for the following reason. The reader will notice that, since those qubits that might require corrections are never measured, all measurements and corrections commute. The entanglement operators too commute with the corrections and the measurement operations when they do not act upon the same qubits. As such all operations commute and all the necessary entanglement operation can be done at once, all the necessary measurement operations can be done at once and all the necessary corrections can be done at one. □

We finish this section by introducing Algorithm 3.1 which uses the methods discussed in the proof of lemma 3.1.1 to build the graph state $\left|\widetilde{G}\right\rangle$.

Algorithm 3.1. Br-sup-graph state creation.

Let the graph G, with vertices $v_1,...,v_n$, describe the entanglement pattern in the graph state $|G\rangle$. Suppose further that G is a br-sub-graph of the graph $\widetilde{G}$, whose vertices are indexed by $\widetilde{v}_1,...,\widetilde{v}_n,b_1,...,b_m$.

Lemma 3.1.1 tells us that there exists a choice of qubits $|b_1\rangle,...,|b_m\rangle$ such that Pauli-Y basis measurements of these qubits, once entangled in the state $\left|\widetilde{G}\right\rangle$, recreates the effect of the bridge and break operations. In doing so this creates the state $|G\rangle$ up to some local rotations about the Z axis through angles from the set $\left\{0,\frac{\pi}{2},\pi,\frac{3\pi}{2}\right\}$.

In this protocol we introduce the vectors $w\in\{0,1\}^m$ and $z\in\{0,1\}^m$. The vector w_i indicates if a break or bridge operations should be applied to the vertex b_i in the process of building G from $\widetilde{G}$. If a break operations is to be performed on the vertex b_i then $w_i=0$ while if it is a break then $w_i=1$. This means if $w_i=0$ then $|b_i\rangle$ is drawn from the set $\{|0\rangle,|1\rangle\}$ while if $w_i=1$ then it is drawn from the set $\{|+\rangle,|-\rangle\}$. The vector z isolates exactly the state of $|b_i\rangle$. If $z_i=0$ then $|b_i\rangle$ is drawn from the set $\{|0\rangle,|+\rangle\}$ while if $z_i=1$ then it is drawn from the set $\{|1\rangle,|-\rangle\}$.

In this protocol we describe the process of building the state $\left|\widetilde{G}\right\rangle$ which meets the conditions of lemma 3.1.1.

Input:

1. Vectors $w\in\{0,1\}^m$ and $z\in\{0,1\}^m$.
2. A description of the graph $\widetilde{G}$
3. The state $|G\rangle$ and a description of G.

Output:

1. The state $\left|\widetilde{G}\right\rangle$ of lemma 3.1.1.

Algorithm:

1. For each vertex b_i generate the state $|b_i\rangle$ depending on the values of w_i and z_i in accordance to the following table.

	$z_i = 0$	$z_i = 1$
$w_i = 1$	$\|+\rangle$	$\|-\rangle$
$w_i = 0$	$\|0\rangle$	$\|1\rangle$

2. Entangle vertices $b_1, ..., b_m$ and the state $|G\rangle$ according to $\widetilde{G}$ and obtain the state $\left|\widetilde{G}\right\rangle$.

As discussed in the proof of Lemma 3.1.1, and using the notation of Algorithm 3.1, the state generated after measuring vertices $b_1, ..., b_m$ in the state $\left|\widetilde{G}\right\rangle$ in the Pauli-Y basis results in the state of expression (3.12). We have used that y_i is the measurement outcome for the qubit at vertex b_i while v_{i_1} and v_{i_2} are the neighbours of b_i in $\widetilde{G}$.

$$\prod_{i=1}^{m} \left(Z_{v_{i_1}}^{z_i} \otimes Z_{v_{i_2}}^{z_i} \right)^{1-w_i} \left(S_{v_{i_1}}^{(-1)^{y_i+z_i}} \otimes S_{v_{i_2}}^{(-1)^{y_i+z_i}} \right)^{w_i} |G\rangle \tag{3.12}$$

3.2 Blind Delegated Graph State Creation

In this section we address problem 2 mentioned in the this chapter's introduction. We develop a protocol used by a computationally weak client to request the generation of a graph state by a powerful server. The protocol does not, however, reveal a full description of the graph to the server. Throughout the reader should keep in mind the intuition introduced at the start of the chapter, which is that the state requested should be secretly moved to from a more general graph state.

We start by formalising three notions mentioned in the paragraph above. Firstly what does it mean to 'secretly' generate a graph state. The measure of what is kept secret from the server during a distributed algorithm is referred to by the terms blindness and leakage [18] defined now.

Definition 3.2.1. *A protocol with input X is* blind while leaking at most $L(X)$ *if the distribution of messages obtained by the server is dependent only on $L(X)$.*

We need, also, to be precise with regards to what is meant by a computationally 'weak' client and 'powerful' server. The clients power will be classical with the exception that they can prepare and send single qubits to the server. They can also receive and send classical information.

The server can receive single qubits and has the power to entanglement, measurement and generate states as they please. They can also send and receive classical information.

Finally, although we remain somewhat vague in this regard, we can discuss further what is meant by a 'general' graph state. For most of this section we make no specific choice of this general graph but note that a br-sup-graph, $\widetilde{G}$, will be regarded as more general than its br-sub-graph G. This is simply because it could have many br-sub-graphs besides G. A 'very general' graph is then one which has many br-sub-graphs.

Although we leave a detailed discussion of the following point to precise implementations of the scheme we develop, we would like to ensure that the graph we begin with, $\widetilde{G}$, is indeed 'very general'. Intuitively, it is this assurance that convinces us that a server, with only the knowledge that the graph state built is described by one of the many br-sub-graph of a graph $\widetilde{G}$, is sufficiently blind. This is simply because it would be harder to guess which graph state was, in fact, built. Figure 3.3 will enlighten the reader on this point.

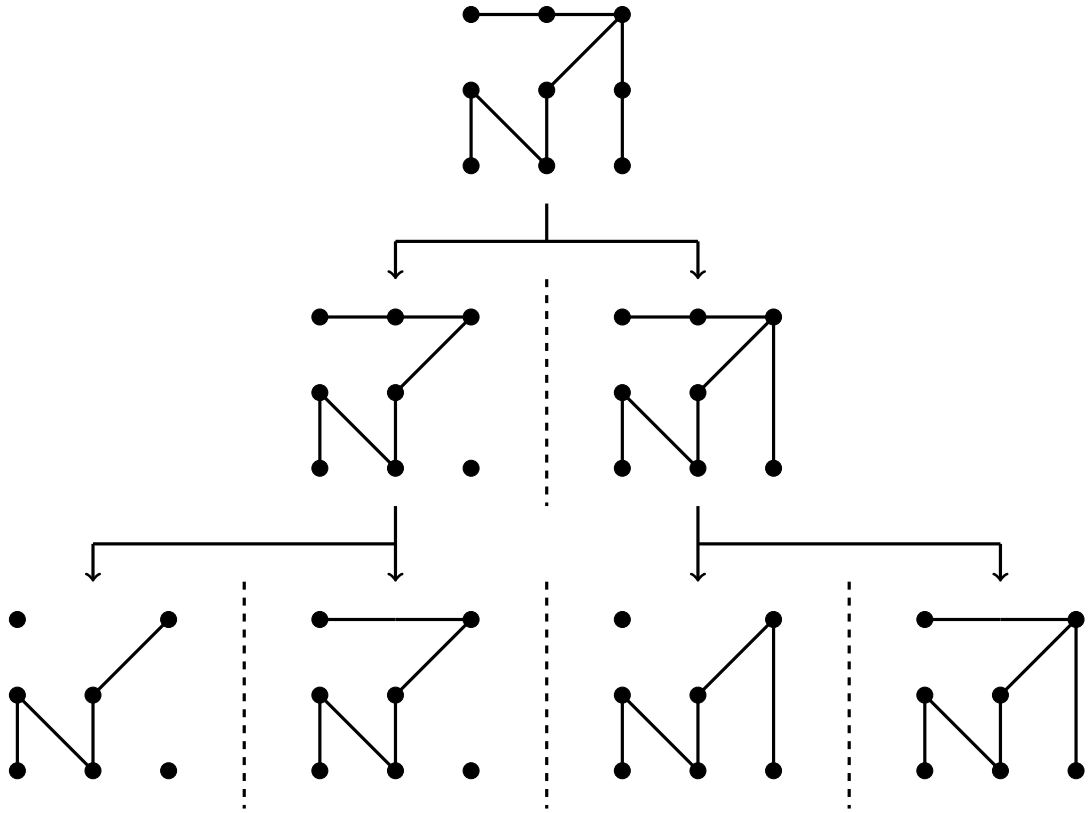

Figure 3.3: An example of the many possible graphs which could result if the only available knowledge was that a bridge or break operation has occurred and not which of the two in fact has.

The reader may wish to skip ahead to Figure 3.8 to see the dotted complete bipartite graph, defined in Definition 3.3.1 and which provides an example of a very general graph.

Throughout the remainder of this section we will appeal, as suggested by the titles of the subsections herein, to abstract cryptography which was introduced in section 1.4. Here the client plays the role of the honest player while the server behaves as the adversary. We proceed by introducing the ideal functionality of the resource we would like to develop.

3.2.1 The Ideal Resource

The protocol we develop is concerned with two graphs. Namely G, which describes the graph state we wish to construct, and $\widetilde{G}$, of which G is a br-sub-graph. The protocol may reveal at most $\widetilde{G}$ to the server. More formally, the ideal blind distributed graph state construction resource, $\mathcal{R}_{gen}$, should be blind while leaking at most $\widetilde{G}$.

As such the ideal resource, $\mathcal{R}_{gen}$, we develop should take the graphs G and $\widetilde{G}$ as inputs from the client and output $\widetilde{G}$ to the server. For no additional information about the form of the graph G to be revealed it must also be assumed that G is equally likely to be any of the br-sub-graphs of $\widetilde{G}$.

The server may wish to perform some deviation from the protocol we request. One can represent this by considering some set of operations F^1 given as input by the server. We will discuss the form of the deviation in greater detail in Section 3.2.2.1 once we have introduced the real resource.

At the end of the protocol the server should retain possession of the final state, ρ_K. This state should consist of a quantum state and a classical binary string, y^1, which we may represent as the quantum state $\left|y^1\right\rangle$. The state may hence be written as in (3.13)

$$\rho_K = \widehat{\rho}_K \otimes \left|y^1\right\rangle \tag{3.13}$$

The client should understand ρ_K to be the state of expression (3.14) after some deviation F^1 is applied and a measurement, of the last m qubits, in the Pauli-Y basis is made. We have that $w,z \in \{0,1\}^m$ and $r \in \{0,1\}^n$, all three of which are known to the client. Here $E_{\widetilde{G}}$ is the entanglement pattern on the qubits $\widetilde{v}_1,...,\widetilde{v}_n,b_1,...,b_m$ as defined by the edges of the graph $\widetilde{G}$.

$$E_{\widetilde{G}} \bigotimes_{i=1}^{n} S^{r_i} |+\rangle \bigotimes_{j=1}^{m} (1-w_j)\left[(1-z_j)|0\rangle + z_j|1\rangle\right] + w_j\left[(1-z_j)|+\rangle + z_j|-\rangle\right] \tag{3.14}$$

This state should be, in the case the server behaves honestly, that of expression

(3.15), where C is a correction and $\left|y^1\right\rangle$ is the state of the last m qubits after measurement.

$$\rho_{K,h} = C\left|G\right\rangle \otimes \left|y^1\right\rangle \tag{3.15}$$

As the correction, C, should be known to the client we require outputs from the protocol be, to the client, the appropriate correction, C, and, to the server, the final state, ρ_K.

None of any of the vectors w, z, r should be known to the server and should each be picked uniformly at random. Hence the ρ_K will be, to the server, that resulting from a measurement, in the Pauli-Y basis, of the last m qubits of equation (3.16) after some deviation, as described by F^1, has been performed. ρ^* is the maximally mixed state.

$$E_{\widetilde{G}} \bigotimes_{i=1}^{n+m} \rho^* \tag{3.16}$$

The above discussion on the different parties inputs, are summarised in Figure 3.4.

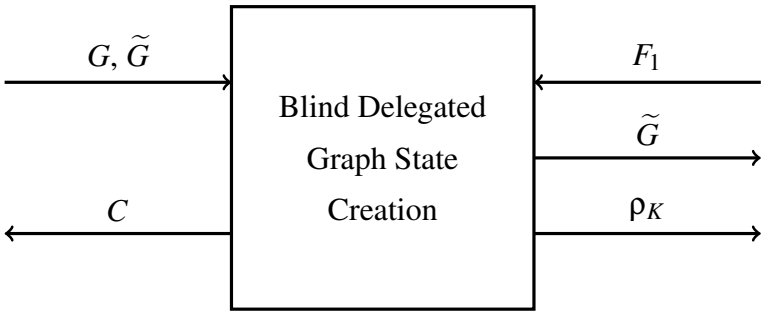

Figure 3.4: Ideal resource, $\mathcal{R}_{gen}$, for a blind delegated graph state creation protocol. G is the graph describing the entanglement pattern in the graph state to be created and is a br-sub-graph of $\widetilde{G}$. C is a unitary corrections to be performed on the output state ρ_K in order to give the state $\left|G\right\rangle$ in the case that the deviation, F_1, is the identity (i.e. the case where the server behaves honestly).

We will finish this section by noting lemma 3.2.1 which was stated less formally in the first paragraph of this section. It follows as, from the server's point of view, the aforementioned Pauli-Y basis measurements are made on the maximally mixed state of expression (3.16) plus any state the server wishes to input or entangle to it themselves. As such the state given as output does not reveal any new information.

Lemma 3.2.1. *The ideal resource, $\mathcal{R}_{gen}$, defined in this section is blind while leaking at most the graph $\widetilde{G}$.*

3.2.2 The Real Resource

In the proof of Lemmas 3.2.2 and 3.2.3 we will show that Algorithm 3.2 defines a real resource, $\mathcal{S}_{gen}$, which is indistinguishable from the ideal blind delegated graph state creation resource, $\mathcal{R}_{gen}$, developed in Section 3.2.1. A pictorial representation of the algorithm can be found in Figure 3.6 and serves as a useful method of comparing to Figure 3.4.

Roughly speaking, the client evolves from a graph state $\left|\widetilde{G}\right\rangle$ to a graph state $|G\rangle$ while the state is in the possession of the server by tricking the them into performing the bridge and break operations of Section 3.1 without revealing which of these two operations they are performing.

To achieve this two things must certainly be true.

1. The states received by the server when a break operation is taking place and when a bridge operation is taking place must be indistinguishable.

2. The operations that the server is asked to perform in both the case that a break operation is to occur and when a bridge operation is to occur must be the same.

Algorithm 3.2 achieves condition 1 by asking the client to send, in the case they want to perform a break operation on a graph state, with equal probability the states $|0\rangle$ and $|1\rangle$ in the place of the break qubit. Similarly the client sends, in the case where they wish to perform a bridge operation on a graph state, with equal probability the states $|+\rangle$ and $|-\rangle$ in the place of the bridge qubit. In this way, all break or bridge qubits look, to the server, like the maximally mixed state. We will denote the states which correspond to the break and bridge qubits by $\rho_1^b, ..., \rho_m^b$.

The states $\rho_1^v, ..., \rho_v^m$ correspond to the vertices $v_1, ..., v_n$ and are sent, with equal probability, in one of any of the four states from the set $\{|+\rangle, S|+\rangle, Z|+\rangle, SZ|+\rangle\}$. It will, in the following section, be explained as to why this choice is necessary. The reader will realise that the state of each qubit, from the point of view of the server, is then the maximally mixed state. This information is expressed in equation (3.17) where $\mathbb{I}_n$ is the identity operator on n qubits and which can be verified by simple matrix expansion.

$$\bigotimes_{j=1}^{n} \sum_{i=0}^{3} \frac{1}{4} S^i |+\rangle \langle +| S^{-i} = \mathbb{I}_n \tag{3.17}$$

As such, all qubits sent to the server look to be in the maximally mixed state, regardless of the operation being performed.

Combined, and followed by the necessary pattern of entanglement, these two sets of qubits build the the maximally mixed state of expression (3.16) presenting some promise for this algorithm.

Condition 2 is met by asking the server to perform a Pauli-Y measurement in both cases. The reader will recall from section 3.1 that the choice of bridge and break qubit is enough then to determine the operation that will be performed. However, not knowing if states from the set $\{|0\rangle, |1\rangle\}$ or states from the set $\{|+\rangle, |-\rangle\}$ were sent means the server cannot know anything about the effect the Pauli-Y measurement they are asked to perform will have. They will only know that either a break or bridge operation has occurred.

So, we are essentially performing, in a distributed way, Algorithm 3.1 with inputs; G, w as is appropriate for the the graph G^1, z chosen uniformly at random, $\widetilde{G}$ and the state of expression (3.18). Notice also that, because of the relation between G and w, choosing G at random from the br-sub-graphs of $\widetilde{G}$ means w will be drawn uniformly at random from its domain.

$$E_G \bigotimes_{i=1}^{n} S^{r_i} |+\rangle \tag{3.18}$$

Before continuing to Algorithm 3.2, where the details of this method lies, we will look to understand how to refer to the deviation F^1 which was introduced earlier. We take the rough understanding of Algorithm 3.2 gained above and conduct the following discussion.

3.2.2.1 The Deviation

Recall the deviation F^1 introduced as part of the ideal resource of section 3.2.1. We model the server's deviation as depicted in Figure 3.5.

We can represent any deviation as an operator F^1 preceding the a unitary operator, $E_{\widetilde{G}}$, controlled by the client. The operator F^1 may undo $E_{\widetilde{G}}$ and apply any operation of its own, including entanglement with its own states.

One can assume that the measurements are made to the correct qubits as they may be commuted by the operator F^1. Since any measurement may be modelled by a unitary operation, absorbed into F^1, followed by local measurements we can also assume any measurement the server wishes is made.

[1] Recall that w has a 0 entry when a break operation is performed on the corresponding qubit and a 1 entry for a bridge. The necessary operations for moving from $\widetilde{G}$ to G then defines w for us.

The reader will also notice that, as hoped, we can represent a honest server, and so the filtered resource, by setting F_1 to be the identity operator.

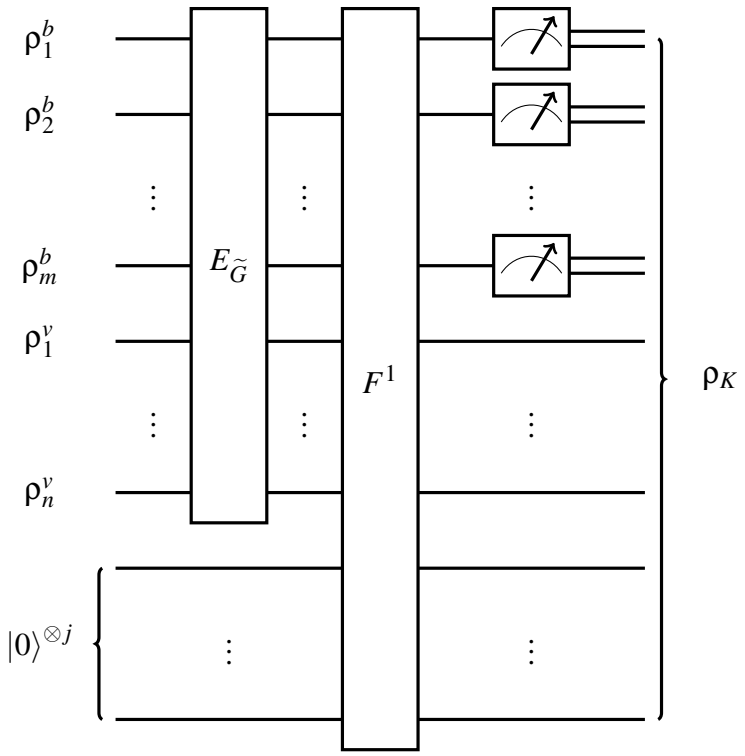

Figure 3.5: The deviation F^1 by a server during the implementation of the real blind delegated graph state creation resource of Algorithm 3.2. The operation F^1 replicates all possible deviations by the server. The operator $E_{\widetilde{G}}$ implements entanglement between the qubits $\widetilde{v}_1,...,\widetilde{v}_n,b_1,...,b_m$ as described by the graph $\widetilde{G}$. Qubits $b_1,...,b_m,\widetilde{v}_1,...,\widetilde{v}_n$ are received from the client while the state $|0\rangle^{\otimes j}$ is built by the server according to the requirements of F^1.

We can now introduce Algorithm 3.2 after which we conduct a discussion about the filtered resource.

Algorithm 3.2. Blind delegated graph state creation.

A client would like to delegate the construction of a graph state $|G\rangle$, up to some known correction C, to a server. The graph G is a br-sub-graph of $\widetilde{G}$. The algorithm is blind while leaking at most $\widetilde{G}$.

The result is the state ρ_K which will, in the case that the server is honest, be the state $C|G\rangle \otimes |y^1\rangle$ where C is some unitary correction and $|y^1\rangle$ is an m qubit Pauli-Y basis state.

We use the notation of w and z as in Algorithm 3.1 while $r_i \in \{0,1,2,3\}$ defies the initial states of the qubit v_i as being the state $S^{r_i}|+\rangle$.

Input:

- **Client:**
  1. A description of the graphs G and $\widetilde{G}$
- **Server:**
  1. The deviation F^1.

Output:

- **Client:**
  1. Corrections, C.
- **Server:**
  1. The graph $\widetilde{G}$.
  2. The final state ρ_K.

Algorithm:

- **Client:**
  1. Generate random string $r \in \{0,1,2,3\}^n$.
  2. Generate random string $z \in \{0,1\}^m$.
  3. Produce the string $w \in \{0,1\}^m$ according to whether a break or a bridge operation is performed to the vertex b_i. As in Algorithm 3.1 this means setting $w_i = 0$ or $w_i = 1$ respectively.
  4. For each vertex v_i initialise the state $S^{r_i}|+\rangle$.
  5. For each vertex b_i initialise the state as described by the table below.

	$z_i = 0$	$z_i = 1$
$w_i = 1$	$\|+\rangle$	$\|-\rangle$
$w_i = 0$	$\|0\rangle$	$\|1\rangle$

  6. Send all prepared qubits to the server.
- **Server:**

7. Receive qubits and $\widetilde{G}$
8. Output the graph $\widetilde{G}$
9. The server now has the necessary inputs to implement the deviated entanglement and measurement operation of section 3.2.2.1. They do so to obtain measurement outcomes y^1.
10. Return y^1 to the client.
11. Output state $\rho_K = \widehat{\rho}_K \otimes \left|y^1\right\rangle$ generated.

- **Client:**

12. Receive y^1.
13. Calculate and output correction.

    If the server is honest, then the final state, $\rho_{K,h}$, will be as in expression (3.19).

$$\left(\prod_{i=1}^{m}\left(Z_{v_{i_1}}^{z_i}\otimes Z_{v_{i_2}}^{z_i}\right)^{1-w_i}\left(S_{v_{i_1}}^{(-1)^{y_i^1+z_i}}\otimes S_{v_{i_2}}^{(-1)^{y_i^1+z_i}}\right)^{w_i}\right)\left(\prod_{j=1}^{n}S_{v_i}^{r_j}\right)|G\rangle\otimes\left|y^1\right\rangle \tag{3.19}$$

    In expression (3.19) we have used the notation that v_{i_1} and v_{i_2} are the neighbours of b_i in the graph $\widetilde{G}$ as used in Algorithm 3.1. We also have that y_i^1 is the measurement outcome for qubit b_i.

    The correction C is then the operator preceding the state $|G\rangle$.

3.2.2.2 The Filtered Real Resource

Recall that the filtered resource, $\mathcal{S}_{gen}\#$ is defined by forcing the servers deviation, F^1, to be the identity and preventing the server from accessing anything other than its inputs and outputs. We now prove lemma 3.2.2 which alludes to part 1 of Definition 1.4.7 which we have referred to as the correctness condition.

Lemma 3.2.2. *The filtered resource, $\mathcal{S}_{gen}\#$, defined in this section is indistinguishable from the filtered real resource, $\mathcal{R}_{gen}\#$, defined in section 3.2.1 when the server behaves honestly.*

Proof. The state knowledge of a distinguisher at the end of the protocol is the total of all of the information obtained throughout. In both cases this can be described by

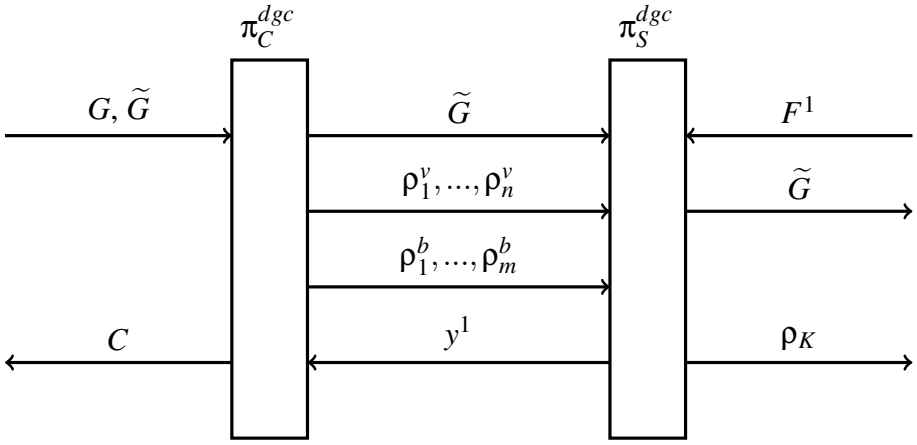

Figure 3.6: Real blind delegated graph state creation resource, $\mathcal{S}_{gen}$. π_C^{dgc} is the client protocol for delegated graph state creation and π_S^{dgc} is the server protocol for this resource. $y^1 \in \{0,1\}^m$ is the measurement outcomes. $\rho_1^v, ..., \rho_n^v$ are initial states of the qubits $v_1, ..., v_n$ and $\rho_1^b, ..., \rho_m^b$ are bridge and break qubits $b_1, ..., b_m$.

the set of variables as seen in 3.20. Here the variables G, C and ρ_K are as previously discussed while $\widetilde{G}_{\text{in}}$ is the description of the graph $\widetilde{G}$ given as input by the client and $\widetilde{G}_{\text{out}}$ is that given as output to the server.

$$G, \widetilde{G}_{\text{in}}, \widetilde{G}_{\text{out}}, C, \rho_K \tag{3.20}$$

Let us begin by recalling how these objects behave in the ideal resource defined in Section 3.2.1. The graphs $\widetilde{G}_{\text{in}}$ and $\widetilde{G}_{\text{out}}$ should match. The state ρ_K should be as in expression (3.15) and the the correction C should be as it appears in that expression. ρ_K should be a measurement of the state of expression (3.14).

In the case of the real protocol $\widetilde{G}_{\text{in}}$ and $\widetilde{G}_{\text{out}}$ certainly match simply by observing the protocol.

The construction of the qubits sent to the server and the deviation ensure us that ρ_K is a measurement on the state of expression (3.14).

Comparing expression (3.19) to (3.15) leaves us satisfied about the remaining condition.

As such the resources $\mathcal{R}_{\text{gen}}\#$ and $\mathcal{S}_{\text{gen}}\#$ are indistinguishable. □

3.2.3 The Simulator

We now move to prove, in the case that the server behaves maliciously, the equivalence of the real resource, $\mathcal{S}_{gen}$, and ideal resource, $\mathcal{R}_{gen}$, defined in Section 3.2.2 and Section

3.2.1 respectively. We draw the reader attention back to Section 1.4 and, in particular, Definition 1.4.7 point 2 to achieve this.

We define a simulator which acts as an interface to the ideal system and produces a indistinguishable interface to that of the real system. The action of the simulator at each stage of the real protocol of Algorithm 3.2 is displayed in Algorithm 3.3 and can be visualised in Figure 3.7.

Algorithm 3.3. Blind delegated graph state creation simulator.

The simulator, σ_{gen}, used to prove the equivalence of the real resource, $\mathcal{S}_{gen}$, of Section 3.2.2 and the ideal resource, $\mathcal{R}_{gen}$, of Section 3.2.1. The simulator acts as an interface to the ideal system and interfaces with the server in a way that is indistinguishable from that of the real system. At each step of the real protocol seen in Algorithm 3.2 we specify the action of the simulator. In all steps not listed below the simulator has no effect on the server.

---

7. - Produce states $\widehat{\rho}^b_1, ..., \widehat{\rho}^b_m$ in the maximally mixed state and send to the server.
   - Produce states $\widehat{\rho}^v_1, ..., \widehat{\rho}^v_n$ in the maximally mixed state and send to the server.
   - Copy graph $\widetilde{G}$ to register.
8. no effect on register
9. - Take in and forward F^1.
   - Receive state ρ_K from ideal system and send to sever.

We can now move to the main result of this section which proves the real resource, $\mathcal{S}_{gen}$, of Section 3.2.2 corresponds to a protocol which is blind while leaking at most the graph $\widetilde{G}$.

Lemma 3.2.3. *The real resource, $\mathcal{S}_{gen}$, defined in section 3.2.2 is indistinguishable from the ideal resource, $\mathcal{R}_{gen}$, of section 3.2.1 when the ideal system is interfaced with through the simulator, σ_{gen}, of Algorithm 3.3.*

Proof. We consider, at each step of Algorithm 3.2, why the state of the real and ideal system are identical. To do so we consider the state that a distinguisher has in their control and note, that in both cases, they are indistinguishable.

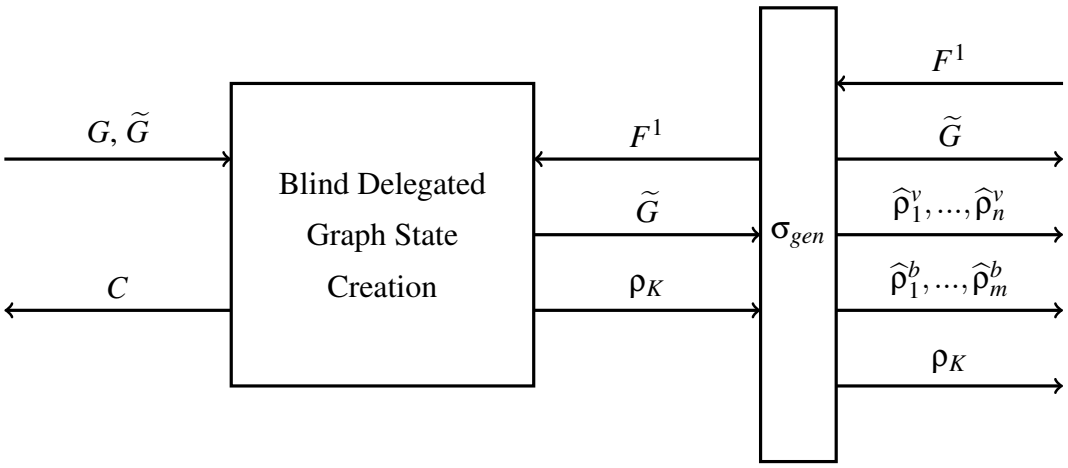

Figure 3.7: Ideal delegated graph creation resource, $\mathcal{R}_{gen}$, with the simulator σ_{gen}. $\widehat{\rho}_1^b, ..., \widehat{\rho}_m^b$ are bridge and break qubits in the maximally mixed state. $\widehat{\rho}_1^v, ..., \widehat{\rho}_n^v$ are stationary qubits in the maximally mixed state. Other notation can be understood by studying Figure 3.4

During steps 1 to 6 the only knowledge the distinguisher has is of the server and clients inputs. Hence the state on their possession is that of expression (3.21).

$$G, \widetilde{G}, F^1 \tag{3.21}$$

At step 7 the server, when interacting with the unfiltered real resource, receives the states $\rho_1^v, ..., \rho_n^v, \rho_1^b, ..., \rho_m^b$. As discussed, the the states at the vertices $v_1, ..., v_n$ are chosen uniformly at random from the set $\{|+\rangle, S|+\rangle, Z|+\rangle, SZ|+\rangle\}$. The states corresponding to vertices $b_1, ..., b_m$ are drawn uniformly at random from either the set $\{|0\rangle, |1\rangle\}$ or the set $\{|+\rangle, |-\rangle\}$. In either case these states are indistinguishable from the maximally mixed states sent by the simulator. As such the state, in both the real and ideal case, is as in expression (3.22).

$$G, \widetilde{G}, F^1, \rho_1^v \otimes ... \otimes \rho_n^v \otimes \rho_1^b \otimes \rho_m^b \tag{3.22}$$

While step 8 has no effect on the distinguisher's state step 9 produces the final state. In the ideal case the output is the deviated measurement of the last m qubits of expression (3.14) as discussed in Sections 3.2.1 and 3.2.2.1. Algorithm 3.2 will reveal this to be the case for the real resource too. Hence, in both the case of the real resource, and the ideal resource interfaced with through a simulator, the total knowledge of the distinguisher is as appears in expression (3.23).

$$G, \widetilde{G}, F^1, \rho_K \tag{3.23}$$

The state of knowledge of the distinguisher does not change until step 13 when the correction is outputted by the client. In the real and ideal cases the outputs are the same and so the final state of knowledge of the distinguisher in both cases is as in expression (3.24).

$$G, \widetilde{G}, F^1, \rho_K, C \tag{3.24}$$

Hence, the state of knowledge of the extinguisher in the real and ideal cases is indistinguishable.

□

With these tools in hand we move to understanding the blind construction of an IQP graph state. From here on, and thanks to Lemma 3.2.3, we can refer to the real and the ideal resource interchangeably.

3.3 IQP Graph State Generation

We draw this chapter to a close by discussing the generation of the IQP graph states defined in Algorithm 2.2. The graph state generation protocol developed in Section 3.2 required two inputs from the server. One of these is the graph, G, describing the entanglement pattern of the graph state we would like to build and another is a graph $\widetilde{G}$ of which G is a br-sub-graph. We already discovered in Section 2.3 that the necessary graph, G, for an IQP computation is the bipartite graph of Definition 2.3.1. We begin this section by defining the graph $\widetilde{G}$ needed for the construction of an IQP graph state. This $\widetilde{G}$ will be in the form of a dotted complete bipartite graph defined now.

Definition 3.3.1. *A* dotted complete bipartite graph *is a graph $\tilde{G} = \left(\tilde{V}, \tilde{E}\right)$ whose vertex set $\tilde{V}$ and edge set $\tilde{E}$ have the following properties. The vertex set $\tilde{V}$ has a partition $\left\{\tilde{P}, \tilde{A}, \tilde{B}\right\}$ such that if $\left|\tilde{P}\right| = n_P$ and $\left|\tilde{A}\right| = n_A$ then $\left|\tilde{B}\right| = n_P n_A$. Labelling elements of the sets in this partition by setting $\tilde{P} = [p_1, ..., p_{n_P}]$, $\tilde{A} = [a_1, ..., a_{n_A}]$ and $\tilde{B} = \left\{b_{(i,j)} : i \in [1, ..., n_P], j \in [1, ..., n_A]\right\}$ gives us the necessary expressive power to define the edge set of $\tilde{G}$. In particular we have the edge set as defined in* (3.25).

$$\begin{aligned} \tilde{E} = &\left\{\left(p_i, b_{(i,j)}\right) : i \in [1, ..., n_P], j \in [1, ..., n_A]\right\} \\ &\cup \\ &\left\{\left(b_{(i,j)}, a_j\right) : i \in [1, ..., n_P], j \in [1, ..., n_A]\right\} \end{aligned} \tag{3.25}$$

The reader may refer to the graph $\widetilde{G}$ in Figure 3.8 for some intuition on the structure of a dotted complete bipartite graph.

We make the following definition to introduce notation which compares a bipartite graph and its dotted complete bipartite counterpart.

Definition 3.3.2. *Define the operator $\sim(\cdot)$ to be that taking a bipartite graph G with a given partition $\{P,A\}$ to a dotted complete bipartite graph $\widetilde{G}$ in the following way. Using the notation in definition 3.3.1 we have, for the image of this operator, $\widetilde{P}=P$, $\widetilde{A}=A$ and $\widetilde{B}=\left[b_{(1,1)},...,b_{(1,n_A)},...,b_{(n_P,1)},...,b_{(n_P,n_A)}\right]$ where $|P|=n_P$ and $|A|=n_A$.*

The edge set $\widetilde{E}$ of $\widetilde{G}$ is that required to meet the conditions of definition 3.3.1 for $\widetilde{G}$ to be a dotted complete bipartite graph.

Although we have used this $\widetilde{G}$ notation before to mean some graph of which G is a br-sub-graph this new notation allows us to be more specific about the exact form of $\widetilde{G}$. From here on we will use this $\widetilde{G}$ notation as described in Definition 3.3.2. We visualise the function defined in Definition 3.3.2 using Figure 3.8. It is clear from this diagram that bipartite graphs are br-sub-graphs of dotted complete bipartite graphs.

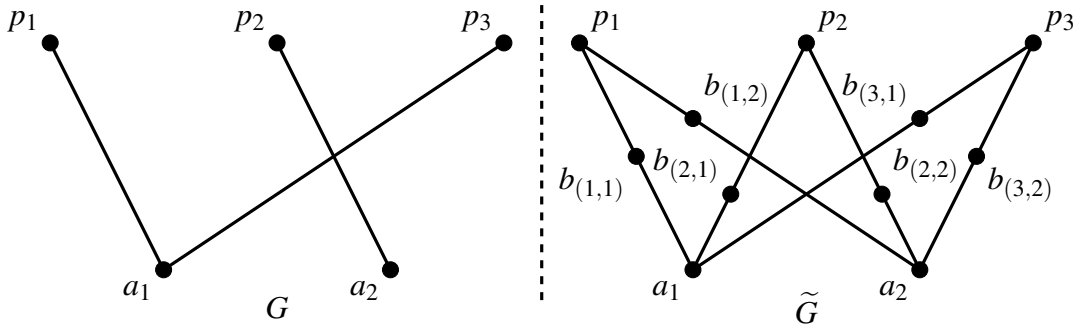

Figure 3.8: An example of the $\sim(\cdot)$ function introduced in Definition 3.3.2. Here $n_P=3$ and $n_A=2$ while the partition used is $P=[p_1,p_2,p_3]$ and $A=[a_1,a_2]$. $\widetilde{G}$ is an example of the dotted complete bipartite graph introduced in definition 3.3.1 with $\widetilde{B}=\left[b_{(1,1)},b_{(1,2)},b_{(2,1)},b_{(2,2)},b_{(3,1)},b_{(3,2)}\right]$.

The reader will now notice that, in fact, the dotted complete bipartite graph of Definition 3.3.1 actually depends only on n_P and n_A. As such we may define a dotted complete bipartite graph independently of the full description of the br-sub-graph it is created from.

Definition 3.3.3. *We use the notation $\widetilde{(n_P,n_A)}$ to refer to a dotted complete graph with $\widetilde{P}=[p_1,...,p_{n_P}]$ and $\widetilde{A}=[a_1,...,a_{n_A}]$ and with $\widetilde{B}$ and $\widetilde{E}$ defined to meet Definition 3.3.1 of being a dotted complete bipartite graph.*

Realising that a br-sup-graph of any bipartite graph can be found in the form of a dotted complete bipartite graph means we now have the necessary client inputs to

blindly create the graph state needed for an IQP computation. When comparing to section 3.2 one may like to note that, using previous notation, $n_A + n_P = n$ and $n_A n_P = m$.

Finally, we make some slight alterations to the blind delegated graph state creation resource. In the case of the IQP computation we talk, rather than about the graph G, about the X-program matrix $\mathbf{P}$. Algorithm 2.2 teaches us about the bijective relationship between the two and so we now talk of inputting $\mathbf{P}$ into the graph generation resource. We then leave it as part of the clients protocol to make the transition to talking about G and proceeding to carry out the graph generation as before.

Before the client would need to input the graph $\widetilde{G}$. In the case of the IQP computation though, the br-sup-graph considered is concerned only with the values n_A and n_P; information which is contained in the description of $\mathbf{P}$. Hence, once again, we leave it to the client to extract this information and construct the necessary graph $\widetilde{G}$ to continue.

Finally, leaking the graph $\widetilde{G}$ to the server is now equivalent to leaking only n_A and n_P as these are the only value on which it depends. We summarise all of these points in the resource seen in Figure 3.9.

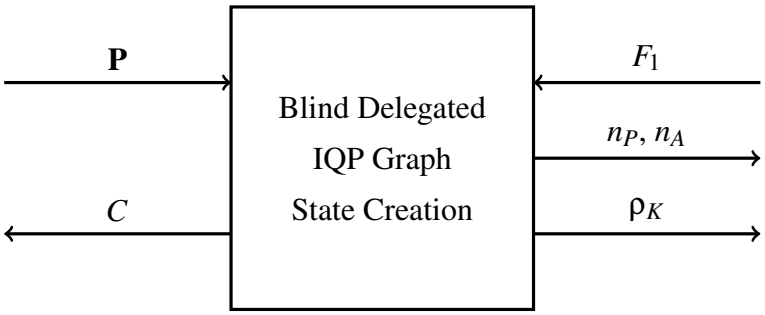

Figure 3.9: Ideal resource, $\mathcal{R}_{geniqp}$, for a blind delegated graph IQP state creation protocol. The notation is as in Figure 3.4 with the following exceptions. $\mathbf{P}$ is the X-program to be implemented. n_P and n_A are the number of primary and ancilla qubits respectively that are needed for the computation.

Chapter 4

A Hypothesis Test

The reader will recall that, in section 2.2, we introduced the IQP hypothesis test of [1]. We understood their method by dividing the problem of performing a hypothesis test into the three conditions, 1.1, 1.2 and 1.3, listed in that section. We identified that, in that case, these three problems translated into condition 2.1, 2.2 and 2.3 were solved by points 3.1, 3.2 and 3.3. We mentioned that the main contribution of our work would be finding a a new solution to condition 1.3 and the discussion of this chapter will focus on that solution.

It is our claim that our solution provides information theoretically secure hiding of the hidden structure as opposed to the computational complexity assumptions used by [1] and recalled in Conjecture 2.2.2. It is the tools developed in Section 3.3 which we use to achieve this. In that section we developed a method of building an IQP graph state without revealing its description to the server. In order to perform a hypothesis test with this resource we must dicuss two things.

1. How to perform a full IQP computation using the blind delegated IQP graph state creation resource

2. Which IQP computation to perform in order to conduct the hypothesis test.

In section 4.1 we address point 1 by extending the the ideas of Section 3.3 to implement a full IQP computation. In Section 4.2 we address point 2 and show that hiding the graph state is enough to successfully fulfil condition 1.3 using a similar X-program to that used by [1].

We will continue to employ the bias technique outlined in Algorithm 2.1 and this again leaves with three conditions for the development of a successful IQP hypothesis test.

4.1 The X-Program send to a server must represent an IQP-hard computation.

4.2 It must be possible for a client, having knowledge of s and the X-program, to calculate the the quantity (2.2).

4.3 The IQP graph state being hidden using the tools of Algorithm 2.1 must conceal the value of s from the server and force them to implement the intended IQP-hard computation.

4.1 The Full Delegated IQP Computation

In this section we will explore how to perform an IQP computation using the blind graph state creation resource developed in Section 3.3. This problem is solved with Algorithm 4.1 but we conduct some discussion before introducing it.

Recall that after implementing the blind delegated IQP graph state creation resource of Section 3.3 the server, in the case they behave honestly, is in possession of the state $\rho_{K,h} = C|\mathbf{P}\rangle \otimes |y\rangle$. The client is in possession of the corrections, C, to the graph state.

To complete the computation measurements should be performed on the state $C|\mathbf{P}\rangle$ but as the corrections, as seen in (3.19), is not known to the server it must be sent to them so that they might correct the measurement basis they use. This is similar to the adaptive measurements of Section 1.3. Sending these new measurement angles should not reveal any new information about the graph state in the possession of the server. We prove that this is the case in the following lemma.

Lemma 4.1.1. *Revealing the correction of expression* (3.19) *gives, to the server, no information about the state in its possession.*

Proof. Consider the correction to be made to the measurement angles of each of the vertices $v_1, ..., v_n$. For a particular vertex v_i this can be derived from (3.19) and is as seen in (4.1). In that expression $\hat{n} = n_A$ in the case that v_i is a primary qubit and $\hat{n} = n_P$ in the case that v_i is an ancilla qubit. $w_{(i,j)}$ and $z_{(i,j)}$ correspond to the vertex $b_{(i,j)}$ and $y^1_{(i,j)}$ is the measurement outcome from measuring that qubit. r_i corresponds to v_i.

$$\left(\prod_{j=1}^{\hat{n}} Z^{\left(1-w_{(i,j)}\right)z_{(i,j)}} S^{w_{(i,j)}(-1)^{y^1_{(i,j)}+z_{(i,j)}}}\right) S^{r_i} \tag{4.1}$$

This might also be written as in expression (4.2).

$$\left(\prod_{j=1}^{\hat{n}} S^{2\left(1-w_{(i,j)}\right)z_{(i,j)}+w_{(i,j)}(-1)^{y^1_{(i,j)}+z_{(i,j)}}}\right) S^{r_i} \tag{4.2}$$

Hence, the number we need to send to the server is that of (4.3) and which informs them of the power to which the S correction should be raised.

$$c_i = \left(\sum_{j=1}^{\hat{n}} 2\left(1-w_{(i,j)}\right)z_{(i,j)} + w_{(i,j)}(-1)^{y^1_{(i,j)}+z_{(i,j)}}\right) + r_i \pmod 4 \tag{4.3}$$

Recalling that the r_i quantity is drawn uniformly at random and so acts as a one-time-pad.

As described by the real resource of section 3.2.2 the server is aware of the value of y^1 but not of w and z. Indeed, it is these quantities which describe the graph and which remain hidden as a result of the one-time-pad described above. No information, then, is revealed about the bracketed component of expression (4.3) and so no information about the structure of the graph is revealed. □

This proof reveals why we chose to initialise stationary qubits the way we did. It was this initialisation that added the one-time-pad.

With Lemma 4.1.1 in hand we can safely assume that we can communicate the measurement corrections to be performed to the server, and so prepare to complete an IQP computation, without the server learning anything about the state in their possession. Note that this only applies up to the point of the measurement and the measurement result itself may reveal some information about the graph state.

In reality, the server may, once again, choose to deviate from performing the measurements we request and we model this with the input F^2. On this occasion the deviation simply occurs before the measurement and may perform any behaviour within the power of the server.

We end this section with Algorithm 4.1 which can be visualised in Figure 4.1 and which uses the techniques discussed in this section.

Algorithm 4.1. Delegated IQP computation

Here we describe an algorithm to delegate an IQP computation to a powerful server. This algorithm uses the blind delegated graph state creation resource of Section 3.2. We continue to use the notation w, z and r as it was used in Algorithm 3.2. It is assumed that the X-program measurement angle, θ, is publicly known.

Input:

- **Client**
  1. The X-program $(\theta, \mathbf{P})$.
- **Server**
  1. Their deviation $F = \{F_1, F_2\}$.

Output:

- **Client:**
  1. The measurement outcomes, y^2.
- **Server:**
  1. The dimensions of the graph state n_A and n_P.
  2. The state ρ_K^2.

Algorithm:

- **Client:**
  1. Call the Blind Delegated Graph State Creation resource with input $\mathbf{P}$.
- **Server**
  2. Receive n_A and n_P.
  3. Input deviation F^1 into Blind Delegated Graph State Creation resource.
  4. Receive ρ_K.
- **Client:**
  5. Receive corrections C.
  6. Send, to the sever, the corrections, $c_1, ..., c_{n_A+n_P}$, to the measurement angles. They are sent in the form of the power to which an S operator should be raised and are as in expression (4.3).
- **Server:**

7. Input deviation F^2 and implement deviated measurement to receive outcome $\rho^2 = \widehat{\rho}^2 \otimes \left|y^2\right\rangle$.
8. Return measurement results, y^2 to the client.
9. Output ρ^2

- **Client:**

10. Receive and output y^2.

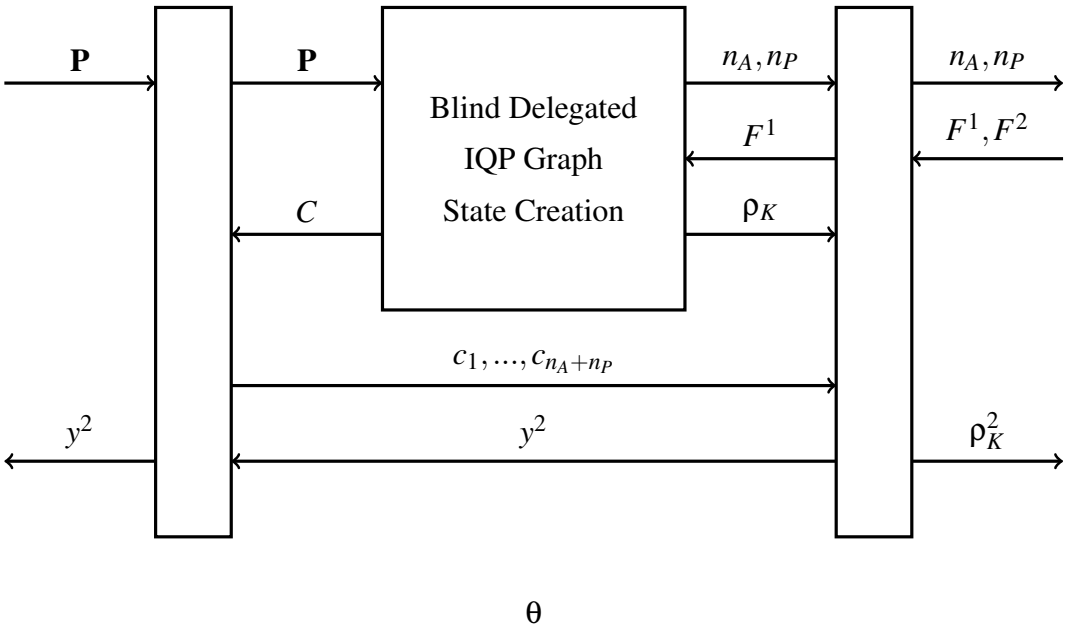

Figure 4.1: Resource for a delegated IQP computation. The notation can be understood by recalling Figure 3.4 and Algorithm 4.1

So we now understand how to implement an IQP computation without revealing anything about the IQP graph state used before it is measured. We move, with these tools in hand, to show how this can be used to hide the secret structure mentioned in condition 1.3. In doing so we discuss and formalise our hypothesis test.

4.2 Our Protocol

We conclude this chapter by introducing and discussing our hypothesis test. The algorithm itself is described in Algorithm 4.3 while it can be visualised in Figure 4.2. You will notice, in the figure and the algorithm, that a hypothesis X-program generation resource is used. In the case of the hypothesis test used by [1] a similar resource is used

and was recalled by us in Algorithm 2.1. We require a slightly different and somewhat simpler resource which we define in Algorithm 4.2.

Algorithm 4.2. IPQ Hypothesis Test X-program Generation.

Here we outline the construction of an X-program which will be used in the hypothesis test that we develop.

Input:

1. q (a prime) chosen so that $q+1$ is a multiple of 8.

Output:

1. X-program, $(\theta, \mathbf{P}) \in [0, 2\pi] \times \mathbb{F}_2^{q \times \frac{q+1}{2}}$.
2. A vector, $\mathbf{s} \in \{0,1\}^{\frac{q+1}{2}}$.

Algorithm:

1. Generate the generator matrix, $\mathbf{G} \in \mathbb{F}_2^{q \times \frac{q+1}{2}}$, for a quadratic residue code, over $\mathbb{F}_2$, of length q.
2. To this matrix, the client appends an additional column consisting entirely of ones. Call this new matrix $\widehat{\mathbf{P}} \in \mathbb{F}_2^{q \times \left(\frac{q+1}{2}+1\right)}$.
3. Reorder the rows of $\widehat{\mathbf{P}}$ randomly.
4. Add multiples of columns of this new matrix to one another randomly to obtain $\mathbf{P}$.
5. Output the X-program $\left(\frac{\pi}{8}, \mathbf{P}\right)$.
6. Suppose this reordering of rows and adding of columns can be achieved by applying the matrix $\mathbf{A} \in \mathbb{F}_2^{\left(\frac{q+1}{2}+1\right) \times \left(\frac{q+1}{2}+1\right)}$. Return the vector of expression 4.4.

$$\mathbf{s} = (0, ..., 0, 1)\, \mathbf{A}^{-T} \in \{0,1\}^{\frac{q+1}{2}} \tag{4.4}$$

The reader will notice that the vector **s** has been designed to be non-orthogonal to all rows of this matrix. Hence the matroid it defines by building the matrix $\mathbf{P_s}$, as discussed in Section 2.2 (i.e. by taking all rows non-orthogonal to **s**), is the same as the matroid constructed in the hypothesis test of Algorithm 2.1. In particular, this means that the bias value is the same as in that case.

The resource developed in Algorithm 4.2 is used by the client to generate the X-program that will be used during the hypothesis test. We introduce this in Algorithm 4.3 now and note that it might be visualised by looking at Figure 4.2.

Algorithm 4.3. An algorithm to perform an IQP hypothesis test.

We describe our IQP Hypothesis test. The output of this test is the value 0 or 1 depending on of the value returned to the client is non-orthogonal or orthogonal respectively to the s value held by the client. The expected value of the output is then the bias.

Input:

- **Client:**
  1. q (a prime) chosen so that $q+1$ is a multiple of 8.
- **Server:**
  1. Deviation $F = \{F_1, F_2\}$.

Output:

- **Client:**
  1. orthogonal $0, 1$
- **Server:**
  1. n_A, n_P.
  2. ρ^2.

Algorithm:

- **Client:**
  1. Call Hypothesis Test X-program Generation resource with input q.

2. Retain **s** and forward **P** to the delegated IQP computation resource.

- **Server:**

1. Receive n_A and n_P.
2. Input deviation $F = \{F_1, F_2\}$ into the Delegated IQP Computation resource.
3. Receive ρ^2

- **Client:**

1. Receive y^2 from the delegated IQP computation resource.
2. Check orthogonality to **s** and output test result. If the y^2 is orthogonal to **s** output 1 and output 0 otherwise.

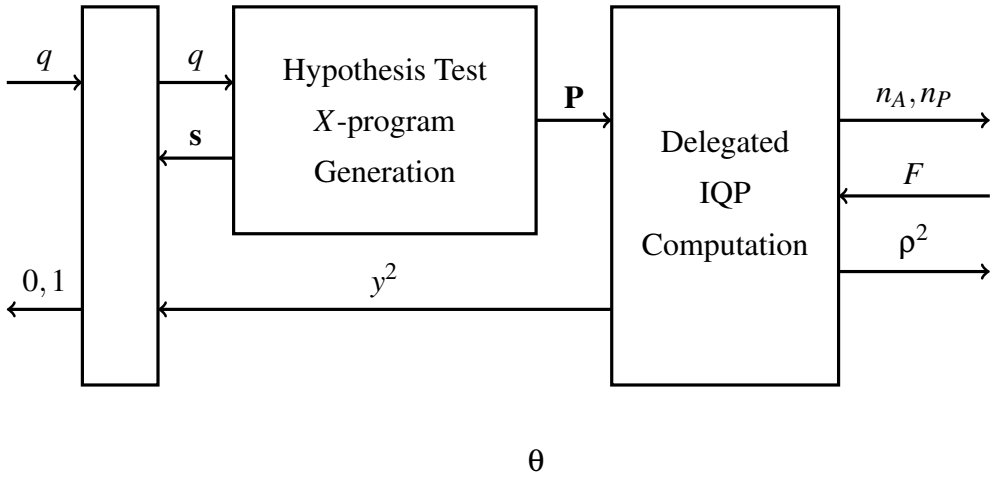

Figure 4.2: Our hypothesis test as described in Algorithm 4.3. The resource uses the Hypothesis Test X-program Generation resource of Algorithm 4.2 and the Delegated IQP Computation resource of Algorithm 4.1. The notation used in this figure is explained in those algorithms.

By using the resource of Algorithm 4.3 multiple, but a constant number of, times and checking the probability of orthogonality we will have completed our hypothesis test.

The reader will notice the similarities of our method to that of [1] so it is left to understand how this method solves conditions 1.1, 1.2 and 1.3 and in what way our solution improves on that of [1]. We outlined the conditions that our hypothesis test must fulfil in points 4.1, 4.2 and 4.3, and we address them each now.

4.1 The X-Program send to a server must represent an IQP-hard computation.

The analysis here is similar to that used before for the hypothesis test of [1]. We discussed, in Section 2.2, how an asymptotically optimal classical simulation of an IQP machine would fail to reproduce the same bias statistics as an IQP machine. Notice that this can only be made harder by the fact that, in our case, the server does not even know the full description of the X-program.

We still have the same problem as in [1] that the particular problem we are asking the server to implement may not be IQP-hard, just outside of classical. However, it may be the case that by completely hiding the X-program, there could be an extension of this work to implementing an IQP-hard problem.

This point is the only one remaining which relies on any conjectures, namely Conjecture 2.2.1, and so strengthening this should certainly be the subject of future work.

4.2 It must be possible for a client, having knowledge of $\mathbf{s}$ and the X-program, to calculate the the quantity (2.2).

That the bias value of the X-program that we developed in this case is the same as that developed in the case of hypothesis test discussed in Section 2.2 was mentioned after the description of Algorithm 4.2. Hence, we know once again, that this condition is met.

4.3 The IQP graph state being hidden using the tools of Algorithm 2.1 must conceal the value of $\mathbf{s}$ from the server.

It is to the strengthening of this condition which our work contributes the most. Up to the point of the second round of measurement the only things learnt by the server are the quantities n_A, n_P. These quantities are the same for all values of $\mathbf{s}$ as the graph depend only on the size of $\mathbf{s}$ and on nothing else.

The construction of $\mathbf{s}$, as seen in Algorithm 4.2, is such that each value of $\mathbf{s}$ is generated with equal probability. As such, and without any structure to the distribution from which the $\mathbf{s}$ quantities are drawn, the server cannot make any predictions as to the value of $\mathbf{s}$.

It could be the case that the measurement result will reveal to the server some information about the graph state or the value of $\mathbf{s}$ as, unlike in the first round of measurements, not all measurement outcomes are equally likely. This would seem not to be a problem though as the only way to extract useful information from a measurement

would be to have the power to perform IQP computations, hence passing the test any way. We claim then to have met this condition.

A final complication which we mention now relates to the blind delegates IQP graph state creation resource of Section 3.3. It was mentioned there that the graph, G, given as input to the resource should be, with equal probability, any one of the br-sub-graphs of the dotted complete bipartite graph $\widetilde{G}$. In the case of the QR code used in Algorithm 4.2 it is not clear that the randomisation that takes place in that algorithm is enough to ensure this is true and it should be the subject of future work to clarify this.

The X-programs are, however, drawn with equal likelihood from the set of possible hypothesis test X-programs (i.e. those that could be built by Algorithm 4.2). It seems likely then that the same hiding analysis could be used in the case where the client gives away that they are performing a hypothesis test. With a more restricted design of $\widetilde{G}$ and the same graph generation resource of section 3.2 similar security could be achieved, again providing new lines of enquiry.

One could also correct this by implementing setting hypothesis test X-programs amongst other X-programs so that the graph states requested are picked uniformly at random from the br-sub-graphs of $\widetilde{G}$. This will, of course, increase the number of run required.

4.3 A Comparison With [1]

Although we have discussed the advantage of our work as being the contribution to condition 4.3 we note also two disadvantages. By using the blind distributed graph state generation resource we have fundamentally tied ourselves to the MBQC architecture of quantum computing. Although not a conceptual problem, it may be the case that, perhaps for engineering reasons, this is not the architecture adopted in the long term. This is not a problem in the hypothesis test of Section 2.2 as they do not specify an architecture to be used.

A second possible draw back is the two step process that is used to complete our hypothesis test. The reader will notice that we require two rounds of measurement to implement our test; one for the bridge and break operations and one for the implementation of the hypothesis test X-program. In the case of the hypothesis test of Section 2.2 this is not the case and it may be implemented in a single round of measurements. It seems that the hypothesis test of Section 2.2 better exploits the property that the operations performed in an IQP computation should commute. In that way

we are demanding more from the server as previously they were not required to store the quantum states for as long a period, for example. The number of rounds of measurements that we request is, however still constant and so it seems to be a modest additional requirement.

Chapter 5

Future Work and Conclusion

Other approaches to performing hypothesis tests have also been successfully developed. The recent work of [39, 40, 41] are relevant and we are now in a position to discuss, understand and compare this work to ours. We do so in Section 5.1 as it provides us with the necessary background to understand proposals for future work made in Section 5.2. We bring this work to a close with a concluding discussion in Section 5.3.

5.1 Related Work

The work of [39] focusses, not on hypothesis tests, but verification of correctness as was the case in in [18]. It may be argued that the main conceptual difference between these two pieces of work is that, while [18] relies on blindness to perform verification, [39] does not.

It is the blindness used in [18], and in this work, which forces the requirement for interaction between the server and client during these verification protocols. The method of achieving blindness also ties these works to using the MBQC architecture. By avoiding the use of blindness this interactiveness and the restriction to MBQC is avoided in [39].

[40] extends the work of [39] from requiring many servers to one and [41] uses a similar method to perform a hypothesis tests on an IQP machine. Their methods are broadly speaking, based on the same technique which we can now summarise.

The local Hamiltonian problem is a decision problem relying on the promise that the energy of a ground state of some given Hamiltonian is above some value or below some smaller value. The problem is to decide which is the case. It is also known [42]

that, for each quantum circuit, one can construct a Hamiltonian whose ground state energy is below some threshold if a specified qubit from the output of the circuit is $|1\rangle$ with high probability. It is in this way that any quantum circuit may be written as a local Hamiltonian problem. In [39] the authors develop a quantum circuit used to perform verification of a computation which is encoded as such a local Hamiltonian problem.

The server is asked to solve the problem as well as to produce a witness for this local Hamiltonian problem. The client can then verify the witness state by performing a set or random measurements upon it. Importantly the verification part can occur at any time after the computation has been performed which may also be seen as another advantage which could not be achieved by the interactive technique of our work.

While we have noted the key difference between this method and our work is the lack of explicit usage of blindness it is interesting too to note the similarities. The techniques used in our work have four steps in common with the work of [39, 40, 41] which we call 'insert randomness', 'server computation', 'send quantum information', 'send classical information'. The difference between the two methods is the order in which these tools are used. For us the order and use of these tools are as follows.

1. *Insert randomness*: Some randomness is added to the qubits generated.

2. *Send quantum information*: The qubits are sent and describe the graph.

3. *Server computation*: The server implements the required measurements.

4. *Send classical information*: The measurement results and corrections are communicated.

In the case of [39, 40, 41] the order is as follows.

1. *Send classical information*: The description of the states to be generated is sent.

2. *Server computation*: The necessary states are created and computations performed.

3. *Send quantum information*: The outputs of the circuit and the witness states are returned.

4. *Insert randomness*: Random measurements are made.

It is interesting that by invoking the randomness towards the end of the computation [39] are able to reduce the number of steps that the procedure takes to complete. In our case this was two while in their case it is just one. Not including randomness until the end means the server does not need help completing the computation as it is being performed and so there is no interaction during the test.

It does seem, however, that by adding the randomness at the end one does introduce a level of 'blindness' to where the computation will be tested. As such the server must obey the computation everywhere. It is not blindness to any aspect of the computation itself which is the kind of blindness we have seen before.

The main draw back of their work is that they require a number of repeats, exponential in the size of the inputs unlike in our case where it is constant.

5.2 Future Work

Continuing on from the discussion of Section 5.1, it would be interesting to understand if it would be possible to reorder the series of events as listed above in order to combine our method and the local Hamiltonian method and achieve the 'best of both worlds' scenario. It could be an interesting avenue of exploration to try to understand if some tools of the local Hamiltonian technique could be added to our bias technique in order to reduce the number of measurements to one round while keeping the other benefits we have achieved.

We also discussed, towards the end of section 4.2, that there is future work to be done in addressing conditions 4.1, 4.2 and 4.3. In particular condition 4.1 remains the only condition relying on conjectures and should be the focus of future developments. The Development of X-programs sampled truly at random, as was discussed after point 4.3 in Section 4.2 should also be investigated further.

It also seems likely that, by altering the X-program used in the hypothesis test, one could reduce the number of qubits needed to perform this hypothesis test. Currently we need $n_A n_P$ bridge and break qubits but reducing this to a logarithmic function of the program size would be an exciting advance.

As well as improving the hypothesis tests on IQP machines it would be of interest to extent these ideas to other forms of quantum computers. In particular the boson sampling machine [10] whose problem set is contained in those solvable by an IQP machine.

5.3 Conclusion

We now summarise the concluding remarks make in the previous sections. In this work we have strengthened on conjectures made in [1] with regard to their IQP hypothesis test. We have developed tool, which is fundamentally linked to measurement based quantum computing, for the blind creation of graph states while only revealing the structure of some graph more general than the graph describing the IQP computation. We also discussed how to present a server with the necessary information to perform a complete IQP computation without giving up any more information than this general graph.

We hope to extend this work to develop a hypothesis test which removes all remaining conjectures regarding the successfulness of such a hypothesis test. It seem the most stubborn problem is to ensure that the server is asked to perform a IQP-hard computation. We also hope that we might be better able to utilise the instantaneous nature of the IQP architecture by designing a hypothesis test only requiring one round of operations and no interaction between the server and the client during the process.

Bibliography

[1] Dan Shepherd and Michael J. Bremner, *Temporally Unstructured Quantum Computation*, Proc. R. Soc. A 465, 1413–1439, 20 February 2009

[2] Richard P. Feynman, *Simulating Physics with Computers*, Int. J. Theor. Phys. 21, 467–488, 1982

[3] I. M. Georgescu, S. Ashhab and Franco Nori, *Quantum Simulation*, Reviews of Modern Physics, 2014

[4] Michael A. Nielsen, Isaac L. Chuang, *Quantum Computing and Quantum Simulation*, 2000

[5] Peter W. Shor *Poly-Time Algorithms for Prime Factorisation and Discrete Logarithms on a Quantum Computer*, SIAM journal on computing 26.5 (1997): 1484-1509

[6] Lov. K. Grover, *A fast quantum mechanical algorithm for database search* Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. ACM, 1996.

[7] Bennett, Charles H., and Gilles Brassard, *Quantum cryptography: Public key distribution and coin tossing* Theoretical Computer Science 560 : 7-11, 2014

[8] E. Knill and R. Laflamme, *Power of One Bit of Information*, Physical Review Letters 81.25 (1998): 5672,

[9] Tomoyuki Morimae, Keisuke Fujii and Joseph F. Fitzsimons *Hardness of Classically Simulating the One-Clean-Qubit Model*, Physical review letters, 2014

[10] Bryan T. Gard, Keith R. Motes, Jonathan P. Olson, Peter P. Rohde and Jonathan P. Dowling, *An Introduction to Boson-Sampling*, arXiv preprint arXiv:1406.6767, 2014

[11] P. Aliferis, F. Brito, D. P. DiVincenzo, J. Preskill, M. Steffen and B. M. Terhal, *Fault-tolerant computing with biased-noise superconducting qubits: a case study*, New J. Phys. 11, 013061, 20 January 2009

[12] Michael J. Bremner, Richard Jozsa and Dan J. Shepherd *Classical Simulation of Commuting Quantum Computations Implies Collapse of the Polynomial Hierarchy*, Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences. The Royal Society, 2010

[13] Bremner, Michael J., Ashley Montanaro, and Dan J. Shepherd. *Average-case complexity versus approximate simulation of commuting quantum computations* arXiv preprint arXiv:1504.07999 (2015).

[14] Raussendorf, Robert, and Hans J. Briegel. *A one-way quantum computer.* Physical Review Letters 86.22 (2001): 5188.

[15] Raussendorf, Robert, Daniel E. Browne, and Hans J. Briegel. *Measurement-based quantum computation on cluster states.* Physical review A 68.2 (2003): 022312.

[16] Danos, Vincent, Elham Kashefi, and Prakash Panangaden *The Measurement Calculus* Journal of the ACM (JACM) 54.2 (2007): 8.

[17] Broadbent, Anne, Joseph Fitzsimons, and Elham Kashefi *Universal Blind Quantum Computation* Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on. IEEE, 2009.

[18] Joseph F. Fitzsimons and Elham Kashefi, *Unconditionally Verifiable Blind Quantum Computation*, arXiv preprint arXiv:1203.5217, 2012

[19] Pretros Walden, *University of Edinburgh Introduction to Quantum Computing course lecture notes.* 2016, https://qcintro.wordpress.com/lectures/

[20] Alastair I.M. Rae *Quantum Mechanics* Institute of Physics, Bristol, England, 1992

[21] Einstein, Albert, Boris Podolsky, and Nathan Rosen. *Can quantum-mechanical description of physical reality be considered complete?* Physical review 47.10 (1935): 777.

[22] Edward Farhi, Jeffrey Goldstone, Sam Gutmann and Michael Sipser, *Quantum Computation by Adiabatic Evolution*, arXiv preprint quant-ph/0001106, 2000

[23] MacKay, David JC. *Information theory, inference and learning algorithms* Cambridge university press, 2003.

[24] Ekert, Artur K. *Quantum cryptography based on Bell's theorem* Physical review letters 67.6 (1991): 661.

[25] König, Robert, et al. *Small accessible quantum information does not imply security* Physical Review Letters 98.14 (2007): 140502.

[26] Renner, Renato. *Security of quantum key distribution* International Journal of Quantum Information 6.01 (2008): 1-127.

[27] Pfitzmann, Birgit, and Michael Waidner. *Composition and integrity preservation of secure reactive systems* Proceedings of the 7th ACM conference on Computer and communications security. ACM, 2000.

[28] Canetti, Ran. *Universally composable security: A new paradigm for cryptographic protocols* Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on. IEEE, 2001.

[29] Ben-Or, Michael, and Dominic Mayers. *General security definition and composability for quantum and classical protocols* arXiv preprint quant-ph/0409062 (2004).

[30] Unruh, Dominique. *Simulatable security for quantum protocols* arXiv preprint quant-ph/0409125 (2004).

[31] Maurer, Ueli, and Renato Renner. *Abstract cryptography* In Innovations in Computer Science. 2011.

[32] Portmann, Christopher, and Renato Renner. *Cryptographic security of quantum key distribution* arXiv preprint arXiv:1409.3525 (2014).

[33] Dunjko, Vedran, et al. *Composable security of delegated quantum computation* International Conference on the Theory and Application of Cryptology and Information Security. Springer Berlin Heidelberg, 2014.

[34] Kashefi, Elham, and Anna Pappa. *Blind Multiparty Quantum Computing.* arXiv preprint arXiv:1606.09200 (2016).

[35] Hoban, Matty J., et al. *Measurement-based classical computation.* Physical review letters 112.14 (2014): 140505.

[36] J. G. Oxley *Matroid Theory.* Oxford University Press, 2011

[37] Papadimitriou, Christos H. *Computational Complexity* John Wiley and Sons Ltd., 2003.

[38] Barz, S., Kashefi, E., Broadbent, A., Fitzsimons, J. F., Zeilinger, A., Walther, P. *Demonstration of blind quantum computing* Science 335.6066 (2012): 303-308.

[39] Fitzsimons, Joseph F., and Michal Hajdušek. *Post hoc verification of quantum computation* arXiv preprint arXiv:1512.04375 (2015).

[40] Morimae, Tomoyuki, and Joseph F. Fitzsimons. *Post hoc verification with a single prover* arXiv preprint arXiv:1603.06046 (2016).

[41] Hangleiter, D., et al. *Direct certification of a class of quantum simulations* arXiv preprint arXiv:1602.00703 (2016).

[42] Kempe, Julia, Alexei Kitaev, and Oded Regev. *The complexity of the local Hamiltonian problem.* SIAM Journal on Computing 35.5 (2006): 1070-1097.